TIBCO® Managed File Transfer Platform Server for z/OS

User's Guide

Version 8.1.0
August 2021

Contents

Data Encryption and Compression

TIBCO® Managed File Transfer Platform Server for z/OS supports both data encryption and data compression within a file transfer.

The Platform Server performs both data encryption and data compression "on the fly". This means that the Platform Server encrypts and compresses data while sending the data to a remote Platform Server destination. And likewise, the remote Platform Server decrypts and decompresses the data when the data is received. No special APIs, preprocessing steps, or postprocessing steps are required for the Platform Server to encrypt or compress data.

Both data encryption and compression can be performed during a single file transfer.

See the following introductions for more details about data encryption and data compression on TIBCO Managed File Transfer (MFT) Platform Server for z/OS:

- Data Encryption
- Data Compression

Data Encryption

You can use TIBCO MFT Platform Server for z/OS to encrypt data while sending the data to a remote destination. The data is then decrypted when it is received at the remote destination.

On TIBCO MFT Platform Server for z/OS, by using the ENCRYPT parameter, you can specify whether to use data encryption, or specify the encryption algorithm to be used for a transfer. However, the Platform Server always encrypts all passwords by using a proprietary encryption algorithm. If you specify an encryption algorithm (Data Encryption Standard, Blowfish, or AES), the Platform Server encrypts the password a second time by using the specified encryption algorithm, and optionally encrypts the data to be transferred. If no encryption algorithm is specified, the Platform Server still encrypts the passwords by using its proprietary encryption algorithm.

> **Note:** No special APIs, preprocessing tasks, or postprocessing steps are required for the Platform Server to encrypt data.

TIBCO MFT Platform Server for z/OS supports the following encryption algorithms:

- Data Encryption Standard (DES)
- Blowfish
- AES

For more information, see Data Encryption Algorithms.

TIBCO MFT Platform Server for z/OS provides the following methods for you to specify an encryption algorithm by using the ENCRYPT parameter:

- GLOBAL definition
- NODE definition
- Transfer Interface definition

For more information, see Defining Data Encryption.

Data Encryption Algorithms

TIBCO MFT Platform Server for z/OS supports three data encryption algorithms: Data Encryption Standard (DES), Blowfish, and AES.

DES Encryption

DES is a block encryption algorithm working in chunks of 8 bytes. The DES key length is fixed at 56 bits. An offshoot of the DES encryption algorithm is Triple DES (3DES), which effectively provides 112-bit encryption by a combination of encryption and decryption functions.

The advantage and disadvantages of DES are as follows:

- Advantage of DES
    - This algorithm is an established standard that has withstood the test of time.
- Disadvantages of DES
    - This algorithm is CPU intensive; therefore, it might slow down file transfers if CPUs are busy or slow. 3DES uses almost three times the CPU cycles that DES uses.
    - The 56-bit key length of the DES encryption algorithm is considered insufficient based on the current computer ability.

Blowfish Encryption

Blowfish is a block encryption algorithm that can use keys from 32 to 448 bits long. The Platform Server uses a 56-bit key if `ENCRYPT=BLOWFISH` or `ENCRYPT=BF` is specified, and uses a 448-bit key if `ENCRYPT=BLOWFISH_LONG` or `ENCRYPT=BFL` is specified.

The advantages and disadvantage of Blowfish are as follows:

- Advantages of Blowfish
    - This algorithm is about six times faster than DES and about fifteen times faster than 3DES.
    - The key length can range from 32 bits to 448 bits.
- Disadvantage of Blowfish
    - This algorithm has not been tested by as many attacks as DES.

AES Encryption

AES is a symmetric block encryption algorithm that uses a key length of 256 bits.

The advantages and disadvantage of Rijndael are as follows:

- Advantages of AES
    - This algorithm is selected as the Advanced Encryption Standard (AES) by the US government.
    - This algorithm supports 256-bit key length.
    - This algorithm is more efficient than DES or 3DES.
- Disadvantage of AES
    - This algorithm is slightly less efficient than Blowfish or Blowfish Long.

Defining Data Encryption

You can use the `ENCRYPT` parameter to specify whether to encrypt data or not, or specify the encryption algorithm for file transfers.

> **Note:** If you do not specify the encryption algorithm to use, the Platform Server still encrypts the password by using its proprietary encryption algorithm.

You can specify the encryption algorithm by using the following methods:

- GLOBAL Definition: Specifies a default encryption algorithm for the entire system.
- NODE Definition: Specifies a default encryption algorithm for a particular node. NODE definition overrides any GLOBAL definitions for a particular node.
- Transfer Interface Definition: Specifies the definitive encryption algorithm for a single transfer request. Transfer Interface definition overrides any GLOBAL or NODE definitions for a particular transfer.

GLOBAL Definition

By using GLOBAL definition, you can specify a default encryption algorithm for the entire system.

Note: GLOBAL definitions are used only when they are not overridden by NODE and Transfer Interface definitions.

The format of GLOBAL definition is as follows:

```
ENCRYPT={NONE | AES | AES128 | DES | 3DES | BLOWFISH | BLOWFISH_LONG |
AES},PASSONLY
```

ENCRYPT={NONE | AES | AES128 | DES | 3DES | BLOWFISH | BLOWFISH_LONG | AES},PASSONLY

This parameter specifies the level of encryption that is used by default for the entire system. The following table lists the description of each parameter option:

Option	Alternate Specification	Description
NONE		No encryption is used. This is the default setting.
AES		AES 256-bit encryption is used.
AES128		AES 128-bit encryption is used.
DES		DES encryption is used. For more information of DES encryption, see DES

Option	Alternate Specification	Description
		Encryption.
3DES		Triple DES encryption is used.
BLOWFISH	BF	Blowfish 56-bit encryption is used. For more information of Blowfish encryption, see Blowfish Encryption.
BLOWFISH_ LONG	BFL	Blowfish 448-bit encryption is used.
AES	RJ/AES	AES 256-bit encryption is used. For more information of AES encryption, see AES Encryption.

When the PASSONLY operand is used, only the password is encrypted by using the specified encryption algorithm. The data is not encrypted.

> **Note:** The PASSONLY operand is only valid for file transfers between z/OS and z/OS systems.

Examples Using GLOBAL Definition

The following examples show how to use GLOBAL definition to specify a default encryption algorithm for the entire system.

Example 1:

```
ENCRYPT=NONE
```

The Platform Server encrypts the password by using a proprietary algorithm. Data is not encrypted.

Example 2:

```
ENCRYPT=BLOWFISH_LONG or ENCRYPT=BFL
```

The Platform Server encrypts the password and data by using Blowfish 448-bit encryption algorithm.

Example 3:

```
ENCRYPT=DES,PASSONLY
```

The Platform Server encrypts the password by using DES encryption algorithm. Data is not encrypted.

NODE Definition

By using NODE definition, you can specify the default encryption algorithm for a particular node.

Note: For a particular node, NODE definitions always override any GLOBAL definitions, and are used only when they are not overridden by Transfer Interface definitions.

The format of NODE definition is as follows:

```
ENCRYPT={NONE | AES | AES128 | DES | 3DES | BLOWFISH | BLOWFISH_LONG |
AES},PASSONLY
```

ENCRYPT={NONE | AES | AES128 | DES | 3DES | BLOWFISH | BLOWFISH_LONG | AES},PASSONLY

This parameter specifies the level of encryption that is used by a particular node. The following table lists the description of each parameter option:

Option	Alternate Specification	Description
NONE		No encryption is used. This is the default setting.
AES		AES 256-bit encryption is used.
AES128		AES 128-bit encryption is used.
DES		DES encryption is used.

Option	Alternate Specification	Description
		For more information of DES encryption, see DES Encryption.
`3DES`		Triple DES encryption is used.
`BLOWFISH`	`BF`	Blowfish 56-bit encryption is used. For more information of Blowfish encryption, see Blowfish Encryption.
`BLOWFISH_LONG`	`BFL`	Blowfish 448-bit encryption is used.
`AES`	`RJ/AES`	AES 256-bit encryption is used. For more information of AES encryption, see AES Encryption.

When the `PASSONLY` operand is used, only the password is encrypted by using the specified encryption algorithm. The data is not encrypted.

Note: The `PASSONLY` operand is only valid for file transfers between z/OS and z/OS systems.

Examples Using NODE Definition

The following examples show how to use NODE definition to specify a default encryption algorithm for a particular node.

Example 1:

```
ENCRYPT=NONE
```

The Platform Server encrypts the password by using a proprietary algorithm. Data is not encrypted.

Example 2:

```
ENCRYPT=BLOWFISH_LONG or  ENCRYPT=BFL
```

The Platform Server encrypts the password and data by using Blowfish 56-bit encryption algorithm.

Example 3:

```
ENCRYPT=AES
```

The Platform Server encrypts the password and data using encryption algorithm.

Transfer Interface Definition

By using Transfer Interface definition, you can specify the definitive encryption algorithm for a single transfer request.

ⓘ **Note:** For a particular transfer request, Transfer Interface definitions always override any GLOBAL definitions and NODE definitions.

TIBCO MFT Platform Server for z/OS provides the following three user interfaces for transfer:

- ISPF
- REXX
- Batch

The REXX and Batch interfaces use the same encryption parameter. In addition, the generalizations made about the REXX and Batch interfaces must be also applied to the ISPF interface. See Interactive Interface for more information about the ISPF interface.

The format of Transfer Interface definition is as follows:

```
ENCRYPT={NONE | AES | AES128 | DES | 3DES | BLOWFISH | BLOWFISH_LONG |
AES}
```

This parameter specifies the level of encryption that is used by a particular transfer request. The following table lists the description of each parameter option:

Option	Alternate Specification	Description
NONE		No encryption is used. This is the default setting.
AES		AES 256-bit encryption is used.
AES128		AES 128-bit encryption is used.
DES		DES encryption is used. For more information of DES encryption, see DES Encryption.
3DES		Triple DES encryption is used.
BLOWFISH	BF	Blowfish 56-bit encryption is used. For more information of Blowfish encryption, see Blowfish Encryption.
BLOWFISH_LONG	BFL	Blowfish 448-bit encryption is used.
AES	RJ/AES	AES 256-bit encryption is used. For more information of AES encryption, see AES Encryption.

For Transfer Interface definition, you can use the PASSONLY parameter to specify whether to encrypt both the password and data or just encrypt the password. This parameter has the following two valid values:

- YES: Only encrypts the password. Data is not encrypted.
- NO: Encrypts both the password and data. This is the default value of the PASSONLY parameter.

Note: The PASSONLY parameter is only valid for file transfers between z/OS and z/OS systems.

Examples Using Transfer Interface Definition

The following examples show how to use Transfer Interface definition to specify a definitive encryption algorithm for a single transfer request.

Note: When the `PASSONLY` parameter is not defined, its default value, `NO`, is used.

Example 1:

```
ENCRYPT=NONE
```

The Platform Server encrypts the password by using a proprietary algorithm. Data is not encrypted.

Example 2:

```
ENCRYPT=BLOWFISH or ENCRYPT=BF
```

The Platform Server encrypts both the password and data by using Blowfish 56-bit encryption algorithm.

Example 3:

```
ENCRYPT=AES
```

The Platform Server encrypts both the password and data using the AES encryption algorithm.

Data Compression

You can use TIBCO MFT Platform Server for z/OS to compress data while sending the data to a remote destination to reduce the number of packets to be sent between systems and to reduce network traffic. The data is then decompressed when it is received at the remote destination.

Note: No special APIs, preprocessing tasks, or postprocessing steps are required for the Platform Server to compress data.

TIBCO MFT Platform Server for z/OS supports the following compression algorithms:

- Run Length Encoding (RLE)
- Limpel Ziv (LZ)
- ZLIB

For more information, see Data Compression Algorithms.

TIBCO MFT Platform Server for z/OS only supports Transfer Interface definition for specifying the compression algorithm used by a particular transfer request. GLOBAL definition and NODE definition are not supported.

For more information, see Defining Data Compression.

Rules of Using Data Compression

Because occasionally the use of data compression might slow down the transfer speed, you must determine whether to use data compression based on the actual situations.

For best results, do not compress a file when the Platform Server nodes are communicating over a high-speed network. For example, when TIBCO® Managed File Transfer Platform Server for Windows is communicating with TIBCO MFT Platform Server for z/OS over a 1gb network, enabling data compression in the data transfer might slow down the transfer because extra CPU cycles are used to compress data.

In the following situations, it is a good practice to use data compression:

- When the network speed is less than 1 mbps.
- When the data to be transferred contains repeating characters or strings. This usually occurs in text files rather than in binary files.
- When DES or 3DES data encryption is used. Both DES encryption and 3DES encryption are heavy users of CPU cycles. The Platform Server compresses data before encrypting data. Therefore, if data compression is used, the amount of data to be encrypted reduces and less CPU cycles are required for data encryption.

Data Compression Algorithms

TIBCO MFT Platform Server for z/OS supports three compression algorithms: RLE (Run Length Encoding), LZ (Limpel Ziv), and ZLIB.

RLE Compression

RLE compression is data dependent; that is, the compression ratio of RLE compression might vary widely based on the type of data being compressed. Use RLE compression if the network bandwidth is not a critical bottleneck for your network and you must save CPU cycles. RLE compression is the default compression algorithm of the Platform Server.

The advantages and disadvantages of RLE compression are as follows:

- Advantages of RLE
    - This algorithm is fast and uses less CPU cycles.
    - This algorithm is suitable for compressing data containing repeating characters such as spaces or nulls.
- Disadvantages of RLE
    - This algorithm is not suitable for compressing binary files which contain few repeating characters.
    - This algorithm typically does not compress data as well as ZLIB compression.

LZ Compression

LZ compression can compress a variety types of data. Use LZ compression, if you want better compression ratio and can spare CPU cycles.

The advantage and disadvantages of LZ compression are as follows:

- Advantage of LZ
    - This algorithm provides better compression ratio than RLE compression. It can compress text files at a ratio of 2:1 or more.
- Disadvantages of LZ
    - This algorithm consumes more CPU cycles than RLE or ZLIB Compression.
    - This algorithm is not suitable for compressing binary files which contain few repeating strings due to the CPU overhead of LZ Compression and decompression. We do not suggest LZ Compression. ZLIB is a better compression choice.

ZLIB Compression

ZLIB compression consists of 9 compression levels. Level 1 provides fast compression but low compression ratio. Levels 7 - 9 produce the best compression quality but are slower.

Typically, level 2 offers the best compromise between compression quality and speed.

The advantages and disadvantages of ZLIB compression are as follows:

- Advantages of ZLIB
    - This algorithm provides various levels of compression and CPU utilization.
    - This algorithm compresses data better than RLE compression and LZ compression.
    - This algorithm uses less CPU cycles than LZ compression.
    - This algorithm decompresses data more efficiently than LZ compression.
- Disadvantages of ZLIB
    - This algorithm uses more CPU cycles than RLE compression.
    - This algorithm is not suitable for compressing binary files which contain few repeating strings.

Defining Data Compression

You can use the `COMPRESS` parameter to specify whether to use data compression, or specify the compression algorithm used for a particular transfer request.

> **Note:** For TIBCO MFT Platform Server for z/OS, you can only use Transfer Interface definition to specify the data compression algorithm used for a particular transfer request. The GLOBAL definitions do not supply a default compression type, but the NODE definition can define a default compression for transfers with that node.

You can implement data compression through the following user interfaces:

- Batch
- ISPF
- REXX

The format of the `COMPRESS` parameter is as follows:

```
COMPRESS={YES | NO | RLE | LZ | ZLIB1 – ZLIB9}
```

The following table lists the description of each parameter option:

Option	Description
YES	The default compression algorithm (RLE) is used for this transfer.
NO	No compression is used for this transfer.
RLE	RLE compression is used for this transfer. **Note:** If you specify YES, RLE compression is used by default. For more information of RLE compression, see RLE Compression.
LZ	LZ compression is used for this transfer. For more information of LZ compression, see LZ Compression.
ZLIB1 – ZLIB9	ZLIB compression is used for this transfer. For more information of ZLIB compression, see ZLIB Compression.

Email Notification

TIBCO MFT Platform Server for z/OS provides the Email interface from where you can set the Platform Server to send an email notification to a user when a transfer is completed successfully or unsuccessfully.

Because TIBCO MFT Platform Server for z/OS is a “lights out” system in many ways, a means of notification must be used when an error occurs. TIBCO MFT Platform Server for z/OS now supports email notification when a transfer is completed. You can configure the Platform Server to send an email to a Platform Server user when a request fails, and to a different Platform Server user when the transfer is completed successfully. For example, you can send an email to the technical support if the request fails, and to the user of the file if the request is completed successfully.

To use the email notification function of the Platform Server, you must configure SMTP on z/OS mainframe, and define the email notification parameters on the Platform Server. For more information about parameters relating to email notification, see Email Notification Parameters.

The following example shows an email notification that was sent after a transfer was completed successfully:

```
Transfer request to IP Address 127.127.127.1
Request Status---------= Success
Transfer Type----------= SEND
Activity Number--------= I827500004
Local User-------------= ZOS1
Remote User------------= WIN1
Local File-------------= DATASET.NAME
Remote File------------= C:\FILE.TXT
Last Message-----------=
PGTF3101I Activity I827500004 successfully transferred 157 records with
127.127.127.1
Process----------------= INTERIM
User Data--------------=
Records Transferred----= 000000157
Bytes Transferred------= 005143632
```

Email Notification Parameters

To configure TIBCO MFT Platform Server for z/OS to send an email notification after a transfer is completed, you must define the email notification parameters.

The following table lists three GLOBAL parameters that you must configure to send an email from the Platform Server:

Parameter	Description
SMTP_JOBNAME	Defines the name of the SMTP started task.
SMTP_LOCALCLASS	Defines the name of the local class of the SMTP started task. **Note:** This parameter must match the LOCALCLASS parameter of the SMTP configuration.
SMTP_SENDER_HOSTNAME	Defines the IP address or the TCP host name. When sending an email, the Platform Server uses the value of this parameter to set the SMTP MAIL FROM command. This parameter value is displayed as the sender of the email when the email is read. The maximum length of this parameter is 40 bytes.

When an email is sent, the sender name defaults to FUSION_GOOD for successful transfers, and FUSION_FAIL for unsuccessful transfers. Your email Inbox can be sorted by the "Sender" (or "From Email Address") to see which requests failed and which requests were successful.

The following table lists the two parameters that you can configure in the Batch, REXX, DNI, and ISPF interfaces:

Parameter	Description
EMAIL_GOOD	Defines the email address to which the Platform Server sends an email when a request is completed successfully.
EMAIL_FAIL	Defines the email address to which the Platform Server sends an email when a request fails.

Data Transfer with SSL

To secure data transfer, TIBCO MFT Platform Server for z/OS supports data transfer with Secure Sockets Layer (SSL).

Communications between servers and clients require secure data transfers over the Internet. Secure data transfers involve more than encrypting data. With SSL, you can perform data encryption, and server and client authentication. In addition, the Platform Server pushes data transfer with SSL one step further by providing support for compliance with HIPAA or FIPS (Federal Information Processing Standard) 140 regulations.

For more information about data transfer with SSL, see the following introductions:

- SSL Authentication
- Data Encryption with SSL
- Compliance with HIPAA or FIPS Regulation

For information of SSL configuration, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Note: The term "SSL" and "TLS" are used interchangeably in this document. Platform Server only supports TLS protocols; it does not support SSLV2 or SSLV3. When the term "SSL" is used, TLS is used for all file transfer activities.

SSL Authentication

By using SSL, TIBCO MFT Platform Server for z/OS supports both server authentication and client authentication to insure that in a file transfer, the client validates the identity of the server and the server validates the identity of the client.

When using TIBCO MFT Platform Server for z/OS with SSL, if you want to send a file to a remote Platform Server, you must authenticate the identity of the remote Platform Server before issuing a transfer request; this is done through server authentication. If you receive a file transfer request from a remote Platform Server, you must authenticate the identity of the remote Platform Server before accepting the data; this is done through client authentication. Data transfer starts only after the authentication is completed.

TIBCO MFT Platform Server for z/OS performs SSL authentication by using digital certificates and certificate authority (CA). For more information of digital certificates and certificate authority, see Digital Certificates and Certificate Authority.

Digital Certificates

By using SSL, TIBCO MFT Platform Server for z/OS uses digital certificates to confirm the identity of the owners of the digital certificates, and uses a received digital certificate to identify the communication partners.

A digital certificate usually consists of the following three components which are all used by TIBCO MFT Platform Server for z/OS:

- Certificate
- Public key
- Private key
- Private key password

Certificate

This component is used by remote users to perform authentication. A certificate includes a public key for a receiver to decrypt data that is encrypted with a private key from the sender. Typically, a certificate includes the following contents:

- Certificate version number
- Certificate serial number
- Information of the certificate authority that issues the certificate
- Public key and encryption algorithm
- Time in which the certificate is valid
- Information about the user, including:
    - Common Name (CN)
    - Locality (L)
    - State (ST)
    - Country (C)
    - Organization Unit (OU)

- Organization (O)

> ℹ **Note:** The Platform Server is not a certificate manager. You cannot use the Platform Server to create certificates or to manage certificates. The certificates are produced by the Certificate Authority (CA). The contents of a certificate are governed by the X.509 certificate specification.

Public Key

The partner's public key is used to encrypt data. Data encrypted with a public key can only be decrypted by the private key associated with the public key.

Private Key

You can use this component to decrypt data that is encrypted by a remote user by using your public key.

Private Key Password

This component protects your private key from being copied and used by other people.

Certificate Authority

By using SSL, TIBCO MFT Platform Server for z/OS uses certificate authority (CA) to certify the certificate that is received from the sender in a data transfer. To make sure the CA that issued the digital certificate can be trusted, the Platform Server with SSL uses a Trusted CA file.

In a data transfer, a CA issues a digital signature and adds the digital signature to the certificate that is received from the sender. A digital signature is simply a piece of data that is encrypted by using the private key of the certificate authority. The receiver uses the public key from the sender to decrypt and validate the data. When the certificate is checked, you can see the CA that created the certificate.

> ℹ **Note:** You must store the certificates received from the CA to a key ring file. For how to generate a key ring file, see Creating Key Ring Files.

To make sure the CA that issues the digital certificate can be trusted, the Platform Server uses a Trusted CA file to define the CAs that SSL trusts. The Platform Server with SSL checks the Trusted CA file to see whether the CA that issues the digital certificate is listed in the file. If the CA is in the Trusted CA file, the Platform Server with SSL uses the public key to decrypt and validate the digital signature in the certificate that is received from the sender in a data transfer. The Platform Server with SSL only accepts a certificate request when the CA that issues the digital certificate is in the Trusted CA file and the digital signature is valid.

Certificates uses the Base64 encoding format. The following example shows a sample certificate that is encoded in the Base64 format:

```
-----BEGIN CERTIFICATE-----
MIICdzCCAeCgAwIBAgIDNYAGMA0GCSqGSIb3DQEBBAUAMIGHMQswCQYDVQ
QGEwJaQTEiMCAGA1UECBMZRk9SIFRFU1RJTkcgUFVSUE9TRVMgT05MWTEd
MBsGA1UEChMUVGhhd3RlIENlcnRpZmljYXRpb24xFzAVBgNVBAsTDlRFU1
QgVEVTVCBURVNUMRwwGgYDVQQDExNUaGF3dGUgVGVzdCBDQSBSb290MB4X
DTAxMDgyNzE3MDI1NFoXDTAxMDkxNzE3MDI1NFowUjELMAkGA1UEBhMCVV
MxETAPBgNVBAoTCFByb2dpbmV0MRQwEgYDVQQLEwtEZXZlbG9wbWVudDEa
MBgGA1UEAxMRUHJvZ2luZXQgRW1wbG95ZWUwgZ8wDQYJKoZIhvcNAQEBBQ
ADgY0AMIGJAoGBALnB6f3CSDhcWMChxsmxqtNG7qL8tsiUaXSslRnRCFXg
tiY3mnZyxcLfr0EzfD9MyyLTENO6VVknE7hlS65uMuU1lrxrRr45xuf0+X
tGzoGD9l8j+Ux0/fmS9xKiyBS5+cBt8xMPHPqWgqESBO9cx1QbRctpZ7FT
c2yPCV3ZpKGjAgMBAAGjJTAjMBMGA1UdJQQMMAoGCCsGAQUFBwMBMAwGA1
UdEwEB/wQCMAAwDQYJKoZIhvcNAQEEBQADgYEABogHOgfpnJClIeybjDDt
KqbWuelhDbnCRJg1HMtioGk6/AUC3ZTGh+Jq6O+PbQ/Y+O7T4LcadFNukJ
12EOcv3C2z31YrbwSn5WaPkilhQMEImmGpQ4tM90XSn+2l6IvS6mtbtLvK
6Qb68cSlpxogugmFN9egZbOezR2DU+5arSc=

-----END CERTIFICATE-----
```

Creating Key Ring Files

The Platform Server supports for creating key ring files by using the provided commands.

You can use the key manager utility (`gskkyman`) or the RACF `RACDCERT` command to create key rings.

> **Note:** It is a good practice to use the RACF `RACDCERT` command.

Data Encryption with SSL

TIBCO MFT Platform Server for z/OS supports data encryption with SSL for file transfers under unsecured network.

Under unsecured network, file transfers are vulnerable to the following two types of attacks:

- The data can be read by unauthorized people.
- The data can be altered by unauthorized people.

The first type of attacks can be resolved by encrypting data. SSL uses a combination of the following two types of encryption algorithms to encrypt data:

- Asymmetrical encryption

  The sending and receiving systems use different keys to encrypt and decrypt the data. For more details, see Asymmetrical Encryption.

- Symmetrical encryption

  The same key is used for both encryption and decryption of data. For more details, see Symmetrical Encryption.

SSL performs its handshake by using asymmetrical encryption; then the software sends the symmetrical encryption key that is encrypted by the asymmetrical encryption algorithm. In this way, both sender and receiver have the same encryption key that is transmitted in a secure form, and the data can be transferred more efficiently.

While to resolve the second type of attacks, you must perform message integrity check to detect whether the data is changed. SSL detects any changes in data by adding a message digest to all transmitted data.

A message digest is a condensed representation of a message. Before a file transfer, SSL uses the MAC algorithm based on Secure Hash Algorithm (SHA) to generate a 20-byte message digest for data. This message digest is then sent together with the encrypted data to the remote Platform Server. At the remote Platform Server, the receiver uses the same algorithm to generate a message digest and compare the generated message digest with that received. If any component of the data is changed, the message digest is also changed. In this way, the receiver checks whether the data is altered before accepting the data.

Asymmetrical Encryption

For asymmetrical encryption, the sending and receiving systems use two different keys (public key and private key) to encrypt and decrypt data.

Typically, data is encrypted using a public key and can only be decrypted by the private key associated with the public key.

The main disadvantage of asymmetrical encryption is that it is slower than symmetrical encryption.

Symmetrical Encryption

For symmetrical encryption, the data is encrypted and decrypted by using the same key.

Symmetrical encryption algorithms include DES, Triple DES (also known as 3DES), Blowfish, and AES. For more information, see Data Encryption Algorithms.

In most cases, symmetrical encryption is more efficient than asymmetrical encryption. But, using symmetrical encryption to send an encryption key over the internet might cause security problems because the receiver must use the same key as the sender to decrypt the data.

Compliance with HIPAA or FIPS Regulation

TIBCO MFT Platform Server for z/OS enforces HIPAA or FIPS 140 regulations as the security policy on initiated and responding data transfers.

HIPAA and FIPS 140 are government standards to certify cryptographic modules that are used to protect information and communications in electronic commerce within a security system. The secure system protects sensitive but unclassified information.

- If you set the security regulation to HIPAA, all files must be transferred by using SSL with the AES or Blowfish Long encryption type, which uses 128-bit or greater key length.
- If you set the security regulation to FIPS 140, all files must be transferred by using SSL with the AES encryption type, which uses 256-bit key length.

If the encryption type you specified to comply with the HIPAA or FIPS 140 security policy for data transfer is not valid, the encryption type is overridden and a message is displayed to inform you that a valid encryption type is used for data transfer. For example, if you use DES encryption for data transfers using HIPPA or FIPS 140 security policy, because DES is not a valid encryption algorithm for HIPAA and FIPS 140, DES encryption is overridden and a message is displayed to inform you that the encryption algorithm is changed to Blowfish Long or Rijndael (AES).

Platform Server SSL/TLS Support

Platform Server supports two different modes of SSL/TLS communication:

- **TLS Mode**: Platform Server initiates a TLS connection with the target Platform Server. Certificates are passed between the Client and Server and the certificates are validated. A Symmetric encryption key is transmitted between the client and the server. The SSL Connection is then terminated. Data is encrypted using the Symmetric key passed in the TLS Session. Sequence numbers and a message digest are added to each packet of data transmitted and are validated by the recipient. TLS mode is supported for all versions of Platform Server, Internet Server and Command Center.
- **TLS Tunnel Mode**: Platform Server initiates a TLS connection with the target Platform Server. Certificates are passed between the Client and Server and the certificates are validated. Transfer data is sent over the TLS session. All data is encrypted using the cipher selected during TLS Negotiation. Message digests and sequence numbers are added by the internal TLS protocol. TLS mode is supported for Platform Server V8 and above and for Internet Server and Command Center V8.1 and above. If you are using Platform Server to communicate over public lines, we suggest using TLS Tunnel mode.

Batch Interface

TIBCO MFT Platform Server for z/OS provides the Batch interface where you can queue transfers to TIBCO MFT Platform Server for z/OS by using Job Control Language (JCL). You can perform a single file transfer or multiple file transfers within one step in a job.

Note: Users of the JCL interface of TIBCO MFT Platform Server for z/OS are called "batch clients".

When you initiate a transfer request on the Platform Server, you must assign a process name for the transfer request. The process name can subsequently be utilized by ISPF operators who want to monitor the status of the transfer. The Platform Server assigns a 10-character activity number for the transfer request that the ISPF or console operators might use to monitor the status of the transfer.

JCL Statements

You can use JCL statements to perform batch jobs and queue file transfers.

Required JCL Statements

This sample JCL includes the required JCL statements. Each of the JCL statements that are used in the sample are explained in the table that follows.

Note: If you create a dataset and then initiate a transfer to send the dataset to a remote Platform Server, we suggest that you add the following parameter to the JOB card: `DSENQSHR=ALLOW` This parameter converts the exclusive ENQ to a shared ENQ. If this parameter is not added, the MFT transfer most likely will fail with a `File in use` error message. Support was added for the `DSENQSHR` parameter in z/OS V2.1.

```
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB DD DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
```

Statement	Description
EXEC PGM=OSIUB000,PARM='data'	Defines the program that must be executed to run the batch interface. The PARM parameter defines the name of the Platform Server started task. You can also use the PARM parameter to override any SYSIN parameters. For more information, see Appendix G. Overriding JCL SYSIN Parameters. **Note:** The EXEC PGM program name must be defined as "OSIUB000".
STEPLIB	Defines the library that contains the Platform Server load modules. **Note:** You must include this statement in the JCL to identify the Platform Server load library. If the Platform Server LOADLIB is already added to the LINKLST, you do not need to add the Platform Server LOADLIB to the STEPLIB.
SYSPRINT	Defines the output report file that records the parameters that are used for the file transfer. If the file is successfully queued, you can read the output report file and find out the transaction number assigned to the job.
SYSIN	Defines the input file that records which file to transfer and to where to send the parameters that govern file transfer activities.

Optional JCL Statements

The following table lists the optional JCL statements:

Statement	Description
FUSCFG	Defines the configuration file that defines the server name, IP

Statement	Description
	address, IP name, or LUName of the started task. **Note:** This statement is only required when the `PARM CONFIG` parameter is defined. For more information on FUSCFG configuration, see "Defining and Configuring the FUSCFG File" in the *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Defining the PARM Field of the EXEC PGM=JCL Card

You can use the `PARM` field in a JCL statement to define the communication method between TIBCO MFT Platform Server for z/OS clients and started tasks, define process parameters and additional parameters, and define the `CONFIG` parameter.

For more information of the `PARM` field configurations, see the following introductions:

- Defining the Communication Method
- Defining Process Parameters and Additional Parameters
- Defining the CONFIG Parameter

Defining the Communication Method

You can define the way the TIBCO MFT Platform Server for z/OS clients communicate with the Platform Server started task in the `PARM` field in a JCL statement.

TIBCO MFT Platform Server for z/OS clients can communicate with the Platform Server started tasks through the following three ways:

- Cross Memory Services
- SNA LU6.2
- TCP

See Supported Communication Methods for more information of the three communication methods.

Supported Communication Methods

TIBCO MFT Platform Server for z/OS clients can communicate with the Platform Server started tasks through three ways: Cross Memory Services, SNA LU6.2, and TCP.

Cross Memory Services

TIBCO MFT Platform Server for z/OS clients can communicate with the Platform Server started tasks through cross memory services.

- Required Parameter

  `SERVER`: Defines the name of the Platform Server started task.

- Optional Parameter

  No optional parameter is required.

- Example

  ```
  // EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
  ```

SNA LU6.2

TIBCO MFT Platform Server for z/OS clients can communicate with the Platform Server started tasks through SNA LU6.2.

- Required Parameters
  - `SERVLUNAME`: Defines an APPLID of the Platform Server. For more information, see "Defining VTAM Resources for Systems Using SNA" in *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.
  - `ACBNAME`: Defines the name of the Platform Server client ACB (Access Control Block) defined to VTAM. For more information, see "Defining VTAM Resources for Systems Using SNA" in *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

    > **Note:** Only the prefix of the name, without the two-character numeric suffix, should be specified.

- Optional Parameter
  - `MODENAME`: Defines the name of the SNA mode with which the session should be

established.

- Example

```
//STEP0001 EXEC PGM=OSIUB000,PARM='SERVLUNAME=FUSNAPPL,ACBNAME=FUSN'
```

TCP

TIBCO MFT Platform Server for z/OS clients can communicate with the Platform Server started tasks through TCP.

- Required Parameters
    - `TCPIPJOBNAME`: Defines the name of the TCP/IP started task.
    - `SERVIPNAME` or `SERVIPADDR`: Defines the IP name or the IP address of the Platform Server.
- Optional Parameter
    - `SERVIPPORT`: Defines the IP port that will be used to connect to the Platform Server. The parameter is set by default to `46464`, and it must match the GLOBAL `TCPIPPORT` parameter.
- Example

```
//STEP0001 EXEC PGM=OSIUB000,
//         PARM=('SERVIPADDR=127.127.127.50,',
//         'TCPIPJOBNAME=TCPIP,SERVIPPORT=2500')
```

Defining Process Parameters and Additional Parameters

In addition to defining the communication method between TIBCO MFT Platform Server for z/OS client and started tasks, you can define any process parameters and additional parameters in the `PARM` field of a JCL statement.

For more information of the Platform Server process parameters, see Batch Interface Parameters.

Parameters specified in the `PARM` field override those specified by using the SYSIN method. For more information on overriding `SYSIN` parameters, see Examples of Overriding SYSIN Parameters.

If you want to update the transfer activity type (SEND or RECEIVE), it must be the second parameter following the SERVER= parameter which defines the communication method. Additional parameters can optionally be added following the activity type. If the activity type is not defined, additional parameters follows the SERVER= parameter. All parameters are separated by commas (,).

Note: When you override a parameter through a PARM definition, all transfers defined by the SYSIN DD statement are overridden.

Examples of Overriding SYSIN Parameters

You can override any SYSIN parameters through PARM definitions.

Note: z/OS platform has a 100-byte limitation on the amount of data that can be entered in the PARM override. This is not a TIBCO MFT Platform Server for z/OS restriction.

The following examples shows how to override the SYSIN parameters through PARM definitions:

Example 1: Override the Fusion1 server to specify an activity type of SEND and a new remote file name

```
 //STEP0001 EXEC PGM=OSIUB000,
//         PARM=('SERVER=FUSION1,SEND,',
//         'REMOTE_FILE=C:\FUSION TEST\TEST OVERIDE.1')
```

Example 2: Override the FUSIONA server through SNA to specify different local and remote file names and a different node name

```
 //STEP0001 EXEC PGM=OSIUB000,
//         PARM=('SERVLUNAME=SERVALU,ACBNAME=FUSN,',
//         'DSN=LOCAL.OS390.FILE,',
//         'REMOTE_FILE=C:\REMOTE FILE,NODE=MYNODE')
```

Example 3: Override the server through TCP to specify a different IP address and IP port for a transfer

```
    //STEP0001 EXEC PGM=OSIUB000,
   //         PARM=('SERVIPADDR=10.10.12.5,TCPIPJOBNAME=TCPIP,',
   //         'IPADDR=10.10.13.12,IPPORT=47000')
```

Example 4: Override information on a transfer request when the initiator has no access or authorization to the dataset pointed to by the SYSIN DD statement.

You must set a process called FUSBATCH in a PROCLIB that includes the `PARM` override call `PARMDATA`:

```
//STEP0001 EXEC
FUSBATCH,PARMDATA=‘NODE=LANODE,DSN=LOCAL.DSN,RUSER=OPER1,
PASS=MYPASS’
```

The following example shows the `PARM` field definition:

```
 // EXEC PGM=OSIUB000,PARM='SERVER=FUSIONA,&PARMDAT'
```

Note: You must include the SYSIN DD statement to specify the PROCESS statement and any other statement that is not defined in the `PARM` override.

Defining the CONFIG Parameter

You can define the `CONFIG` parameter in the `PARM` field in a JCL statement to allow Platform Servers to submit requests to a Platform Server for z/OS. This is a usage of the `PARM` override facility.

Required Parameter

`CONFIG`: Defines the name of the CONFIG entry within the FUSCFG configuration entry. It can contain up to 24 characters, and it must match the `CONFIG` parameter within the FUSCFG member of the FUSCFG DD statement.

The FUSCFG CONFIG entry can be defined to communicate through TCP, SNA, or Cross Memory Services. The advantage of using these methods of communication is that all configuration data for communicating to the Platform Server started task is controlled within a single file. If the connectivity information for a started task changes, the FUSCFG entry can be changed so that individual batch jobs do not have to be changed.

Note: The FUSCFG file used by the Batch interface is the same as the FUSCFG information used by the ISPF and REXX interfaces. The same file can be used by the Batch interface as well as by the ISPF and REXX interfaces. For more information on the FUSCFG file, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Example of Defining the CONFIG Parameter

```
//STEP0001 EXEC PGM=OSIUB000,
//PARM='CONFIG=CONFIGENTRYNAME'
```

Defining the Batch Interface Parameters

You can configure TIBCO MFT Platform Server for z/OS parameters by using either the `PARM` field of the `EXEC PGM=JCL` card or the SYSIN DD statement.

The limitations of the JCL PARM statement will restrict you to a limited number of parameters, so we suggest defining the transfer parameters in the SYSIN DD.

PROCESS Statement

A PROCESS statement, also called process card, indicates the process name and is contained in a client job step.

The JCL client of TIBCO MFT Platform Server for z/OS communicates with the server through cross-memory services to queue transfer requests. Every transfer request to a Platform Server is assigned a process name by the user who initiates the request. The process name can be used subsequently by operators who want to monitor the progress of the transfer. The process name is also displayed in a message at the end of every file transfer.

The process name is indicated by a process card. A client job step can contain one or more process cards. Each process is indicated by a PROCESS statement, which is a record in the SYSIN file.

The following example shows a client job step containing two process cards:

```
//SYSIN DD  *
PROCESS,PAYROLL,TRANSFER,SEND
:
: parameters

PROCESS,INVENTORY,TRANSFER,RECEIVE
:
: parameters
/*
```

The following table lists the three positional parameters of the PROCESS card:

Parameter	Description
`Process Name`	This parameter is the first positional parameter and defines the name of the process. The process name is the name by which this file transfer is known to the System Server. This parameter can contain 1 - 8 characters.
`Activity Type`	This parameter is the second positional parameter and defines the activity type. You must set this parameter to `TRANSFER` when sending, receiving, submitting, or running a script. You can also set this parameter to `PING` if you want to check whether a TIBCO MFT Platform Server for z/OS is active or not.
`Activity`	This parameter is the third positional parameter and defines the activity to perform. The supported activities are as follows: • `SEND`: Indicates sending a file from the local system to the remote system. • `RECEIVE`: Indicates receiving a file from the remote system to the local system. • `SUBMIT`: Indicates locally submitting a file transfer to the internal reader. The job is submitted to the CPU where

Parameter	Description
	the Platform Server started task is running. The submitted job runs under the authorization of the user that is running the Batch interface. • SCRIPT: Indicates scheduling a script to run in the Platform Server started task. A node can be defined with the CONTENTION_WINNERS parameter to limit the number of concurrent scripts that can be run synchronously. For example, by defining the CONTENTION_WINNERS parameter to 1 and specifying the NODE parameter in the process, you can force scripts to be run single-threaded. The NODE parameter is ignored other than to limit the number of concurrent requests for a script.

Format of PROCESS Statements

Parameters in a PROCESS statement might be continued by using one of the following notations:

- Put a non-space in column 72. Data is continued in column 1 of the next record.
- Append a space plus space combination (+) to the end of a line. The first non-blank character of the next line is appended to the last non-blank character of the first line.
- Append a space plus plus combination (++) to the end of a line. A space is added between the first non-blank character of the next line and the last non-blank character of the first line.

Parameters can also be enclosed in single or double quotation marks to accommodate imbedded quotations.

Examples of Process Statements

The following examples show how to define the PROCESS cards.

Example 1:

```
PROCESS MONTHLY,TRANSFER,RECEIVE
```

In this example, the process is named as `MONTHLY`, and is defined to receive a file from a remote computer. After this statement is issued, operators can refer to the process in commands by this name.

Example 2:

```
PROCESS PAYROLL,TRANSFER,SEND
```

In this example, the process is named as `PAYROLL`, and is defined to send a file to a remote computer.

Example 3:

```
PROCESS LOCALJOB,TRANSFER,SUBMIT
```

In this example, the process is named as `LOCALJOB`, and is defined to submit a job to the internal reader.

Example 4:

```
PROCESS RUNSCRPT,TRANSFER,SCRIPT
```

In this example, the process is named as `RUNSCRPT`, and is defined to schedule a script to run in the TIBCO MFT Platform Server for z/OS started task.

Example 5:

```
PROCESS,,PING
```

This process checks whether a TIBCO MFT Platform Server for z/OS is active. It returns 0 if the server is active, and returns a non-zero number if the server is not active. The `PROCESS,,PING` request can also be called from within the script program OSIUB000 or OSIUC000, or the API interface (FUSCFAPI).

> **Note:** To perform this ping action, the TIBCO MFT Platform Server for z/OS must be version 6.5.1 or higher.

SYSIN DD Statement

You can use the SYSIN DD statement to define the input file that records what file to transfer and where to send any other parameters that govern a file transfer activity.

For the parameters that can be defined in the SYSIN DD statement, see SYSIN Coding Parameters.

For the syntax rules of the SYSIN DD statement, see SYSIN DD Statement Syntax Rules.

SYSIN DD Statement Syntax Rules

To define valid SYSIN DD statements, you must follow the syntax rules of the SYSIN DD statement.

Parameters are specified in the data record referenced by the SYSIN DD statement.

When defining parameters in the SYSIN DD statement, you must comply with the following syntax rules:

- The process, process name, activity type, and activity must be specified on the first line delimited by a comma (,).
- Additional parameters must be specified one per line.
- Parameters that have imbedded spaces must use double quotation marks (") as delimiters at the start and end of the parameters.
- When a line starts with an asterisk (*), the line is considered a comment.
- Parameters can be continued in one of the following two ways:
    - Place a "space plus space" combination (" + ") at the end of a parameter. The parameter is continued on the next line at the first non-blank character. This is the best practice of continuing data because it is the simplest to implement.
    - Append a space plus plus combination (++) to the end of a line. A space is added between the first non-blank character of the next line and the last non-blank character of the first line.
    - Place a continuation mark in column 72 (non-blank character) and start the next line in column 1. This can be repeated to go on to a third or fourth line, if necessary.

SYSIN Coding Parameters

You can define file transfers by specifying parameters in the SYSIN DD statement.

Note: You can also specify the parameters in the `PARM` field on the JCL EXEC statement. Parameters specified in the `PARM` field override those specified by using the SYSIN DD statement.

The following table lists the parameters that you can specify in a SYSIN DD statement.

Parameter	Alternate Specification	Description
`ALLOC_DIR`		Defines the number of directory blocks for a new file.
`ALLOC_PRI`		Defines the primary allocation quantity for a new file.
`ALLOC_SEC`		Defines the secondary allocation quantity for a new file.
`ALLOC_TYPE`		Defines the type of allocation for a new file.
`AVAIL`		Defines whether TAPE is supported.
`BLOCKSIZE`	`BLKSIZE`	Defines the DCB block size for a new file.
`CALLJCL`		Requests the Platform Server to call a user program defined on the remote system with JCL linkage. This parameter should only be used when the remote system is z/OS.
`CALLPROG`		Requests the Platform Server to call a user program defined on the remote system using standard linkage conventions. This parameter should only be used when the remote system is z/OS.

Parameter	Alternate Specification	Description
CHKPT	CKPT / CHECKPOINT	Defines the checkpoint interval.
COMPRESS		Defines the compression options.
CONTINUE		Defines whether transfers continue when a transfer fails.
COSNAME		Defines the Class of Service (COS) entry name.
CRCCHECK	CRC	Defines whether CRC checking will be turned on for this transfer.
CRLF		Defines the CRLF/LF options.
DATACLASS	DC / DCLASS	Defines the SMS data class.
DATASET_TYPE		Defines the type of the new dataset to create.
DATE		Defines the date to start the transfer.
DCB		Defines RECFM, LRECL, and BLKSIZE.
DCB_LIKE		Defines the DCB model DSN.
DESCRIPTION	DESC	Defines the transfer description.
DIRTEST		Shows files for a directory transfer.
DSN	LF / LFILE / LDSLOCAL_FILE	Defines the name of the local dataset.
DSORG		Defines the dataset organization.
EFFECT		Defines the file creation options.

Parameter	Alternate Specification	Description
EMAIL_FAIL		Defines the email address to which to send an email notification for unsuccessful requests.
EMAIL_GOOD		Defines the email address to which to send an email notification for successful requests.
ENCRYPT		Defines the data encryption options.
EVERY		Defines the repeat cycle of a request.
EVERY_COUNT		Defines the number of times that an EVERY request can be scheduled to execute.
EVERY_EXDAYS		Defines the EVERY expiration date.
EVERY_EXTIME		Defines the EVERY expiration time.
EVERY_MISSED		Defines the EVERY control.
EXEC	REXXEXEC / COMMAND	Defines a command or exec for execution on the remote system.
EXECPREPROC		Defines whether directory transfer preprocessing actions are executed on the parent transfer or the child transfers. **Note:** This parameter overrides the value defined in the GLOBAL EXECPREPROC parameter.
EXECPOSTPROC		Defines whether directory transfer postprocessing actions are executed on the parent transfer or the child transfers.

Parameter	Alternate Specification	Description
		Note: This parameter overrides the value defined in the GLOBAL EXECPOSTPROC parameter.
EXPDT		Defines the transfer expiration date.
EXPTM		Defines the transfer expiration time.
FILEERRORINT		Defines the wait interval when a retryable file error occurs. This parameter defines how long a transfer waits before being retried. **Note:** When defined, this parameter overrides the Node and GLOBAL FILE_ERROR_TRY_INTERVAL parameters. When not defined, the GLOBAL FILE_ERROR_TRY_INTERVAL parameters will be used. **Note:** The actual interval can be up to 60 seconds or more than the value specified, depending on when the error occurs and when the dispatcher runs.
FILEERRORMAX		Defines the maximum number of retryable file errors. **Note:** When defined, this parameter overrides the Node and GLOBAL FILE_ERROR_TRY_COUNT parameters. When not defined, the GLOBAL FILE_ERROR_TRY_COUNT parameters will be used.
FILE_EXPDT		Defines the file expiration date.

Parameter	Alternate Specification	Description
FILE_RETPD		Defines the file retention period.
HOLD		Defines the Hold option.
IPADDR		Defines the IP address of the remote system.
IPNAME		Defines the IP name of the remote system.
IPPORT		Defines the IP port of the remote system.
IUNIT		Defines the input tape unit.
IVOLUME		Defines the input dataset volume.
LENGTH	LRECL	Defines the DCB logical record length for the new file.
LIST		Defines the name of the distribution list.
LOCALLABEL	LLABEL	Defines the label used on the initiating Platform Server when sending or receiving a tape file.
LOCALCTFILE	LCT	Defines the name of the local Translation table.
LOGON_DOMAIN	DOMAIN	Defines the Windows domain where the remote user resides.
LPASS	LOCAL_PASS	Defines the local password.
LUSER	LOCAL_USER	Defines the local user ID.
MAINTAINBDW	MBDW	Defines whether to maintain RECFM=VB BDW when sending or receiving files.
MAINTAINRDW	MRDW	Defines whether to maintain RECFM=VB RDW

Parameter	Alternate Specification	Description
		when sending or receiving files.
MEMBER		Defines the member names.
MGTCLASS	MC / MCLASS	Defines the SMS management class
MQ_FAIL		Defines the Platform Server to notify MQ on failure.
MQ_GOOD		Defines the Platform Server to notify MQ on success.
MQ_NOTIFY		Defines when the Platform Server sends MQ notifications.
MQ_WAIT_MESC		Defines the number of minutes that MQ waits for data.
NODE		Defines the node name of the remote system.
NODECLASS		Defines the node class.
NOTIFY		Defines the name of the user to notify when the file transfer is completed.
NOTIFY_TYPE		Defines the type of user ID to notify.
OVOLUME		Defines the output dataset volume.
PASSONLY		Defines the Platform Server to encrypt only the password.
PERMIT_ACTNS		Defines the Windows-permitted actions.
POST_ACTION	PPA	Defines the postprocessing action.

Parameter	Alternate Specification	Description
PRIORITY		Defines the transfer schedule priority.
PROCNAME		Defines the process name.
PURGE		Defines the timeout request option.
RDSIP		Defines the remote file disposition.
RECFM		Defines the DCB record format for the new file.
REMOTECTFILE	RCT	Defines the name of the remote Translation table.
REMOTELABEL	RLABEL	Defines the label used on the responder Platform Server when sending or receiving a tape file.
REMOTE_FILE	RF / RFILE / RDSN	Defines the name of the file on the remote system.
REMOTE_PASS	RPASS	Defines the password for the remote user on the remote system.
REMOTE_UNIT	UNIT	Defines the unit for file creation.
REMOTE_USER	RUSER	Defines the name of the user on the remote system.
REMOVETRAIL	RMTRAIL	Defines whether to remove trailing blanks and nulls.
RESOLVE_GDG		Defines the GDG resolution indicator.
RETENTIONPRD		Defines the retention period for the transfer request.

Parameter	Alternate Specification	Description
RETRY	TRY	Defines the number of times that a request will be tried.
RETRYINT		Defines the retry interval.
RMTRAIL		Defines whether to remove the trailing spaces.
RSACCELERATOR	RSA	Defines whether transfers go to an RSA host.
RSCOMPRESS	RSC	Defines whether RSA uses compression.
RSENCRYPT	RSE	Defines whether RSA uses encryption.
RSHOST	RSH	Defines the RSAccelerator host name or IP address.
RSMAXSPEED	RSMAX	Defines the maximum speed that RSA uses.
RSPORT		Defines the RSAccelerator port number.
RSPROTOCOL	RSP	Defines the protocol that RSA uses.
SCRIPT		Defines the name of the script file in the started task.
SENDRECFMU		Defines whether to maintain RECFM=VB BDW when sending or receiving files.
SPACE		Defines the space for a new file.
SPACERELEASE		Defines the release unused space flag.
SSL		Defines the SSL options. SSL is now an alias of the TLS parameter. See the TLS parameter.

Parameter	Alternate Specification	Description
SPACE		Defines the space for a new file.
SPACERELEASE		Defines the release unused space flag.
STORCLASS	SC / SCLASS	Defines the SMS storage class.
SUBDIR		Defines whether to use subdirectories.
SUBMIT		Defines the Platform Server to submit JCL to the remote system.
TAPE		Notifies the Platform Server that this is an uncatalogued tape transfer.
TCP_TIMEOUT	TCPTIMEOUT	Defines the TCP timeout, in minutes, for this transfer. **Note:** This parameter overrides the TCP_ TIMEOUT defined in the GLOBAL configuration.
TEMPLATE		Defines the Windows template.
TEXTEOFTERM		Defines the CRLF/LF end of file options.
TIME		Defines the time at which to start a transfer.
TLS		Defines whether TLS is used for the request. (SSL is now an alias of the TLS parameter.)
TRANS_TYPE		Defines the transfer type.
TRANSFER		Defines the file transfer direction.
TRCLASSNAME	TRCLASS	Defines the transfer class.

Parameter	Alternate Specification	Description
TRUNCATE		Defines the truncation options when receiving a file from Windows, UNIX, IBM i, or MFT Internet Server when record delimiters are defined.
TYPE		Defines the TEXT or BINARY file.
UNIXPERM		Defines the permissions that are set when you create a file on a target UNIX system.
UTF8BOM		Defines whether the UTF-8 BOM (0xefbbbf) is added at the start of the converted data or is removed from the start of the data before conversion.
VOLUME	VOL / VOLSER	Defines the volume for file creation.
VSAM_KEYLEN		Defines the VSAM KSDS key length.
VSAM_KEYPOS		Defines the VSAM KSDS relative key position.
VSAM_LIKE		Defines the VSAM model DSN.
VSAM_LRECL		Defines the VSAM record length.
VSAM_RRSLOT		Defines the VSAM RRDS slot number option.
VSAM_REUSE		Defines the VSAM REUSE option.
VSAM_TYPE		Defines the VSAM file organization.
WAIT		Defines the transfer WAIT options.
XMEMBER		Defines the excluded member names.

ALLOC_DIR

Default	`0`
Allowable Values	`0` - `32000`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the number of directory blocks that should be allocated when a file is created. If no file is created, this parameter is ignored. Normally, TIBCO MFT Platform Server for z/OS automatically determines the number of directory blocks that a file has allocated and passes that information to its partner.

You can use this parameter to perform the following operations:

- Override the number of directory blocks on a local file.
- Specify the number of directory blocks to allocate when the partner is not TIBCO MFT Platform Server for z/OS.

ALLOC_PRI={primary_space}

Default	Not Applicable
Allowable Values	`0` - `999999`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the primary allocation quantity in tracks, cylinders, kilobytes, or megabytes as indicated in the `ALLOC_TYPE` field.

ALLOC_SEC

Default	Not Applicable
Allowable Values	`0 - 999999`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the secondary allocation quantity in tracks, cylinders, kilobytes, or megabytes as indicated in the `ALLOC_TYPE` field.

ALLOC_TYPE={T | C | M | K | B}

Default	`T`
Allowable Values	`T`, `C`, `M`, `K`, `B`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter instructs z/OS for creating new files. This parameter is ignored when sent to TIBCO MFT Platform Server for Windows.

The valid values of this parameter are as follows:

- `T`: dataset size is expressed in tracks.
- `C`: dataset size is expressed in cylinders.
- `M`: dataset size is expressed in megabytes.
- `K`: dataset size is expressed in kilobytes.
- `B`: dataset size is expressed in blocks.

AVAIL={IMMED | DEFER}

Default	`IMMED`
Allowable Values	`IMMED`, `DEFER`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether TAPE is supported.

The valid values of this parameter are as follows:

- `IMMED`: specifies that the remote file is available to users as soon as the file transfer is completed. This is the default and the best practice.
- `DEFER`: requests that the remote file availability be deferred if the remote system uses this option. Most remote systems do not use this option.

Note: In the responder function, TIBCO MFT Platform Server for z/OS treats `DEFER` as tape.

BLOCKSIZE={nnnnn | 0}

Default	`0`
Allowable Values	`0` - `32760`
Minimum	`0`
Maximum	`32760`
Alternate Specification	`BLKSIZE`

This parameter specifies the size of the block. For FB, the block size must be a multiple of the record length "`l`"; for VB, the block size can be any value up to the record length plus four. The maximum value of this parameter is `32760`. When this parameter is set to `0`, the operating system determines the optimum block size.

CALLJCL

You can use this function to call a user-written program directly from a script. Only one parameter can be passed to the program being called. The program being called must be defined either in LINKLIB or in STEPLIB of the job running the script. The parameter is passed to the user-written program by using the same linkage that is used when passing a JCL PARM statement to a program. Only character variables can be passed to the user program. Character data is passed as a string of characters.

The program called by the CALLJCL function, should not update any of the variables. Variables cannot be updated from the called program. The only data that can be passed back to the script file is the return code. Variable *%RC* is updated with the return code set by the called program.

If the program defined in the CALLJCL function is not found, an error is displayed and the script is terminated. If an abend occurs while in control of the user-written program, error messages are displayed and the script is terminated.

The format of this function is `CALLJCL` *`program_nameparameter_data`*. For example, `CALLJCL USERPGM %CHARDATA`. This command calls the USERPGM program and passes a parameter to the program. Assume that the variable is set in the following way: `%CHARDATA = "TESTDATA"`. When control is passed to USERPGM, R1 points to the address of an address list containing an address:

```
R1 —> Address (parameter 1) —> X'0008' (the length of data passed), TESTDATA
(the data passed)
```

ℹ **Note:** USERPGM can be written in any language that supports z/OS linkage conventions.

CALLPROG={"program_parameters"}

with this batch SYSIN parameter, you call a user program defined on the remote system.

ℹ **Note:** `TRANS_TYPE` must be `COMMAND` if this parameter is specified; any other value will result in an error and the transaction will not be queued.

This parameter is supported only when communicating with TIBCO MFT Platform Server z/OS. If you supply parameters with the program to be called, the entire string must be enclosed in double quotation marks. The program to be called must be specified in the

STEPLIB statement of the remote TIBCO MFT Platform Server for z/OS started task. The linkage conventions used for this parameter are the same as the linkage conventions for programs called by another program. R1 points to an address which points to the parameter data entered on the command. Nulls are placed at the end of the data to show where the entered data ends.

For example, `CALLPROG="TESTPROG parameters"`.

CHKPT={Y | N | nnn}

Default	`N`
Allowable Values	`Y`, `N`, `nnn`
Minimum	`1`
Maximum	`360`
Alternate Specification	`CKPT`

You can use this parameter to specify whether to take checkpoints during this transfer. Where, `nnn` specifies the time in minutes that each checkpoint takes. If `Y` is specified, the interval defaults to five minutes.

ℹ **Note:** Checkpoints are taken at time intervals; therefore, you can start a transfer backup at the point of failure if the transfer is interrupted.

ℹ **Note:** Checkpoint restart processing is not supported when spanned files are used (`RECFM=VS` or `RECFM=VBS`).

COMPRESS={YES | NO | RLE | LZ | ZLIB1 - ZLIB9}

Default	`NO`
Allowable Values	`YES`, `NO`, `RLE`, `LZ`, `ZLIB1 - ZLIB9`

Minimum	Not Applicable
Maximum	Not Applicable

You can use this parameter to specify whether to use data compression, or specify the data compression algorithm to use.

Use `COMPRESS=NO` on high-speed links because compression actually slows down file transfer speeds. Use `COMPRESS=YES` when sending data over slow-speed links (speeds of T1 and below).

For more information on data compression on TIBCO MFT Platform Server for z/OS and data compression algorithms, see Data Compression.

CONTINUE={YES | NO}

Default	`NO`
Allowable Values	`YES`, `NO`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether transfers continue if a transfer fails. This parameter is used only when directory transfers are performed, and is ignored for non-directory transfer requests.

If you specify `CONTINUE=NO` and one of the file transfers within the directory fails, the Platform Server stops the processing. If you specify `CONTINUE=YES` and one of the file transfers within the directory fails, the Platform Server continues with the processing.

COSNAME={class_of_service_name}

Default	Not Applicable
Allowable Values	Class of Service name with 1 - 8 character.

Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the Class of Service name. The value of the COSNAME parameter must match the name of a Platform Server class of service that is enabled at startup or by the ENABLE operator command.

CRCCHECK or CRC

Default	NO
Allowable Values	YES, NO
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether CRC checking is turned on for this node. CRC checking is performed against the data that is sent over the network to ensure that the data has not been corrupted. The CRC is not performed against the file itself. ASCII to EBCDIC conversion, translation, and LF/CRLF change the contents of the file between the sender and receiver so the CRC is not performed against the file contents. Valid values are:

- YES: Performs CRC checking by default.
- NO: Does not perform CRC checking by default. NO is the default value

The CRCCHECK parameter on the Batch parameter overrides the Global CRCCHECK settings.

Note: If the partner Platform Server does not support CRC checking, CRC is computed but is not checked against the partner's CRC value computed.

CRLF={YES | NO | CRLF | LF | USSLF| KEEPCRLF}

Default	NO
Allowable Values	YES, NO, CRLF, LF, USSLF, KEEPCRLF
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the default CRLF option. This parameter is only used for Initiator requests when you do not enter the CRLF parameter.

The valid values of this parameter are as follows:

- YES or CRLF: indicates that records are delimited by Carriage Return/Line Feed (typically PC-based systems).

  Note: When you send a file to UNIX with CRLF delimiters, Platform Server for UNIX converts CRLF to LF. If you want to maintain CRLF on UNIX, see the KEEPCRLF parameter value.

- NO: indicates that the file contains no record delimiters.
- LF: indicates that records are delimited by line feeds (typically UNIX based systems).
- USSLF: indicates that you can send a z/OS USS file and terminate the USS file records with an ASCII LF(0x0A).
- KEEPCRLF: indicates that a file is sent to UNIX and wants to maintain CRLF delimiters. Platform Server for z/OS sends data with CRLF delimiters and Platform Server for UNIX does NOT convert CRLF to LF.

DATACLASS={SMS_data_class}

Default	None
Allowable Values	Data Class name with 1 - 8 characters.

Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	DC, DCLASS

This parameter represents the z/OS Data Class as defined to the Data Facility/System Managed Storage. In addition, it is used to indirectly select file attributes such as record format and logical record length. The value of this parameter can contain 1 - 8 numeric, alphabetic, or national characters ($, #, or @). The first character must be an alphabetic or national character.

DATASET_TYPE={LIBRARY | PDS | DSN | VSAM | LARGE}

Default	None
Allowable Values	LIBRARY, PDS, DSN, VSAM, LARGE
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the type of file to be created.

The valid values of this parameter are as follows:

- LIBRARY: indicates creating a PDS/E file.
- PDS: indicates creating a z/OS PDS file.
- DSN: indicates creating a sequential z/OS file.
- VSAM: indicates creating a VSAM file.
- LARGE: use this option for disk datasets with more than 65535 tracks per volume.

Note: When this parameter is not specified, TIBCO MFT Platform Server for z/OS creates a file of the same type as the source file.

DATE={yyyyddd | yyddd}

Default	Current date
Allowable Values	Current date - infinity
Minimum	Current date
Maximum	NONE

This parameter defines the Julian date at which the file transfer is eligible to commence. For example, 9 January, 2014 can be specified as `DATE=2014009` or `DATE=14009`. When this parameter is not specified, this parameter defaults to the current date.

Note: Let this parameter default to the current date; unless, you have a requirement to delay the file transfer.

DCB=({RECFM=xx,LRECL=nnn,BLKSIZE=nnn,DSORG=zz | Model DCB DSN})

This parameter defines the DCB information for a file being created. For a SEND transfer, the DCB information is associated with the remote system. For a RECEIVE transfer, the DCB information is associated with the local system.

The `DCB` parameter includes the following two formats:

- `DCB`={RECFM=xx,LRECL=nnn,BLKSIZE=nnn,DSORG=zz}: defines the `RECFM`, `LRECL`, `BLKSIZE`, and `DSORG` parameters in a single parameter. You can also set these parameters in the individual `RECFM`, `LRECL`, `BLKSIZE`, and `DSORG` parameters.

  Note: The `BLKSIZE` parameter is not required. If you do not specify it, the z/OS system selects an optimum block size for the file being created.

- `DCB`={ModelDSN}: defines a z/OS dataset that operates as a Model for the DCB attributes for the file being created.

> ℹ **Note:** The ModelDSN you specified must exist; otherwise, the transfer fails. This parameter can also be set by using the `DCB_LIKE` parameter.

DCB_LIKE={Model_DCB_DSN}

This parameter defines a z/OS dataset that operates as a Model for the DCB attributes for the file being created.

> ℹ **Note:** The ModelDSN you specified must exist; otherwise, the transfer fails. This parameter can also be set by using the `DCB` parameter.

DESCRIPTION={description}

Default	None
Allowable Values	Any string of up to 25 characters.
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	`DESC`

This parameter defines a description consisting of any alphabetical, numeric, or national characters of up to 25 characters. The description will be logged into the history files that describe this file transfer, and will also be displayed in the System Log at the end of the transfer.

> ℹ **Note:** Embedded spaces are allowed when enclosed in quotes.

This parameter is optional.

DIRTEST={YES | NO}

Default	`NO`
Allowable Values	`YES`, `NO`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the files to be transferred without actually performing the file transfer request. This parameter is used only when directory transfers are performed, and is ignored for non-directory transfer requests.

When this parameter is set to `YES`, all of the information about the transfer is extracted. Instead of executing the transfer, a message that indicates the local and remote files to be processed is written. When this parameter is set to `NO`, transfers are performed.

> **Note: Note:** The `DIRTEST` parameter requires the `WAIT=YES` definition. In addition, it returns file names only when the request is queued via cross memory by using `SERVER=`*`taskname`*; if the request is queued via TCPIP or SNA, no file names are returned.

DSN={data_set_name}

Default	None
Allowable Values	Any valid DSN of 1 - 54 characters.
Minimum	`1`
Maximum	`255`
Alternate Specification	`LF`, `LFILE`, `LDSN`, `LOCAL_FILE`

This parameter defines the dataset name of the local file. It can be any valid dataset name of 1 - 54 characters. When queuing a request to send a file to a remote open system, the dataset must exist at the time the request is queued and the person queuing the transfer must have the authority to access the file. The DSN parameter must be entered on only one line and cannot be continued onto another record.

Note: When specifying Generation Data Groups (GDG) with the DSN= parameter, you can specify a generation dataset by using either its true name (that is, with the G0000V00 qualifier) or the relative generation number. The alias name will be turned into the true name at the time the file transfer is scheduled. For example, if you specify the generation number -1, the dataset transferred will be the one that is -1 at queuing time, not necessarily at transmission time. This is to ensure that the proper generation is sent.

DSORG={PS | DA | PO | PDSE | PE | VS | USS}

Default	Default comes from the Source file being sent
Allowable Values	PS, DA, PO, PDSE, PE, VS, USS
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the dataset organization, and is only used when creating a file on a z/OS system.

The valid values are as follows:

- PS: Creates a sequential file.
- DA: Creates a BDAM file.
- PO: Creates a PDS.
- PDSE: Creates a PDSE.
- PE: Creates a PDSE (the same as PDSE)
- VS: Creates a VSAM file.
- USS: Creates a UNIX System Services file.

EFFECT={C | R | A | CR | CA | CN}

Default	CR
Allowable Values	C, R, A, CR, CA, CN
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the file creation options. The LDISP and RDISP parameters might override this parameter. Whichever parameter defined last in the batch job takes precedence.

The valid values are as follows:

- C: Creates the remote file. If the remote file already exists, abort the transfer.
- R: Replaces the remote file. If the remote file does not exist, abort the transfer.
- A: Appends the contents of the source file to the end of the remote file.
- CR: Creates the remote file or replaces it if it already exists.
- CA: Creates the remote file or appends the contents of the source file to the end of the remote file if it already exists.
- CN: Creates a file if it does not exist or replaces it if it already exists. If the directory where the file will be created does not exist, create it.

If the remote system is another z/OS computer and you are sending to a member of a partitioned dataset, specify CR or R. Specifying C will not work because the PDS itself already exists. If you specify A or CA for a PDS-PDS transfer, the member will be sent only if it does not exist in the recurring file. Otherwise, an error message is returned and the transfer fails.

EMAIL_FAIL={email_addresses}

This parameter defines the email address or email addresses to which to send an email notification when a TIBCO MFT Platform Server for z/OS file transfer request is unsuccessful. You can specify either a single email address or up to 4 email addresses separated by semicolons (;). The maximum length of this parameter is 64 bytes.

If you do not specify this parameter, no email will be sent when a request fails. This parameter is mutually exclusive with the `MQ_FAIL` parameter.

> **Note:** This parameter is in addition to the GLOBAL and NODE `EMAIL_FAIL` parameter. It does not override or replace the GLOBAL or NODE parameter.

EMAIL_GOOD={email_addresses}

This parameter defines the email address or email addresses to which to send an email notification when a TIBCO MFT Platform Server for z/OS file transfer request is successful. You can specify either a single email address or up to 4 email addresses separated by semicolons (;). The maximum length of this parameter is 64 bytes.

If you do not specify this parameter, no email will be sent when a request is successful. This parameter is mutually exclusive with the `MQ_GOOD` parameter.

> **Note:** This parameter is in addition to the GLOBAL and NODE `EMAIL_GOOD` parameter. It does not override or replace the GLOBAL or NODE parameter.

ENCRYPT={NONE | DES | 3DES | BLOWFISH | BF | BLOWFISH_LONG | BFL | AES}

This parameter defines the level of encryption to be used for this single transfer. If specified, this parameter overrides the `ENCRYPT` parameter specified in the GLOBAL or NODE definitions for this transfer.

The valid values are as follows:

- `NONE`: No encryption is used for this transfer.
- `DES`: DES encryption is used for this transfer.
- `3DES`: Triple DES encryption is used for this transfer.
- `BLOWFISH`| `BF`: Blowfish 56-bit encryption is used for this transfer.
- `BLOWFISH_LONG` | `BFL`: Blowfish 448-bit encryption is used for this transfer.
- `AES`: 256-bit encryption is used for this transfer.
- `NEVER`: No encryption is used for this node. Even if you override the `ENCRYPT` option in

the batch interface, encryption is turned off for this node.

For more information on this parameter, see Data Encryption.

EVERY

Default	Not Applicable
Allowable Values	DAY, HOUR, WEEKDAY, WEEKEND, SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, DDxx, DDLAST, DDFIRST, xxDAYS, xxHOURS, xxMINUTES
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines that repeat cycle of a request; when the request is completed, it is re-scheduled based on the values defined by this parameter.

The valid values are as follows:

- DAY: The request is scheduled every day of the week (at 24 hour intervals).
- HOUR: The request is scheduled every hour.
- WEEKDAY: The request is scheduled every weekday (Monday through Friday).
- WEEKEND: The request is scheduled every weekend day (Saturday and Sunday).
- SUNDAY: The request is scheduled every Sunday.
- MONDAY: The request is scheduled every Monday.
- TUESDAY: The request is scheduled every Tuesday.
- WEDNESDAY: The request is scheduled every Wednesday.
- THURSDAY: The request is scheduled every Thursday.
- FRIDAY: The request is scheduled every Friday.
- SATURDAY: The request is scheduled every Saturday.
- DDxx: the request is scheduled on the *xx* day of every month. If a month does not include day xx (for example, April 31), that month is skipped and the request is not

scheduled until the next month that contains day *xx*.

- DDLAST: The request is scheduled on the last day of the month.
- DDFIRST: The request is scheduled on the first day of the month.
- xxDAYS: The request is scheduled to run every xx days (01 - 99).
- xxHOURS: The request is scheduled to run every xx hours (01 - 99).
- xxMINUTES: The request is scheduled to run every xx minutes (01 - 99).

EVERY_COUNT={every_execution_count}

Default	0 (no count)
Allowable Values	0 - 32767
Minimum	0
Maximum	32767

This parameter defines the number of times that an EVERY request can be scheduled to execute. This parameter is meaningful only when the EVERY parameter is specified.

When this parameter is not defined or is specified as 0, no limit is assigned for the number of times that an EVERY request can be executed. When this parameter is specified as a non-zero value, the EVERY request will be removed from the request queue when the EVERY_COUNT value is reached.

EVERY_EXDAYS={number_of_expiration_days}

Default	0
Allowable Values	1 - 9999
Minimum	1
Maximum	9999

This parameter defines the number of days that a request scheduled by the EVERY parameter can remain in the request queue before the request expires in the request queue. This parameter is meaningful only when the EVERY parameter is specified.

If the default value of 0 is used, the default Platform Server expiration is used.

EVERY_EXTIME={hhmm}

Default	0
Allowable Values	0001 - 2359
Minimum	0001
Maximum	2359

This parameter defines the number of hours and minutes that each request scheduled by the EVERY parameter can remain in the request queue before each request expires in the request queue. This parameter is meaningful only when the EVERY parameter is specified.

If the default value of 0 is used, the EVERY_EXDAYS value or the default Platform Server expiration is used.

EVERY_MISSED={SCHEDULE | BYPASS}

This parameter defines whether an EVERY request can be scheduled at the Platform Server startup when the interval defined by the EVERY request has already passed. This parameter is meaningful only when the EVERY parameter is specified.

This can happen when Platform Server started task is brought down and up. For example, if you specify a request to run at 10:00 a.m. every day and the started task is brought down at 9:50 a.m. and brought up again at 10:10 a.m., the EVERY_MISSED parameter is used to determine whether the request should be scheduled at the startup, even though the actual interval has expired.

The valid values are as follows:

- SCHEDULE: Schedules the request at startup.
- BYPASS: Does not schedule the request at startup, and schedule it at the next interval.

EXEC={remote_command_to_execute}

This batch SYSIN parameter specifies a command or exec to be executed on the remote system.

> **Note:** When this parameter is defined, `TRANS_TYPE` must be `COMMAND`; otherwise, error will occur and the transaction will not be queued.

This parameter is supported only when communicating with TIBCO MFT Platform Server for z/OS version 5.4 and above. If you supply parameters with the command exec to be executed, the entire string must be enclosed in double quotation marks. For example, `EXEC="TESTEXEC parameters"`. When the remote system is z/OS, the command must be a REXX exec. When running on UNIX, it must be an executable UNIX command.

> **Note:** If the `DSN` (or `LFILE`) parameter is defined, any output created by the remote exec will be saved in the file defined by that parameter. Only UNIX and Windows systems provide any output from command execution.

EXECPREPROC={PARENT | CHILD}

This parameter defines whether directory transfer preprocessing actions are executed on the parent transfer or the child transfers.

The valid values are as follows:

- `PARENT`: Executes preprocessing on the parent transfer.
- `CHILD`: Executes preprocessing on the child transfers.

> **Note:** This parameter overrides the `EXECPREPROC` value defined in the GLOBAL configuration.

EXECPOSTPROC={PARENT | CHILD}

This parameter defines whether directory transfer postprocessing actions are executed on the parent transfer or the child transfers.

The valid values are as follows:

● PARENT: Executes postprocessing on the parent transfer.

● CHILD: Executes postprocessing on the child transfers.

This parameter overrides the EXECPREPROC value defined in the NODE and GLOBAL configurations.

EXPDT={yyddd | yyyyddd}

Default	30 days from today
Allowable Values	date of today - nnn
Minimum	date of today
Maximum	Not Applicable

This parameter defines the date, in Julian format, at which the file transfer will be purged off of the work queue if the file transfer is not yet completed. This parameter defaults to thirty days from now. It is a good practice to use the default value.

EXPTM={HHMM | +HHMM}

This parameter defines the expiration time of a file transfer.

You can specify this parameter in the following two formats:

- *HHMM*: Sets the expiration time to HHMM.
- *+HHMM*: Adds the value of this parameter to the current date or time (or the request date or time if the request date or time is not the current date or time) to get the expiration date and time of a file transfer.

> **Note:** When used in +HHMM format, this parameter also overrides any preceding EXPDT parameters.

FILEERRORINT={10 - 9999}

This parameter defines the wait interval when a retryable file error occurs. It defines how long a transfer waits before being retried. When defined, this parameter overrides the NODE and GLOBAL `FILE_ERROR_TRY_INTERVAL` parameters.

The valid values are from `10` to `9999`. When not defined, the NODE `FILE_ERROR_TRY_INTERVAL` parameter is used. When the NODE `FILE_ERROR_TRY_INTERVAL` is not defined, the GLOBAL `FILE_ERROR_TRY_INTERVAL` parameter is used.

Note: The actual interval can be up to 60 seconds or more than the value specified, depending on when the error occurs and when the dispatcher runs.

FILEERRORMAX={1 - 9999}

This parameter defines the maximum number of retryable file errors. When the number of retryable file errors reaches this limit, the transfer is terminated. When defined, this parameter overrides the NODE and GLOBAL `FILE_ERROR_TRY_COUNT` parameters.

The valid values are from `1` to `9999`. When not defined, the NODE `FILE_ERROR_TRY_COUNT` parameter is used. When the NODE `FILE_ERROR_TRY_COUNT` is not defined, the GLOBAL `FILE_ERROR_TRY_COUNT` parameter is used.

FILE_EXPDT={YYDDD | YYYYDDD}

This parameter defines the date, in Julian format, at which the file expires. This parameter is typically used in tape processing to prevent a tape from being overwritten. Take special care when using this parameter with a disk file. The default is no expiration date on the file.

FILE_RETPD={DDDD}

This parameter defines the number of days after which the file expires. This parameter is typically used in tape processing to prevent a tape from being overwritten. Take special care when using this parameter with a disk file. The default is no expiration date on the file.

HOLD={YES | NO}

This parameter defines whether the transfer commence immediately.

If you specify `HOLD=YES`, the transfer will not commence until explicitly released by an operator. It is a good practice to specify `HOLD=NO`.

IPADDR={xxx.xxx.xxx.xxx}

This parameter defines the IPV4 or IPV6 IP address of the remote destination. If this parameter is specified, the transfer is TCP/IP.

The parameter must be a valid IPV4 or IPV6 IP Address.

Note: This parameter is mutually exclusive with `IPNAME`.

IPNAME={domain_name}

This parameter defines the IP name of the remote destination. If this parameter is specified, the transfer is TCP/IP.

The name can be up to 64 bytes long (for example, `JohnD.XYZ.COM`).

Note: This parameter is mutually exclusive with `IPADDR`.

If you want to use the DNS lookup feature (`IPNAME=…`), you must concatenate the IBM C Runtime libraries in the STEPLIB. Also, if you want to use DNS lookup, the SYSTCPD DD name must be in the Platform Server JCL and must point to the TCPIP DATA config file.

IPORT={nnnnn | 0}

This parameter defines the IP port of the remote destination. The valid range is `0` - `65535` and the default is `0`. If the default value of `0` is used, the IP port defined (or defaulted to) in the GLOBAL `TCPIPPORT` parameter is used.

IUNIT={8_character_unit_name}

This parameter defines the unit name for the input tape file, and is only used when sending uncataloged tape files. This parameter is ignored for all RECEIVE transfers or for SEND transfers where the input dataset is cataloged or on a disk.

This parameter does not have a default value.

> **Note:** When this parameter is specified, the `IVOLUME` parameter and `TAPE=YES` must also be specified.

IVOLUME={volume_name}

This parameter defines the volume name that is used for the input dataset for a transfer. when this parameter is used, the catalog is not used to allocate the input dataset. The Platform Server verifies that the dataset exists on the `IVOLUME` specified. This parameter must be used with extreme care because most sites use only cataloged datasets. This parameter is valid only the Platform Server is sending a file to a remote system, or is receiving a file from another TIBCO MFT Platform Server z/OS.

This parameter can contain 1 - 6 characters.

LDISP={SHR | OLD | DNEW | MOD | NEW | NEWR | NEWA},{KEEP | CATLG | DELETE},{KEEP |CATLG | DELETE}

This parameter defines the disposition of the local dataset. It is broken up into three parameters.

The following table lists the dataset status options:

Transfer Direction	Option	Short Description
Send or Receive	SHR	Requests shared access to the dataset.

Transfer Direction	Option	Short Description
Receive	`OLD`	Requests exclusive access to the dataset.
	`DNEW`	Deletes the file if it already exists, and then re-creates the file; creates the file If the file does not exist.
	`MOD`	Appends data to the end of the dataset.
	`NEW`	Creates dataset. If dataset already exists, returns error.
	`NEWR`	Creates dataset. If dataset already exists, replaces the dataset.
	`NEWA`	Creates dataset. If dataset already exists, appends data to the end of the dataset.

The following table lists the normal disposition options:

Transfer Direction	Option	Short Description
Send	`KEEP`	Keeps the dataset after the transfer is completed successfully. If a dataset is created, do not catalog the dataset. **Note:** Use this option with great care.
Receive new dataset	`CATLG`	Catalogs the dataset If a dataset is created. If a dataset is not created, keeps the dataset.
Send	`DELETE`	Deletes the dataset after the transfer is completed successfully.

The following table lists the error disposition options:

Transfer Direction	Option	Short Description
Send	`KEEP`	Keeps the dataset after the transfer is completed successfully. If a dataset is created, do not catalog the dataset. **Note:** Use this option with great care.
Receive new dataset	`CATLG`	Catalogs the dataset If a dataset is created. If a dataset is not created, keeps the dataset.
Send	`DELETE`	Deletes the dataset after the transfer is completed unsuccessfully.

This parameter works like the mainframe `LDISP` parameter. You only need to specify the first parameter for a RECEIVE transfer.

The `EFFECT` parameter might override the `LDISP` and `RDISP` parameters. Whichever parameter defined last in the batch job takes precedence.

> **Note:** In some conditions, the error disposition of `DELETE` cannot be processed. The dataset deletion is performed in the Dynamic deallocation routine. If the dataset has not been allocated, it cannot be deleted. For example, if a request fails because the dataset is not available, the dynamic allocation fails; therefore, the dataset cannot be deleted because it is not allocated.

LENGTH={nnnnn | 0}

Default	`0`
Allowable Values	`0` - `32760`

Minimum	`4 for RECFM=F|FB`
	`5 for RECFM=V|VB`

Maximum	`32760`

Alternate Specification	`LRECL`

This parameter defines the maximum logical record length, also called the string length, which is used to encode the data records of the file.

It is a good practice to omit this parameter if you are sending or receiving a file into a file that already exists because the Platform Server will determine the appropriate length. If you are receiving (initiator receive) a file that does not exist, you must indicate the length if you do not want the Platform Server to use the default value.

LIST={node_name}

Default	Not Applicable
Allowable Values	Symbolic name with 1 - 8 characters
Minimum	1-character symbolic name
Maximum	8-character symbolic name

This parameter defines the symbolic name of the list of destinations for this transmission. This parameter must match one of the member names of the configuration library that defines the list of remote computers.

ℹ **Note:** The `LIST` parameter cannot be used in conjunction with the NODE `IPADDR` or `IPNAME` parameter.

LOCALLABEL=(nn,SL)

Default	(1,SL)
Allowable Values	The file label number, `SL`
Minimum	N/A

Maximum	N/A
Alternate Specification	LLABEL

This parameter defines the label used on the initiating Platform Server when sending or receiving a tape file.

NN defines the file label number. `SL` is the only label type currently supported.

Note: For sending a tape file, if the label information is in the catalog, this parameter is not required.

Note: This parameter is ignored for disk files.

LOCALCTFILE={Translation_table_used_locally}

Default	Not Applicable
Allowable Values	`NULL` , `NONE`, Translation table name with 1 - 16 characters
Minimum	1
Maximum	16
Alternate Specification	LCT

This parameter defines the local translation table name. The translation table name can contain 1 - 16 characters.

Translation tables are defined in the CONFIG DD statement, and can be enabled at startup or enabled through the ENABLE operator command. The `LOCALCTFILE` translation table is used when `TYPE=TEXT` transfers are defined. This parameter is ignored when `TYPE=BINARY` is specified. A value of `NULL` or `NONE` indicates that no conversion takes place.

The `LOCALCTFILE` parameter on the file transfer request overrides the `DEFAULT_LOCALCTFILE` parameter.

To use z/OS Unicode Conversion Services, the following format of this parameter is required.

```
LOCALCTFILE=CC:xxxxx:yyyyy
```

Where:

xxxxx defines the CCSID of the Local data - usually the EBCDIC CCSID.

yyyyy defines the CCSID of the Remote data - usually the ASCII CCSID

For information on the CCSID values to use, see the IBM manual: *z/OS Unicode Services User's Guide and Reference*.

Example: `LOCALCTFILE=CC:01140:01208`

- For a Send Request, this parameter converts data from EBCDIC CCSID 1140 to ASCII UTF-8 CCSID 01208.
- For a Receive Request, this parameter converts data from ASCII UTF-8 CCSID 01208 to EBCDIC CCSID 1140.

When using this parameter, set the following parameter so that the partner does NOT perform any conversion:

REMOTECTFILE=NULL

LOGON_DOMAIN={domain}

Default	None
Allowable Values	1 - 16 characters
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	`DOMAIN`

This parameter defines the Windows domain to be used to authenticate a particular file transfer request that is initiated by the mainframe.

Note: This field is Windows-specific and is only valid when sending files to a Windows machine.

LPASS={local_password}

Default	Not Applicable
Allowable Values	Password with 1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the password associated with the local user ID. This parameter is ignored if the `LUSER` parameter is not defined. If the `LUSER` parameter is defined without a password, the Platform Server checks whether the user that initiates the transfer request is authorized to log on as a SURROGATE for the defined `LUSER`. For more information on SURROGATE logon checking, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

LRECL={nnnnn | 0}

Default	`0`
Allowable Values	`0 - 32760`
Minimum	`0`
Maximum	`32760`
Alternate Specification	`LENGTH`

This parameter defines the maximum logical record length, also called the string length, which is used to encode the data records of the file.

It is a good practice to omit this parameter if you are sending or receiving a file into a file that already exists because the Platform Server will determine the appropriate length. If you are receiving (initiator receive) a file that does not exist, you must indicate the length if you do not want the Platform Server to use the default value.

LUSER={local_user_ID}

Default	Not Applicable
Allowable Values	User ID with 1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the local user ID under whose authorization the file transfer is run. The `LUSER` parameter can be used together with the `LPASS` parameter. If the `LUSER` parameter is defined without a password, the Platform Server checks whether the user that initiates the transfer request is authorized to log on as a SURROGATE for the defined `LUSER`. For more information on SURROGATE logon checking, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

MAINTAINBDW={YES | NO | BLOCKVS}

Default	`NO`
Allowable Values	`YES`, `NO`, `BLOCKVS` **Note:** Used when transferring VBS files between z/OS systems when the LRECL exceeds 32k.
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether `RECFM=VB BDW` (Block Descriptor Word) is maintained when sending or receiving files. This parameter must only be specified when the file being transferred is in `RECFM=VB` format. The BDW and the RDW (Record Descriptor Words) are maintained when sending data to a remote system or receiving data from a remote system. If the data being sent or received is not in the proper BDW/RDW format, the transfer fails.

MAINTAINRDW={YES | NO}

Default	`NO`
Allowable Values	`YES`, `NO`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether `RECFM=VB RDW` (Record Descriptor Word) will be maintained when sending or receiving files. This parameter must only be specified when the file being transferred is in `RECFM=VB` format. The RDW is maintained when sending data to a remote system or receiving data from a remote system. If the data being sent or received is not in the proper RDW format, the transfer fails.

MEMBER={member_name_to_include}

Default	None
Allowable Values	Up to 25 member names with 8 characters
Minimum	Not Applicable
Maximum	25 member names

This parameter defines the names of the members to be included in a PDS file transfer. You can enter this parameter multiple times on a single file transfer to define up to 25 member names. This parameter can only be used for z/OS to z/OS transfers, and is ignored when the dataset names on both ends of the file transfer are not PDS or PDSE files.

Note: This parameter is mutually exclusive with the `XMEMBER` parameter.

MGTCLASS={SMS_Management_Class}

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	MC, MCLASS

This parameter defines the z/OS Management Class as defined to the Data Facility/System Managed Storage. This parameter can contain 1 - 8 characters, including numeric, alphabetic, or national characters ($, #, or @). The first character must be an alphabetic or national character.

MQ_FAIL={$MQ:XXXX:mmmmmmmmmm}

This parameter defines the MQ (Message Queue) to which Platform Server writes a record when a request fails. If this parameter is not specified, no record will be written to the MQ when a request fails.

Note: This parameter is mutually exclusive with the EMAIL_FAIL parameter.

$MQ is a fixed literal and must be the first 3 bytes; XXXX defines the MQ Queue Manager name; and mmmmmmmmmm defines the MQ name, and can contain 1 - 55 characters.

Note: This parameter is in addition to the GLOBAL MQ_FAIL parameter. It does not override or replace the GLOBAL parameter.

MQ_GOOD={$MQ:XXXX:mmmmmmmmmm}

This parameter defines the MQ (Message Queue) to which the Platform Server writes a record when a request is completed successfully. If this parameter is not specified, no record is written to the MQ when a request is successful.

Note: This parameter is mutually exclusive with the EMAIL_GOOD parameter.

$MQ is a fixed literal and must be the first 3 bytes; XXXX defines the MQ Queue Manager name; and mmmmmmmmmmm defines the MQ name, and can contain 1 - 55 characters.

Note: This parameter is in addition to the GLOBAL MQ_GOOD parameter. It does not override or replace the GLOBAL parameter.

MQ_NOTIFY={INITIATOR | RESPONDER | BOTH | NONE}

This parameter defines when the Platform Server sends MQ notifications.

The valid values are as follows:

- NONE: No MQ notification is sent.
- INITIATOR: MQ notification is sent on Initiator tasks.
- RESPONDER: MQ notification is sent on Responder tasks.
- BOTH: MQ notification is sent on both Initiator and Responder tasks.

Note: The GLOBAL MQ_GOOD and MQ_FAIL parameters are also required before notifications take place.

MQ_WAIT_MSEC

This parameter defines the interval in milliseconds that MQ waits for additional data before returning EOF. If this parameter is not defined, the default value of 5000 is used, which causes MQ to wait for additional data for 5 seconds before returning EOF.

The maximum value allowed is 99999999. When 0 is specified, MQ does not wait for additional data and returns EOF after reading the last record.

NODE={node_name}

Default	Not Applicable
Allowable Values	Symbolic names with 1 - 32 characters
Minimum	1-character symbolic name
Maximum	32-character symbolic name

This parameter defines the symbolic name of the target node for this transmission. This parameter might match one of the member names of the configuration library that defines the remote computer. If the member does not exist in the configuration library, the node parameter specified is used as the remote LU name. The Platform Server initiates the transfer to the LU name specified.

NODECLASS

Default	0
Allowable Values	0 - 255
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the node class to be used for this file transfer request. This parameter is only used when the NODE definition associated with the request defines the DEFAULT_NODECLASS parameter with a non-zero value. If the transfer request does not use a NODE definition, or if the NODE definition specifies DEFAULT_NODECLASS=0, NODECLASS processing is not performed. If you define the NODECLASS parameter with a value greater than the NODE WINNERS parameter, the request is flagged as an error and the request is retried at the next error interval.

NOTIFY={user_ID}

Default	None
Allowable Values	Names with 1 - 8 characters
Minimum	1-character name
Maximum	8-character name

This parameter defines the name of the user to notify when this file transfer is completed, either successfully or unsuccessfully. It is a good practice to specify this parameter as either your own user ID, or that of one of your Operations Support team.

Note: This parameter is used in conjunction with the `NOTIFY_TYPE` parameter.

NOTIFY_TYPE={TSO}

Default	`TSO`
Allowable Values	`TSO`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the type of user ID to notify when the file transfer is completed. Currently, the only option supported is TSO.

Note: This parameter is used in conjunction with the `NOTIFY` parameter.

OVOLUME={output_volume}

Default	None
Allowable Values	Volume names with 1 - 6 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the volume that is used for the output dataset for a transfer. If this parameter is used, the catalog is not used to allocate the output dataset. The Platform Server verifies that the dataset exists on the output volume specified.

This parameter must be used with extreme care, because most sites use only cataloged datasets. If you use this parameter with a cataloged dataset, and specify a different volume than the one the cataloged dataset resides on, a duplicate uncataloged dataset will be created on the `OVOLUME` volume.

Note: This parameter is valid only when Platform Server is receiving a file from a remote system or is sending a file to another TIBCO MFT Platform Server z/OS.

PERMIT_ACTNS={S | H | A | R | C | Z | E}

Default	None
Allowable Values	`S`, `H`, `A`, `R`, `C`, `Z`, `E`
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	`PermittedActions`

This parameter defines the Windows-permitted action.

The valid values are as follows:

- `S` (SYSTEM_FILE): Indicates that the file is a system file and can only be viewed by the operating system instead of the user.
- `H` (HIDDEN_FILE): Indicates that the file cannot be seen by the user.
- `A` (ARCHIVE_FILE): Select this option if you want to mark a file that has changed since it was last backed up.
- `R`: Indicates that the file being accessed can only be viewed by the user. No changes can be made to the file.
- `C`: Indicates that you can create a file as compressed on the remote system. This attribute is only available on NTFS partitions. If the receiving file system is not NTFS, the option is ignored.
- `Z`: Appends a CR/LF (0x0d, 0x0a) to the end of the file, followed by the DOS End of File character, Control Z (0x1a). If a trailing Control Z or CR/LF is already present, it does not add them again. This feature is only available when Carriage Return/Line Feed processing is enabled.
- `E`: Appends a DOS End of File character, Control Z (0x1a).

Note: The previous fields are Windows-specific and are only valid when sending down to a Windows machine.

POST_ACTION={S | F | P},{L | R},{CALLJCL | CALLPGM | SUBMIT | COMMAND},{data_to_be_processed}

Default	None
Allowable Values Parameter 1	`S`: Success, `F`: Failure, `P`: Pre-Processing
Allowable Values Parameter 2	`L`: Local, `I`: Initiator, `R`: Remote, `R`: Responder **Note:** Local and Initiator are the same. Remote and Responder are the same.
Allowable Values Parameter 3	`CALLJCL`: CALL with JCL linkage

	CALLPGM: CALL with Program linkage SUBMIT: Submit JCL to internal reader COMMAND: Execute Command
Allowable Values Parameter 4	Data to be processed
Alternate Specification	PPA

This parameter defines the postprocessing or preprocessing actions that can be performed for a file transfer. Up to 4 postprocessing actions can be defined for an individual file transfer. Actions can be performed based on the success or failure of a file transfer as well as be performed locally or remotely. Any combination of up to four S | F, L | R, and command types are allowed. The table below indicates the type of actions that can be performed, and the data necessary for each action:

Action	Support	Data
CALLJCL	z/OS	Program name (1 - 8 bytes) followed by data
CALLPGM	z/OS	Program name (1 - 8 bytes) followed by data
SUBMIT	z/OS	DSN (MEMBER) or MEMBER name
COMMAND	Windows, UNIX	Command name followed by data

Note: The difference between CALLJCL and CALLPGM is the way that the parameters are passed to the called program. CALLJCL uses JCL linkage in which the parameter length is the first two bytes of the data passed. CALLPGM uses standard IBM linkage conventions for calling programs.

Parameters can be delimited by commas or colons. If any spaces are imbedded in the data, enclose the entire parameter in double quotation marks.

Note: For the SUBMIT action, if only a member name is defined, the GLOBAL JOB_SUBMIT_DSN parameter must be defined.

The POST_ACTION data field is limited to 256 characters. To alleviate that restriction and to make it easier to define the parameter data, the Platform Server supports substitutable parameters within the POST_ACTION data. You can use the substitutable parameters to define fields such as the local file name. In this way, you do not have to define the file name twice, and it takes fewer characters to use a substitutable parameter than an actual file name. For more information on substitutable parameters, see Appendix L. Post Processing Action (PPA) Substitutable Parameters. For information on preprocessing actions, see Appendix L. Preprocessing Actions.

PRIORITY={3 | n}

Default	3
Allowable Values	0 - 9 (decimal numbers only)
Minimum	0
Maximum	9

This parameter defines the priority of the file transfer. 0 indicates the lowest priority and 9 indicates the highest priority.

PROCNAME={process_name}

Default	Name specified in a PROCESS statement
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the process name used for the file transfer. The process name can be specified by the PROCESS statement, or via the PROCNAME parameter. If both are specified, the PROCNAME parameter overrides the value specified in the PROCESS statement.

PURGE={YES | NO}

Default	NO
Allowable Values	YES , NO
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the action to be taken when a request times out. This parameter is used only when the WAIT =YES and TIMEOUT parameters are also specified.

If PURGE=YES is specified, a request will be purged from the queue when the TIMEOUT interval expires before the request is completed. If PURGE=NO is specified (or defaulted to), the request will not be purged from the queue when the TIMEOUT interval expires. Either way, a return code will notify you that a request times out.

RDISP={SHR | OLD | DNEW | MOD | NEW | NEWR | NEWA},{KEEP | CATL | DELETE},{KEEP | CATL | DELETE}

This parameter defines the disposition of the dataset on the remote Platform Server system. This parameter is only supported on transfer requests to other z/OS systems. The parameter is divided into three parameters.

The following table lists the dataset status options:

SHR	Requests shared access to the dataset.
OLD	Requests exclusive access to the dataset.
DNEW	Deletes the file if it already exists, then re-creates the file; creates the file if the file does not exist.
MOD	Appends data to the end of the dataset.
NEW	Creates dataset. If the dataset already exists, returns error.

`NEWR`	Creates dataset. If the dataset already exists, replaces the dataset.
`NEWA`	Creates dataset. If the dataset already exists, appends data to the end of the dataset.

The following table lists the normal deposition options:

`KEEP`	Keeps the dataset after the transfer is completed successfully. If a dataset is created, do not catalog the dataset. **Note:** Use this option with great care.
`CATL`	If a dataset is created, catalogs the dataset. If a dataset is not created, keeps the dataset.
`DELETE`	Deletes the dataset after the transfer is completed successfully.

The following table lists the error deposition options:

`KEEP`	Keeps the dataset after the transfer is completed unsuccessfully. If a dataset is created, do not catalog the dataset. **Note:** Use this option with great care.
`CATLG`	If a dataset is created, catalogs the dataset. If a dataset is not created, keeps the dataset.
`DELETE`	Deletes the dataset after the transfer is completed unsuccessfully.

The `RDISP` parameter works like the mainframe `RDISP` parameter.

The `EFFECT` parameter might override the `RDISP` and `LDISP` parameters. Whichever parameter defined last in the batch job takes precedence.

Note: In some conditions, the error disposition of DELETE cannot be processed. The dataset deletion is performed in the Dynamic deallocation routine. If the dataset is not allocated, the delete operation cannot be performed. For example, if a request fails because the dataset is not available, the dynamic allocation fails; therefore, the dataset cannot be deleted because it is not allocated. Another good example is if the request fails with a network error because the remote system is not available; obviously, the dataset cannot be deleted because communication with the remote system is not established.

RECFM={F | FB | V | VB | U | VS | VBS}

Default	V
Allowable Values	F, FB, V, VB, U, VS, VBS
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the significance of the character logical record length (semantics of LRECL boundaries). You can specify fixed, variable, or system default.

The valid values are as follows:

- F: Each string contains exactly the number of characters defined by the string length parameter.
- FB: All blocks and all logical record are fixed in size. One or more logical records reside in each block.
- V: The length of each string is less than or equal to the string length parameter.
- VB: Blocks as well as logical record length can be of any size. One or more logical records reside in each block.

> **Note:** Using checkpoint restart when writing to variable length blocked records might result in a D37 abend if insufficient secondary extents are defined. It is a good practice to make sure sufficient secondary extents are allocated or turn checkpoint restart off for variable length blocked records.

- `U`: Blocks are of variable size. No logical records are used. The logical record length is displayed as zero. This record format is usually only used in load libraries. Block size must be used if you are specifying `U`.
- `VS`: Records are variable and can span logical blocks. `RECFM=VS` is not supported when checkpoint restart is used.
- `VBS`: Blocks as well as logical record length can be of any size. One or more logical records reside in each block. Records are variable and can span logical blocks. `RECFM=VBS` is not supported when checkpoint restart is used.

REMOTECTFILE={remote_file_name}

Default	Not Applicable
Allowable Values	Translation table names with 1 - 16 characters, `NULL`, or `NONE`
Minimum	1
Maximum	16
Alternate Specification	RCT

This parameter defines the remote Translation table name. The Translation table name must be defined on the remote system. You can enter a Translation table name of 1 - 16 characters. The `REMOTECTFILE` translation table is used when `TYPE=TEXT` transfers are defined. This parameter is ignored when `TYPE=BINARY` is specified. A value of `NULL` or `NONE` indicates that no conversion will take place.

The `REMOTECTFILE` parameter on the file transfer request overrides the `DEFAULT_REMOTECTFILE` parameter.

To use z/OS Unicode Conversion Services, the following format of this parameter is required.

```
REMOTECTFILE=CC:xxxxx:yyyyy
```

Where:

xxxxx defines the CCSID of the Local data - that is, z/OS data read or written to disk.

yyyyy defines the CCSID of the Remote data - that is, data sent to or received from a partner

For information on the CCSID values to use, see the IBM manual: *z/OS Unicode Services User's Guide and Reference*.

- For a Send Request (that is, Read), xxxxx is the Source CCSID and yyyyy is the target CCSID.
- For a Receive Request, (that is, Write), yyyyy is the Source CCSID and xxxxx is the target CCSID.

When using this parameter, set the following parameter so that the local system does NOT perform any conversion:

LOCALCTFILE=NULL

REMOTELABEL=(nn,SL)

Default	(1,SL)
Allowable Values	The file label number, SL
Minimum	N/A
Maximum	N/A
Alternate Specification	RLABEL

This parameter defines the label used on the target Platform Server when sending or receiving a tape file.

NN defines the file label number. SL is the only label type currently supported.

> **Note:** For sending a tape file, if the label information is in the catalog, this parameter is not required.

Note: This parameter will be ignored for disk files.

REMOTE_FILE={remote_file_name}

Default	None
Allowable Values	Any combination of up to 255 characters (including embedded spaces). The file name must be enclosed in double quotation marks if it contains embedded spaces.
Minimum	1 character
Maximum	255 characters
Alternate Specification	`RF`, `RFILE`, `RDSN`

This parameter defines the name of the file on the remote system that is the subject of the activity (the name of the file on the remote computer). The file name can include any combination of up to 255 characters (including embedded spaces) as long as it is a valid name on the remote system. If the remote file name contains embedded spaces, the file name must be enclosed in double quotation marks.

If a file name goes past column 71 of the batch SYSIN file, you must place a continuation mark in column 72 (non-blank character) and start the next line in column 1. This can be repeated to go on to a third or fourth line, if necessary.

TIBCO MFT Platform Server for z/OS supports you to specify a directory as the remote file name by ending the remote file name with a backslash (\). If this is the case, the host dataset name is appended to the remote file name. Therefore, the host dataset name becomes the file name in the directory specified as the remote file name.

For example, you are sending a file from the host to the PC, and `DSN=PROD.SALES.A0405` and `REMOTE_FILE=C:\SALES\` are specified. The PC file name will be `C:\SALES\PROD.SALES.A0405`.

REMOTE_PASS

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	RPASS

This parameter defines the password on the remote system. It is normally used together with the REMOTE_USER parameter to specify the user ID and password of a user on a remote system. This parameter is required, unless no security is defined on the remote system or the user ID on the remote system does not require a password.

When the remote system is TIBCO MFT Platform Server for z/OS, you can optionally change the password of the remote user ID. This is done by typing the old password followed by a slash and the new password.

REMOTE_UNIT={unit_name}

Default	0
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	UNIT

This parameter defines the unit where the dataset will be written. This parameter is only used when the file is being written to a z/OS system.

REMOTE_USER={remote_user_ID}

Default	None
Allowable Values	Characters, 0 - 20 bytes in length
Minimum	0
Maximum	9
Alternate Specification	RUSER

This parameter defines the user ID to be sent to the remote computer for verification purposes. It can be any characters and 0 - 20 bytes in length. This ID must match the ID defined on the remote system. Most systems use this as a security check.

When you specify *PROFILE, the Platform Server checks the PROFILE dataspace for a profile that matches the NODE/IPNAME/IPADDR/LIST value.

REMOVETRAIL={YES | NO}

Default	NO
Allowable Values	YES , NO
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	RMTRAIL

This parameter is used only when sending or receiving text files to or from a UNIX or Windows machine and works differently depending on whether you are sending or receiving a file.

When sending text files to remote systems with CRLF=YES and this parameter is specified as YES, the z/OS system removes all trailing spaces and nulls before the file is transmitted. This causes the transfer to run faster and take up less disk space on the remote system.

This parameter is ignored when communicating with TIBCO MFT Platform Server z/OS or for binary transfers.

When receiving text files and this parameter is specified as `YES`, the z/OS system removes trailing spaces under the following circumstances:

- When Receiving Text files from UNIX or Windows
- When the `RECFM` parameter is `V` or `VB`.

RESOLVE_GDG={IMMEDIATE | DEFERRED}

Default	`IMMEDIATE`
Allowable Values	`IMMEDIATE` , `DEFERRED`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether a GDG is resolved immediately (when the transfer is scheduled) or deferred (when the transfer is executed). By default, the Platform Server resolves the GDG name immediately.

RETENTIONPRD={nnn}

This batch parameter defines the number of days that an activity can remain on the work queue before it expires and is purged. The value must be `0` - `365` days. If `0` is specified, the file never expires.

RETRY={nn | 0}

Default	`0`
Allowable Values	`0` - `999`
Minimum	`0`

Maximum	999
Alternate Specification	TRY

This parameter defines the maximum number of times that the file transfer can be attempted before it is purged from the Platform Server work queue. If defined, this parameter overrides the value defined for `DEFAULT_TRY_COUNT` in the GLOBAL definition.

> **Note:** If the `TRY` count is exceeded for transfers that receive recoverable file errors, transfers is terminated regardless of the `FILE_ERROR_TRY_COUNT` setting.
>
> `RETRY=1` indicates that the Platform Server tries the transfer once and then purges the transfer no matter whether the transfer is successful or not.
>
> `RETRY=0` means the number of tries defaults to the value defined by the `DEFAULT_TRY_COUNT` parameter in the GLOBAL definition.

RETRYINT={nnM | nnH | 0}

The parameter defines the retry interval of a file transfer. The parameter can be specified in terms of hours (for example, `RETRYINT=2H`) or minutes (for example, `RETRYINT=0M`). If neither `H` or `M` is specified, the parameter is assumed to be in minutes (for example, `RETRYINT=5` is the same as `RETRYINT=5M`).

When the Platform Server receives a network error attempting to transfer a file, the request and node definition are marked as an error. The Global `RESETINTERVAL` parameter determines when errors are reset. When the errors are reset, the Platform Server retries the request. In some instances, you do not want to retry the request so often. For example, if you send data to customers overnight and some of the customers turn their PCs off at night, you might want to retry the transfers less frequently. You can specify `RETRYINT=1H` to try sending the requests every hour.

> **Note:** This parameter does not override the error settings for the request and NODE. A request is sent when both the request and node are not in an error state.

RSACCELERATE={YES | NO}

You can set this parameter to `Y` to force a transfer to be conducted through a TIBCO MFT Platform for Windows Platform TIBCO Accelerator server using the TIBCO Accelerator technology which greatly improves data transfer speeds over IP networks with high latency.

RSCOMPRESS={YES | NO | BEST | DEFAULT | FAST}

When conducting file transfers through an TIBCO Accelerator, you can configure the TIBCO Accelerator server to compress the data being transferred. The TIBCO Accelerator uses a proprietary compression compatible with ZLIB.

By setting this parameter to `DEFAULT`, your file receives the greatest compression and might take slightly longer to transfer; by setting this parameter to `FAST`, your file is less compressed but is sent out faster.

RSENCRYPT={YES | NO}

When conducting file transfers through an RSAccelerator (RSA), the RSA server encrypts the data with a 256-bit Blowfish encryption key when this parameter is set to `YES`. When this parameter is set to `NO`, no encryption is used.

RSHOST={NO | host}

This parameter defines the IP address or host name of the TIBCO MFT Platform Server for Windows with TIBCO Accelerator enabled. By defining a host on the command line or in a transfer template, you override the `RSHost` value configured in the `config.txt` file if `RSHost` is defined. If you set this parameter to `NO` and set `RSAccelerator` to `YES`, the value configured for `RSHost` in the `config.txt` file is used.

RSMAXSPEED

When you conduct file transfers through an RSAccelerator (RSA), this parameter defines the maximum speed in Kilobytes per second to be used by the RSA server when you set this parameter in your command line or transfer template. This parameter can be 256 - 100000 kbps.

RSPORT={NO | port}

This parameter defines the port number the TIBCO MFT Platform Server for Windows Platform TIBCO Accelerator server is listening on for transfers using the TIBCO Accelerator technology. By defining a port number on the command line or in a transfer template, you override the `RSPort` value configured in the `config.txt` file. The default value of this parameter is `9099`. If you set this parameter to `NO` and set `RSAccelerator` to `YES`, the value configured for `RSPort` in the `config.txt` file is used.

RSPROTOCOL={UDP | PDP | TCP}

When conducting file transfers through an RSAccelerator (RSA), you can specify the RSA server to use its own enhanced version of User Datagram Protocol (`UDP`), TIBCO Accelerator's parallel implementation of TCP which is called Parallel Delivery Protocol (`PDP`), or straight TCP (`TCP`) by using this parameter.

SCRIPT={script_file_name}

Default	None
Allowable Values	Up to 255 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the name of the script file that should be scheduled to run in TIBCO MFT Platform Server for z/OS started task. This parameter is only valid when the activity defined on the PROCESS card is `SCRIPT`. Variable parameters can be passed to the SCRIPT file by appending a comma and the variable parameters needed.

You can specify this parameter in the following two ways:

- Specify the fully qualified dataset name.

  For example, `SCRIPT=MY.PDS(SCRIPT),%DSN=ZOS.ACCT.FILE`.

- Specify the member name only.

  For example, `SCRIPT=SCRIPT,%DSN=ZOS.ACCT.FILE`.

Note: When this method is used, the SCRIPT DD statement must be defined to the started task and the member defined by the `SCRIPT` parameter must exist in the datasets defined by the SCRIPT DD statement.

This schedules the script SCRIPT to run, and also passes a variable parameter to the script called `%DSN`.

SENDERCFMU={YES | NO | NOBDW}

Default	None
Allowable Values	`YES`, `NO`, `NOBDW`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether to maintain RECFM=VB BDW when sending or receiving files.

The valid values are as follows:

- `YES`: When `YES` is specified for a send file transfer, the file is opened using `RECFM=U`. Each block is written as an individual record. The main use of this parameter is to retain the `RECFM=VB` block and record structure when sending a file to a UNIX or Windows platform.
- `NO`: When `NO` is specified for a send file transfer, the file is not opened using `RECFM=U`. The default value is `NO`.
- `NOBDW`: When `NOBDW` is specified for a send file transfer, the file is opened using `RECFM=U`. Each block is written as an individual record. This parameter is the same as `YES`, except that the `BDW` is not sent.

Note: This parameter must be used only when `TYPE=BINARY` is specified.

SPACE=({CYL | TRK | KIL | MEG},({primary},{secondary},{directory}),{RLSE})

Default	None
Allowable Values	See the following explanation of this parameter.
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the values used when creating a new file on a z/OS system. The values can also be specified in the ALLOC_TYPE, ALLOC_PRI, ALLOC_SEC, and ALLOC_DIR parameters. If both parameters are specified, the last parameter specified takes precedence.

The SPACE parameter format is similar to (although not as flexible as) the z/OS JCL SPACE parameter. The values specified as *PRI*, *SEC*, and *DIR* are the primary, secondary and directory block allocation. If RLSE is specified, space will be released after the transfer is completed. If RLSE is not specified, space will not be released after the transfer is completed.

If the SPACE parameter is not defined, the allocation parameters associated with the input file is used for the output file. CYL indicates that allocation is done in Cylinders. TRK indicates that allocation is done in tracks. KIL indicates that allocation is done in Kilobytes (thousand bytes), while MEG indicates that allocation is done in Megabytes (million bytes).

SPACERELEASE={YES | NO}

Default	YES
Allowable Values	YES, NO
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether unused space is released when the Platform Server creates a file. The default value of YES indicates releasing unused space, NO indicates not releasing unused space. This parameter can also be specified by the SPACE parameter. If you specify the SPACE parameter, the default value is the same as specifying SPACERELEASE=NO. Specifying RLSE on the SPACE parameters is the same as specifying SPACERELEASE=YES.

STORCLASS={SMS_storage_class}

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	SC, SCLASS

This parameter defines the z/OS Storage Class as defined to the Data Facility/System Managed Storage, which is used to indicate what type of media the host file is kept on, and what the backup, restore, and archive policies of the installation are. The value is 1 - 8 character long and can contain numeric, alphabetic, or national characters ($, #, or @). The first character must be alphabetic or national character.

SUBDIR={YES | NO | ALL | SEQAVSAM}

Default	NO
Allowable Values	YES , NO, ALL, SEQAVSAM
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether files within subdirectories are processed. This parameter is used only when directory transfers are performed, and is ignored for non-directory transfer requests.

When a directory structure is defined, this parameter refers to the subdirectories within the directory structure. When z/OS datasets are defined, this parameter defines whether VSAM files or members of a PDS are transferred. For more information of this parameter, see Appendix M. Directory Transfers.

The valid values are as follows:

- `YES`: Sequential and PDS files are selected to be transferred.
- `NO`: Subdirectories are not selected to be transferred.
- `ALL`: Sequential files, PDS files, and VSAM files are selected to be transferred.
- `SEQAVSAM`: Sequential files and VSAM files are selected to be transferred.

SUBMIT= {"jcl.ibrary(member),symbolic parameters" | "member,symbolic parameters"}

By using this batch SYSIN parameter, you can submit a job on the remote system. This parameter is different from `TRANS_TYPE=JOB` in that the JCL to run actually sits on the remote system. This parameter is supported only when communicating with TIBCO MFT Platform Server z/OS. If you supply symbolic parameters with the job to be submitted, the entire string must be enclosed in double quotation marks.

> **Note:** You must set the `TRANS_TYPE` parameter to `COMMAND` if this parameter is specified; otherwise, an error will occur and the transaction will not be queued.

You can submit this parameter in the following two ways:

- By specifying the entire dataset name in the `SUBMIT` parameter.
- By specifying only the member name in the `SUBMIT` parameter. If this option is selected, you must define the GLOBAL **JOB_SUBMIT_DSN** parameter to define the name of the dataset that contains the member to be submitted.

Symbolic parameters can be supplied in the `SUBMIT` parameter along with the member or dataset name. Symbolic parameters must start with an ampersand (&). When the remote system is reading the JCL to be submitted, it scans the JCL for a match on the symbolic parameters defined. If a match is found, the data assigned to the symbolic parameter is substituted for the symbolic parameter in the JCL to be submitted. The symbolic parameters in the JCL must be delimited by one of the following characters for substitution to complete successfully: space, comma, equal sign (=), parenthesis plus comma plus parenthesis (,) or period. If a period is found at the end of a parameter, the period is removed before the parameter substitution takes place.

For example, SUBMIT="TESTJCL,&FILE=TEST.FILE,&RFILE=PROD.FILE".

For example, SUBMIT="JCL.FILE(TESTJCL),&FILE=TEST.FILE,&RFILE=PROD.FILE"

Based on the previous examples, if member TESTJCL contains the following statements:

```
//FUSJOB  JOB
555,'TEST',MSGCLASS=X,REGION=5M,CLASS=A
//BR14     EXEC PGM=IEFBR14
//DD1  DD  DSN=&FILE,DISP=SHR
//DD2  DD  DSN=&RFILE.,DISP=SHR
```

After data substitution, the data submitted into the internal reader is as follows:

```
//FUSJOB  JOB
555,'TEST',MSGCLASS=X,REGION=5M,CLASS=A
//BR14     EXEC PGM=IEFBR14
//DD1  DD  DSN=TEST.FILE,DISP=SHR
//DD2  DD  DSN=PROD.FILE,DISP=SHR
```

TAPE={YES | NO}

This parameter is only used when sending uncataloged tape files. It notifies the Platform Server that the request is for a tape. When this parameter is specified, you must also specify the IUNIT and IVOLUME parameters. This parameter is ignored for all receive transfers or for send transfers where the input dataset is cataloged or on a disk.

The default value is NO.

TCP_TIMEOUT, TCPTIMEOUT

This parameter defines the TCP timeout, in minutes, for this transfer.

The valid values are from 0 to 1440.

Note: This parameter overrides the TCP_TIMEOUT defined in the GLOBAL configuration.

ℹ **Note:** Platform Server negotiates this value with its transfer partner. If the `TCP Timeout` value of the transfer partner is higher than the value specified by this parameter or the GLOBAL `TCP_TIMEOUT`, then the partner value is used.

TEMPLATE

Default	Not Applicable
Allowable Values	Not Applicable
Minimum	1 byte
Maximum	30 bytes

This parameter defines the file that is used by the receiving partner as a template for its Access Control List. The ACL of this file is copied to the ACL of the destination file. The file specified must be readable by the partner that is receiving the File to File transfer. This field is Windows-specific and is only valid when sending files down to a Windows machine.

TEXTEOFTERM={YES | NO}

This parameter defines whether the Platform Server adds a LF or CRLF delimiter to the last record of a Text transfer to UNIX or Windows computers. When you specify `NO`, the Platform Server does not add LF or CRLF to the last record. When you specify `YES`, the Platform Server appends LF or CRLF to the last record sent to Windows or UNIX for a text transfer. If this parameter is not specified, the GLOBAL `TEXTEOFTERM` parameter is used. If both parameters are missing, the Platform Server does not add LF or CRLF to the last record of a Text transfer to UNIX or Windows.

TLS={YES | NO | TUNNEL}

Default	`NO`
Allowable Values	`YES`, `NO`, `TUNNEL`

Minimum	Not Applicable
Maximum	Not Applicable. Alternate Specification SSL

This parameter defines whether SSL/TLS should be used for a request. This definition overrides the GLOBAL or NODE definitions. The valid values are:

- `YES`: MFT Platform Server attempts to use SSL to authenticate both ends of the connection.
- `NO`: MFT Platform Server does not use SSL authentication.
- `TUNNEL`: All transfer data is sent over an encrypted TLS tunnel connection.

TIME={hhmmss}

Default	`000000`
Allowable Values	Not Applicable
Minimum	`000000`
Maximum	`235959`

This parameter defines the time of day after which the file transfer is eligible to commence. It defaults to midnight on the eligible date (this means that the transfer is eligible immediately). It is expressed in military time (`000000` - `235959`). It is a good practice to let this parameter default to `000000`.

TIMEOUT={number_of_seconds}

Default	`0` (no timeout)
Allowable Values	`0` - `32767`

Minimum	`0`
Maximum	`32767`

This parameter defines the number of seconds that a request waits for completion before the request times out. If this parameter is not specified, the batch job waits until the request is completed. If this parameter is specified, the batch job waits for this number of seconds before reporting the request as timed out. The `PURGE` parameter defines the action to be taken when a request times out.

> **Note:** This parameter is used only when `WAIT =YES` is also specified.

TRANS_TYPE={FILE | JOB | PRINT | COMMAND}

This parameter defines the type of transfer being sent or received.

Default	`FILE` (no timeout)
Allowable Values	`FILE`, `JOB` , `PRINT` , `COMMAND`
Minimum	Not Applicable
Maximum	Not Applicable

The valid values are as follows:

- `FILE`: stores the contents of the file transfer in a file (default).
- `JOB`: sends the output of the file transfer directly to the system to be executed as a job. The data is sent to the target system as is; no symbol substitution is performed.

  > **Note:** When you send a file to the Windows machine to be executed as a job, the job is executed in the `\Windows\SYSTEM32\` directory. Consider this when writing your batch jobs if you want to change the directory in which the batch job is executed.

- `PRINT`: sends the file that is being transferred directly to the print queue or spool on the remote side.

- COMMAND: indicates that the transfer is a command to be executed on the remote system. When you specify TRANS_TYPE=COMMAND, you must also specify one of the following parameters: REXXEXEC, CALLJCL, CALLPROG, or SUBMIT. The APPLICATION parameter must be FUSION or it must be completely left out when the REXXEXEC, CALLJCL, CALLPROG, or SUBMIT parameter is used.

  Note: This option is not supported by DNI.

TRANSFER={SEND | RECEIVE | SUBMIT | SCRIPT}

This parameter defines the direction of the file transfer. Typically, this parameter is defined on the PROCESS statement. However, this parameter can be used on the JCL EXEC PARM statement to override the value defined on the PROCESS statement.

The valid values are as follows:

- SEND: indicates sending a file from the local system to the remote system.
- RECEIVE: indicates receiving a file from the remote system to the local system.
- SUBMIT: indicates locally submitting a file transfer to the internal reader. The job is submitted on the CPU where the Platform Server started task is running. The submitted job runs under the authorization of the user that is running the Batch interface program.
- SCRIPT: indicates scheduling a script to run in the Platform Server started task.

  Scripts are run single threaded by default. By defining the CONTENTION_WINNERS parameter to a value higher than 1 and then defining the NODE parameter in the script, simultaneous transfers can take place. The NODE parameter is ignored other than to limit the number of concurrent requests for a script.

  For example,

```
 START FTMSCMD /Send /RI=CFUSERID /RW=CFPASSWD
/DS:ABC /COMMAND /RE="FUSSCRPT
LF=CF.PROCESS.LIB
(TESTABC)~%%RING=TT01~%%INDS=ABCDE1.ABCTEST.ABCD10.ABC10.ZIP~%%SVR=
ABC20
NODE=xxxxx"
```

TRCLASSNAME={transfer_class_name}

Default	Not Applicable
Allowable Values	Transfer classes with 1 - 12 characters defined in TRCLASS DD statement
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the transfer class name. Transfer classes are used to limit the number of concurrent transfers. Transfer classes are defined in the TRCLASS DD statement. Transfer classes defined by this parameter must match a defined transfer class. If the transfer class is not defined, the default transfer class that supports unlimited concurrent transfers is used. Transfer class names are not case sensitive and is always displayed in uppercase.

TRUNCATE ={YES | NO | WRAP}

Default	`NO`
Allowable Values	`YES`, `NO`, `WRAP`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter is only used when receiving a file from Windows, UNIX, IBM i, or MFT Internet Server when record delimiters are defined. This parameter is ignored for a SEND or when communicating to another z/OS system.

The valid values are:

- `NO`: If the record received from the partner is greater than the LRECL, a truncate error occurs and the transfer is terminated with an error.
- `YES`: If the record received from the partner is greater than the LRECL, the record is truncated and any data after the LRECL is lost.
- `WRAP`: If the record received from the partner is greater than the LRECL, the record is

truncated and any data after the LRECL is written to the next record. If the record received has no delimiters within the first 32760 bytes, a truncate error occurs and the transfer is terminated with an error. In cases like this, you should turn off delimiter checking, so records will be written without any delimiter checking.

TYPE={TEXT | BINARY | DATA|ASCII}

This parameter defines the data type which indicates whether data conversion is performed for the transfer. The default value is `BINARY`.

The valid values are as follows:

- `TEXT`: EBCDIC to ASCII data conversion is performed.
- `BINARY`: No data conversion is performed.
- `DATA`: Used only with the `KNFUSION` protocol. This is the equivalent of `TEXT` with the `FUSION` protocol.
- `ASCII`: ASCII is the same as TEXT. EBCDIC to ASCII data conversion is performed.

UNIXPERM

Defines the permissions that will be set when creating a file on a target UNIX system.

Default: None

Allowable values: 3 numeric characters. Each character should be between 0 and 7.

UTF8BOM={ADD | REMOVE | BOTH |NONE}

Default	`NONE`
Allowable Values	`ADD`, `REMOVE`, `BOTH`, `NONE`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether the UTF-8 BOM (0xefbbbf) is added at the start of the converted data or is removed from the start of the data before conversion. This parameter is used only when the `LCT` parameter defines that Unicode services are used to convert data. This parameter can be used for a z/OS Initiated Send or Receive transfers, although it is more commonly used when sending a file to a target system that requires a BOM.

The valid values are:

- `ADD`: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. This value is typically used when sending data to a target system (like Linux) that requires the UTF-8 BOM at the start of the file.
- `REMOVE`: The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data. If the start of the file does not include the BOM, no changes are made. This value is typically used when receiving data from a target system (like Linux) that contains the UTF-8 BOM at the start of the file.
- `BOTH`: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data.
- `NONE`: No UTF-8 BOM processing is performed. The UTF-8 BOM is NOT inserted at the start of the converted data or removed before conversion.

VOLUME={volume1_name,volume2_name,….,volume5_name}

Default	Not Applicable

Allowable Values	Up to 5 volumes
Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	`VOL`, `VOLSER`

This parameter defines the volumes where the dataset can be written. This parameter is only used if the file is being written to a z/OS system. Up to 5 volumes can be defined.

> **Note:** You can specify more than one volume only for disk datasets that are being created.

VSAM_KEYPOS={VSAM_relative_key_position}

Default	Relative key position of source file
Allowable Values	`0 - 32760`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the position of the key relative to zero. This is only used for VSAM KSDS files, and is ignored for other VSAM file types. This field is used only when creating a VSAM file. If this field is not defined, the key position of the source file is used.

> **Note:** This field is a displacement. Therefore, for a key in the first byte of a record, the `VSAM_KEYPOS` parameter must be defined as `0`.

VSAM_KEYLEN={VSAM_key_length}

Default	Key length of source file
Allowable Values	`1 - 255`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the key length. This is only used for VSAM KSDS files, and is ignored for other VSAM file types. This field is used only when creating a VSAM file. If this field is not defined, the key length of the source file is used.

VSAM_LIKE={VSAM_model_cluster_data_set_name}

Default	None
Allowable Values	44-byte character dataset name
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the file that is used as a model for creating the VSAM file. This field is used only when creating a VSAM file. If this field is not defined, the attributes of the sending file are used.

VSAM_LRECL={VSAM_record_length}

Default	Source file record length
Allowable Values	`0 - 32760`
Minimum	Not Applicable

Maximum	Not Applicable

This parameter defines the maximum record length of a VSAM file. This field is used only when creating a VSAM file. If this parameter is not defined, the source file record length is used to create the file.

VSAM_TYPE={VSAM_file_type}

Default	None
Allowable Values	`ESDS` , `RRDS`, `KSDS`, `LDS`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the type of VSAM file to be created. This field is used only when creating a VSAM file. If this parameter is not defined, the source file type is used when creating the file.

VSAM_REUSE={YES | NO}

Default	`NO`
Allowable Values	`YES`, `NO`
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether the file is opened with the REUSE option. The REUSE option overrides the file when it is used. It can be used under the following two circumstances:

- When the file is defined with the `IDCAMS REUSE` parameter.
- When the file is being loaded for the first time.

If the VSAM_REUSE parameter is specified under any other circumstances, VSAM returns an OPEN error, and the transfer is terminated.

VSAM_RRSLOT={YES | NO}

Default	NO
Allowable Values	YES, NO
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines whether a RRDS to RRDS file transfer maintains the RRDS slot numbers. This parameter is used only when sending or receiving a RRDS file. When this parameter is defined as YES, the Platform Server inserts the RRDS slot number in the first 4 bytes of the record read when sending a RRDS file; the Platform Server is informed that the RRDS slot number is in the first 4 bytes of the record to be written when receiving a record.

WAIT={YES | NO | ASYNC | ASYNCWAIT}

Default	NO
Allowable Values	YES, NO, ASYNC, ASYNCWAIT
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the job to wait for the file transfer to finish before proceeding. If you specify WAIT=YES in the parameters of a batch job, the batch job does not continue processing until the transfer is completed. If the transfer is not completed successfully, the step ends and a code is returned indicating the reason for the failure. Do not specify WAIT=YES when distribution list transfers are performed (that is, when you are using a distribution list).

The valid values are as follows:

- YES: the transfer request waits until the transfer is completed.
- NO: the transfer request is completed when the request is scheduled.
- ASYNC: used for directory transfers. The job waits until all transfers are scheduled. However, the job does not wait for transfers to be completed.
- ASYNCWAIT: used for directory transfers only. Transfers are submitted asynchronously and the job is not completed until all transfers are completed.

  > **Note:** A maximum of 500 transfers can be submitted asynchronously. If more than 500 transfers are submitted as a result of the directory transfer, transfers over 500 are submitted with WAIT=YES.

> **Note:** Pay attention to the following points, when using ASYNC or ASYNCWAIT on a directory receive transfer:
>
> - You cannot receive multiple files to a PDS. The PDS can become corrupted if multiple transfers are written to the PDS at the same time. You must use WAIT=YES when writing to a PDS.
> - When receiving multiple files to a LIBRARY, the library must exist. It cannot be created by the transfer.

See the following example:

```
//BATCHJOB JOB 555,'USER NAME',MSGCLASS=X,NOTIFY=USERID,REGION=4M

//*

//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=CFSERVER'

//STEPLIB DD DISP=SHR,DSN=FUSION.LOADLIB

//SYSUDUMP DD SYSOUT=*

//SYSPRINT DD SYSOUT=*

//SYSIN  DD *

**************************************************************
```

```
* Send and wait till it is done               *

**************************************************************

  PROCESS,REPORT,TRANSFER,SEND

    DSN=USER1.SALES.REPORT

    REMOTE_FILE=C:\FILES\RPT1

    TYPE=BINARY

    EFFECT=CR

    NODE=WINSRV1

    WAIT=YES

    REMOTE USER=Administrator

    RPASS=password

    RETRY=1

**************************************************************

  PROCESS,BATCH,TRANSFER,SEND

    DSN=USER1.JCL(PCBATCH)
    REMOTE_FILE=C:\FILES\JCL1

    TYPE=BINARY

    EFFECT=CR

    NODE=WINSVR1

    REMOTE USER=Administrator

    RPASS=password

    RETRY=1
```

In this example, a single step includes two transfers. If the first transfer fails, this step terminates, and all transfers following it are not executed. When the same process statements are executed in the scripting program (OSIUC000), the step does not terminate after a failed transfer.

Use this method when the second transfer is dependent on the outcome of the first transfer. A transfer that is not dependent upon the outcome of a previous transfer, accomplished by specifying the `WAIT` parameter, cannot be included in the same step (as a transfer that includes the parameter). This ensures that the job is executed regardless of the outcome of the previous transfer.

For how this parameter can be used with the z/OS IF/THEN/ELSE/ENDIF JCL parameters to initiate additional transfers at the end of a transfer based upon the success or unsuccessful completion of the previous step, see Appendix F. WAIT Parameter for IF/THEN/ELSE/ENDIF.

XMEMBER={members_to_exclude}

Default	None
Allowable Values	Up to 25 8-character member names
Minimum	Not Applicable
Maximum	25 member names

This parameter defines the member names to be excluded from a PDS file transfer. You can enter this parameter on a single file transfer request to define up to 25 member names. This parameter can only be used for z/OS to z/OS transfers. This parameter is ignored when the dataset names on both ends of the file transfer are not PDS or PDSE files.

Note: This parameter is mutually exclusive with the `MEMBER` parameter.

Print Support/Remote Job Submission Parameters

The Print Support/Remote Job Submission parameters are used for defining the print task with detailed instructions.

The following table lists supported Print Support and Remote Job Submission parameters:

Parameters	Alternate Specification	Short Description
`COPIES`		The number of copies to print
`DESTINATION`	`DEST`	The report destination
`FCB`		The report form control buffer
`FORM`		The report form name
`PRINTER_NAME`		The report printer name
`SYSOUT`		The report SYSOUT class
`USERNAME`		The report user name
`WRITER`		The report external writer name

`COPIES`={*nnn* | 1}

Default	`1`
Allowable Values	`1` - `999`
Minimum	`1`
Maximum	`999`

This parameter defines the number of copies to print of a particular report on the remote computer.

`DESTINATION`={*report_destination*}

Default	None
Allowable Values	1 - 8 characters

Minimum	Not Applicable
Maximum	Not Applicable
Alternate Specification	DEST

This parameter defines the destination of the job submitted to the internal reader.

FCB**={*form_control_buffer*}**

Default	None
Allowable Values	1 - 4 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the form control buffer name as defined to JES. This field is applied when the remote computer is a z/OS system.

FORM

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the form name upon which the report is printed on the remote computer.

PRINTER_NAME

Default	None
Allowable Values	1 - 255 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the name of the remote printer to which the job is sent.

SYSOUT

Default	None
Allowable Values	A character of A - Z or 0 - 9
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the printer class to be assigned on the remote computer when data is written into the remote printer queue.

USERNAME

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the user name assigned to a job submitted to the internal reader.

WRITER

Default	None
Allowable Values	1 - 8 characters
Minimum	Not Applicable
Maximum	Not Applicable

This parameter defines the external writer name that is used to process this printer file on the mainframe. This is the name of a service program on the mainframe, which is given control when it is time to process this file from the printer queue. The service program, which is written by the customer, decides how it wants to process this print file.

Note: Do not specify a value for this parameter unless directed by the systems analyst on the mainframe.

Sample JCL Statements

The sample JCL statements show how to queue file transfers to TIBCO MFT Platform Server for z/OS.

Example 1

The following example shows how to send and create a file on the remote server:

```
//SEND1 JOB 555,'USER NAME',MSGCLASS=X,REGION=4M
//**************************************************************
//*       THIS SENDS A FILE TO A REMOTE COMPUTER
//**************************************************************
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB  DD  DISP=SHR,DSN=FUSION.LOADLIB
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
**************************************************************
   PROCESS,SEND1,TRANSFER,SEND
        LF=HLQ.FILE.NAME
        RF=C:\FILES\SEND1.BIN
        AVAIL=IMMED
        CRLF=Y
        TEXT=YES
        EFFECT=CR
        HOLD=NO
        NODE=REMNODE
        NOTIFY=TSOUSER
        NOTIFY_TYPE=TSO
        REMOTE_USER=RMTUSER
        RPASS=PASSWORD
        RETRY=1
        TYPE=TEXT
```

Example 2

The following example shows how to receive, create, or append a file on the remote system:

```
//RECEIVE1 JOB 555,'USER NAME',MSGCLASS=X,REGION=4M
//********************************************************************
//*       THIS RECEIVES A FILE FROM THE REMOTE COMPUTER
//********************************************************************
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB  DD  DISP=SHR,DSN=FUSION.LOADLIB
//SYSUDUMP DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
**************************************************************
   PROCESS,RECEIVE1,TRANSFER,RECEIVE
        DSN=HLQ.FILE.NAME
        REMOTE_FILE=C:\FILES\RECEIVE1.BIN
        AVAIL=IMMED
        CRLF=Y
        EFFECT=CA
        HOLD=NO
        NODE=REMNODE
        NOTIFY=TSOUSER
        NOTIFY_TYPE=TSO
        REMOTE_USER=RMTUSER
        RPASS=PASSWORD
        RETRY=1
        TYPE=TEXT
```

Return Codes

Return codes are sent back to the initiator to report whether the transfer request is successful or not.

The following return codes are supported by the job step:

Return Code	Description
0	Successful: Request was scheduled successfully (WAIT=NO). Successful: Request was completed successfully (WAIT=YES).
<> 0	Unsuccessful

The following table lists the return codes that are set in message PGTB4142I when the Batch interface transfers are completed:

Return Code	Description
0	Successful: Request was scheduled successfully (WAIT=NO). Successful: Request was completed successfully (WAIT=YES).
4	Unsuccessful: Parameter error.
8	Unsuccessful: Load for module failed.
11	Unsuccessful: Dataset for file SEND was not cataloged.
12	Unsuccessful: Abend occurred.
14	Unsuccessful: Inconsistent EFFECT parameter.
20	Unsuccessful: Server was not available.
24	Unsuccessful: Security access was denied for local dataset.
21	Unsuccessful: Request timed out with no purge.

Return Code	Description
22	Unsuccessful: Request timed out with purge.
128	Unsuccessful: Network error occurred when scheduling request.
252	Unsuccessful: NODE definition was required by GLOBAL REQUIRE_NODE_DEFINITION.
253	Unsuccessful: User profile was not found.
254	Unsuccessful: Profile dataspace was not defined.
255	Unsuccessful: List was not found.

REXX Interface

TIBCO MFT Platform Server for z/OS provides the REXX interface from which you can use REXX to queue transactions, inquire on transfers, and perform operator control functions to the Platform Server. All the features of the REXX language are supported.

Executing REXX Requests

You can use the REXX execs to execute transfer requests on TIBCO MFT Platform Server for z/OS in five ways.

You can use the following five ways to execute REXX execs:

- Using the ISPF Command Shell
- Using the Batch TMP: IKJEFT01
- Using the REXX Batch Processor (IRXJCL)
- Using Platform Server Address Space
- Using the REXX_EXEC Parameter

Using the ISPF Command Shell

You can execute the Platform Server REXX execs directly on an ISPF command line of the ISPF Command Shell panel.

To run the REXX execs, enter the command on a command line on the ISPF Command Shell panel and press Enter.

Note: To open the ISPF Command Shell panel, select option 6 on the ISPF Primary Option menu.

Examples

The following two examples show how to run the REXX execs on an ISPF command line:

Example 1:

```
FUSSEL TRN=IA19500038
```

This command displays the last message of the specified transaction ID.

Example 2:

```
FUSSEND IPADDR=192.192.192.100 IPPORT=46464 LF=local.file RF=remote.file
INTERVAL=3 SAY CFG=fusion wait
```

This command sends a local file (`LF`) to a remote file (`RF`) using TCP/IP.

Advantage and Disadvantages

The advantage and disadvantages of using the ISPF Command Shell are as follows:

- Advantage
    - You can run the REXX execs by simply entering the command on a command line.
- Disadvantages
    - You cannot use this method in an unattended production environment.
    - You cannot fully use the functionality of the REXX language, because only ISPF commands can be executed from the command line.

      If you must execute a REXX exec with TIBCO MFT Platform Server for z/OS REXX commands imbedded, you must create a member in the Platform Server EXECS library that contains the REXX exec, and then execute that exec from the command line.

Using the Batch TMP (IKJEFT01)

You can execute the Platform Server REXX execs by submitting JCL that executes the batch terminal monitor program (TMP), IKJEFT01.

The batch TMP (IKJEFT01) can be used to execute TSO and REXX commands. The SYSTSIN input file defines the REXX execs to be executed. The Platform Server REXX execs can be executed directly from within the SYSTSIN file or called by another REXX exec. However, only commands can be executed directly from the SYSTSIN file; no REXX keywords can be used within this environment. This means that you can execute TSO and Platform Server commands directly within the SYSTSIN file, but if you want to use REXX keywords such as

IF or DO, you must add the exec to a member of the EXECS, and execute that exec from the IKJEFT01 SYSTSIN.

Examples

The following examples show how to run REXX execs from the batch TMP.

Example 1:

```
//jobcard  JOB ,'  ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IKJEFT01
//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
***fusion requests or TSO commands***

//
```

This example shows a sample JCL needed for processing the batch TMP.

Example 2:

```
//jobcard  JOB ,'  ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IKJEFT01
//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD *
FUSSEND IPADDR=192.192.192.100 IPPORT=46464 LF=local.file +
RF=remote.file INTERVAL=3
        SAY CFG=fusion wait
//
```

This example shows a sample JCL that can be used to execute a Platform Server REXX exec.

Example 3:

```
//jobcard  JOB ,'  ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IKJEFT01
//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
```

```
//SYSTSIN   DD *
SENDPROD
//
```

This example shows a sample JCL that can be used to execute a Platform Server REXX exec that is a member of the SYSEXEC file. Member SENDPROD of SYSEXEC then contains the following REXX exec:

```
/*     REXX      */
FUSSEND 'IPADDR=192.192.192.100 IPPORT=46464
LF=local.file’,‘RF=remote.file INTERVAL=3
         SAY CFG=fusion wait’
IF RC = 0 THEN DO
   SAY   '0 RETURN CODE FROM FUSSEND'

   EXIT 0

 END
ELSE   DO
   SAY   'RETURN CODE FROM FUSSEND=' RC
   EXIT 8

 END
```

Note: The parameters of the `FUSSEND` command are enclosed in single quotation marks, which is required when a REXX exec contains a Platform Server command. This is a restriction of the TSO and REXX environments.

Advantages and Disadvantage

The following are advantages and disadvantage of using the batch TMP:

- Advantages
    - You can use this method in an unattended production environment.
    - Return code processing can be used to check the status of the request.
    - Multiple Platform Server requests can be executed from the SYSTSIN file.
    - You can execute all TSO commands such as `ALLOC` and `FREE` in this environment.

- Disadvantage
    - It is less efficient than the REXX batch processor, IRXJCL.

Using the REXX Batch Processor (IRXJCL)

You can execute the Platform Server REXX execs by submitting JCL that executes the REXX batch processor, IRXJCL.

The REXX batch processor (IRXJCL) can be used to execute REXX commands. The input to the IRXJCL program are embedded in the PARM statement of the EXEC JCL card. The Platform Server execs can be entered directly on the PARM JCL statement or called by another exec. However, only REXX execs can be executed directly from the PARM statement; no REXX keywords can be used within this environment. This means that you can execute the Platform Server commands directly within the PARM statement, but if you want to use REXX keywords such as `IF` and `DO`, you must add the exec to a member of the EXECS, and execute that exec from the IRXJCL PARM.

Examples

The following examples show how to run REXX execs from the REXX batch processor:

Example 1:

```
//jobcard  JOB ,'  ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IRXJCL,
//  PARM=’exec to be executed’
//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD DUMMY
//
```

This example shows a sample JCL needed for processing the REXX batch processor.

Example 2:

```
//jobcard  JOB ,'  ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IRXJCL,
//  PARM=’FUSSEND IPADDR=192.192.192.100 IPPORT=46464 LF=local.file
//             RF=remote.file INTERVAL=3 SAY CFG=fusion wait’
//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
```

```
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD DUMMY
//
```

This example shows a sample JCL of using the REXX batch processor to execute a Platform Server REXX exec.

Note: The PARM statement cannot exceed 100 characters. This is a restriction of the JCL processor. If the PARM statement cannot be contained on a single line, you must perform the following actions:

- Enclose the PARM statement in single quotation marks.
- Code the characters up to column 71.
- Continue the PARM statement on the next JCL card starting in column 16.

Example 3:

```
//jobcard  JOB ,'   ',MSGCLASS=X,CLASS=A
//S1   EXEC PGM=IRXJCL,PARM=’SENDPROD’

//STEPLIB   DD DSN=FUSION.LOADLIB,DISP=SHR
//SYSEXEC   DD DSN=FUSION.EXECS,DISP=SHR
//SYSTSPRT  DD SYSOUT=*
//SYSTSIN   DD DUMMY
//
```

This example shows a sample JCL of executing a REXX exec that is a member of the `EXECS` file. Member SENDPROD of the Platform Server EXECS library then contains the following REXX exec:

```
/*     REXX      */
FUSSEND 'IPADDR=192.192.192.100 IPPORT=46464 LF=local.file’,
‘RF=remote.file INTERVAL=3
         SAY CFG=fusion wait’
IF RC = 0 THEN DO
   SAY   '0 RETURN CODE FROM FUSSEND'

   EXIT 0
```

```
 END
ELSE   DO
   SAY   'RETURN CODE FROM FUSSEND=' RC
   EXIT 8

 END
```

Note: The parameters of the `FUSSEND` command are enclosed in single quotation marks, which are required when a REXX exec includes a command. This is a restriction of the TSO and REXX environments.

Advantages and Disadvantages

The following are advantages and disadvantages of using the REXX batch processor:

- Advantages
    - You can use this method in an unattended production environment.
    - Return code processing can be used to check the status of the request.
    - You can perform a process override to change the PARM statement.
- Disadvantages
    - Only one command can be entered on the PARM statement.
    - The PARM statement restriction of 100 characters might limit the command requested.
    - You cannot execute certain TSO commands such as `ALLOC` and `FREE` in this environment.

Using Platform Server Address Space

You can schedule the execution of REXX execs within the Platform Server address space.

You can use one of the following two ways to run REXX execs in the Platform Server address space:

- Use the FUSQREXX exec to queue a REXX exec.
- Use the Batch interface to queue a REXX exec.

When running REXX execs within the Platform Server address space, conform to the following rules:

- `CFG=*` or `CONFIG=*` must be defined to configure the Platform Server for processing all requests in the local Platform Server.
- The REXX requests are single-threaded. This means that you must not execute any command that might cause the REXX exec to be put into a wait state for more than a few seconds. You cannot use any of the following requests:
  - WAIT
  - MWAIT
- Because REXX execs are not running in the TSO environment, certain REXX commands such as `ALLOC` and `FREE` do not work. For more information on the commands that do not work in an z/OS environment, see z/OS TSO/E REXX Reference manual.

Typically, the types of execs that can be run within the Platform Server address space are requests to queue a transfer, queue a local job submission, or inquire on the status of a transfer. Other REXX commands such as `IF`, `DO`, and `EXECIO` are also supported. If you want to access a file, the file must be allocated within a JCL statement of the Platform Server started task. For example, the SYSEXEC DD statement that points to the EXECS library must be added to a JCL statement of the Platform Server started task.

Using the FUSQREXX Exec to Queue a REXX Exec

You can queue a REXX exec to be executed within the Platform Server address space by using the FUSQREXX exec.

> **Note:** When the command field contains imbedded spaces, all FUSQREXX parameters must be delimited by a comma.

Example

The following example shows a sample FUSQREXX exec:

```
FUSQREXX COMMAND=FUSINQ TRN=IA038* CFG=*,CFG=FUSION_STC
```

This command causes the following two parameters to be passed to the FUSQREXX exec:

- `CFG=*`: Indicates that when the FUSINQ exec is executed, the `config` file points to the started task under which the exec is running.
- `CFG=FUSION_STC`: Informs the Platform Server REXX to queue the transfer to the server defined by the FUSCFG member in the Platform Server EXECS library.

This command queues a REXX exec to the server defined by the FUSCFG member in the Platform Server EXECS library.

The command to be executed is as follows:

```
FUSINQ TRN=IA03812456 REPORT=LONG CFG=*
```

Using the Batch Interface to Queue a REXX Exec

You can queue a REXX exec to be executed within the address space by using the Batch interface.

The following example shows the input required to queue a REXX exec to run in the Platform Server address space:

```
PROCESS,REXXPROC,TRANSFER,REXX
COMMAND=”FUSINQ TRN=IA038* CFG=*”
DATE=2008303
```

This example queues the following command to the Platform Server started task. The command is eligible for execution on Julian date 2008303.

```
FUSINQ TRN=IA038* CFG=*
```

As a result, this example input inquires all transactions starting with `IA038`. `CFG=*` indicates that the REXX exec is running in the same address space as the started task.

Return Code Processing When REXX Execs Are Queued

The Platform Server bases the request return code on two values returned by the REXX command processor.

A return code indicates whether the exec is processed without errors. Sample errors could be syntax errors that are not intercepted by the SIGNAL, or execs that are not defined in the SYSEXEC file. To determine the return code, consider the following two situations:

- If the Platform Server receives a non-zero return code from the REXX command processor, that is the return code that the Platform Server saves.
- If the Platform Server receives a zero return code from the REXX command processor, it scans the data returned by the REXX processor on the EXEC or RETURN statement. Up to 64 bytes of data can be returned and displayed on the REXX exec completion message. The Platform Server checks the first 5 bytes of that message for a return code. The 5 bytes are evaluated in the following manners:
    - If the 5 bytes of data are numeric, the Platform Server converts that data to a return code value.
    - If the Platform Server finds one or more numeric characters followed by a space or a null (0x00), the Platform Server converts that data into a return code.
    - If no numeric data is found (followed by a null or space), the Platform Server assigns a return code value of 0.

> **Note:** The Platform Server assigns a 1-byte value to contain the return code. Therefore, any return code value greater than or equal to 256 is computed by dividing 256 into the return code and assigning the remainder as the return code. Therefore, a return code of 256 becomes a 0 return code, while a return code of 257 becomes a return code of 1.

Using the REXX_EXEC Parameter

You can run REXX execs by specifying the `REXX_EXEC` parameter. In this case, the Platform Server executes the REXX exec when a file transfer is completed and purged from an active queue, and passes the transaction number and request status (either `COMPLETE` or `FAILED`) to the REXX exec.

Example

Member POSTPROC of the Platform Server EXECS library shows an example of executing a REXX exec when a Platform Server request is completed.

```
FUSSEND IPADDR=14.0.0.0 LF=local.file RF=remote.file REXX_EXEC=POSTPROC
```

When the request is completed, the Platform Server executes the following command:

```
POSTPROC IA23500010 COMPLETE
```

Where, `IA23500010` is the transaction number that is created when the `FUSSEND` exec is queued, and `COMPLETE` indicates that the request is completed successfully.

The POSTPROC member in the Platform Server EXECS library is as follows:

```
/*     REXX                                                      */
/* ------------------------------------------------------------- */
/*     EXEC which runs when a request completes and is purged    */
/*     from the MFT Platform Server request queue
*/
/* ------------------------------------------------------------- */
signal on syntax
/* ------------------------------------------------------------- */
/*         STEP 1                                                */
/*         get values passed to the Exec from FUSION             */
 PARSE VALUE '' WITH trn status other
 PARSE ARG           trn status other
/* ------------------------------------------------------------- */
/*         STEP 2                                                */
/*         Now get all Request fields and put them in the        */
/*             REXX external Queue                               */
 FUSSEL 'CFG=* TRN='trn ' QUEUE'
/*  FUSSEL 'CFG=* TRN='trn ' QUEUE'  */
 if RC <> 0 then do
say 'non zero return code from TBPOST FUSSEL request==>' RC
       exit 12
      end
/* ------------------------------------------------------------- */
/*         STEP 3                                                */
/*          now read the REXX external Queue and display the     */
/*              parameters                                       */
do while queued() > 0
   parse pull input
   say input
end
/* ------------------------------------------------------------- */
/*         STEP 4                                                */
/*   now check the transfer status and exit with return code   */
 if status = 'COMPLETE' then retncode = 0
     else retncode = 8
 exit retncode
/* ------------------------------------------------------------- */
syntax:
```

```
Say 'Error in line exec TBPOST  ' sigl ' ' errortext(rc)
Say ' Line : ' sourceline(sigl)
exit 1000
```

In this example, the `POSTPROC` exec defines the following four steps:

1. Parse the command line to get the `TRN` and `STATUS` command line parameters.
2. Issue the `FUSSEL` command to get all of the fields associated with the request.

   > **Note:** `CFG=*` indicates that the exec is running within the Platform Server address space. `QUEUE` defines the Platform Server to write all output to the external REXX queue.

3. Read the external REXX queue and display all the parameters. The output is sent to the SYSTSPRT DD statement of the Platform Server started task.
4. Exit from the exec with a return code, based on whether the request status is `COMPLETE` or `FAILED`.

The output from this `POSTPROC` exec is as follows:

```
retncode===> 0 PGTF3101I Activity IA22500019 successfully transferred
25603 records with node 190.190.190.100
trans.transnum = IA22500019
trans.sysname = 190.190.190.100
trans.local dsn = LOCAL.FILE
trans.rdsn = remote.file
trans.ruserid = FUSNUSER
trans.recfm = VB
trans.type = TRTYPE
trans.maxrecl = 04100
trans.effect = CREATE/REPLACE
trans.status = COMPLETE
trans.avail = IMMEDIATE
trans.time.eligible =
trans.date.eligible =
trans.time.started = 172157
trans.date.started = 2008295
trans.date.interrupt = 2008295
trans.time.interrupt = 172209
trans.pri =
trans.userdata =
trans.chkintvl = 000000
trans.chkpt.count = 000000
trans.process = FILEXFER
trans.expdt = 2008325
trans.retry.limit = 00001
trans.tries = 00001
trans.last.message = PGTF3101I Activity IA22500019 successfully
transferred 25603 records with node 190.190.190.100
trans.xfer.type = FILE
trans.record.count = 000000025603
trans.byte.count = 000002073701
trans.compressed.byte.count = 000000000000
trans.local.user = IBMUSER
trans.logon.domain =
trans.translation = BINARY
trans.crlf = NONE
trans.compress.type = NONE
trans.notify.type =
trans.notify.user =
trans.file.avail = IMMEDIATE
trans.compress.ratio =
trans.application.type = ALL
trans.status = COMPLETE
trans.ip.port = 46464
rexx.exec.... = POSTPROC
```

After the `POSTPROC` exec is completed, the Platform Server displays the following message on the z/OS console:

```
PGTE2221I Data: 0 returned by REXX exec: POSTPROC
```

In this case, `POSTPROC` exec is completed with a return code of zero. The returned data passed by the exec is displayed in the PGTE2221I message.

REXX Execs

TIBCO MFT Platform Server for z/OS REXX execs can be divided into four subsets.

The four subsets of REXX execs are as follows:

- Queue a Platform Server request to the address space. (This is also known as scheduling a request within the address space.)
    - FUSQREXX: Queues a REXX exec to be executed.

  - FUSQSUB: Queues a local job submission to be executed.
  - FUSRECV: Queues a receive request.
  - FUSRECVT: Queues a receive request for a text file.
  - FUSSCMD: Queues a send request for commands.
  - FUSSCRPT: Schedules a script to run in the Platform Server started task.
  - FUSSEND: Queues a send request.
  - FUSSENDT: Queues a send request for a text file.
  - FUSSMEM: Queues transfers to send multiple members to a remote node.
  - FUSSMFIL: Queues transfers to send multiple files to a remote node.
  - FUSWAIT: Waits on requests queued with the `MWAIT` operand.
  - FUSWCLR: Clears out the wait queue.
- Get summary information on the Platform Server requests based on a variety of selection criteria.
  - FUSINQ: Inquires on transfers based on selection criteria.
- Perform operator control or detailed inquiry based on the Platform Server transaction number.
  - FUSDEL: Deletes a Platform Server request.
  - FUSHOLD: Holds a Platform Server request.
  - FUSREL: Releases a Platform Server request that is in the HOLD state.
  - FUSSEL: Performs detailed inquiry on a Platform Server request.
  - FUSSUSP: Suspends an active Platform Server request.
- Perform Platform Server message and node inquiry.
  - FUSMSG: Inquires on a Platform Server message.
  - FUSNODE: Inquires on node entries of a Platform Server.

REXX Parameter Syntax Rules

You must follow the syntax rules when using the REXX parameters.

The syntax rules of the REXX parameters are as follows:

- The CONFIG and CFG parameters define the configuration files for all Platform Server REXX commands except the FUSWCLR command which does not require a configuration.
- The Platform Server REXX parameters are typically separated by a space or comma. When separated by a comma, no commas are allowed in the parameter values. Because the PPA parameter value requires commas, PPA is not allowed when using comma delimited parameters. When using space delimited parameters, parameter values can be enclosed in single or double quotation marks. If a parameter value contains any spaces, it must be enclosed in quotes. We strongly suggest using space delimited parameters and enclosing parameter values in single or double quotation marks where necessary.

  For example:

  ```
  FUSSEND IPADDR=14.0.0.0,LF=local.file,RF=remote.file,wait
  FUSSEND IPADDR=14.0.0.0 LF=local.file RF=remote.file wait
  FUSSEND IPADDR=14.0.0.0 LF=local.file RF=remote.file wait
  PPA="S,L,CALLJCL,TESTPGM parameters"
  FUSSEND IPADDR=14.0.0.0 LF=local.file RF=remote.file wait
  PPA='F,R,COMMAND,FailCMD parameters'
  ```

- Parameters can be entered in uppercase or lowercase. The Platform Server performs translation when necessary.
- If the REMOTE_USER or RUSER parameter is not defined, the Platform Server sets the remote user ID to *PROFILE by default. For more information of remote user profiles, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.
- To get help information, enter the Platform Server REXX command followed by a space and a question mark (?). For example, FUSINQ ?. For this example, a help screen is displayed with an overview of the exec and the parameters that are accepted by the exec.
- The Platform Server REXX execs that queue a request accept any parameter defined to the Platform Server Batch interface. For more detailed information of the parameters which are accepted by the Platform Server execs that queue a transfer, see Batch Interface.
- Scripts are run single-threaded by default. You can execute multiple transfers simultaneously by setting the CONTENTION_WINNERS parameter to a value higher than

1 and then setting the NODE parameter in the script. The NODE parameter is ignored other than to limit the number of concurrent requests for a script. For example:

```
START FTMSCMD /Send /RI=CFUSERID /RW=CFPASSWD /DS:ABC /COMMAND
/RE="FUSSCRPT
LF=CF.PROCESS.LIB
(TESTABC)~%%RING=TT01~%%INDS=ABCDE1.ABCTEST.ABCD10.ABC10.ZIP~%%SVR=
ABC20 NODE=xxxxx"
```

Queuing Requests to Platform Server Started Task

You can use REXX execs to queue Platform Server requests to the Platform Server started task.

The REXX execs used are as follows:

- FUSNODE: displays information on enabled NODE definitions.
- FUSQREXX: queues a REXX exec to be executed within the address space.
- FUSQSUB: queues a request to submit a job locally within the address space of the Platform Server.
- FUSRECV: queues a request to receive a file from a remote partner.
- FUSRECVT: queues a request to receive a text file from a remote partner.
- FUSSCMD: queues a request to send a command to a remote partner.
- FUSSCRPT: queues a request to schedule a script request to run by the platform server.
- FUSSEND: queues a request to send a file to a remote partner.
- FUSSENDT: queues a request to send a text file to a remote partner.
- FUSSMEM: sends multiple members of a PDS to a node of a remote platform server.
- FUSSMFIL: queues transfers to send multiple files to a remote node.

FUSNODE

You can use the FUSNODE exec to display information on enabled NODE definitions.

Required Parameters

This exec does not has required parameters.

Optional Parameters

The following table lists the optional parameters of the `FUSSCMD` exec:

Parameter	Description
`CONFIG` | `CFG`	Defines the configuration file used by the `FUSNODE` exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
`NODE`	Defines the NODE definition on which detailed information is to be displayed. If this parameter is not specified, a list of enabled NODE definitions is displayed.
`INIHOLD`	Modifies the defined NODE definition to turn on the Initiator Hold flag. This indicates that initiator requests to this node will not be dispatched. The `NODE` parameter must be defined.
`INITREL`	Modifies the defined NODE definition to reset the Initiator Hold flag. This indicates that Initiator requests to this node will be dispatched. The `NODE` parameter must be defined.
`RESPHOLD`	Modifies the defined NODE definition to turn on the Responder Hold flag. This indicates that the responder requests from this node will be rejected with a recoverable error. The `NODE` parameter must be defined.
`RESPREL`	Modifies the defined NODE definition to reset the Responder Hold flag. This indicates that the responder requests from this node will be accepted. The `NODE` parameter must be defined.

Return Codes

The following table lists the return codes of the `FUSNODE` exec:

Return Code	Description
0	Indicates that the Platform Server has successfully displayed the message.
5	Indicates that the NODE was not enabled.
Any other return code	Indicates a network error occurred when processing a request.

Example

The following is an example of using the FUSNODE exec to display information on enabled NODE definitions:

```
FUSNODE N=TESTNODE
```

The output of this example is as follows:

```
ONode Name...........= TESTNODE
IP Address..........= 192.192.100.10
IP.Port.............= 46464
STATUS..............= Normal
SSL.................= No
Protocol............= *Default
Parallel............= YES
Cont.Winners........= 00005
Cont.Losers.........= 00005
Encrypt.............= NONE
retncode===> 0
utput
```

FUSQREXX

You can use the `FUSQREXX` exec to queue a REXX exec to be executed within the address space.

Required Parameter

The following table lists the required parameter of the `FUSSCMD` exec:

Parameter	Description
`COMMAND`	Defines the REXX command to execute. This command can be up to 255 bytes. **Note:** If the `COMMAND` parameter contains imbedded spaces, all the Platform Server parameters must be delimited by commas instead of the normal space delimiter.

Optional Parameters

The following table lists the optional parameters of the `FUSQREXX` exec:

Parameter	Description
`CONFIG` | `CFG`	Defines the configuration file used by the exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
`DATE`	Defines the start date of the request in Julian format, ccyyddd.
`DESCRIPTION`	Defines the user data for the transfer in 25 bytes.
`EVERY`	Defines the interval at which the request will be executed.
`EVERY_COUNT`	Defines the number of times that an `EVERY` request can be executed.
`EVERY_EXDAYS`	Defines the expiration interval in days for a transfer scheduled by an `EVERY` request.

Parameter	Description
EVERY_EXTIME	Defines the expiration interval in hours and minutes for a transfer scheduled by an EVERY request.
EVERY_MISSED	Defines whether a request is scheduled when the interval defined by the EVERY request has passed.
HOLD	Defines whether a request is put on hold when it is queued. The valid values are as follows: • Yes: puts the transfer on hold when it is queued. • No: does not put the transfer on hold when it is queued. This is the default option.
INTERVAL	Defines the status scan interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
MWAIT	Puts a request on a queue, and returns to the caller before the request is completed.
PROCESS	Defines the process name that is associated with the request.
PURGE	Purges uncompleted transfers when the TIMEOUT interval expires. **Note:** This parameter is ignored unless the WAIT parameter is specified.
REXX_EXEC	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec will only be executed when the Platform Server request is purged from the active queue.
SAY	Writes a status record on each status scan.

Parameter	Description
	Note: This parameter is ignored unless the WAIT parameter is also specified.
TIME	Defines the start time of the request in 24-hour clock format, hhmm.
TIMEOUT	Defines the timeout interval. Default =300 seconds. See TIMEOUT parameter in "Waiting for Transfer Requests to Complete". **Note:** This parameter is ignored unless the WAIT parameter is specified.
WAIT	Waits for this request to be completed before returning to REXX.

FUSQSUB

You can use the FUSQSUB exec to queue a request to submit a job locally within the address space of the Platform Server.

Required Parameters

The following table lists the required parameter of FUSSCMD exec:

Parameter	Description
LFILE | LF | DSN	Defines the file that the Platform Server submits to the internal reader.

Optional Parameters

The following table lists the optional parameters of the FUSQSUB exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.

Parameter	Description
DATE	Defines the start date of the request in Julian format, ccyyddd.
DESCRIPTION	Defines the user data for the transfer in 25 bytes.
EVERY	Defines the interval at which the request will be executed.
EVERY_COUNT	Defines the number of times that an EVERY request can be executed.
EVERY_EXDAYS	Defines the expiration interval in days for a transfer scheduled by an EVERY request.
EVERY_EXTIME	Defines the expiration interval in hours and minutes for a transfer scheduled by an EVERY request.
EVERY_MISSED	Defines whether a request is scheduled when the interval defined by the EVERY request has passed.
HOLD	Defines whether a request is put on hold when it is queued. The valid values are as follows: • Yes: Puts the transfer on hold when it is queued. • No: Does not put the transfer on hold when it is queued. This is the default option.
INTERVAL	Defines the status scan interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
MWAIT	Puts a request on a queue, and returns to the caller before the request is completed.
PROCESS	Defines the process name that is associated with the request.
PURGE	Purges uncompleted transfers when the TIMEOUT interval expires.

Parameter	Description
	Note: This parameter is ignored unless the `WAIT` parameter is specified.
`REXX_EXEC`	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec is only executed when the Platform Server request is purged from the active queue.
`SAY`	Writes a status record on each status scan. **Note:** This parameter is ignored unless the `WAIT` parameter is also specified.
`TIME`	Defines the start time of the request in 24-hour clock format, hhmm.
`TIMEOUT`	Defines the timeout interval. **Note:** This parameter is ignored unless the `WAIT` parameter is specified.
`WAIT`	Waits for this request to be completed before returning to REXX.

Delimiter

You can pass a symbolic parameter to the `FUSQSUB` exec by using a tilde character (~) instead of a comma.

Example

The following example shows how to pass a symbolic parameter to the `FUSQSUB` exec by using a tilde character (~):

```
FUSQSUB LF=YOUR.JCL.LIBRARY(MEMBER)~&LF=ABC.DEF
```

FUSRECV

You can use the FUSRECV exec to queue a request to receive a file from a remote partner.

Required Parameters

The following table lists the required parameters of the FUSRECV exec:

Parameter	Description
LFILE | LF | DSN	Defines the dataset that the Platform Server updates with data received by the remote system.
RFILE | RF | REMOTE_FILE	Defines the name of the file on the remote Platform Server system. The remote system will read this file and send the data to the local Platform Server that will write the data to the LFILE file.
IPADDR or IPNAME or LIST or NODE	IPADDR: Defines the dotted decimal IP address of the remote Platform Server node. IPNAME: Defines the IP name of the remote Platform Server node. It can be up to 24 bytes. LIST: Defines the name of the distribution list. NODE: Defines the name of the destination node for a request. The LUname is used if NODE is not defined to the Platform Server.

Optional Parameters

The following table lists the optional parameters of the FUSSCMD exec:

Optional Parameters	Description
ALLOC_TYPE	Defines the allocation type for creating datasets. Valid values are: • C: Cylinders • T: Tracks • M: Megabytes

Optional Parameters	Description
	- K: Kilobytes
ALLOC_DIR	Defines the number of directory blocks that should be allocated when a dataset is created.
ALLOC_PRI	Defines the primary allocation quantity when creating a dataset.
ALLOC_SEC	Defines the secondary allocation quantity when creating a dataset.
AVAIL	Defines whether TAPE is supported. - IMMED: The remote file is available to users as soon as the file transfer is completed. This is the default and the best practice. - DEFER: The remote file availability is deferred. The Platform Server treats this as a TAPE.
BLOCKSIZE	Defines the size of the block when creating a dataset.
CKPT	Defines whether to take checkpoints during this transfer. Valid values are as follows: - nn: Defines the time in minutes that each checkpoint takes. Allowable values are 1 - 9999 minutes. - YES: Uses a 5-minute checkpoint interval. - NO: Does not use checkpoint. This is the default option.
COMPRESS	Defines whether to use data compression, or defines the data compression algorithm to use. Valid values are as follows: - NO: No compression is used. This is the default option. - YES: Uses RLE compression. - RLE: Uses RLE compression.

Optional Parameters	Description
	• LZ: Uses LZ compression.
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server uses the default configuration.
COPIES	Defines the number of copies for a report. **Note:** This parameter is used only when TRANS_TYPE=PRINT is specified.
CRCCHECK(CRC)	Defines whether CRC checking will be turned on by default. CRC checking is performed against the data that is sent over the network to ensure that the data has not been corrupted. The CRC is not performed against the file itself. ASCII to EBCDIC conversion, translation and LF/CRLF change the contents of the file between the sender and receiver so the CRC is not performed against the file contents. Valid values are: • YES: Performs CRC checking by default • NO: Does not perform CRC checking by default The CRCCHECK parameter on the Node and batch parameters override the Global CRCCHECK settings. **Note:** If the partner Platform Server does not support CRC checking, CRC is computed but will not be checked against the partner's CRC value computed.
CRLF	Defines whether records are delimited by CRLF (Carriage Return/Line Feed). Valid values are as follows: • YES: Records are delimited by CRLF.

Optional Parameters	Description
	- NO: Records are not delimited by CRLF. This is the default option.
DATACLASS	Defines the SMS data class when creating a dataset.
DATASET_TYPE	Defines the type of dataset to be created. Valid values are: - PDS: Creates z/OS PDS. - LIBRARY: Creates z/OS PDS/E. - DSN: Creates z/OS sequential files. - VSAM: Creates z/OS VSAM files.
DATE	Defines the start date of the request in Julian format, ccyyddd.
DESCRIPTION	Defines the user data for the transfer in 25 bytes.
DESTINATION	Defines the destination for a report.
DISP	Defines the type of allocation that the Platform Server performs for the request. Valid values are: - SHR: Allocates as DISP=SHR. - OLD: Allocates as DISP=OLD.
EFFECT	Defines how the Platform Server open a dataset when writing to the dataset. Valid values are: - C: Creates a file. - R: Replaces a file. - A: Appends a file.

Optional Parameters	Description
	• CR: Creates or replaces a file. This is the default option. • CA: Creates or appends a file.
EMAIL_FAIL	Defines the email address to which to send an email notification when a request is completed unsuccessfully.
EMAIL_GOOD	Defines the email address to which to send an email notification when a request is completed successfully.
ENCRYPT	Defines the level of encryption that is used by the transfer. If specified, this parameter overrides the ENCRYPT parameter specified in the GLOBAL or NODE definitions for this transfer. Valid values are: • NONE: No encryption is used for this transfer. • DES: DES encryption is used. • 3DES: triple DES encryption is used. • BF: Blowfish 56-bit encryption is used. • BLOWFISH: Blowfish 56-bit encryption is used. • BFL: Blowfish 448-bit encryption is used. • BLOWFISH_LONG: Blowfish 448-bit encryption is used. • AES: AES 256-bit encryption is used. For more information on this parameter, see Data Encryption.
EVERY	Defines the interval at which the request will be executed.
EVERY_COUNT	Defines the number of times that an EVERY request can be executed.
EVERY_EXDAYS	Defines the expiration interval in days for a transfer scheduled by an EVERY request.

Optional Parameters	Description
EVERY_EXTIME	Defines the expiration interval in hours and minutes for a transfer scheduled by an EVERY request.
EVERY_MISSED	Defines whether a request is scheduled when the interval defined by the EVERY request has passed.
EXPDT	Defines the expiration date of a file transfer request, in Julian format, CCYYDDD. The default value is 30 days from now.
EXPTM	Defines the expiration time for a file transfer request.
FCB	Defines a form control buffer for a report. **Note:** This parameter is used only when TRANS_TYPE=PRINT is specified.
FORM	Defines the name of the form for a report. **Note:** This parameter is used only when TRANS_TYPE=PRINT is specified.
HOLD	Defines whether a request is put on hold when it is queued. Valid values are: • Yes: Puts the transfer on hold when it is queued. • No: Does not put the transfer on hold when it is queued. This is the default option.
INTERVAL	Defines the status scan interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.

Optional Parameters	Description
IVOLUME	Defines the volume that is used for input dataset processing by the Platform Server. When this parameter is specified, the catalog is not used to allocate the dataset.
LDISP	Defines local status, normal disposition, and error disposition. The values of the LDISP parameter must be delimited by a colon instead of a comma.
LENGTH	The record length of a file when creating a dataset.
LOCALLABEL	Defines the label used on the initiating Platform Server when sending or receiving a tape file. The input format is (*nn*,SL) where *nn* defines the file label number and SL is the only label type currently supported. For sending a tape file, if the label information is in the catalog, this parameter is not required. This parameter is ignored for disk files.
LOGON_DOMAIN	Defines the name of the Windows domain to which to log on.
MGTCLASS	Defines the SMS management class when creating a dataset.
MQ_GOOD	Defines the MQ (Message Queue) to which the Platform Server writes a record when a request is completed successfully.
MQ_FAIL	Defines the MQ (Message Queue) to which Platform Server writes a record when a request fails.
MWAIT	Puts a request on a queue and returns to the caller before the request is completed.
NODE	Defines the name of the destination node for a request. The LUname is used if the NODE parameter is not defined to the Platform Server.

Optional Parameters	Description
NOTIFY	Defines the TSO user to notify when the request is completed.
NOTIFY_TYPE	Defines a type of user to notify when the request is completed. Currently, the only option supported is TSO.
OVOLUME	Defines the volume that is used for that is used for the output dataset for a transfer. When this parameter is specified, the catalog is not used to allocate the dataset.
PPA	Defines the preprocessing or postprocessing actions that are performed for the file transfer. The format of the PPA command is: `PPA={"S | F |P, L | R, CALLJCL | CALLPGM | SUBMIT | COMMAND data_to_be_processed"}` Where Status is: S: Success, F: Failure, P: Pre-Processing Where Source is: L: Local (Initiator), R: Remote (Responder) Where Action is: CALLJCL: Call program on z/OS with JCL linkage CALLPGM: Call program on z/OS with standard linkage SUBMIT: Submit job on z/OS COMMAND: Execute a command Where Data is: The data to be processed
PROCESS	Defines the process name that is associated with the request.
PRIORITY	Defines the priority for file transfers.

Optional Parameters	Description
	The allowable values are 0 - 9. The default value is 3.
PRINTER_NAME	Defines the name of a remote printer.
PURGE	Defines whether to purge uncompleted transfers when the TIMEOUT interval expires. **Note:** This parameter is ignored unless the WAIT parameter is specified.
RDISP	Defines the remote status, normal disposition, and error disposition. The values of the RDISP parameter must be delimited by a colon instead of a comma.
RECFM	Defines the platform record format when creating a dataset. Valid values are: • F: Fixed • FB: Fixed block • V: Variable • VB: Variable block • U: Undefined • FS: Fixed standard • FBS: Fixed blocked standard • VS: Variable spanned • VBS: Variable blocked spanned
REMOTELABEL	Defines the label used on the target Platform Server when sending or receiving a tape file. The input format is (*nn*,SL) where *nn* defines the file label number and SL is the only label type currently supported. For sending a tape file, if the label information is in the

Optional Parameters	Description
	catalog, this parameter is not required. This parameter is ignored for disk files.
RETENTIONPRD	Defines the number of days that the Platform Server leaves a request on an active queue before purging the request. The allowable values are 1 - 9999. The default value is 30 days.
RETRY	Defines the number of times that the Platform Server tries a request when a network error occurred. The default value of 1 means that the Platform Server purges the transfer when the first network error occurs. For best results set this parameter to 1 if you specify the WAIT parameter.
RETRYINT	Defines the retry interval of the file transfer. This parameter can be specified in terms of hours, for example, RETRYINT=2H or minutes, for example, RETRYINT=0M.
REXX_EXEC	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec is only executed when the Platform Server request is purged from the active queue.
RMTRAIL	Defines whether to remove the trailing blanks and spaces when sending a TEXT file to the Platform Server for UNIX or Windows. Valid values are: • YES: Removes trailing spaces and nulls. • NO: Does not remove trailing spaces and nulls. This is the default option.
RPASS	Defines the password for the user on a remote computer.

Optional Parameters	Description
	Note: This parameter can only be defined when the RUSER parameter is also defined.
RUSER	Defines the name of the user on the remote computer. If this parameter is not defined, the default value *PROFILE is used.
SAY	Writes a status record on each status scan. **Note:** This parameter is ignored unless the WAIT parameter is also specified.
STORCLASS	Defines the SMS storage class when creating the dataset.
SYSOUT	Defines the SYSOUT class for a report. **Note:** This parameter is used only when TRANS_TYPE=PRINT is specified.
TEMPLATE	Defines the Windows template for a file transfer.
TIME	Defines the start time of the request in 24-hour clock format, hhmm.
TIMEOUT	Defines the timeout interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
TLS	Defines whether TLS (SSL) can be used for this request. This definition overrides the GLOBAL or NODE definitions. Valid values are: • YES: TLS is used for this transfer.

Optional Parameters	Description
	• NO: TLS is not used for this transfer. • TUNNEL: All data transmitted is sent over an encrypted TLS Tunnel.
TRANS_TYPE	Defines the type of transfer being sent or received. Valid values are: • FILE: Data is written to a file. This is the default option. • PRINT: Data is sent to a printing queue. • JOB: Data is run as a job. • COMMAND: Data is executed as a command.
TRUNCATE(TRUNC)	This parameter is only used when receiving a file from Windows, UNIX, IBM i, or MFT Internet Server when record delimiters are defined. This parameter is ignored for a SEND or when communicating to another z/OS system. Valid values are: • NO: If the record received from the partner is greater than the LRECL, a truncate error occurs and the transfer is terminated with an error. • YES: If the record received from the partner is greater than the LRECL, the record is truncated and any data after the LRECL is lost. • WRAP: If the record received from the partner is greater than the LRECL, the record is truncated and any data after the LRECL is written to the next record. If the record received has no delimiters within the first 32760 bytes, a truncate error occurs and the transfer is terminated with an error. In cases like this, you should turn off delimiter checking, so records will be written without any delimiter checking.
TYPE	Defines the type of data in a file, and whether ASCII - EBCDIC conversion occurs.

Optional Parameters	Description
	Valid values are: • TEXT: Removes trailing blanks and nulls, and perform data conversion. • DATA: The same as TEXT except that trailing blanks and nulls are not removed. • BINARY: Does not perform data conversion. This is the default option.
USERNAME	Defines the user name associated with a report.
UTF8BOM	Defines whether the UTF-8 BOM(0xefbbbf) is added at the start of the converted data or is removed from the start of the data before conversion. This parameter is used only when the LCT parameter defines that Unicode services are used to convert data. This parameter can be used for a z/OS Initiated Send or Receive transfers, although it is more commonly used when sending a file to a target system that requires a BOM. Valid values are: • ADD: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. This value is typically used when sending data to a target system (like Linux) that requires the UTF-8 BOM at the start of the file. • REMOVE: The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data. If the start of the file does not include the BOM, no changes are made. This value is typically used when receiving data from a target system (like Linux) that contains the UTF-8 BOM at the start of the file. • BOTH: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data. • NONE: No UTF-8 BOM processing is performed. The UTF-8 BOM is not inserted at the start of the converted data or

Optional Parameters	Description
	removed before conversion.
`VOLUME`	Defines the volumes where the dataset can be written. Up to 5 volumes can be defined. If more than one volume is defined, the volumes must be delimited by colons.
`VSAM_RRSLOT`	Defines whether the RRDS slot number is included in a VSAM RRDS transfer.
`VSAM_KEYPOS`	Defines the VSAM KSDS key position relative to 0.
`VSAM_KEYLEN`	Defines the VSAM KSDS key length.
`VSAM_TYPE`	Defines the type of VSAM file to be created. Valid values are: • ESDS • KSDS • RRDS
`VSAM_LIKE`	Defines the model DSN for VSAM file creation.
`VSAM_LRECL`	Defines the record length used for VSAM file creation.
`VSAM__REUSE`	Defines whether to use the VSAM RESUSE option.
`WAIT`	Waits for this request to be completed before returning to REXX.
`WRITER`	Defines the name of the z/OS external writer for a report. **Note:** This parameter is only when `TRANS_TYPE=PRINT` is specified.

Example

The following example shows how to use the `FUSRECV` exec to receive a file from a remote partner:

```
FUSRECV IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.FILE.RECV1
RF=C:\FUSRECV1.TXT INTERVAL=3 SAY WAIT CRLF=YES TYPE=TEXT
```

The output of this example is as follows:

```
TRANSACTION=IA01500041 STATUS=ACTIVE   RECORD=000000000 BYTES=000000000

 TRANSACTION=IA01500041 STATUS=COMPLETE RECORD=000000052 BYTES=000004098

 ***Fusion retcode.......=  0

 ***Fusion TransNum......=  IA01500041

 ***Completed transfers..=  01

 ***Successful transfers.=  01

 ***Failed transfers.....=  00

 ***Purged transfers.....=  00

 ***Last Message.........=  PGTF3101I Activity IA01500041 successfully
transferred 52
                            records with remote node 127.127.127.1
Request Queued successfully with TRANSNUM= IA01500041
```

FUSRECVT

You can use the FUSRECVT exec to queue a request to receive a text file from a remote partner.

The FUSRECVT exec is identical to the FUSRECV exec, except that Fusion automatically adds the following configurations:

- TYPE=TEXT: Informs the Platform Server to convert the data from ASCII to EBCDIC if the data is received from an ASCII system.
- CRLF=YES: Indicates that the records must be delimited by CRLF (Carriage Return/Line Feed).

When sending data between TIBCO MFT Platform Server for z/OS nodes, the FUSRECV and FUSRECVT execs are identical.

The parameters used with the FUSRECVT exec are the same as those used with the FUSRECVT exec. For more information about the parameters, see FUSRECV.

Example

The following example shows how to use the FUSRECVT exec to receive a file from a remote partner:

```
FUSRECVT IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.FILE.RECV1
RF=C:\FUSRECV1.TXT INTERVAL=3 WAIT SAY
```

Note: The FUSRECV exec and FUSRECVT exec examples perform the same exact transfer.

FUSSCMD

You can use the FUSSCMD exec to queue a request to send a command to a remote partner.

Required Parameters

The following table lists the required parameters of the FUSSCMD exec:

Parameter	Description
REXXEXEC or CALLJCL or CALLPROG or SUBMIT	REXXEXEC: Defines a command or exec to be executed on the remote platform. No parameter can be passed to the command. CALLJCL: Defines the name of the program to call, and the parameters that are to be passed to the program. CALLPROG: Defines the name of the program to call, and the parameters that are to be passed to the program. SUBMIT: Defines the name of the JCL to submit, and any symbolic parameters associated with the JCL. The parameters must be delimited by colons (:) instead of comma (,).
IPADDR or IPNAME or LIST or NODE	IPADDR: Defines the dotted decimal IP address of the remote Platform Server node. IPNAME: Defines the IP name of the remote Platform Server node. It can be up to 24 bytes. LIST: Defines the name of the distribution list.

Parameter	Description
	NODE: Defines the name of the destination node for a request. The LUname is used if NODE is not defined to the Platform Server.

Optional Parameters

The following table lists the optional parameters of the FUSSCMD exec:

Optional Parameters	Description
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server uses the default configuration.
DATE	Defines the start date of the request in Julian format, ccyyddd.
DESCRIPTION	Defines the user data for the transfer in 25 bytes.
EMAIL_FAIL	Defines the email address to which to send an email notification when a request is completed unsuccessfully.
EMAIL_GOOD	Defines the email address to which to send an email notification when a request is completed successfully.
ENCRYPT	Defines the level of encryption that is used by the transfer. If specified, this parameter overrides the ENCRYPT parameter specified in the GLOBAL or NODE definitions for this transfer. Valid values are: • NONE: No encryption is used for this transfer. • DES: DES encryption is used. • 3DES: Triple DES encryption is used. • BF: Blowfish 56-bit encryption is used. • BLOWFISH: Blowfish 56-bit encryption is used. • BFL: Blowfish 448-bit encryption is used.

Optional Parameters	Description
	- BLOWFISH_LONG: Blowfish 448-bit encryption is used. - AES: AES 256-bit encryption is used. For more information on this parameter, see Data Encryption.
EXPDT	Defines the expiration date of a file transfer request, in Julian format, CCYYDDD. The default value is 30 days from now.
EXPTM	Defines the expiration time for a file transfer request.
INTERVAL	Defines the status scan interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
NOTIFY	Defines the TSO user to notify when the request is completed.
NOTIFY_TYPE	Defines a type of user to notify when the request is completed. Currently, the only option supported is TSO.
PPA	Defines the preprocessing or postprocessing actions that are performed for the file transfer. The format of the PPA command is: `PPA={"S | F |P, L | R, CALLJCL | CALLPGM | SUBMIT | COMMAND data_to_be_processed"}` Where Status is: S: Success, F: Failure, P: Pre-Processing Where Source is: L: Local (Initiator), R: Remote (Responder) Where Action is: CALLJCL: Call program on z/OS with JCL linkage

Optional Parameters	Description
	CALLPGM: Call program on z/OS with standard linkage SUBMIT: Submit job on z/OS COMMAND: Execute a command Where Data is: The data to be processed
PROCESS	Defines the process name that is associated with the request.
PRIORITY	Defines the priority for file transfers. The allowable values are 0 - 9. The default value is 3.
PRINTER_NAME	Defines the name of a remote printer.
PURGE	Defines whether to purge uncompleted transfers when the TIMEOUT interval expires. **Note:** This parameter is ignored unless the WAIT parameter is specified.
RETENTIONPRD	Defines the number of days that the Platform Server leaves a request on an active queue before purging the request. The allowable values are 1 - 9999. The default value is 30 days.
RETRY	Defines the number of times that the Platform Server tries a request when a network error occurred. The default value of 1 means that the Platform Server purges the transfer when the first network error occurs. For best results set this parameter to 1 if you specify the WAIT parameter.
RETRYINT	Defines the retry interval of the file transfer. This parameter can be specified in terms of hours, for example, RETRYINT=2H or minutes, for example, RETRYINT=0M.

Optional Parameters	Description
REXX_EXEC	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec is only executed when the Platform Server request is purged from the active queue.
RPASS	Defines the password for the user on a remote computer. **Note:** This parameter can only be defined when the RUSER parameter is also defined.
RUSER	Defines the name of the user on the remote computer. If this parameter is not defined, the default value *PROFILE is used.
SAY	Writes a status record on each status scan. **Note:** This parameter is ignored unless the WAIT parameter is also specified.
TIME	Defines the start time of the request in 24-hour clock format, hhmm.
TIMEOUT	Defines the timeout interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
WAIT	Waits for this request to be completed before returning to REXX.

Example

The following example shows how to use the FUSSCMD exec to queue a request to send a command to a remote partner:

```
FUSSCMD NODE=SNANODE SUBMIT=JCL.LIB(ABC123) INTERVAL=3 SAY WAIT
```

The output of this example is as follows:

```
TRANSACTION=IA01500042 STATUS=ACTIVE   RECORD=000000000 BYTES=000000000

TRANSACTION=IA01500042 STATUS=COMPLETE RECORD=000000000 BYTES=000000000

***Fusion retcode.......=  0

***Fusion TransNum......=  IA01500042

***Completed transfers..=  01

***Successful transfers.=  01

***Failed transfers.....=  00

***Purged transfers.....=  00

***Last Message.........=  PGTF3101I Activity IA01500042 successfully
transferred 0 records with remote node SNANODE
Request Queued successfully with TRANSNUM= IA01500042
```

FUSSCRPT

You can use the `FUSSCRPT` exec to queue a request to schedule a script request to run by the Platform Server.

Required Parameter

The following table lists the required parameter of the `FUSSCMD` exec:

Parameter	Description
`SCRIPT`	Defines the name of the script file to be executed. If the script file contains imbedded commas, the commas must be replaced with colons (:). The `FUSSCRPT` exec will then replace the colons with commas.

Optional Parameters

The following table lists the optional parameters of the `FUSSCRPT` exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
DATE	Defines the start date of the request in Julian format, ccyyddd.
DESCRIPTION	Defines the user data for the transfer in 25 bytes.
EVERY	Defines the interval at which the request will be executed.
EVERY_COUNT	Defines the number of times that an EVERY request can be executed.
EVERY_EXDAYS	Defines the expiration interval in days for a transfer scheduled by an EVERY request.
EVERY_EXTIME	Defines the expiration interval in hours and minutes for a transfer scheduled by an EVERY request.
EVERY_MISSED	Defines whether a request is scheduled when the interval defined by the EVERY request has passed.
HOLD	Defines whether a request is put on hold when it is queued. The valid values are as follows: • Yes: puts the transfer on hold when it is queued. • No: does not put the transfer on hold when it is queued. This is the default option.
INTERVAL	Defines the status scan interval. **Note:** This parameter is ignored unless the WAIT parameter is specified.
MWAIT	Puts a request on a queue, and returns to the caller before the request is completed.

Parameter	Description
PPA	Defines the preprocessing or postprocessing actions that are performed for the file transfer. The format of the PPA command is: `PPA={"S | F |P, L | R, CALLJCL | CALLPGM | SUBMIT | COMMAND data_to_be_processed"}` Where Status is: S: Success, F: Failure, P: Pre-Processing Where Source is: L: Local (Initiator), R: Remote (Responder) Where Action is: CALLJCL: Call program on z/OS with JCL linkage CALLPGM: Call program on z/OS with standard linkage SUBMIT: Submit job on z/OS COMMAND: Execute a command Where Data is: The data to be processed
PROCESS	Defines the process name that is associated with the request.
PURGE	Purges uncompleted transfers when the TIMEOUT interval expires. **Note:** This parameter is ignored unless the WAIT parameter is specified.
REXX_EXEC	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec will only be executed when the Platform Server request is purged from the active queue.
SAY	Writes a status record on each status scan.

Parameter	Description
	Note: This parameter is ignored unless the WAIT parameter is also specified.
TIME	Defines the start time of the request in 24-hour clock format, hhmm.
TIMEOUT	Defines the timeout interval. Default =300 seconds. See TIMEOUT parameter in "Waiting for Transfer Requests to Complete". **Note:** This parameter is ignored unless the WAIT parameter is specified.
WAIT	Waits for this request to be completed before returning to REXX.

Delimiter

You can pass a symbolic parameter to the FUSSCRPT exec by using a tilde character (~) instead of a comma.

Example

The following example shows how to pass a symbolic parameter to the FUSSCRPT exec by using a tilde character (~):

```
FUSQSUB LF=YOUR.JCL.LIBRARY(MEMBER)~&LF=ABC.DEF
```

FUSSEND

You can use the FUSSEND exec to queue a request to send a file to a remote partner.

Required Parameters

The following table lists the required parameters of the FUSSCMD exec:

Parameter	Description
LFILE | LF | DSN	Defines the dataset that the Platform Server updates with data

Parameter	Description
	received by the remote system.
RFILE | RF | REMOTE_FILE	Defines the name of the file on the remote Platform Server system. The remote system will read this file and send the data to the local Platform Server that will write the data to the LFILE file.
IPADDR or IPNAME or LIST or NODE	IPADDR: Defines the dotted decimal IP address of the remote Platform Server node. IPNAME: Defines the IP name of the remote Platform Server node. It can be up to 24 bytes. LIST: Defines the name of the distribution list. NODE: Defines the name of the destination node for a request. The LUname is used if NODE is not defined to the Platform Server.

Optional Parameters

The optional parameters of the FUSSEND exec are as follows: includes the following optional parameters:

Optional Parameters	Description
ALLOC_TYPE	Defines the allocation type for creating datasets. Valid values are: • C: Cylinders • T: Tracks • M: Megabytes • K: Kilobytes
ALLOC_DIR	Defines the number of directory blocks that should be allocated when a dataset is created.

Optional Parameters	Description
ALLOC_PRI	Defines the primary allocation quantity when creating a dataset.
ALLOC_SEC	Defines the secondary allocation quantity when creating a dataset.
AVAIL	Defines whether TAPE is supported. • IMMED: The remote file is available to users as soon as the file transfer is completed. This is the default and the best practice. • DEFER: The remote file availability is deferred. The Platform Server treats this as a TAPE.
BLOCKSIZE	Defines the size of the block when creating a dataset.
CKPT	Defines whether to take checkpoints during this transfer. Valid values are: • nn: Defines the time in minutes that each checkpoint takes. Allowable values are 1 - 9999 minutes. • YES: Uses a 5-minute checkpoint interval. • NO: Does not use checkpoint. This is the default option.
COMPRESS	Defines whether to use data compression, or defines the data compression algorithm to use. Valid values are: • NO: No compression is used. This is the default option. • YES: Uses RLE compression. • RLE: Uses RLE compression. • LZ: Uses LZ compression. • ZLIB1 - ZLIB9: Uses ZLIB compression.

Optional Parameters	Description
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server uses the default configuration.
COPIES	Defines the number of copies for a report. **Note:** This parameter is used only when TRANS_TYPE=PRINT is specified.
CRCCHECK(CRC)	Defines whether to perform CRC checking on the transfer. Valid values are: • YES: CRC checking is used. • NO: No CRC checking is used.
CRLF	Defines whether records are delimited by CRLF (Carriage Return/Line Feed). Valid values are: • YES: Records are delimited by CRLF. • NO: Records are not delimited by CRLF. This is the default option.
DATACLASS	Defines the SMS data class when creating a dataset.
DATASET_TYPE	Defines the type of dataset to be created. Valid values are: • PDS: Creates z/OS PDS. • LIBRARY: Creates z/OS PDS/E. • DSN: Creates z/OS sequential files. • VSAM: Creates z/OS VSAM files.

Optional Parameters	Description
DESCRIPTION	Defines the user data for the transfer in 25 bytes.
DESTINATION	Defines the destination for a report.
DISP	Defines the type of allocation that the Platform Server performs for the request. Valid values are: • SHR: Allocates as DISP=SHR. • OLD: Allocates as DISP=OLD.
EFFECT	Defines how the Platform Server open a dataset when writing to the dataset. Valid values are: • C: Creates a file. • R: Replaces a file. • A: Appends a file. • CR: Creates or replaces a file. This is the default option. • CA: Creates or appends a file.
EMAIL_FAIL	Defines the email address to which to send an email notification when a request is completed unsuccessfully.
EMAIL_GOOD	Defines the email address to which to send an email notification when a request is completed successfully.
ENCRYPT	Defines the level of encryption that is used by the transfer. If specified, this parameter overrides the ENCRYPT parameter specified in the GLOBAL or NODE definitions for this transfer. Valid values are: • NONE: No encryption is used for this transfer.

Optional Parameters	Description
	- DES: DES encryption is used. - 3DES: Triple DES encryption is used. - BF: Blowfish 56-bit encryption is used. - BLOWFISH: Blowfish 56-bit encryption is used. - BFL: Blowfish 448-bit encryption is used. - BLOWFISH_LONG: Blowfish 448-bit encryption is used. - AES: AES 256-bit encryption is used. For more information on this parameter, see Data Encryption.
EVERY	Defines the interval at which the request will be executed.
EVERY_COUNT	Defines the number of times that an EVERY request can be executed.
EVERY_EXDAYS	Defines the expiration interval in days for a transfer scheduled by an EVERY request.
EVERY_EXTIME	Defines the expiration interval in hours and minutes for a transfer scheduled by an EVERY request.
EVERY_MISSED	Defines whether a request is scheduled when the interval defined by the EVERY request has passed.
EXPDT	Defines the expiration date of a file transfer request, in Julian format, CCYYDDD. The default value is 30 days from now.
EXPTM	Defines the expiration time for a file transfer request.
FCB	Defines a form control buffer for a report.

Optional Parameters	Description
	Note: This parameter is used only when `TRANS_TYPE=PRINT` is specified.
`FORM`	Defines the name of the form for a report. **Note:** This parameter is used only when `TRANS_TYPE=PRINT` is specified.
`HOLD`	Defines whether a request is put on hold when it is queued. Valid values are: • `Yes`: Puts the transfer on hold when it is queued. • `No`: Does not put the transfer on hold when it is queued. This is the default option.
`INTERVAL`	Defines the status scan interval. **Note:** This parameter is ignored unless the `WAIT` parameter is specified.
`IVOLUME`	Defines the volume that is used for input dataset processing by the Platform Server. When this parameter is specified, the catalog is not used to allocate the dataset.
`LDISP`	Defines local status, normal disposition, and error disposition. The values of the `LDISP` parameter must be delimited by a colon instead of a comma.
`LENGTH`	The record length of a file when creating a dataset.
`LOCALLABEL`	Defines the label used on the initiating Platform Server when sending or receiving a tape file. The input format is (*nn*,`SL`)

Optional Parameters	Description
	where *nn* defines the file label number and SL is the only label type currently supported. For sending a tape file, if the label information is in the catalog, this parameter is not required. This parameter is ignored for disk files.
LOGON_DOMAIN	Defines the name of the Windows domain to which to log on.
MGTCLASS	Defines the SMS management class when creating a dataset.
MQ_GOOD	Defines the MQ (Message Queue) to which the Platform Server writes a record when a request is completed successfully.
MQ_FAIL	Defines the MQ (Message Queue) to which Platform Server writes a record when a request fails.
MWAIT	Puts a request on a queue, and returns to the caller before the request is completed.
NODE	Defines the name of the destination node for a request. The LUname is used if the NODE parameter is not defined to the Platform Server.
NOTIFY	Defines the TSO user to notify when the request is completed.
NOTIFY_TYPE	Defines a type of user to notify when the request is completed. Currently, the only option supported is TSO.
OVOLUME	Defines the volume that is used for that is used for the output dataset for a transfer. When this parameter is specified, the catalog is not used to allocate the dataset.
PERMIT_ACTNS	Defines the permitted actions for Windows. Valid values are:

Optional Parameters	Description
	• S: System • H: Hidden file • A: Archive file • R: Read only • C: NTFS compression • Z: EOF CRLF. Adds CRLF or CTRL-/Z to file end. • E: EOF. Adds CTRL-Z to file end.
PPA	Defines the preprocessing or postprocessing actions that are performed for the file transfer. The format of the PPA command is: `PPA={"S | F |P, L | R, CALLJCL | CALLPGM | SUBMIT | COMMAND data_to_be_processed"}` Where Status is: S: Success, F: Failure, P: Pre-Processing Where Source is: L: Local (Initiator), R: Remote (Responder) Where Action is: CALLJCL: Call program on z/OS with JCL linkage CALLPGM: Call program on z/OS with standard linkage SUBMIT: Submit job on z/OS COMMAND: Execute a command Where Data is: The data to be processed
PROCESS	Defines the process name that is associated with the request.
PRIORITY	Defines the priority for file transfers.

Optional Parameters	Description
	The allowable values are `0 - 9`. The default value is `3`.
`PRINTER_NAME`	Defines the name of a remote printer.
`PURGE`	Defines whether to purge uncompleted transfers when the `TIMEOUT` interval expires. **Note:** This parameter is ignored unless the `WAIT` parameter is specified.
`RDISP`	Defines the remote status, normal disposition, and error disposition. The values of the `RDISP` parameter must be delimited by a colon instead of a comma.
`RECFM`	Defines the platform record format when creating a dataset. Valid values are: • `F`: Fixed • `FB`: Fixed block • `V`: Variable • `VB`: Variable block • `FS`: Fixed standard • `VS`: Variable spanned • `VBS`: Variable blocked spanned
`REMOTELABEL`	Defines the label used on the target Platform Server when sending or receiving a tape file. The input format is (*nn*,`SL`) where *nn* defines the file label number and `SL` is the only label type currently supported. For sending a tape file, if the label information is in the catalog, this parameter is not required. This parameter is ignored for disk files.

Optional Parameters	Description
RETENTIONPRD	Defines the number of days that the Platform Server leaves a request on an active queue before purging the request. The allowable values are 1 - 9999. The default value is 30 days.
RETRY	Defines the number of times that the Platform Server tries a request when a network error occurred. The default value of 1 means that the Platform Server purges the transfer when the first network error occurs. For best results set this parameter to 1 if you specify the WAIT parameter.
RETRYINT	Defines the retry interval of the file transfer. This parameter can be specified in terms of hours, for example, RETRYINT=2H or minutes, for example, RETRYINT=0M.
REXX_EXEC	Defines a REXX exec that will be executed when the Platform Server request is completed, either successfully or unsuccessfully. **Note:** This exec will only be executed when the Platform Server request is purged from the active queue.
RMTRAIL	Defines whether to remove the trailing blanks and spaces when sending a TEXT file to the Platform Server for UNIX or Windows. Valid values are: • YES: Removes trailing spaces and nulls. • NO: Does not remove trailing spaces and nulls. This is the default option.
RPASS	Defines the password for the user on a remote computer. **Note:** This parameter can only be defined when the RUSER parameter is also defined.

Optional Parameters	Description
RUSER	Defines the name of the user on the remote computer. If this parameter is not defined, the default value `*PROFILE` is used.
SAY	Writes a status record on each status scan. **Note:** This parameter is ignored unless the `WAIT` parameter is also specified.
STORCLASS	Defines the SMS storage class when creating the dataset.
SYSOUT	Defines the SYSOUT class for a report. **Note:** This parameter is used only when `TRANS_TYPE=PRINT` is specified.
TEMPLATE	Defines the Windows template for a file transfer.
TIME	Defines the start time of the request in 24-hour clock format, hhmm.
TIMEOUT	Defines the timeout interval. See TIMEOUT parameter in Waiting for Transfer Requests to Complete. **Note:** This parameter is ignored unless the `WAIT` parameter is specified.
TLS	Defines whether TLS (SSL) can be used for this request. This definition overrides the GLOBAL or NODE definitions. Valid values are: • `YES`: TLS is used for this transfer. • `NO`: TLS is not used for this transfer. • `TUNNEL`: All data transmitted is sent over an encrypted

Optional Parameters	Description
	TLS Tunnel connection.
TRANS_TYPE	Defines the type of transfer being sent or received. Valid values are: • FILE: data is written to a file. This is the default option. • PRINT: data is sent to a printing queue. • JOB: data is run as a job. • COMMAND: data is executed as a command.
TRUNCATE(TRUNC)	This parameter is only used when receiving a file from Windows, UNIX, IBM i, or MFT Internet Server when record delimiters are defined. This parameter is ignored for a SEND or when communicating to another z/OS system. Valid values are: • NO: If the record received from the partner is greater than the LRECL, a truncate error occurs and the transfer is terminated with an error. • YES: If the record received from the partner is greater than the LRECL, the record is truncated and any data after the LRECL is lost. • WRAP: If the record received from the partner is greater than the LRECL, the record is truncated and any data after the LRECL is written to the next record. If the record received has no delimiters within the first 32760 bytes, a truncate error occurs and the transfer is terminated with an error. In cases like this, you should turn off delimiter checking, so records will be written without any delimiter checking.
TYPE	Defines the type of data in a file, and whether ASCII - EBCDIC conversion occurs. Valid values are:

Optional Parameters	Description
	• TEXT: Removes trailing blanks and nulls, and perform data conversion. • DATA: The same as TEXT except that trailing blanks and nulls are not removed. • BINARY: Does not perform data conversion. This is the default option.
USERNAME	Defines the user name associated with a report.
UTF8BOM	Defines whether the UTF-8 BOM(0xefbbbf) is added at the start of the converted data or is removed from the start of the data before conversion. This parameter is used only when the LCT parameter defines that Unicode services are used to convert data. This parameter can be used for a z/OS Initiated Send or Receive transfers, although it is more commonly used when sending a file to a target system that requires a BOM. Valid values are: • ADD: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. This value is typically used when sending data to a target system (like Linux) that requires the UTF-8 BOM at the start of the file. • REMOVE: The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data. If the start of the file does not include the BOM, no changes are made. This value is typically used when receiving data from a target system (like Linux) that contains the UTF-8 BOM at the start of the file. • BOTH: The UTF-8 BOM (0xEFBBBF) is added at the start of the converted data. The UTF-8 BOM (0xEFBBBF) is removed from the start of the file before converting the data. • NONE: No UTF-8 BOM processing is performed. The UTF-8 BOM is not inserted at the start of the converted data or

Optional Parameters	Description
	removed before conversion.
VOLUME	Defines the volumes where the dataset can be written. Up to 5 volumes can be defined. If more than one volume is defined, the volumes must be delimited by colons.
VSAM_RRSLOT	Defines whether the RRDS slot number is included in a VSAM RRDS transfer.
VSAM_KEYPOS	Defines the VSAM KSDS key position relative to 0.
VSAM_KEYLEN	Defines the VSAM KSDS key length.
VSAM_TYPE	Defines the type of VSAM file to be created. Valid values are: • ESDS • KSDS • RRDS
VSAM_LIKE	Defines the model DSN for VSAM file creation.
VSAM_LRECL	Defines the record length used for VSAM file creation.
VSAM__REUSE	Defines whether to use the VSAM RESUSE option.
WAIT	Waits for this request to be completed before returning to REXX.
WRITER	Defines the name of the z/OS external writer for a report. **Note:** This parameter is only when TRANS_TYPE=PRINT is specified.

Example

The following example shows how to use the FUSSEND exec to queue a request to send a file to a remote partner:

```
FUSSEND NODE=SNANODE LF=LOCAL.FILE1 RF=C:\FUSSEND.TXT INTERVAL=3
SAY WAIT TYPE=TEXT CRLF=YES
```

The output of this example is as follows:

```
TRANSACTION=IA01500042 STATUS=ACTIVE   RECORD=000000000
BYTES=000000000
TRANSACTION=IA01500042 STATUS=COMPLETE RECORD=000000052
BYTES=000004160
***Fusion retcode.......=  0
***Fusion TransNum......=  IA01500042
***Completed transfers..=  01
***Successful transfers.=  01
***Failed transfers.....=  00
***Purged transfers.....=  00
***Last Message.........=  PGTF3101I Activity IA01500042
successfully transferred 52 records with remote node SNANODE
Request Queued successfully with TRANSNUM= IA01500042
```

FUSSENDT

You can use the FUSSENDT exec to queue a request to send a text file to a remote partner.

The FUSSENDT exec is identical to the FUSSEND exec, except that the Platform Server automatically adds the following configurations:

- TYPE=TEXT: Indicates that the Platform Server converts the data from EBCDIC to ASCII if the data is received by an ASCII based system.
- CRLF=YES: Indicates that the records must be delimited by CRLF (Carriage Return/Line Feed).

When sending data between TIBCO MFT Platform Server for z/OS nodes, the FUSSEND and FUSSENDT execs are identical.

The parameters used with the FUSSENDT exec are the same as those used with the FUSSEND exec. For more information about the parameters, see FUSSEND.

Example

The following example shows how to use the FUSSENDT exec to send a text file to a remote partner:

```
 FUSSENDT NODE=SNANODE LF=LOCAL.FILE1 RF=C:\FUSSEND.TXT INTERVAL=3 WAIT
SAY
```

Note: The `FUSSEND` exec and the `FUSSENDT` exec examples perform exactly the same transfer.

FUSSMEM

You can use the `FUSSMEM` exec to send multiple members of a PDS to a remote Platform Server node.

You can select which members of the PDS are sent by specifying a wildcard within the member name. The PDS file and optionally the member name are specified in the `LFILE` parameter. For example,

```
LFILE=MY.PDS(MEMBERS) RF=C:\PROD\ACCT\MEMBER.TXT
IPPORT=46464 IPADDR=127.0.0.1
```

If no member name is defined within the parentheses in the `LFILE` parameter, all members are selected. Up to 1000 members can be selected. If you select more than 1000 members, a message is displayed and no transfer is queued.

Wildcard Parameters

Fusion supports the following wildcard parameters:

- `*`: Represents a string of unspecified characters.
- `?`: Represents a single unspecified character.

You can use both of these parameters in a member name. For example:

```
LFILE=PROD.SOURCE(A??B*).
```

In this example, all members starting with `A` in position one and `B` in position four are sent.

Substitutable Parameter

You can use the `%MEM` substitutable parameter in the `RFILE` parameter. In this case, the Platform Server substitutes the member name for the `%MEM` parameter.

For example, when members ABC123 and XYZ456 are selected and RFILE=C:\prod\acct\%mem.asm is specified, the output is as follows:

```
RFILE=C:\prod\acct\ABC123.asm
RFILE=C:\prod\acct\XYZ456.asm
```

If the %MEM parameter is not defined in the RFILE parameter, the Platform Server adds the member name to the end of the RFILE parameter. If the RFILE parameter does not end in a backslash (\), the Platform Server adds a backslash before adding the member name to the end of the RFILE parameter.

For example:

```
FUSSMEM IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.PDS.FILE
RF=C:\%MEM.TXT
INTERVAL=3 SAY WAIT
```

The output of this example is as follows:

```
TRANSACTION=IA01500043 STATUS=ACTIVE   RECORD=000000000 BYTES=000000000
TRANSACTION=IA01500043 STATUS=COMPLETE RECORD=000000017 BYTES=000001360
***Fusion retcode.......=  0
***Fusion TransNum......=  IA01500043
***Completed transfers..=  01
***Successful transfers.=  01
***Failed transfers.....=  00
***Purged transfers.....=  00
***Last Message.........=  PGTF3101I Activity IA01500043 successfully
transferred 17 records with remote node 127.127.127.1
Request Queued successfully with TRANSNUM= IA01500043 for Member:
MEMBER1
```

Note: This output is received for each member of the PDS.

The TEST Parameter

When you use the TEST parameter in the FUSSMEM exec, the exec displays and tests the LFILE and RFILE parameters to be passed to the Platform Server instead of queuing the request. It is a good practice to add this parameter when creating new requests to ensure that the correct members are sent and the correct RFILE names are created. After you verify that all the information of the exec is correct, you can remove the TEST parameter, and schedule the file transfers.

For example,

```
FUSSMEM LF=MY.PDS RF=C:\ACCT\%MEM.TAX TEST NODE=NODE1
```

If two members are in the MY.PDS file, the output of this example is as follows:

```
LFILE=MY.PDS(NY2008)  RFILE=c:\acct\NY2008.tax
LFILE=MY.PDS(FED)     RFILE=c:\acct\FED.tax
```

Note: No transfers are queued when the TEST parameter is on the command line. After you validate that the parameters are correct, you can remove the TEST parameter and execute the FUSSMEM command.

The WAIT and MWAIT Parameters

You can use the WAIT and MWAIT operands in this exec. The Platform Server only supports to queue up to 40 transfers before a FUSWAIT command is executed for an address space.

```
FUSSMEM IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.PDS.FILE
RF=C:\%MEM.TXT
INTERVAL=3 SAY WAIT TEST
```

In this example, the FUSSMEM exec shows the remote file name that is resolved, but does not perform the transfer.

The output of this example is as follows:

```
LFILE=LOCAL.PDS.FILE(MEMBER1)  RFILE=C:\MEMBER1.txt
LFILE=LOCAL.PDS.FILE(MEMBER2)  RFILE=C:\MEMBER2.txt
LFILE=LOCAL.PDS.FILE(MEMBER3)  RFILE=C:\MEMBER3.txt
LFILE=LOCAL.PDS.FILE(MEMBER4)  RFILE=C:\MEMBER4.txt
```

FUSSMFIL

You can use the FUSSMFIL exec to queue transfers to send multiple files to a remote node.

You can use this exec to send multiple files to a remote system in one command. The FUSSMFIL exec uses the same parameters as the FUSSEND exec. This exec uses the LFILE parameter as a high-level prefix and scans the catalog for all files that match the high-level prefix. Then, it substitutes information of the LFILE parameter into the RFILE parameter, and queues a file transfer for each dataset that matches the high-level qualifier.

The Platform Server only supports to initiate up to 500 transfers on any single FUSSMFIL transfer. If more than 500 files match the LFILE prefix, an error message is displayed and no transfers is queued.

Note: You can use the WAIT and MWAIT operands in this exec. The Platform Server only supports to queue up to 40 transfers before a FUSWAIT command is executed for an address space.

Wildcard Parameter

The LFILE parameter accepts a single wildcard parameter, *.

You must use at least one full z/OS prefix before the first wildcard. For example, LF=PROD.* is acceptable, while LF=PROD* is illegal and will not be processed.

The following examples show how to use the wildcard parameter in the FUSSMFIL exec:

- LF=FUSION.* or LF=FUSION: Sends all files starting with FUSION.
- LF=FUSION.ABC* or LF=FUSION.ABC.*: Sends all files starting with FUSION.ABC.

Note: It is more efficient to put the wildcard character after the period than before the period. It is more efficient to use LF=FUSION.PROD.* than using LF=FUSION.PROD*. Because the first setting reads only the entries starting with FUSION.PROD in the catalog, while the second setting reads all entries starting with FUSION in the catalog to see whether a match on FUSION.PROD* exists.

Substitutable Parameters

Three substitutable parameters can be set with the FUSSMFIL exec. You must define one of these parameters within the RFILE parameter; otherwise, the request fails.

Parameter	Description
%LFILE	Substitutes the entire LFILE parameter. For example:

Parameter	Description
	`FUSSMFIL NODE=SNANODE LF=LOCAL.FILE.CRE* RF=C:\%LFILE INTERVAL=3 SAY WAIT` The output of this example is the same as that of the `FUSSEND` or `FUSRECV` exec. You get the output for each file that is transferred.
`%SUFFIX`	Substitutes the data after the high-level prefix defined. For example: `FUSSMFIL IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.FILE.CRE* RF=C:\ACCT.%SUFFIX INTERVAL=3 SAY WAIT` The output of this example is the same as that of the `FUSSEND` or `FUSRECV` exec. You get the output for each file that is transferred.
`%NOHLQ`	Substitutes all data except the high-level qualifier (HLQ). For example: `FUSSMFIL NODE=SNANODE LF=LOCAL.FILE.CRE* RF=C:\ACCT.%NOHLQ INTERVAL=3 SAY WAIT` The output of this example is the same as that of the `FUSSEND` or `FUSRECV` exec. You get the output for each file that is transferred.

In the following example, you can extract four files from the catalog:

`LF=PROD.TESTV*`

The following table shows the remote files for each `RF` command:

	RF=%LFILE	RF= ACCT.%SUFFIX	RF= ACCT.%NOHLO(s/b NOHLQ).bkup
PROD.TESTVB	PROD.TESTVB	ACCT.B	ACCT.TESTVB.BKUP
PROD.TESTVBA	PROD.TESTVBA	ACCT.BA	ACCT.TESTVBA.BKUP

	RF=%LFILE	**RF= ACCT.%SUFFIX**	**RF= ACCT.%NOHLO(s/b NOHLQ).bkup**
PROD.TESTVB2	PROD.TESTVB2	ACCT.B2	ACCT.TESTVB2.BKUP
PROD.TESTVB3	PROD.TESTVB3	ACCT.B3	ACCT.TESTVB3.BKUP

The TEST Parameter

When you use the TEST parameter in the FUSSMFIL exec, the exec displays and tests the LF and RF files to be passed to the Platform Server instead of queuing the request. In this way, you can ensure that only the correct files are queued before a transfer request is queued.

For example:

```
FUSSMFIL lf=prod.acct98.* rf=acct.%nohlq ipaddr=14.0.0.0 ipport=46464
test,
```

The response is as follows:

```
LFILE=PROD.ACCT98.PAYROLL     RFILE=acct.ACCT98.PAYROLL
LFILE=PROD.ACCT98.TAXES           RFILE=acct.ACCT98.TAXES
LFILE=PROD.ACCT98.TAXES.BKUP  RFILE=acct.ACCT98.TAXES.BKUP
```

Note: No transfers are queued when the TEST parameter is used on the command line. After you validate that the parameters in this exec are correct, you can remove the TEST parameter and execute the FUSSMFIL exec.

The following example queues a transfer to send multiple files to a remote node, substitutes the entire %LFILE parameter, and tests the local files (LF) and the remote files (RF) that are queued:

```
FUSSMFIL NODE=SNANODE LF=LOCAL.FILE.CRE* RF=C:\%LFILE INTERVAL=3 SAY
WAIT TEST
```

The output of this example is as follows:

```
LFILE=LOCAL.FILE.CREATE.FILE1  RFILE=C:\LOCAL.FILE.CREATE.FILE1
LFILE=LOCAL.FILE.CREATE.FILE2  RFILE=C:\LOCAL.FILE.CREATE.FILE2
```

```
LFILE=LOCAL.FILE.CREATE.FILE3  RFILE=C:\LOCAL.FILE.CREATE.FILE3
LFILE=LOCAL.FILE.CREATE.FILE4  RFILE=C:\LOCAL.FILE.CREATE.FILE4
```

The following example queues a transfer to send multiple files to a remote node, substitutes the data after the high-level prefix defined, and tests the local files (`LF`) and the remote files (`RF`) that are queued:

```
FUSSMFIL IPADDR=127.127.127.1 IPPORT=46464 LF=LOCAL.FILE.CRE*
RF=C:\ACCT.%SUFFIX INTERVAL=3 SAY WAIT TEST
```

The output of this example is as follows:

```
LFILE=LOCAL.FILE.CREATE.FILE1  RFILE=C:\acct.ATE.FILE1
LFILE=LOCAL.FILE.CREATE.FILE2  RFILE=C:\acct.ATE.FILE2
LFILE=LOCAL.FILE.CREATE.FILE3  RFILE=C:\acct.ATE.FILE3
```

FUSFTP

With the `FUSFTP` exec, users familiar with the FTP user interface can use the Platform Server. This exec uses standard FTP parameters and translates these parameters into Platform Server requests. The GET and PUT requests are translated into the Platform Server RECEIVE and SEND requests.

The format of the `FUSFTP` exec is as follows:

`FUSFTP ipname ipport`

The Platform Server supports FTP parameters in the same way as they are supported by FTP. For example, the `ASCII` parameter changes the transfer type to `A` (text), while the `BINARY` parameter changes the transfer type to `I` (image).

The Platform Server supports the following FTP parameters:

- `APPEND`
- `ASCII`
- `BINARY`
- `BYE`
- `CLOSE`

- DEBUG
- END
- GET
- HELP
- LOCSITE
- OPEN
- PASSWORD
- PUT
- QUIT
- RECV
- SEND
- SITE
- STATUS
- TRACE
- USER
- VERBOSE

In addition to the FTP parameters, the Platform Server also supports some optional parameters with which you can perform functions that are not typically supported by FTP.

The following table lists the optional parameters supported by the FUSFTP exec:

Parameter	Description
CONFIG	Defines the name of the Platform Server REXX configuration file from which you can get information about the Platform Server started task. **Note:** This parameter is required only when you want to communicate with a Platform Server started task other than the default Platform Server started task.
NODE	Informs the Platform Server that the remote Platform Server node is

Parameter	Description
	defined by a NODE definition.
IPNAME	Informs the Platform Server that the remote Platform Server node is actually an IP name.
IPPORT	Defines the IP Port of the remote Platform Server system.

The FUSFTP exec supports the Platform Server User Profile capability. If a remote user is not defined by the USER parameter, the default value *PROFILE which uses the Platform Server User Profile capability is used. For more information on Platform Server user profiles, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

The following example shows a FUSFTP exec connection:

```
------------------------------------------------------------------------
-
                              ISPF Command Shell

Enter TSO or Workstation commands below:



===> tso fusftp

No connection: Use OPEN command to get connection
To override current userid/password, use commands: user, pass
open 192.192.100.10
 will connect to ipaddr 192.192.100.10
ipport 5000
 using IPPORT 5000
ascii
 type is ASCII
verbose
 Verbose is ON
status
 Using default server config
 ipaddr is 192.192.100.10
 ipport is 5000
 Local user is TSOUSER
 Remote user is *PROFILE
 Type is ASCII
 Append is OFF
 Verbose is ON
 Trace is OFF
 Debug is OFF
put test.local.file1 test.remote.file2

 TRANSACTION=IB16500000 STATUS=ACTIVE   RECORD=000000000 BYTES=000000000

 TRANSACTION=IB16500000 STATUS=COMPLETE RECORD=000000010 BYTES=000000810

 ***Fusion retcode.......=  0

 ***Fusion TransNum......=  IB16500000

 ***Completed transfers..=  01

 ***Successful transfers.=  01

 ***Failed transfers.....=  00

 ***Purged transfers.....=  00

***Last Message.........=  PGTF3101I Activity IB16500000 successfully
transferred 10
                           records with remote node 192.192.100.10
 Request Queued successfully with TRANSNUM= IB16500000
bye
 ***FUSFTP exec now ending***
```

Note: The data that is displayed in bold font is the data entered by the user, the data that is displayed in normal font is the data returned by the `FUSFTP` exec.

Waiting for Transfer Requests to Complete

When you run a REXX exec to queue a request to the Platform Server address space, the request typically ends when the request is successfully queued. If you want to wait for the completion of the request, you can specify the Platform Server wait parameters.

The Platform Server supports two types of wait parameters:

- `MWAIT`: Adds the request to a list of requests to be waited on and returns control to the Platform Server REXX exec. When you are ready to wait for all of the requests on the list to complete, issue the FUSWAIT exec.

For more information of this parameter, see Waiting for Multiple Requests to Complete.

For more information of the FUSWAIT exec, see FUSWAIT.

> **Note:** You can use the FUSWCLR exec to clear out all requests on the MWAIT list. For more information of the FUSWCLR, see FUSWCLR.

- WAIT: waits and returns to the Platform Server REXX exec until the request is completed, either successfully or unsuccessfully.

  For more information of this parameter, see Waiting for a Single Request to Complete.

For best results, when you use the WAIT parameter, set the retry count to the default value of 1. In such cases, the request might not be inactive for a long period of time waiting for a retry. It is a good practice to use the default values of the DATE and TIME parameters so that the request is executed immediately.

The following table lists the parameters that can be specified on a WAIT request, or on the FUSWAIT REXX exec:

Parameter	Description
INTERVAL={*number_of_seconds* | 10}	Defines the number of seconds that the Platform Server REXX exec waits between status checks of the requests. The Platform Server REXX exec inquires on the transaction until it is completed successfully or unsuccessfully. After every check, the REXX exec waits the number of seconds defined in the INTERVAL parameter. Valid values are 1 - 9999 and the default value is 10. **Note:** If you do not specify this parameter, the default value is used. Setting this value too low can cause excessive checks on the request, while setting it too high can cause excessive waits between checks.
TIMEOUT={*number_of_*	Defines how long the REXX exec continues to check the status of

Parameter	Description
seconds | 300}	the requests. When the time you defined in this parameter expires, the Platform Server stops scanning for the request status. Based on the PURGE parameter, the Platform Server either purges all requests that are still on the queue, or returns directly without purging any requests. The valid values are 1 - 9999 and the default value is 300. **Note:** If you do not specify this parameter, the default value is used.
PURGE	Defines the Platform Server to purge all requests that are not completed on the queue when the timeout interval expires. **Note:** If this parameter is not specified, the Platform Server takes no action on the uncompleted requests when the timeout interval expires.
SAY	Defines the Platform Server REXX exec to write a status record each time after the Platform Server checks the status of a transfer. The status record includes the transaction number, the current status, and the current record and byte counts. **Note:** If this parameter is not specified, the status record is not displayed. If this parameter is specified for a TSO user, the record is sent to the terminal. If this parameter is specified on a batch mode, the record is sent to the DD statement, SYSTSPRT.

Waiting for a Single Transfer Request to Complete

To wait for a single transfer to complete, add the WAIT parameter to the REXX exec that queues the request.

The INTERVAL and TIMEOUT values use their default values unless otherwise specified. When the timeout expires and the PURGE parameter is specified, the uncompleted requests are purged from the queue. If you want to record status, specify the SAY parameter. For more information of the parameters of the WAIT request, see Waiting for Transfer Requests to Complete.

Examples

The following examples shows how to set the Platform Server to wait for a single transfer request to complete by using the WAIT parameter.

- Queue a request and wait for the request to be completed:

```
 FUSSEND IPADDR=190.190.190.100 LF=local.file RF=remote.file WAIT
```

  The output of this example is as follows:

```
***Fusion retcode.......=  0
***Fusion TransNum......=  IA21500000
***Completed transfers..=  01
***Successful transfers.=  01
***Failed transfers.....=  00
***Purged transfers.....=  00
***Last Message.........=  PGTF3101I Activity IA21500000
successfully
transferred 6403 records with remote node 190.190.190.100
Request Queued successfully with TRANSNUM= IA21500000
```

- Update the status on each scan of the file by adding the SAY parameter:

```
FUSSEND IPADDR=190.190.190.100 LF=local.file RF=remote.file WAIT
SAY
```

  The output of this example is as follows:

```
TRANSACTION=IA21500001 STATUS=ACTIVE   RECORD=000000000
BYTES=000000000
TRANSACTION=IA21500001 STATUS=ACTIVE   RECORD=000006403
BYTES=000518501
TRANSACTION=IA21500001 STATUS=COMPLETE RECORD=000006403
BYTES=000518501
***Fusion retcode.......=  0
***Fusion TransNum......=  IA21500001
***Completed transfers..=  01
***Successful transfers.=  01
```

```
***Failed transfers.....=  00
***Purged transfers.....=  00
***Last Message.........=  PGTF3101I Activity IA21500001
successfully transferred 6403 records with remote node
190.190.190.100
Request Queued successfully with TRANSNUM= IA21500001
```

Note: Any time a wait operation is performed either by specifying the WAIT parameter on a request or by executing the FUSWAIT exec, the Platform Server displays a return code as well as the numbers of completed, successful, unsuccessful, and purged transfers. If a transfer is not completed by the TIMEOUT interval and you do not specify the PURGE parameter, the transfer request is not listed in any of the summary totals.

Waiting for Multiple Requests to Complete

You can use the Platform Server REXX to queue up to 40 transfers and then wait for the transfer requests to complete. To add a request to the requests that the Platform Server waits for completion, add the MWAIT parameter to the exec that queued the transfer request.

For example:

```
FUSSEND IPADDR=190.190.190.100 LF=local.file RF=remote.file MWAIT
```

The FUSSEND exec returns 0 indicating that the request is queued successfully. But this does not mean the transfer request is completed successfully. You must issue the FUSWAIT exec to check the completion of the requests.

For more information of the FUSWAIT exec, see FUSWAIT.

Note: You can use the FUSWCLR exec to clear out all requests on the MWAIT list. For more information of the FUSWCLR exec, see FUSWCLR.

FUSWAIT

You can use the FUSWAIT exec to wait for 1 - 40 transfers that are queued with the MWAIT parameter to complete.

Required Parameter

No parameter is required by the FUSWAIT exec.

Optional Parameters

The following table lists the optional parameters of the FUSWAIT exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the FUSWAIT exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
INTERVAL	Defines the number of seconds between the status scans of a request. Default = 10 seconds. For more information, see Waiting for Transfer Requests to Complete.
PURGE	Defines the Platform Server to purge all requests that are not completed when the TIMEOUT expires. For more information, see Waiting for Transfer Requests to Complete.
SAY	Displays the updated status after each time the request status is checked. **Note:** The Platform Server only scans uncompleted transfers for status checks; if the transfers are completed, the Platform Server does not scan them for status checks. For more information, see Waiting for Transfer Requests to Complete.
TIMEOUT	Defines the number of seconds that the Platform Server continues to scan requests that are not completed. Default = 300 seconds. For more information, see Waiting for Transfer Requests to Complete.

Example

The following example shows how to use the FUSWAIT exec to wait for more than one transfers to complete.

```
FUSSEND IPADDR=190.190.190.100 LF=local.file RF=remote.file1 MWAIT
FUSSEND IPADDR=190.190.190.101 LF=local.file RF=remote.file2 MWAIT
FUSSEND IPADDR=190.190.190.102 LF=local.file RF=remote.file3 MWAIT
```

In this example, three transfers are queued and are ready to be waited on. The following command defines the Platform Server to wait on the three transfers to complete, scan the status in every five seconds, write status updates, stop scanning the requests after 300 seconds, and then purges the uncompleted requests.

```
FUSWAIT INTERVAL=5 SAY TIMEOUT=300 PURGE
```

The output of this example is as follows:

```
TRANSACTION=IA21500005 STATUS=ACTIVE   RECORD=000023072 BYTES=001868639
TRANSACTION=IA21500006 STATUS=ACTIVE   RECORD=000012242 BYTES=000991439
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500005 STATUS=COMPLETE RECORD=000025603 BYTES=002073701
TRANSACTION=IA21500006 STATUS=ACTIVE   RECORD=000025087 BYTES=002031839
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500006 STATUS=COMPLETE RECORD=000025603 BYTES=002073701
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TRANSACTION=IA21500007 STATUS=INACTIVE RECORD=000000000 BYTES=000000000
TIMEOUT EXPIRED WAITING FOR TRANSACTION TO COMPLETE
TRANSACTION IA21500007  PURGED FROM QUEUE DUE TO PURGE PARAMETER
***Fusion retcode.......=  4
***Total Transfers......=  03
***Completed transfers..=  02
***Successful transfers.=  02
***Failed transfers.....=  00
***Purged transfers.....=  01
```

```
FUSWAIT RC==>  4 Partial success of MFT Platform Server MWAIT
```

FUSWCLR

You can use the `FUSWCLR` exec to clear out all the Platform Server requests that are put on the Platform Server `MWAIT` list for this address space.

The Platform Server requests stay on the `MWAIT` list until one of the following situations occurs:

- You execute the `FUSWAIT` REXX exec.
- You execute the `FUSWCLR` REXX exec.
- The job step terminates.

This means that a TSO user can queue a request at 9:00 a.m. and specify the `MWAIT` parameter, and then checks the status through the `FUSWAIT` exec at 5:00 p.m. before leaving for the day.

The `FUSWCLR` exec does not have required and optional parameters.

Example

The following example clears out all the requests that have been put on the `MWAIT` list:

```
FUSWCLR
```

The output of this example is as follows:

```
FUSWCLR RC==> 0 WCLR completed successfully
```

If no transfers are specified with the `MWAIT` parameter, you get the following messages after issuing the `FUSWCLR` exec:

```
PGTE2206E FUSMWCLR failed - no transfers have been queued with MWAIT since the
step started FUSWCLR RC==> 8 No transfers on WAIT queue
```

Inquiring on Transfer Requests

You can use the `FUSINQ` REXX to get summary information on the Platform Server requests based on a variety of selection criteria.

This exec is useful for ad hoc inquiry of transfers. These requests can be submitted daily to get the status of the transfers that are completed within a certain time period, or with other similar characteristics.

In addition, you can add an asterisk (*) to some FUSINQ parameters to perform generic filtering. For more information of generic filtering, see Generic Filtering of Data.

FUSINQ

You can use the FUSINQ exec to inquire on the status of file transfers.

Required Parameters

This exec does not have required parameters.

Optional Parameters

The following table lists the optional parameters of the FUSINQ exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the FUSINQ exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
DAYS	Defines the number of days for how long the Platform Server scans the requests. The default value is 1. **Note:** This parameter is ignored if both the STARTDATE and ENDDATE parameters are provided.
DBYTE | DB	Defines the CyberFusion to display the current byte count.
DMSG | DM	Defines the CyberFusion to display the most recent message prefix.
ENDDATE | EDATE	Defines the last day on which the Platform Server scans the requests. The format of this parameter is YYYYDD. The default value is the

Parameter	Description
	date of today. You can also specify `TODAY` or `TOD` for the date of today, and `YESTERDAY` or `YES` for the date of yesterday.
`ENDTIME` | `ETIME`	Defines the last time at which the Platform Server scans the requests. The format of this parameter is `HHMM` (24-hour system). The default value is `2400`, which is the last minute of the day.
`EXCEPTIONS` | `EXC`	Defines whether exceptions are used as a selection criteria. When this operand is specified, records that are on the active queue are not extracted because they do not have a status of successful or unsuccessful. Valid values are: • `U`: Extracts only unsuccessful transfers. • `S`: Extracts only successful transfers. **Note:** By default, both successful and unsuccessful transfers are inquired.
`HISTORYQ` | `HQ`	Defines the type of records that are extracted. The Platform Server has a history file and a queue of active and inactive transfers. This parameter determines whether the history file, queue, or both are used to extract transfers. The default value is `B`, which indicates extracting data from both the history file and queue. Valid values are: • `H`: Indicates extracting data from the history file. • `Q`: Indicates extracting data from the queue. • `B`: Indicates extracting data from the history file and queue.
`LOCALIZE` | `FILE` | `LF`	Defines the name of the local file for a file transfer request of the Platform Server.

Parameter	Description
LOCALISER | LUSER	Defines the name of the user ID that queued the Platform Server request.
MAXXFER	Defines the maximum number of transfers returned by the FUSINQ exec. The default value is 500. The valid values are 1 - 10000.
PROCESS | PRO	Defines the Platform Server process name for the request.
REMOTESYS | RSYS	Defines the IPNAME, IPADDR, LUNAME, LIST, or NODE parameter that is requested when the request is initiated.
STARTDATE | SDATE	Defines the date on which the Platform Server starts to scan the requests. The format of this parameter is YYYYDDD. The default value is the date of today. You can also specify TODAY or TOD for the date of today, and YESTERDAY or YES for the date of yesterday.
STARTTIME | STIME	Defines the time at which the Platform Server starts to scan the requests. The format of this parameter is HHMM (24-hour system). The default value is 0000, which is first minute of the day.
TRANSNUM | TRN	Defines the transaction number associated with the request. The Platform Server creates a unique transaction number for all requests that are initiated either locally or remotely.
USERDATA | UDATA	Defines the user data for a Platform Server request. The user data is defined on a Platform Server queue request by adding the DESCRIPTION parameter to the transfer.

Note: For more information on inquiry date processing, see Inquiry Date Processing.

Request Inquiry Security

Typically, you can inquire on requests that you initiated. But only the Platform Server REXX or ISPF administrators can inquire on requests initiated by other users. You can use GLOBAL `BOSSID` parameter to determine a Platform Server administrator and restrict the users who can administer the Platform Server work queue.

By specifying a facility class name in the `BOSSID` parameter, you grant rights to administer the Platform Server work queue to the users that have access to this facility class.

If `ANY` is specified or the `BOSSID` parameter is not specified, you grant anyone the rights to access the work queue; all valid users have the same rights and privileges.

> **Note:** You might have to consult your mainframe security specialist for a facility that can be used to limit access to the Platform Server work queue.

Generic Filtering of Data

You can add an asterisk (*) to some `FUSINQ` parameters to perform generic filtering. When generic filtering is performed, the Platform Server only checks the data before the asterisk. If the data matches, the request is considered as matching the filter criteria.

The following parameters supports generic filtering of data:

- `LOCALFILE`
- `LOCALUSER`
- `PROCESS`
- `REMOTESYS` (except that an IP address cannot contain an asterisk)
- `TRANSNUM`
- `USERDATA`

Fusion Filtering Rules

The following rules apply to data being filtered by the `FUSINQ` parameters:

- If a parameter is not included, no filtering is done based on that parameter. This means that no requests are bypassed because of that parameter.
- If a parameter fails any of the filters defined by the `FUSINQ` parameters, the request is

bypassed. A request must pass all filters before it is reported by the FUSINQ command.

- The Platform Server typically translates all data to uppercase before filtering is performed.

Inquiry Date Processing Rules

You can enter the STARTDATE, ENDDATE, and DAYS parameters to limit the number of days that the Platform Server scans the requests. The Platform Server follows certain rules when calculating the inquiry date.

Note: For more information of these parameters, see FUSINQ.

The following rules are followed when the Platform Server these parameters (assume that the date of today is 2008200):

- If both the STARTDATE and ENDDATE parameters are specified, the Platform Server ignores the DAYS parameter.
- If the STARTDATE, ENDDATE, and DAYS parameters are not defined, the Platform Server sets the STARTDATE and ENDDATE parameters to the date of today.

  For example, FUSINQ is treated as FUSINQ SDATE=2008200 EDATE=2008200.
- If the STARTDATE and ENDDATE parameters are not specified and the DAYS parameter is specified, the Platform Server sets the ENDDATE parameter to the date of today and the STARTDATE parameter to the date that is computed by subtracting the number of days defined by the DAYS parameter from the date of today.

  For example, FUSINQ DAYS=5 is treated as FUSINQ SDATE=2008196 EDATE=2008200.
- If the STARTDATE parameter and the DAYS parameter are specified, but the ENDDATE parameter is not specified, the Platform Server sets the ENDDATE parameter to the date that is computed by adding the number of days that is defined by the DAYS parameter to the STARTDATE parameter.

  For example, FUSINQ SDATE=2008190 DAYS=5 is treated as FUSINQ SDATE=2008190 EDATE=2008194.
- If the STARTDATE parameter is not specified, but the ENDDATE parameter and the DAYS parameter are specified, the Platform Server sets the STARTDATE parameter to the date computed by subtracting the number of days defined by the DAYS parameter

from the ENDDATE parameter.

For example, FUSINQ EDATE=YES DAYS=5 is treated as FUSINQ SDATE=2008195 EDATE=2008199.

Examples of the FUSINQ Exec

The following examples of the FUSING exec show how to use this exec to inquire and report on requests.

Example 1

```
FUSINQ EXC=U DAYS=2
```

This command defines the Platform Server to display unsuccessful transfers for yesterday and today.

The output of this example is as follows:

```
**SELECTION CRITERIA**
       STARTDATE= 2008293
       ENDDATE= 2008294
       EXCEPTIONS= U
       number of transfers returned= 11
       2008.293 09:38:48 RA20500001 REXXPROC FAILED 14.0.0.0 IBMUSER
ACCT.TESTVB2
       2008.293 09:38:48 IA20500001 REXXPROC FAILED 14.0.0.0 IBMUSER
ACCT.TESTVB
       2008.293 13:18:33 IA20500007 REXXPROC FAILED 190.190.190.100 FUSUSER
ACCT.TESTVB
       2008.293 13:23:52 IA20500008 REXXPROC FAILED 190.190.190.100 FUSUSER
ACCT.TESTVB
       2008.293 13:29:23 IA20500009 REXXPROC FAILED 190.190.190.100 FUSUSER
ACCT.TESTVB
       2008.293 13:32:36 IA20500010 REXXPROC FAILED 190.190.190.100 FUSUSER
ACCT.TESTVB
       2008.293 14:00:24 IA20500011 REXXPROC FAILED 190.190.190.100 IBMUSER
ACCT.TESTVB
       2008.294 09:42:04 IA21500007 REXXPROC FAILED 190.190.190.100 IBMUSER
ACCT.TESTVB
       2008.294 09:50:29 IA21500010 REXXPROC FAILED 190.190.190.100 IBMUSER
ACCT.TESTVB
       retncode===> 0
```

The output includes the following information:

- Transfer completion dates for completed transfers and start dates for requests on the active queue
- Transfer completion times for completed transfers and start times for requests on the active queue
- Transaction number
- Process name
- Request status
- Remote system name or IP address
- Local user ID
- Local dataset name

Example 2

```
FUSINQ EDATE=TOD DAYS=2 or FUSINQ EDATE=TOD SDATE=YES
```

These two commands define the Platform Server to inquire on requests in the active queue and requests that are completed yesterday or today.

Example 3

```
FUSINQ TRN=IA18* UDATA=PRODACCT* PROCESS=SENDP1 HQ=H
```

This command defines the Platform Server to report on requests that meet the following selection criteria:

- Transaction number starts with `IA18`.
- User data starts with `PRODACCT`.
- Process name is `SENDP1`.
- Only requests that are completed (the history file) are checked.
- Because the `SDATE`, `EDATE`, and `DAYS` parameters are not defined, only requests that are completed today are scanned.

Example 4

```
FUSINQ LF=PROD.ACCT1* DAYS=10
```

This command defines the Platform Server to report on requests that meet the following selection criteria:

- Local file starts with `PROD.ACCT1`.
- Requests of the last 10 days are scanned.
- Active requests are scanned.

Example 5

```
FUSINQ LUSER=ACCTUSER SDATE=2008190 DAYS=20
```

This command defines the Platform Server to report on requests that meet the following selection criteria:

- User `ACCTUSER` is the local user (specifically, the user that submitted the request) on the Platform Server system.
- Start the inquiry on 2008190, and end the inquiry on 2008209.
- Active requests are scanned.

Performing Operator Control or Detailed Inquiry

You can use execs to perform operator control and detailed inquiry on transfers based on transaction numbers.

The following operator control functions are supported:

- FUSDEL: deletes a Platform Server request.
- FUSHOLD: holds a Platform Server request that is currently inactive.
- FUSREL: releases a Platform Server request that is in the hold state.
- FUSSEL: performs detailed inquiry on a request.
- FUSSUSP: suspends a request that is currently active.

These operator control functions use the Platform Server transaction number as the sole parameter for identifying the request. The `FUSDEL`, `FUSHOLD`, `FUSREL`, `FUSSEL`, and `FUSSUSP` execs only check the active queue for a request that matches the transaction number. The FUSSEL exec checks both the active queue and the history queue to find a match on the transaction number.

In addition, you can use the GLOBAL BOSSID parameter to restrict the users who can administer the Platform Server work queue. For more information of how to secure operator control, see Securing Operator Control.

Operator Control Security

Typically, you can perform operator control on requests that you initiated. Only the Platform Server REXX or ISPF administrators can perform operator control on requests initiated by other users. You can use the GLOBAL BOSSID parameter to determine a Platform Server administer and restrict the users who can administer the Platform Server work queue.

By specifying a facility class in the GLOBAL BOSSID parameter, you grant rights to administer the Platform Server work queue to the users that have access to this facility class.

If ANY is specified or the GLOBAL BOSSID parameter is not specified, you grant anyone the rights to access the work queue; all valid users have the same rights and privileges.

Note: You might have to consult your mainframe security specialist for a facility that can be used to limit access to the Platform Server work queue.

FUSDEL

You can use the FUDEL exec to delete a Platform Server request.

Required Parameter

The following table lists the required parameter of the FUSDEL exec:

Parameter	Description
TRANSN | TRN	Defines the fully qualified transaction number (10 bytes) associated with the request.

Optional Parameter

The following table lists the optional parameter of the `FUSDEL` exec:

Parameter	Description
`CONFIG` | `CFG`	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.

Return Codes

The return codes of the `FUSDEL` exec are as follows:

- A zero return code indicates that the Platform Server has marked the request for deletion.
- Non-zero return codes indicate one of the following situations:
    - The Platform Server cannot find the request to be deleted on the active queue.
    - The user is not authorized to delete the Platform Server request identified by the `TRANSNUM` parameter.
    - The request was already marked for deletion.
    - A communication error occurred when performing the task.

Example

The following example shows how to use the `FUSDEL` exec to delete a Platform Server request:

```
FUSDEL TRANSNUM=IA01500040
```

The output of this example is as follows:

```
retncode===> 0 DELETE completed successfully
```

If the transfer is already completed, the following message is displayed when you issue this exec:

```
retncode===> 4 TRANSNUM not found on ACTIVE Queue
```

FUSHOLD

You can use the FUSHOLD exec to hold a Platform Server request that is currently inactive.

Required Parameter

The following table lists the required parameter of the FUSHOLD exec:

Parameter	Description
TRANSN | TRN	Defines the fully qualified transaction number (10 bytes) associated with the request.

Optional Parameter

The following table lists the optional parameter of the FUSDEL exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.

Return Codes

The return codes of the FUSHOLD exec are as follows:

- A zero return code indicates that the Platform Server has put the request on the hold status.
- Non-zero return codes indicate one of the following situations:
  - The Platform Server cannot find the request to be held on the active queue.
  - The user is not authorized to hold the Platform Server request identified by TRANSNUM parameter.
  - The request was in a state that cannot be held, such as ACTIVE or DELETE PENDING.
  - A communication error occurred when performing the task.

Example

The following example shows how to use the FUSHOLD exec to hold a Platform Server request that is currently inactive:

```
FUSHOLD TRANSNUM=IA01500044
```

The output of the example is as follows:

```
retncode===> 0 HOLD completed successfully
```

FUSREL

You can use the FUSREL exec to release a Platform Server request that is in the held state, and allow it to be executed.

Required Parameter

The following table lists the required parameter of the FUSDEL exec:

Parameter	Description
TRANSN | TRN	Defines the fully qualified transaction number (10 bytes) associated with the request.

Optional Parameter

The following table lists the optional parameter of the FUSDEL exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.

Return Codes

The return codes of the FUSREL exec are as follows:

- A zero return code indicates that the Platform Server has released the request.
- Non-zero return codes indicate one of the following situations:
    - The Platform Server cannot find the request to be released on the active queue.
    - The user is not authorized to release the Platform Server request identified by `TRANSNUM` parameter.
    - The request was in a state that cannot be released, such as ACTIVE or DELETE PENDING.
    - A communication error occurred when performing the task.

Example

The following example shows how to use the `FUSREL` exec to release a Platform Server request that is in the held state:

```
FUSREL TRANSNUM=IA01500042
```

The output of the example is as follows:

```
retncode===> 0 RELEASE completed successfully
```

FUSSUSP

You can use the `FUSSUSP` exec to suspend an active Platform Server request, and put it in the held status.

Required Parameter

The following table lists the required parameter of the `FUSDEL` exec:

Parameter	Description
`TRANSN` | `TRN`	Defines the fully qualified transaction number (10 bytes) associated with the request.

Optional Parameter

The following table lists the optional parameter of the `FUSDEL` exec:

Parameter	Description
`CONFIG` | `CFG`	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.

Return Codes

The return codes of the `FUSDEL` exec are as follows:

- A zero return code indicates that the Platform Server has suspended the request.
- Non-zero return codes indicate one of the following situations:
  - The Platform Server cannot find the request to be suspended on the active queue.
  - The user is not authorized to suspend the Platform Server request identified by `TRANSNUM` parameter.
  - The request was in a state that cannot be suspended, such as INACTIVE, HOLD, or DELETE PENDING.
  - A communication error occurred when performing the task.

Example

The following example shows how to use the `FUSSUSP` exec to suspend an active Platform Server request:

```
FUSSUSP TRANSNUM=IA01500042
```

The output of the example is as follows:

```
retncode===> 0 SUSPEND completed successfully
```

FUSSEL

You can use the `FUSSEL` exec to perform detailed inquiry on a Platform Server request.

Required Parameter

The following table lists the required parameter of the FUSDEL exec:

Parameter	Description
TRANSN | TRN	Defines the fully qualified transaction number (10 bytes) associated with the request.

Optional Parameters

The following table lists the optional parameters of the FUSSEL exec:

Parameter	Description
CONFIG | CFG	Defines the configuration file used by the Platform Server exec. If this parameter is not defined, the Platform Server REXX uses the default configuration.
QUEUE	Defines the Platform Server to write all of the request fields back onto the REXX external data queue. User execs can then read the external data queue and report on it by using user defined REXX code. **Note:** This option is normally set only when the FUSSEL exec is called from a user REXX exec.
REPORT	Defines the Platform Server to create a report to display the request fields. You can specify REPORT to display the most important fields about a transfer, or specify REPORT=LONG to display all of the fields associated with a transfer request.

Return Codes

The return codes of the FUSSEL exec are as follows:

- A zero return code indicates that the Platform Server has found the request and displayed fields from the request.

- Non-zero return codes indicate one of the following situations:
    - The Platform Server cannot find the request to be selected on the active queue or the history file.
    - The user is not authorized to select the Platform Server request identified by the `TRANSNUM` parameter.
    - The request was in a state that cannot be selected, such as LOGGING.
    - A communication error occurred when performing the task.

Example

Member POSTPROC of the Platform Server EXECS library shows the following example of how the `FUSSEL` exec can be called with the `QUEUE` parameter from within a user REXX exec. This exec also shows how the user exec can process data in the REXX external data queue.

```
FUSSEL TRN=IA21500006
```

The output of the example is as follows:

```
retncode===> 0 PGTF3101I Activity IA21500006 successfully transferred
25603
 records with remote node 190.190.190.100
```

```
FUSSEL TRN=IA21500006 REPORT
```

The output of the example is as follows:

```
transnum............ =  IA21500006
remote.system.name.. =  190.190.190.100
dsn................. =  ACCT.TESTFILE
status.............. =  COMPLETE
Remote.file.name.... =  ACCT.TEST.FILE3
local.user.......... =  ACCTUSER
remote.userid........=  WINUSER1
transfer type....... =  SEND
status.............. =  COMPLETE
time.completed...... =  09:42:21
date.completed...... =  2008.294
```

```
process............. =  REXXPROC
last.message........ =  PGTF3101I Activity IA21500006 successfully
transferred 25603
                        records with node 190.190.190.100
record.count........ =  000000025603
byte.count.......... =  000002073701
retncode===> 0 PGTF3101I Activity IA21500006 successfully transferred
25603 records
             with remote node 190.190.190.100
```

```
FUSSEL TRN=IA21500006 REPORT=LONG
```

The output of the example is as follows:

```
transnum............ =  IA21500006
remote.system.name.. =  190.190.190.100
dsn................. =  ACCT.TESTFILE
Remote.file.name.... =  ACCT.TEST.FILE3
local.user.......... =  ACCTUSER
remote.userid........=  WINUSER1
recfm............... =  VB
transfer type....... =  SEND
maxrecl..............=  04100
effect.............. =  CREATE/REPLACE
status.............. =  COMPLETE
avail................=  IMMEDIATE
time.eligible....... =    :  :
date.eligible....... =      .
time.started........ =  09:41:59
date.started........ =  2008.294
time.completed...... =  09:42:21
date.completed...... =  2008.294
priority............ =
userdata............ =
chkintvl............ =  000000
chkpt.count......... =  000000
process............. =  REXXPROC
expdt............... =  2008324
retry.limit......... =  00001
tries............... =  00001
last.message........ =  PGTF3101I Activity IA21500006 successfully
transferred 25603
                                  records with remote node
190.190.190.100
```

```
xfer.type........... =  FILE
record.count........ =  000000025603
byte.count.......... =  000002073701
compressed.byte.count=  000000000000
logon.domain........ =
translation......... =  BINARY
crlf................ =  NONE
compress.type....... =  NONE
notify.type......... =
notify.user......... =
file.avail.......... =  IMMEDIATE
compress.ratio...... =
application.type.... =  ALL
status.............. =  COMPLETE
ip.port............. =  46464
retncode===> 0 PGTF3101I Activity IA21500006 successfully transferred
25603 records
                         with remote node 190.190.190.100
```

Inquiring on Platform Server Messages

You can use the FUSMSG exec to inquire on Platform Server messages.

For more information of the FUSMSG exec, see FUSMSG.

FUSMSG

You can use the FUSMSG exec to inquire on the text of a Platform Server message.

Required Parameter

The following table lists the required parameter of the FUSSEL exec:

Parameter	Description
MESSAGE | MSG | M	Defines the number of the Platform Server message that you want to display. You must enter at least four characters. The Platform Server searches for a match on the characters entered. If a match is found, the message text and the message explanation are displayed. If the message is not found, an error message is displayed.

Optional Parameter

This exec does not have optional parameter.

Return Codes

The return codes of the `FUSMSG` exec are as follows:

- A return code of `0` indicates that the Platform Server has successfully displayed the message.
- A return code of `4` indicates that the message was not found.
- A return code of `5` indicates that the FUSMSG DD statement was not defined.
- A return code of `8` indicates that an IO error occurred when accessing the FUSMSG VSAM file.
- Any other return code indicates an error occurred when processing the FUSMSG VSAM file.

Example

The following example shows how to use the `FUSMSG` exec to inquire on the text of a Platform Server message:

```
FUSMSG M=PGTF3108
```

The output of the example is as follows:

```
PGTF3108I xxxx RC=yyyy zzzzz Record Count=nnnn
```

The output message includes the transaction number for the transfer, the return code, the node, the name of the file being transferred, the number of records transferred, the transfer type (send or receive), and a description of the transfer.

Dataset Name Initiation (DNI)

TIBCO MFT Platform Server for z/OS provides the DNI interface from which you can initiate data transfer requests by simply creating a dataset. You do not have to issue any calls to Platform Server programs or APIs, and you do not have to run any batch jobs or submit any JCL. To use the DNI interface, you only have to configure the DNI interface when installing the Platform Server. The configuration can also be easily updated at anytime.

The following example shows how the Platform Server DNI works:

The Platform Server administrator sets up the following configuration in a file referenced by the DNICFG DD statement in the Platform Server startup JCL:

```
REQUEST=SEND
DSN=TESTDNI.FILE1
REMOTE_FILE=TESTDNI.NEW.FILENAME
NODE=NYNODE
FAIL_ACTION=RENAME
FAIL_NEWDSN=TESTDNI.FAIL.FILE1
GOOD_ACTION=RENAME
GOOD_NEWDSN=TESTDNI.GOOD.FILE1
```

The Platform Server scans the z/OS catalog for a dataset called `TESTDNI.FILE1`. When this dataset is found, the Platform Server waits until it is no longer allocated to a job. The Platform Server then initiates a SEND transfer to the NYNODE node. When the transfer is completed, the Platform Server checks the status of the transfer request. If the transfer is successful, the Platform Server renames the dataset to `TESTDNI.GOOD.FILE1`. If the transfer is unsuccessful, the Platform Server renames the dataset to `TESTDNI.FAIL.FILE1`.

This is a simplified example, because the Platform Server has a powerful set of substitutable parameters and wildcards that can be used to reduce the amount of configuration parameters that must be created.

DNI Security

When using DNI, you must consider two security issues: local security and remote security.

Local Security

Local security refers to security on the system where DNI is executing.

When the Platform Server is accessing files on requests initiated by DNI, the user associated with the Platform Server started task is used when accessing files locally. This means that the Platform Server started task must have authority to read, write, or alter all the datasets controlled by DNI.

Remote Security

Remote security refers to security on the system that DNI is communicating with.

When communicating with remote Platform Server nodes, a user ID and password is required for authentication before a request can be accepted. The user ID and password can be specified in the DNI configuration member by using the `REMOTE_USER` or (`RUSER`) and `REMOTE_PASS` or (`RPASS`) parameters. Alternatively, the Platform Server user profiles can be used to create the remote user ID and remote password combination. The local user ID that is used to select the correct user profile is the user ID associated with the Platform Server started task. See *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide* for more information on Platform Server user profiles.

Preparing for Using DNI

Before using the Platform Server DNI, you have to finish the prerequisite tasks.

- Create the DNIACT (DNI Active Queue) dataspace.
- Create the DNICFG dataset.
- Update the Platform Server startup JCL to include the DNICFG and DNIACT DD names.
- Configure the DNI GLOBAL parameters.

For more information on the prerequisite tasks, see *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

DNI Global Parameters

To use the Platform Server DNI, you must configure the following GLOBAL parameters:

Parameter	Description
`DNI_INTERVAL`	Defines the interval in minutes at which the Platform Server scans the

Parameter	Description
	catalog for new datasets.
`DNI_USERID`	Defines the user ID that is used for DNI processes. The default value is the started task user ID.

For more information about these parameters, see *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Configuring DNI

To initiate data transfers by using the DNI interface, you must configure the DNI interface.

A typical DNI process includes the following operations:

- Creating DNI config members
- Defining DNI parameters
- Activating DNI config members

The Platform Server DNI supports most of the configuration parameters supported by the Batch interface; the `EXPDT` and `WAIT` parameters are not supported. If the `EXPDT` or `WAIT` parameter is specified, the Platform Server displays an error message and the DNI configuration is not activated. DNI also supported some additional parameters that are not supported by the Batch interface but are required by the DNI interface because of the hands-off nature of DNI.

The DNI configuration parameters are typically placed in a PDS. Each member contains the necessary DNI parameters to describe the DSN template to be scanned in the z/OS catalog, the parameters required to perform a file transfer, and parameter required to perform postprocessing of the request. Because of the powerful set of substitutable parameters, one DNI configuration member can be used to send many different datasets to many different Platform Server nodes.

Configuration parameters are in one of the following types:

- Parameters supported in the same way as the Batch interface
- DNI Extensions to the Batch interface

Creating DNI Configuration Members

Before initiating data transfer requests from the Platform Server DNI, you must create DNI Config members for the transfer requests in the DNICFG dataset.

Each DNI configuration member contains the necessary DNI parameters to describe the DSN template to be scanned in the z/OS catalog, the parameters required to perform a file transfer, and the parameters required to perform postprocessing actions of the request. By using the substitutable parameters, one DNI configuration member can be used to send many different datasets to many different Platform Server nodes.

Rules for Creating DNI Configuration Members

Each DNI configuration member contains all of the parameters that are used to schedule requests for a single file template. When defining parameters, you must follow the specified rules.

Note: Individual parameters must be specified only once per member.

Parameters can be defined by using the following rules:

- An asterisk (*) in column 1 indicates a comment. All data, including the continuation flag, is ignored.
- Columns 1 - 71 can contain data. Parameters can start anywhere in columns 1 - 71.
- Column 72 is the continuation column. When this column is turned on, the data will be continued in column 1 of the next record. If this column is turned on, all data up to column 71 must be entered.
- Only one parameter is supported per record.
- Parameter values end at the first space or null. If you want to imbed a space in a parameter value, the entire parameter value must be enclosed in double quotation marks.

Examples of valid parameters:

```
REQUEST=SEND
```

```
REQUEST=“SEND”
```

Examples of continuation parameters:

```
REMOTE_FILE=C:\WINDOWS\APPLICAITON\-
TESTDATA.TXT
```

```
REMOTE_FILE="C:\WINDOWS\APPLICATION\-
       TEST DATA FOR ACCOUNTING."
```

Defining DNI Parameters

After creating a DNI configuration, you can define all the parameters that are used to schedule requests for a single file template.

The Platform Server DNI supports all the configuration parameters supported by the Batch interface except the `EXPDT` and `WAIT` parameters. If the `EXPDT` or `WAIT` parameter is specified, the Platform Server displays an error message and the DNI configuration is not activated.

DNI also supports some additional parameters that are not supported by the Batch interface but are required by the DNI interface because of the hands-off nature of DNI.

Configuration parameters are in one of the following types:

- Parameters supported in the same way as the Batch interface
- DNI Extensions to the Batch interface

Supported Batch Interface Parameters

The Platform Server DNI supports all the configuration parameters supported by the Batch interface except the `EXPDT` and `WAIT` parameters.

The following parameters are supported by the DNI interface. For a full description of the parameters see SYSIN Coding Parameters.

`ALLOC_DIR`, `ALLOC_PRI`, `ALLOC_SEC`, `ALLOC_TYPE`, `APPLICATION`, `AVAIL`, `BLOCKSIZE`, `CHKPT`, `COMPRESS`, `CRLF`, `CRCCHECK`, `DATACLASS`, `DATASET_TYPE`, `DATE`, `DCB_LIKE`, `DESCRIPTION`, `DSN`, `EFFECT`, `ENCRYPT`, `EMAIL_GOOD`, `EMAIL_FAIL`, `EXECPREPROC`, `EXECPOSTPROC`, `EXPTM`, `FILEERRORMAX`, `FILEERRORINT`, `FILE_EXPDT`, `FILE_RETPD`, `HOLD`, `IPADDR`, `IPNAME`, `IPPORT`, `LDISP`, `LIST`, `LPASS`, `LOCALCTFILE`, `LENGTH`, `LOCALLABEL`, `LOGON_DOMAIN`, `LUSER`, `MAINTAINBDW`, `MAINTAINRDW`, `MGTCLASS`, `MQ_FAIL`, `MQ_GOOD`, `NODECLASS`, `NODE`, `NOTIFY`, `NOTIFY_TYPE`, `PASSONLY`, `PERMIT_ACTNS`, `POST_ACTION`, `PRIORITY`, `RDISP`, `RECFM`, `REMOTECTFILE`, `REMOTE_`

FILE, REMOTELABEL, REMOTE_PASS, REMOTE_UNIT, REMOTE_USER, REMOVETRAIL, RETENTIONPRD, RETRY, RETRYINT, RSCOMPRESS, RSENCRYPT, RSHOST, RSMAXSPEED, RSPORT, RSPROTOCOL, SPACE, SPACERELEASE, STORCLASS, TCP_TIMEOUT, TEMPLATE, TIME, TLS, TRANS_TYPE, TRCLASSNAME, TRUNCATE, TYPE, UNIXPERM, UTF8BOM, VOLUME

DNI Parameter Extensions to the Batch Interface

The Platform Server DNI supports some parameters that are not supported by the Platform Server Batch interface. These parameters are extensions to the batch commands in that with those parameters, the Platform Server DNI supports features that are not supported by the Batch interface.

Valid Parameters in Both the Batch and DNI Interfaces

The following two parameters are valid in both the Batch and DNI interfaces, but are specified in different ways. These two parameters are typically defined on the batch PROCESS statement:

- REQUEST={SEND/RECEIVE}: This required parameter defines whether a request is a send or receive request.
- PROCESS={*process_name*}: This parameter defines the process name for a transfer request.

Unsupported Functions of the Batch Interface

You can define the following parameters to specify the actions that the Platform Server takes when a request is completed. Because DNI is a "lights out" interface, you must either delete or rename the dataset when a request is completed so that you can process additional files with the same dataset name. This postprocessing capability can be accomplished by the use of the postprocessing configuration parameters.

The DNI interface supports the following postprocessing configuration parameters. See DNI Postprocessing Configuration Parameters for details of these parameters.

- FAIL_ACTION
- FAIL_NEWDSN
- GOOD_ACTION
- GOOD_NEWDSN

These DNI parameters are used for postprocessing on DNI datasets and defines what DNI does when a transfer request is completed. It is important to understand what the word *completion* means to DNI. *completion* means that the request is purged from the Platform Server transaction queue. A request is purged from the Platform Server transaction queue under the following circumstances:

- A request is completed successfully and is purged from the queue.
- A request is completed with an unrecoverable error and is purged from the queue.
- A request is completed with a recoverable error, and is purged from the queue when the number of attempts exceeds the value defined in the `TRY` parameter.

> **Note:** When a DNI request is defined to a Platform Server distribution list through the `LIST` parameter, the request is considered to be failed even if only a single transfer in the distribution list fails. The DNI request is considered successful only when all requests in the distribution list are completed successfully.

DNI Postprocessing Configuration Parameters

You can specify the DNI postprocessing configuration parameters to define the actions to be performed when a transfer request is completed.

`FAIL_ACTION`**={RENAME | DELETE}**

This required parameter defines what action to take when a DNI transfer request fails and is purged from the Platform Server queue. The following two options are supported:

- `DELETE`: Indicates deleting the z/OS dataset.
- `RENAME`: Indicates renaming the z/OS dataset to the dataset that is defined by the `FAIL_NEWDSN` parameter.

> **Note:** If `FAIL_ACTION=RENAME` is specified, the `FAIL_NEWDSN` parameter is required.

`FAIL_NEWDSN`**={*new_z/OS_data_set_name*}**

When a DNI request fails and `FAIL_ACTION=RENAME` is specified, the local z/OS file is renamed to the file defined by the `FAIL_NEWDSN` parameter. This parameter supports the DSN substitutable parameters. This parameter is required when `FAIL_ACTION=RENAME` is specified.

For more information on the DNI substitutable parameters, see Substitutable Parameters.

For example,

```
FAIL_ACTION=RENAME
FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
```

GOOD_ACTION**={RENAME | DELETE}**

This required parameter defines what action to take when a DNI transfer request is completed successfully. The following two options are supported:

- DELETE: Indicates deleting the z/OS dataset.
- RENAME: Indicates renaming the z/OS dataset to the dataset that is defined by the GOOD_NEWDSN parameter.

ℹ **Note:** If GOOD_ACTION=RENAME is specified, the GOOD_NEWDSN parameter is required.

GOOD_NEWDSN**={*new_z/OS_dataset_name*}**

When a DNI request is completed successfully and GOOD_ACTION=RENAME is specified, the local z/OS file is renamed to the file defined by the GOOD_NEWDSN parameter. This parameter supports the DSN substitutable parameters. This parameter is a required when GOOD_ACTION=RENAME is specified.

See Substitutable Parameters for more information of the DNI substitutable parameters.

For example,

```
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
```

Note:
When a file is renamed by specifying FAIL_ACTION=RENAME or GOOD_ACTION=RENAME, the following operations are performed:

- If a dataset exists with a new DSN, the new DSN is deleted and uncataloged.
- The dataset is renamed to the name that is defined by the FAIL_NEWDSN or GOOD_NEWDSN parameter.
- The new dataset name is cataloged.

Defining Local File Wildcard Parameters

By defining the DNI wildcard parameters, you can use the DNI interface to its fullest. You can use the local file wildcard to define a single DNI configuration member for supporting many different datasets that can be sent or received from many different Platform Server nodes. The local wildcards are defined in the DSN parameter, or the LF or LFILE parameter.

For more details on the local file wildcard parameters, see Local File Wildcard Parameters.

Local File Wildcard Parameters

The following wildcard parameters are supported:

Wildcard Parameter	Description

%	This parameter indicates a match on a single character. If a character is not present, a match does not occur. You can include up to eight % wildcard characters in a single qualifier.
	For example,
	`DSN=TESTDNI.NODE%%.*`
	`DSN=TESTDNI.%FILE.%%XYZ`
	If you specify:

Wildcard Parameter	Description

	`DSN=TESTDNI.AB%%.SOURCE`
	DNI returns:
	`TESTDNI.AB12.SOURCE`
	`TESTDNI.ABCD.SOURCE`
	DNI does not return:
	`TESTDNI.AB1.SOURCE`

*	This parameter indicates a match on any data within the qualifier. The asterisk (*) can be preceded by other characters, and followed by additional qualifiers. You cannot specify asterisk as the first qualifier in the z/OS dataset name.
	For example,
	`DSN=TESTDNI.*.SOURCE`
	`LF=PRODDNI.SOURCE.*`
	If you specify:
	`DSN=TESTDNI.AB*.SOURCE`
	DNI returns:
	`TESTDNI.ABC123.SOURCE`
	`TESTDNI.AB.SOURCE`
	DNI does not return:
	`TESTDNI.AA123.SOURCE`

Wildcard Parameter	Description

**	A double asterisk (**) indicates a match on dataset name from this point on. The double asterisk cannot be the first character in the dataset name. It must be the only data in the dataset qualifier and must be the last qualifier in the file name. You cannot add any characters following the double asterisk and you must use it behind a period.
	Valid examples:
	`DSN=TESTDNI.SOURCE.**`
	`LF=TESTDNI.NYNODE.TEXT.**`
	Invalid example:
	`DSN=TESTDNI.**.SOURCE`
	If you specify:
	`DSN=TESTDNI.ABC.**`
	DNI returns:
	`TESTDNI.ABC.TAX.DATA.2008`
	`TESTDNI.ABC.SOURCE`
	DNI does not return:
	`TESTDNI.ABC123.TAX.DATA.2008`

Examples

The following examples show how to define local file wildcards in the DSN parameter.

Example 1:

```
DSN=TESTDNI.AB%%123.SOURCE
```

DNI returns:

```
TESTDNI.AB12123.SOURCE
TESTDNI.ABCD123.SOURCE
```

DNI does not return:

```
TESTDNI.ABCD123.SOURCE
```

Example 2:

```
DSN=TESTDNI.ABC.*
```

DNI returns:

```
TESTDNI.ABC.SOURCE
TESTDNI.ABC.ACCT
```

DNI does not return:

```
TESTDNI.ABC.SOURCE.DATA
```

Example 3:

```
DSN=TESTDNI.AB*.SOURCE
```

DNI returns:

```
TESTDNI.ABC123.SOURCE
TESTDNI.AB.SOURCE
```

DNI does not return:

```
TESTDNI.AA123.SOURCE
```

Example 4:

```
DSN=TESTDNI.AB*.**
```

DNI returns:

```
TESTDNI.ABC123.TAX.DATA.2008
TESTDNI.AB.TAX.DATA.2008
TESTDNI.AB.SOURCE
```

DNI does not return:

```
TESTDNI.AA123.TAX.DATA
```

Additional Parameters

The following two additional parameters can be used in the DSN field to override the `NODE` or `IPNAME` parameter in the request that is queued.

> **Note:** When the `%IPNAME` or `%NODE` parameter is specified in the DNI configuration, DNI substitutes an asterisk wildcard (*) for the parameter when scanning the z/OS catalog for a dataset match.

Parameter	Description
`%IPNAME`	Overrides the `IPNAME` parameter in the request that is queued. Because z/OS only supports 8-digit qualifiers, the `IPNAME` parameter is limited to 8 characters.
`%NODE`	Overrides the node name in the request that is queued. If the node is not defined to a Platform Server, this parameter is used as the SNA `LUNAME`. For example: `DSN=TESTDNI.%NODE.TEXT.DATA` In this example, DNI internally changes this parameter to `TESTDNI.*.TEXT.DATA`. Assume that the catalog scan returns the following datasets:

Parameter	Description

	`TESTDNI.NYNODE.TEXT.DATA`
	`TESTDNI.CHICAGO.TEXT.DATA`
	`TESTDNI.LA.TEXT.DATA`
	DNI will queue the following three transfer requests:
	• Request 1 is with the `TESTDNI.NYNODE.TEXT.DATA` local file to the NYNODE node.
	• Request 2 is with the `TESTDNI.CHICAGO.TEXT.DATA` local file to the CHICAGO node.
	• Request 3 is with the `TESTDNI.LA.TEXT.DATA` local file to the LA node.

The `%IPNAME` processing is identical to that of the `%NODE` parameter, except that instead of overriding the `NODE` parameter, the `IPNAME` parameter is overridden.

Defining MQ Support

The DNI interface supports MQ (Message Queues).

To define a MQ as a source DNI file, you must specify the `DSN` or `LF` parameter in the following way:

```
DSN=$MQ:queue_manager_name:queue_name
```

Where, `$MQ` indicates the file defined is a MQ; *queue_manager_name* defines the name of the queue manager and can contain 1 - 4 characters; and *queue_name* defines the name of the queue and can contain a maximum of 64 characters.

For example:

```
 DSN=$MQ:MQ1:FUSION.MESSAGE.QUEUE
```

In this example, the queue manager name is `MQ1`, and the queue name is `FUSION.MESSAGE.QUEUE`.

When a MQ is defined to DNI, the `FAIL_ACTION` and `GOOD_ACTION` parameters must not be defined. You cannot perform any postprocessing actions on MQs.

You can define the following additional parameter when using MQs under DNI:

`MQ_DEPTH`: defines the minimum number of records that must be in an MQ before DNI initiates a request to process the data. The value of this parameter can be `0` - `999,999`. If this parameter is set to `0` or `1`, the Platform Server initiates a task to process the data as soon as the data is written into the message queue. When this parameter is set to any other values, the Platform Server initiates a task to process data only when the MQ contains at least the number of records defined by this parameter.

Defining Substitutable Parameters

Because you can define the local file name with wildcards, it is not always possible to predefine some fields. To solve this issue, DNI supports substitutable parameters.

Based on the properties of the file transfer request, you can use substitutable parameters to override the contents in the following three fields:

`REMOTE_FILE`, `RFILE`, or `RF`

`FAIL_NEWDSN`

`GOOD_NEWDSN`

When a substitution is performed, the rules and regulations of the target system must be adhered to. The `FAIL_NEWDSN` and `GOOD_NEWDSN` parameters contain the names that the local file is renamed to when a request is completed. Therefore, these files must conform to the following z/OS file conventions:

- Each qualifier in a z/OS file must contain 1 - 8 characters. The qualifiers must not start with a number.
- The names of z/OS datasets cannot exceed 44 characters. If the dataset name exceeds 44 characters, the Platform Server DNI displays an error message and the request is not queued to DNI.
- When a Platform Server is communicating with another version of Platform Server for z/OS, the remote dataset must be 44 digits or less. Because the Platform Server does not know the platform of the target system, a request with a long dataset name will be rejected only when the transfer is initiated.

> **Note:** When defining the `GOOD_NEWDSN` and `FAIL_NEWDSN` parameters, the file names defined for these parameters must not start with the Local File Name. Otherwise, the renamed file may be transferred again. See the example below.

Example of an Invalid Definition

The following example is an invalid definition:

```
DSN=PRODDNI.ACCT.TAX*.**
GOOD_ACTION=RENAME
GOOD_NEWDSN=%LFILE.GOOD
```

Assume that the `PRODDNI.ACCT.TAX2008.NYBRANCH` dataset is created. When the transfer is completed successfully, the file is renamed to `PRODDNI.ACCT.TAX2008.NYBRANCH.GOOD`. The file is then retrieved from the catalog by the Platform Server, and I renamed to `PRODDNI.ACCT.TAX2008.NYBRANCH.GOOD.GOOD` when the transfer is completed successfully. The Platform Server repeats this operation until the file name exceeds 44 bytes and DNI recognizes the error.

The correct way to define the files is as follows:

```
DSN=PRODDNI.ACCT.TAX*.**
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
```

When the transfer is completed successfully, the file is renamed to `PRODDNI.GOOD.ACCT.TAX2008.NYBRANCH`.

Another way to rename the dataset is to use a new high-level qualifier:

```
DSN=PRODDNI.ACCT.TAX*.**
GOOD_ACTION=RENAME
GOOD_NEWDSN=GOODDNI.%NOHLQ
FAIL_ACTION=RENAME
FAIL_NEWDSN=FAILDNI.%NOHLQ
```

If the transfer is successful, the file is renamed to `GOODDNI.ACCT.TAX2008.NYBRANCH`. If the transfer is unsuccessful, the file is renamed to `FAILDNI.ACCT.TAX2008.NYBRANCH`.

Substitutable Parameters

The following table lists the substitutable parameters supported by the DNI interface:

Parameter	Description	Example
%ACB	Defines the name of the ACB that the Platform Server opens when the Platform Server SNA interface is initialized. It is defined by the Platform Server GLOBAL APPC-APPLID parameter. The value of this parameter can contain 1 - 8 characters. If the APPC-APPLID parameter is not defined, no data is substituted for this parameter.	APPC-APPLID=FUSNACB REMOTE_FILE=c:\temp\%ACB.data The remote file is changed to REMOTE_FILE=c:\temp\FUSNACB.data.
%FILE	Defines the Platform Server to substitute the file name. On a USS file, the file name is the text after the last slash. For an z/OS file, the file name is the same as the z/OS file name and returns the same result as the %LFILE token.	Example 1: RF=/usr/data/%FILE If the local file name is /tmp/acct/taxdata.txt, the remote file name is translated to RF=/usr/data/taxdata.txt. Example 2: RF=/usr/data/%FILE If the local file name is PROD.ACCT.DATA.TAXDATA.TXT, the remote file name is translated to RF=/usr/data/PROD.ACCT.DATA.TAXDATA.TXT.

Parameter	Description	Example
`%GDATE`	Defines the Platform Server to substitute the Gregorian date at the time the DNI request is to be queued. The format of the date is YYMMDD. **Note:** If you want to include the century, use the `%GDATEC` parameter.	`GOOD_NEWDSN=TESTDNI.D%GDATE.FILE1` If the current date is December 31, 2008, the file name is changed to `GOOD_NEWDSN=TESTDNI.D051231.FILE1`.
`%GDATEC`	Defines the Platform Server to substitute the Gregorian date at the time the DNI request is to be queued. The format of the date is CCYYMMDD.	`REMOTE_FILE=C:\TESTDNI\ACCT\%GDATEC.tax` If the current date is December 31, 2008, the file name is changed to `REMOTE_FILE=C:\TESTDNI\ACCT\20081231.tax`.
`%JDATE`	Defines the Platform Server to substitute the Julian date at the time the DNI request is to be queued. The format of the date is YYDDD.	`REMOTE_FILE=C:\TESTDNI\ACCT\%JDATE.tax` If the current date is December 31, 2008 (2008366), the file name is changed to `REMOTE_FILE=C:\TESTDNI\ACCT\08366.tax`.

Parameter	Description	Example
	Note: If you want to include the century, use the `%JDATEC` parameter.	
`%JDATEC`	Defines the Platform Server to substitute the Julian date at the time the DNI request is to be queued. The format of the date is CCYYDDD.	`REMOTE_FILE=C:\TESTDNI\ACCT\%JDATEC.tax` If the current date is December 31, 2008 (2008366), the file name is changed to `REMOTE_FILE=C:\TESTDNI\ACCT\2008366.tax`.
`%JOBN`	Defines the Platform Server to substitute the started task name for this parameter. The started task name can contain 1 - 8 characters.	`REMOTE_FILE=C:\TESTDNI\ACCT\%JOBN.source` If the Platform Server started task name is `FUSION`, the file name is changed to `REMOTE_FILE=C:\TESTDNI\ACCT\FUSION.source`.
`%LFILE`	Defines the Platform Server to substitute the current local file (or `LF` or `DSN`) name. This name can be up to 44 characters in length.	`REMOTE_FILE=C:\PRODDNI\%LFILE` If the local file name for the DNI task is `PRODDNI.ACCT.TAX.DATA`, the file name is changed to `REMOTE_FILE=C:\PRODDNI\PRODDNI.ACCT.TAX.DATA`.
`%LLQ`	`LLQ` stands for low-level qualifier. This	`REMOTE_FILE=C:\PRODDNI\acct\tax2008.%LLQ` If the local file name for the DNI task is

Parameter	Description	Example
	parameter defines the Platform Server to substitute the last qualifier in the current local file. This qualifier can contain 1 - 8 characters.	`PRODDNI.ACCT.TAX.DATA`, the file name is changed to `REMOTE_FILE= C:\PRODDNI\acct\tax2008.DATA`.
`%NOHLQ`	`NOHLQ` stands for no high-level qualifier. This parameter defines the Platform Server to strip off the high-level qualifier. The local file name following the high-level qualifier is substituted for this parameter.	`REMOTE_FILE=C:\acct\%NOHLQ` If the local file name for the DNI task is `PRODDNI.ACCT.TAX.DATA`, the file name is changed to `REMOTE_FILE=C:\acct\ACCT.TAX.DATA`.
`%PROC`	Defines the Platform Server to substitute the process name defined by the DNI config PROCESS statement. This parameter can contain 0 - 8 characters.	`REMOTE_FILE=C:\acct\%PROC.dat` If `PROCESS=ABC123` is specified in the `DNICFG` parameter member, the file name is changed to `REMOTE_FILE=C:\acct\ABC123.dat`.
`%Q01%` - `Q22`	`Q` stands for qualifier. A 44-byte	`REMOTE_FILE=PRODDNI.%Q02.%Q03.%Q06.%Q05.%Q04` If the DNI file that matches the DSN wildcard parameter is `TESTDNI.NY.ACCT.TEXT.TAX.Y2008`, the

Parameter	Description	Example
	z/OS dataset can have up to 22 qualifiers, although most datasets typically have far fewer than 22 qualifiers. A qualifier can contain 1 - 8 characters. The first character in the local file is %Q01. The qualifier counter is updated each time a period is encountered. To specify the high-level qualifier, simply specify %Q01. If no data is present for a qualifier, the qualifier is specified as the qualifier name by default. If you specify qualifier Q22 and qualifier Q22 does not exist in the Platform Server, the value Q22 is used in qualifier substitution.	qualifiers are as follows: - %Q01 TESTDNI - %Q02 NY - %Q03 ACCT - %Q04 TEXT - %Q05 TAX - %Q06 Y2008 **Note:** Qualifiers %Q07 - %Q22 are not defined and as a result substitute the values Q07 - Q22. In this case, the file name is changed to REMOTE_FILE=PRODDNI.NY.ACCT.Y2008.TAX.TEXT.
%SYSID	Defines the Platform Server to substitute the z/OS SYSID. This field can	REMOTE_FILE=PRODDNI.%SYSID.BACKUP.DATA If the z/OS SYSID is SYSA, the file name is changed to REMOTE_FILE=PRODDNI.SYSA.BACKUP.DATA.

Parameter	Description	Example
	contain 1 - 4 characters.	
%TIME	Defines the Platform Server to substitute the time when the DNI request is queued. The format of the time is HHMMSS.	GOOD_NEWDSN=PRODDNI.T%TIME.BACKUP.DATA If the DNI request was queued at 11:15:23, the file name is changed to GOOD_ NEWDSN=PRODDNI.T111523.BACKUP.DATA.
%UDATA	Defines the fusion to substitute the user data defined by the DNI configuration DESCRIPTION statement. This parameter can contain 0 - 25 characters.	REMOTE_FILE=C:\acct\%UDATA.dat If DESCRIPTION=ABCDEFGHIJ0123456789 is specified in the DNICFG parameter member, the file name is changed to REMOTE_ FILE=C:\acct\ABCDEFGHIJ0123456789.dat.

Activating DNI Members Automatically

You can add the DNI members to the member DNISTART of the DNICFG ddname to activate those DNI members at the Platform Server startup.

Multiple members can be defined on a single line. Members are delimited by a comma and lines are terminated by a space. An asterisk (*) in column one indicates a comment.

For example:

```
DNI1,DNI2
DNI3,DNITEST,
DNI
```

This example activates members DNI1, DNI2, DNI3, DNITEST, and DNI at Platform Server initialization.

Defining DNI Operator Commands

You can use DNI supported operator commands to control data transfer requests.

The Platform Server DNI supports the following operator commands:

- DNIENA,member
- DNIDISA,member
- DNIDISP,dni transaction number
- DNIPURGE,dni transaction number

The commands must be issued in the `F FUSION,`*`command_name`*`,`*`command_processor`* format. Where, *command_name* specifies the DNI command, and *command_processor* specifies the operand to be passed to the DNI command processor if required.

Enabling a DNI Configuration Member

You can use the `DNIENA` operator command to enable a DNI definition member.

The format of this command is `DNIENA,`*`member`*. The member to be enabled must be specified on the operator command.

Disabling a DNI Configuration Member

You can use the `DNIDISA` operator command to disable a DNI definition member that is activated at DNI initialization or through the `DNIENA` operator command.

The format of this command is `DNIDISA,`*`member`*. The member name to be disabled must be specified on the operator command.

Displaying DNI Requests

You can use the `DNIDISP` operator command to display DNI requests information.

The format of this command is `DNIDISP,`*`dni_transaction_number`*. Where, *dni_transaction_number* is optional on the `DNIDISP` command. If *dni_transaction_number* is not specified, a summary list of all requests on the DNIACT queue is displayed. If *dni_transaction_number* is specified, detailed information of that request is displayed. Remember that a record is on the DNIACT queue from the time that the request is queued until the requests are completed.

> **Note:** *dni_transaction_number* specified must be an 8-digit DNI transaction number, rather than a 10-digit transaction number.

For example:

```
F FUSION,DNIDISP
PGTE2650I DNI TRN=61600014  DSN=TESTDNI.TYLER.TEXT.DNIFILE1
PGTE2650I DNI TRN=61600015  DSN=TESTDNI.TYLER.TEXT.DNIFILE2

F FUSION,DNIDISP,61600014
```

The output of this example is as follows:

```
PGTE2651I DNI TRN......: 61600014
PGTE2651I LOCAL DSN....: TESTDNI.TYLER.TEXT.DNIFILE1
PGTE2651I VOLUME.......: TCB100
PGTE2651I QUEUE TRN....: I616900014
PGTE2651I PROCESS......: DNISEND
PGTE2651I GOOD ACTION..: RENAME
PGTE2651I GOOD NEWDSN..: TESTDNI.GOOD.TYLER.TEXT.DNIFILE1
PGTE2651I FAIL ACTION..: RENAME
PGTE2651I FAIL NEWDSN..: TESTDNI.FAIL.TYLER.TEXT.DNIFILE1
PGTE2651I #TRN QUEUED..:1
PGTE2651I #TRN GOOD....:0
PGTE2651I #TRN FAILED..:0
```

Purging a Request

You can use the `DNIPURGE` operator command to purge a request from an active DNI queue.

The format of this command is `DNIPURGE,`*`dni_transaction_number`*. The 8-digit DNI transaction number must be issued on the command line. The DNI transaction number is a subset of the Platform Server transaction number and can be displayed by using the `DNIDISP` operator command.

Suggestions for DNI Dataset Names

For best results, you can create a new high-level qualifier (HLQ) and use it solely for the Platform Server DNI. Although the Platform Server can utilize high-level qualifiers that are shared with other systems, there exists the potential for files to inadvertently be created with a name that matches the DNI DSN qualifier.

For example, if you want to send the `ACCT.CUSTDATA.CUST1` and `ACCT.CUSTDATA.CUST2` files to remote nodes, you can specify `DSN=ACCT.CUSTDATA.CUST*`; if another user creates the `ACCT.CUSTDATA.CUSTLIST` file, this qualifier matches the DSN qualifier specified and a transfer will be queued as a result.

This example shows how a problem might occur. Wildcard parameter `*`, `%`, and `**` can be combined to form complicated templates, which must be understood by all users creating datasets. This is why it is a good idea to reserve an HLQ for DNI.

It is a good practice to use a qualifier that includes the words GOOD and FAIL (or a similar representation) just after the HLQ. In such cases, it is easy for a user to see whether any requests failed by performing a dataset scan through ISPF 3.4 or LISTCAT on `HLQ.FAIL`.

Examples of Using DNI

Because of the flexible nature of DNI, it can be used in many ways. Your environment dictates how DNI can be most effectively used at your site. The following examples show how the Platform Server DNI can be used in some different scenarios.

Defining DNI for Each Node in Binary and Text Transfers

In this example, you have to define text and binary transfers for each remote Platform Server system. When a transfer is completed, it is renamed to indicate whether it works or fails.

Sample DNI definitions are as follows:

`NODE1TXT`

```
REQUEST=SEND
DSN=PRODDNI.NODE1TXT.**
REMOTE_FILE=C:\PRODDNI\%NOHLQ
TYPE=TEXT
IPNAME=NYOFFICE
```

```
TRY=5
FAIL_ACTION=RENAME
FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
```

`NODE1BIN`

```
REQUEST=SEND
DSN=PRODDNI.NODE1BIN.**
REMOTE_FILE=C:\PRODDNI\%NOHLQ
TYPE=BINARY
IPNAME=NYOFFICE
TRY=5
FAIL_ACTION=RENAME
FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
```

If you have twenty different remote Platform Server systems, you may have to create forty different DNI definitions (twenty for text and twenty for binary).

Using the `%NODE` and `%IPNAME` Substitutable Parameters

In this example, you have to define text and binary transfers for the Platform Server system. Take special care when creating file names because components of the file name will be used as the Platform Server node name or IP name.

When a transfer is completed, it is renamed to indicate whether it works or fails.

> **Note: Note:** When using this method, you must utilize the Platform Server user profiles to define the remote user ID and password combination.

Sample DNI definitions are as follows:

```
TEXT
----
REQUEST=SEND
DSN=PRODDNI.%NODE.TEXT
REMOTE_FILE=C:\PRODDNI\%NOHLQ
TYPE=TEXT
NODE=XXX
```

```
TRY=5
FAIL_ACTION=RENAME
FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ

BINARY
------
REQUEST=SEND
DSN=PRODDNI.%NODE.BINARY
REMOTE_FILE=C:\PRODDNI\%NOHLQ
TYPE=BINARY
NODE=XXX
TRY=5
FAIL_ACTION=RENAME
FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
GOOD_ACTION=RENAME
GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
```

These two definitions support an unlimited number of remote Platform Server partners. The second qualifier of the dataset name is used as the Platform Server node name. This node name must be configured as a Platform Server node. See *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide* for more information on configuring Platform Server NODE definitions.

In this example, you can substitute the `%IPNAME` parameter for the `%NODE` parameter. If you use the `%IPNAME` parameter, you do not have to create any Platform Server NODE definitions.

Using the Email Related Parameters

Some transfer requests require an action to be performed when data is sent to a remote location. Using the `EMAIL_FAIL` and `EMAIL_GOOD` parameters, you can send a message to two different email addresses, based on whether the request is successful or unsuccessful.

Sample DNI definitions are as follows:

```
TEXT
----
       REQUEST=SEND
       DSN=PRODDNI.ABC.COMPANY.DATA
       REMOTE_FILE=C:\PRODDNI\%NOHLQ
       TYPE=TEXT
       IPNAME=ABC.COMPANY.COM
       TRY=5
       FAIL_ACTION=RENAME
       FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
       GOOD_ACTION=RENAME
       GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
       EMAIL_GOOD=theiruser@abc.company.com
       EMAIL_FAIL=techsup@mycompany.com
```

In this example, if the request is successful, the dataset `PRODDNI.ABC.COMPANY.DATA` will be renamed to `PRODDNI.GOOD.ABC.COMPANY.DATA` and an email will be sent to `theiruser@abc.company.com`.

If the request is unsuccessful, the dataset `PRODDNI.ABC.COMPANY.DATA` will be renamed to `PRODDNI.FAIL.ABC.COMPANY.DATA` and an email will be sent to `techsup@mycompany.com`.

> **Note: Note:** In this example, you can tell one user that a file has arrived and another user that the request has failed. If some users must perform some actions when data is received, you can use the `EMAIL_GOOD` parameter to notify them that the request was successful. You can also use the `EMAIL_FAIL` parameter to notify technical support if a request has failed.

Using the LIST Function

You can use Fusion to send a single file to a list of recipients, which is called the distribution list. The distribution list must be defined to the Platform Server before you initiate a request to that list. A distribution list can consist of different Platform Server nodes, SNA LUnames, IP addresses, and IP names. See *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide* for more information about distribution lists, which is also called multi-casting.

First, you must define a distribution list. Sample DNI definitions are as follows:

```
LIST1
 -----
       TYPE=LIST
       NODE=SNANODE,TCPNODE
       IPADDR=127.128.129.1
       IPNAME=yourhost.companya.com
       IPPORT=2500
       IPADDR=127.128.129.2
```

You can set up a DNI configuration member that consists of the following parameters:

```
REQUEST=SEND
       DSN=PRODDNI.ACCT.DATA
       REMOTE_FILE=C:\PRODDNI\%NOHLQ
       TYPE=TEXT
       LIST=LIST1
       TRY=5
       FAIL_ACTION=RENAME
       FAIL_NEWDSN=%Q01.FAIL.%NOHLQ
       GOOD_ACTION=RENAME
       GOOD_NEWDSN=%Q01.GOOD.%NOHLQ
       EMAIL_GOOD=ACCT@mycompany.com
       EMAIL_FAIL=techsup@mycompany.com
```

When the `PRODDNI.ACCT.DATA` file is created, DNI schedules a file send request to all of the Platform Server systems defined in LIST1. The following nodes are processed in the file transfer:

- SNANODE
- TCPNODE
- IPADDR 127.128.129.1: uses the IP port defined in the Platform Server Global file.
- IPNAME yourhost.companya.com: uses the IP port defined in the Platform Server Global file.
- IPADDR 127.128.129.2: uses IP port 2500.

If a request to one of those nodes is successful, an email is sent to `ACCT@mycompany.com`. If a request to one of those nodes is unsuccessful, an email is sent to `techsup@mycompany.com`.

When all of the requests are completed, either successfully or unsuccessfully, the DNI request are considered completed. If all requests are completed successfully, the file is renamed to `PRODDNI.GOOD.ACCT.DATA`. If any of the requests fail, the file is renamed to `PRODDNI.FAIL.ACCT.DATA`.

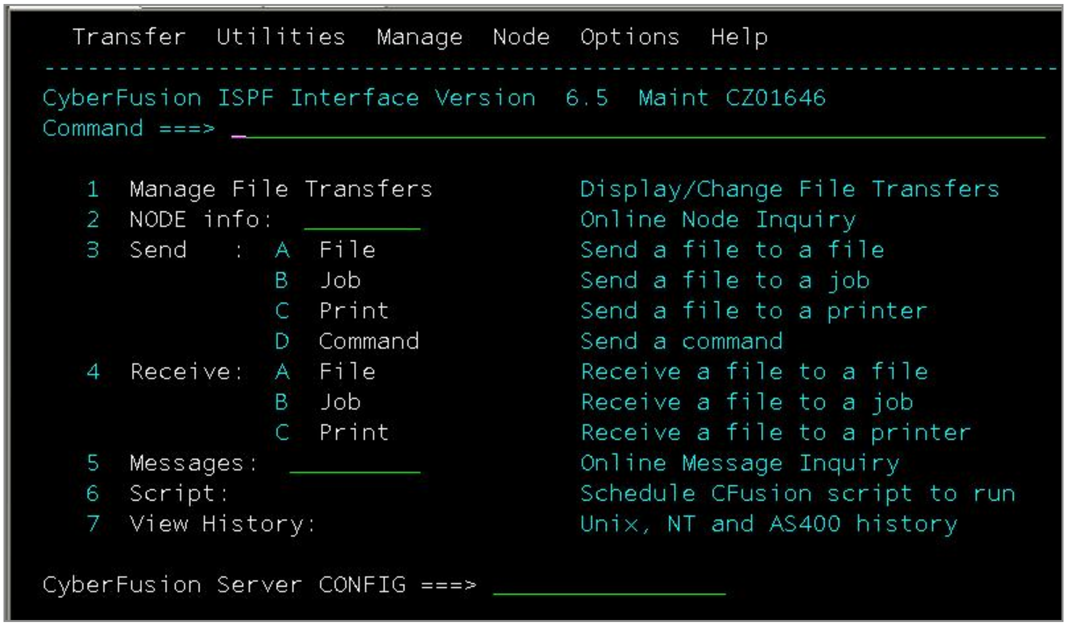
ISPF Interface

The Platform Server ISPF interface, also called the TIBCO MFT Platform Server for z/OS client, is designed for ease-of-use.

After the interface has been used to execute an action, only simple edits (for example, entering the dataset name of the latest dataset to be transferred) are required to execute a transaction that has similar characteristics to the last transaction.

To navigate the interface, select options from the main menu or use the pull-down menus to facilitate the file transfer procedures.

The following figure shows the primary window and command prompt of the Interactive interface:



```
  Transfer  Utilities  Manage  Node  Options  Help
------------------------------------------------------------------------
CyberFusion ISPF Interface Version  6.5  Maint CZ01646
Command ===> ____________________________________________________________

   1  Manage File Transfers         Display/Change File Transfers
   2  NODE info:  ________          Online Node Inquiry
   3  Send    :  A  File            Send a file to a file
                 B  Job             Send a file to a job
                 C  Print           Send a file to a printer
                 D  Command         Send a command
   4  Receive:   A  File            Receive a file to a file
                 B  Job             Receive a file to a job
                 C  Print           Receive a file to a printer
   5  Messages:  _________          Online Message Inquiry
   6  Script:                       Schedule CFusion script to run
   7  View History:                 Unix, NT and AS400 history

CyberFusion Server CONFIG ===> ________________
```

From the primary window, you can also change the Platform Server information for the server to which the transaction is to be queued. A transfer can be queued to an STC (started task) via a PC call, an LU name via SNA, an IP address or an IP name via TCP.

Communication Methods

The Platform Server Interactive interface can communicate with the Platform Server started tasks through three methods: program call, SNA LU6.2, and TCP.

For information of how to define the communication method between the Platform Server Interactive interface and started tasks, see *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Program Call

This is the most efficient way to communicate with the Platform Server; however, to use this communication method, the Platform Server client must be running the same z/OS instance as the Platform Server.

SNA LU6.2

To use this communication method, the Platform Server client APPLID (for more information, see "Defining VTAM Resources for Systems Using SNA" in *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*) must be defined on the system which runs the Platform Server client. If the Platform Server client APPLIDs are defined on each z/OS mainframe, the z/OS instance on which the Platform Server or client is running is unimportant.

TCP

To use this communication method, TCP must be running on the systems on which the Platform Server client and server are executed.

Initiating the ISPF Interface

After installation, you can initiate the interface at a TSO prompt or on an ISPF command line.

You can initiate the interface in one of following two ways:

- At a TSO prompt, type Fusion and press Enter.
- On an ISPF command line, type TSO Fusion and press Enter.

After you initiate the interface, the Platform Server primary window is displayed. In the primary window, you can move the cursor to the command prompt by pressing the Tab key or by using the arrows to position the cursor on the menu choice.

The following actions are available from the command prompt:

Actions	Description
Transfer	Sends a file to a file, job, or printer; or send a command; or receive a file to a file, job, or printer.
Utilities	Browses or edits local datasets.
Manage	Manages file transfers, including filtering on the work queue and history file .
Node	Displays the list of remote node definitions.
Options	Changes the Platform Server date and time formatting.
Help	Learns how to use the Platform Server and view other information relevant for this application.

The Platform Server CONFIG information is defined in the FUSCFG member of the Platform Server EXECS library. This member gives the user-defined name of the configuration, the name of the started task, and the name of the audit dataset. The name of the user-defined configuration is the name used in the Platform Server CONFIG field.

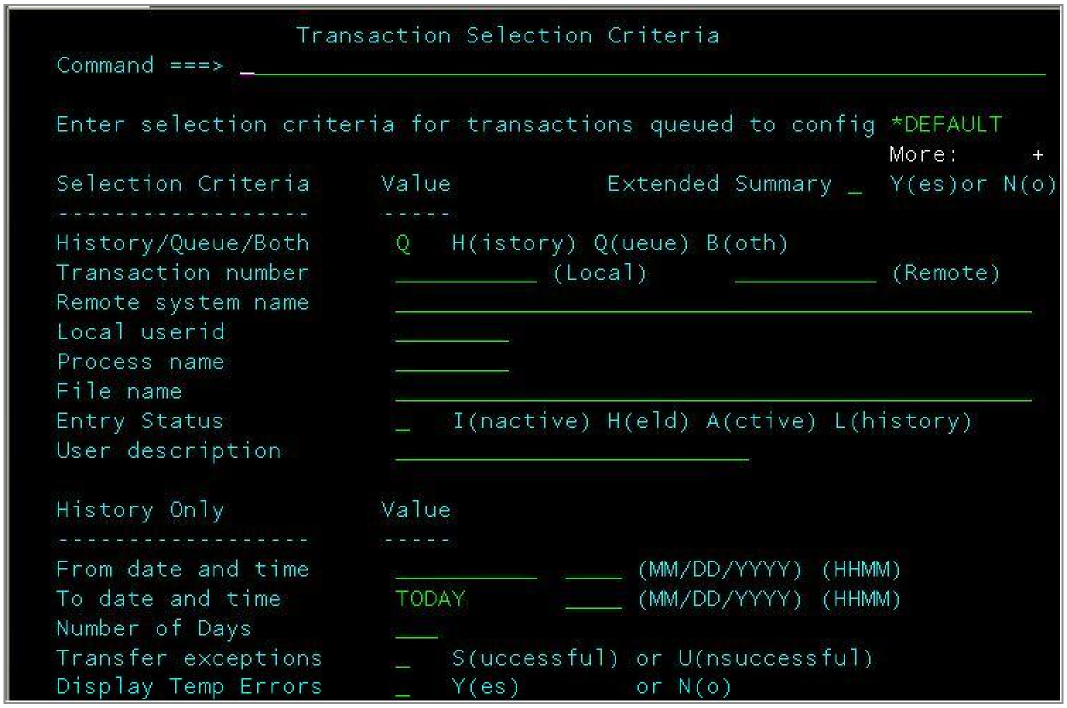
The menu options are described first and the command prompt options are described later.

Managing File Transfers

When you choose the **Manage File Transfers** option from the main menu of the Platform Server primary window to manage file transfers, including filtering on the work queue (transactions not yet completed), history file, or both, you can choose to review, release, or purge entries from the work queue; or to review past transactions on the audit file.

> **Note:** The files that are to be managed are based on the **CyberFusion Server CONFIG** field that can be found on the primary window (see Interactive Interface).

The following figure shows the Transaction Selection Criteria panel of the **Manage File Transfers** option:



```
                 Transaction Selection Criteria
Command ===> _________________________________________________________

Enter selection criteria for transactions queued to config *DEFAULT
                                                           More:     +
Selection Criteria      Value          Extended Summary _  Y(es)or N(o)
------------------      -----
History/Queue/Both       Q   H(istory) Q(ueue) B(oth)
Transaction number       __________ (Local)      __________ (Remote)
Remote system name       _____________________________________________
Local userid             ________
Process name             ________
File name                _____________________________________________
Entry Status             _   I(nactive) H(eld) A(ctive) L(history)
User description         _________________________

History Only            Value
------------------      -----
From date and time       __________  ____ (MM/DD/YYYY) (HHMM)
To date and time         TODAY       ____ (MM/DD/YYYY) (HHMM)
Number of Days           ___
Transfer exceptions      _   S(uccessful) or U(nsuccessful)
Display Temp Errors      _   Y(es)        or N(o)
```

Defining Transaction Selection Criteria

You can use the Transaction Selection Criteria panel to set the transaction selection criteria to view a wide range of activities and to limit the number of transactions displayed.

Note: The files that are to be managed are based on the **CyberFusion Server CONFIG** field that can be found on the primary window (see Interactive Interface).

For example, if you want to view the past transactions of the local user ID USER1, you set `H` in the **History/Queue/Both** field and `USER1` in the **Local userid** entry field, and press Enter; then, only transactions that USER1 initiated are displayed.

To display all transactions, clear all the entry fields in the Transaction Selection Criteria panel and press Enter.

A maximum of 4000 transfers can be displayed on the ISPF panels. This is to insure that the ISPF user and more importantly, the started task does not run out of memory when an ISPF user selects transfers to display.

Transaction Selection Criteria Parameters

The following table lists the parameters of the Transaction Selection Criteria panel:

Parameter	Description
`Display Temp Errors`	Defines whether the Platform Server will return temporary errors. By default, only permanent errors are returned. This parameter only applies when an error is retried. When the retries are exhausted, the Platform Server always writes an audit record when the request is purged from the active queue.
`Entry Status`	Defines the status of the entry or entries you want to select. You can either leave this field blank to select all status or specify one of the following options to refine the search: • `I`: Shows only inactive transactions. • `H`: Shows only held transactions. • `A`: Shows only active transactions. • `L`: Shows only history or logged transactions.
`Extended Summary`	Defines whether to turn the summary screen into 2 lines per transfer. When you specify `Yes`, the date and time are added to the information displayed and the local file name is displayed on a separate line.
`File name`	Enter the file name or any part of the file name followed by an asterisk (*) to refine the search.
`From date and time`	Defines the Platform Server to search for transactions that were logged at and after this specified date and time (optional). The date and time must be in the formats shown to the right of the entry fields. **Note:** This field only applies to history searches.

Parameter	Description
`History/Queue/Both`	Defines whether you want to view past transactions that are logged, current transactions that are still in the work queue, or both past and current transactions. Valid values are: • `H`: Shows only history transactions. • `Q`: Shows only transactions in the work queue. • `B`: Shows both history transactions and transactions in the work queue.
`Local userid`	Enter the local user ID of the user that submitted the transaction or any part of the local user ID followed by an asterisk (*) to refine the search.
`Number of Days`	Defines the number of days back that you want to search. If you enter a `From date and time` value, the Platform Server searches for transactions that were logged at and after the `From date and time` and goes the specified number of days forward. If you specified a `To date and time` value, the Platform Server searches for transactions that were logged at and before the `To date and time` and goes the specified number of days backward. **Note:** The **Number of Days** field defaults to 1 day unless you modify the **From date and time** field.
`Process name`	Enter the process name or any part of the process name followed by an asterisk (*) to refine the search.
`Remote system name`	Enter the remote system name or any part of the remote system name followed by an asterisk (*) to refine the search.
`To date and time`	Defines the Platform Server to search for transactions that were logged at and before this specified date. The date

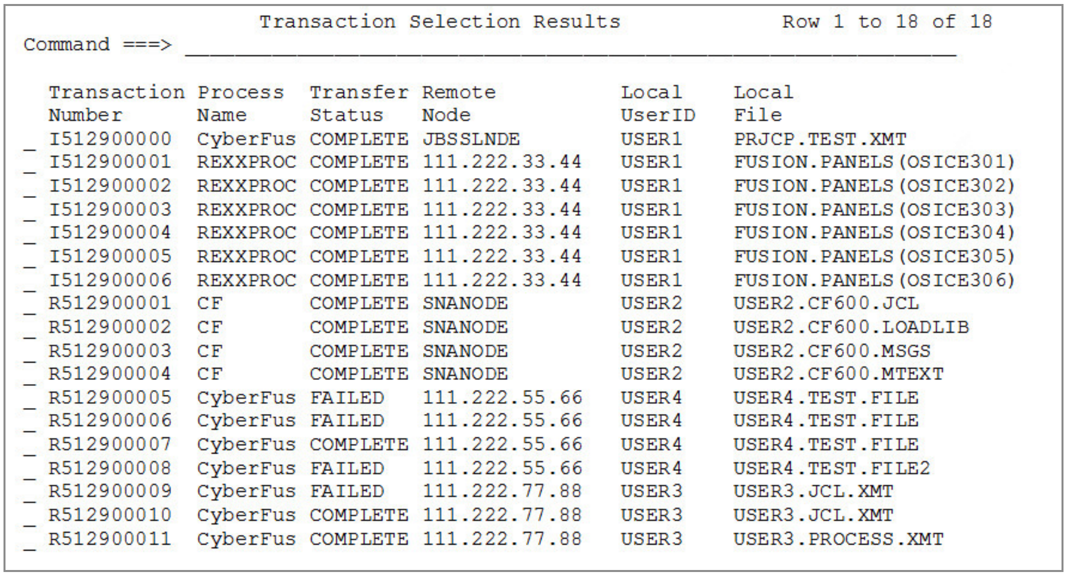
Parameter	Description
	must be in the format shown to the right of the entry field. **Note:** This field only applies to history searches.
`Transaction number`	Defines the transaction number you want to select. You can either leave this field blank to select all transaction numbers or type part of a transaction number followed by an asterisk (*) to refine the search. For example, I5* only selects the transfers initiated in the month of May. This field includes the following two subfields: • **Local**: Defines the transaction number on the local system. • **Remote**: Defines the transaction number of the remote system. **Note:** This parameter is valid only when `H` is specified in the **History/Queue/Both** field.
`Transfer Exceptions`	Defines the Platform Server to return successful, unsuccessful, or both transactions. If you leave this field blank, both successful and unsuccessful transactions are returned. Valid values are: • `S`: Shows only successful transactions. • `U`: Shows only unsuccessful transactions. **Note:** This field only applies to history searches.
`User description`	Enter a file transfer description (of up to 25 alpha, numeric, or national characters). You can enter any part of the description followed by an

Parameter	Description
	asterisk (*) to refine the search. **Note:** No imbedded spaces are allowed in this field.

Manipulating Transaction Selection Results

You can perform various operations on the transactions displayed in the Transaction Selection Results panel.

After you execute the search, the results are displayed in the Transaction Selection Results panel, as shown in the following figure:



```
                   Transaction Selection Results             Row 1 to 18 of 18
Command ===> ______________________________________________________________

  Transaction Process  Transfer Remote          Local    Local
  Number      Name     Status   Node            UserID   File
_ I512900000  CyberFus COMPLETE JBSSLNDE        USER1    PRJCP.TEST.XMT
_ I512900001  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE301)
_ I512900002  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE302)
_ I512900003  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE303)
_ I512900004  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE304)
_ I512900005  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE305)
_ I512900006  REXXPROC COMPLETE 111.222.33.44   USER1    FUSION.PANELS(OSICE306)
_ R512900001  CF       COMPLETE SNANODE         USER2    USER2.CF600.JCL
_ R512900002  CF       COMPLETE SNANODE         USER2    USER2.CF600.LOADLIB
_ R512900003  CF       COMPLETE SNANODE         USER2    USER2.CF600.MSGS
_ R512900004  CF       COMPLETE SNANODE         USER2    USER2.CF600.MTEXT
_ R512900005  CyberFus FAILED   111.222.55.66   USER4    USER4.TEST.FILE
_ R512900006  CyberFus FAILED   111.222.55.66   USER4    USER4.TEST.FILE
_ R512900007  CyberFus COMPLETE 111.222.55.66   USER4    USER4.TEST.FILE
_ R512900008  CyberFus FAILED   111.222.55.66   USER4    USER4.TEST.FILE2
_ R512900009  CyberFus FAILED   111.222.77.88   USER3    USER3.JCL.XMT
_ R512900010  CyberFus COMPLETE 111.222.77.88   USER3    USER3.JCL.XMT
_ R512900011  CyberFus COMPLETE 111.222.77.88   USER3    USER3.PROCESS.XMT
```

When the transaction list is displayed, you can issue the following action commands in the input field to the far left of the transactions:

- P: Purges a transaction.
- S: Selects the transaction to view its details. (Only available from a history search.)
- H: Holds a scheduled transaction.
- R: Releases a scheduled transaction that is being held from running.
- V: Views all attempts to run the transaction. (Only available from a History search.)

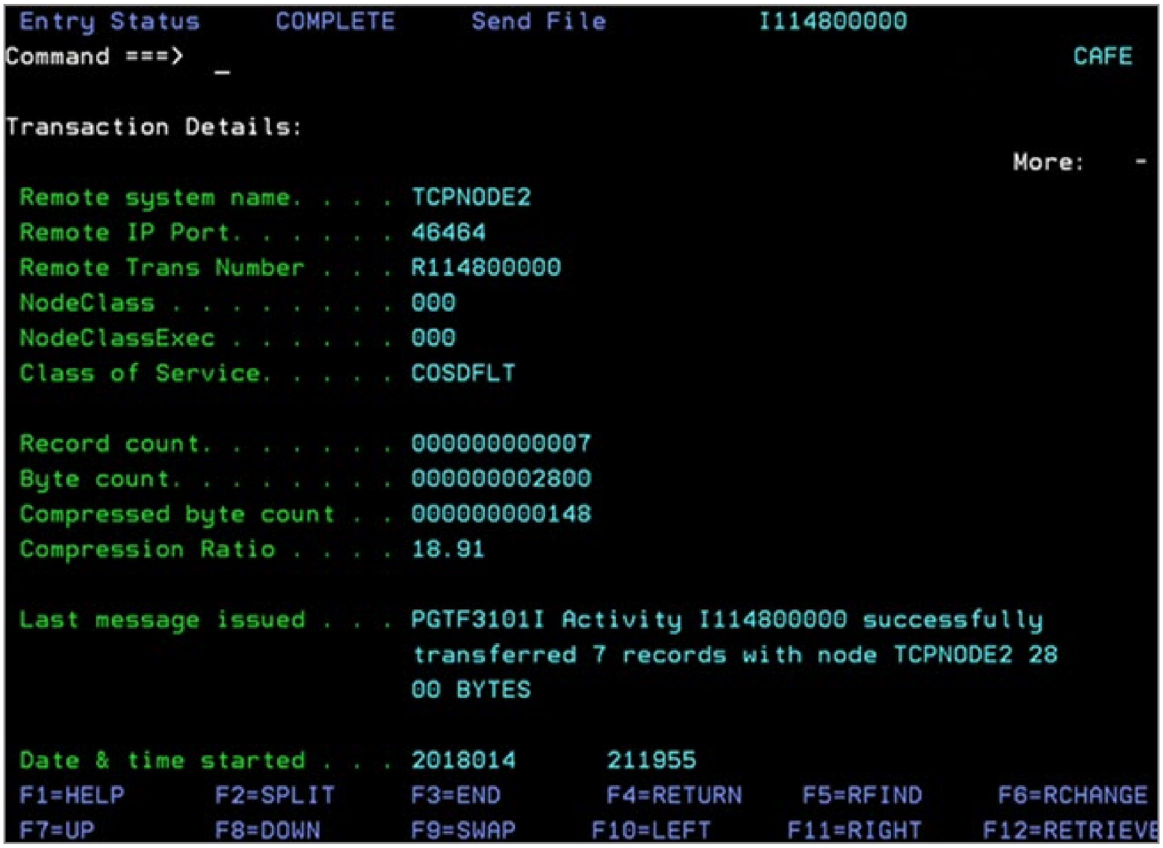
Note: See Viewing Transaction Details and Viewing the Summary of Transactions Attempts for details about how to view transaction details and the summary of transaction attempts.

Viewing Transaction Details

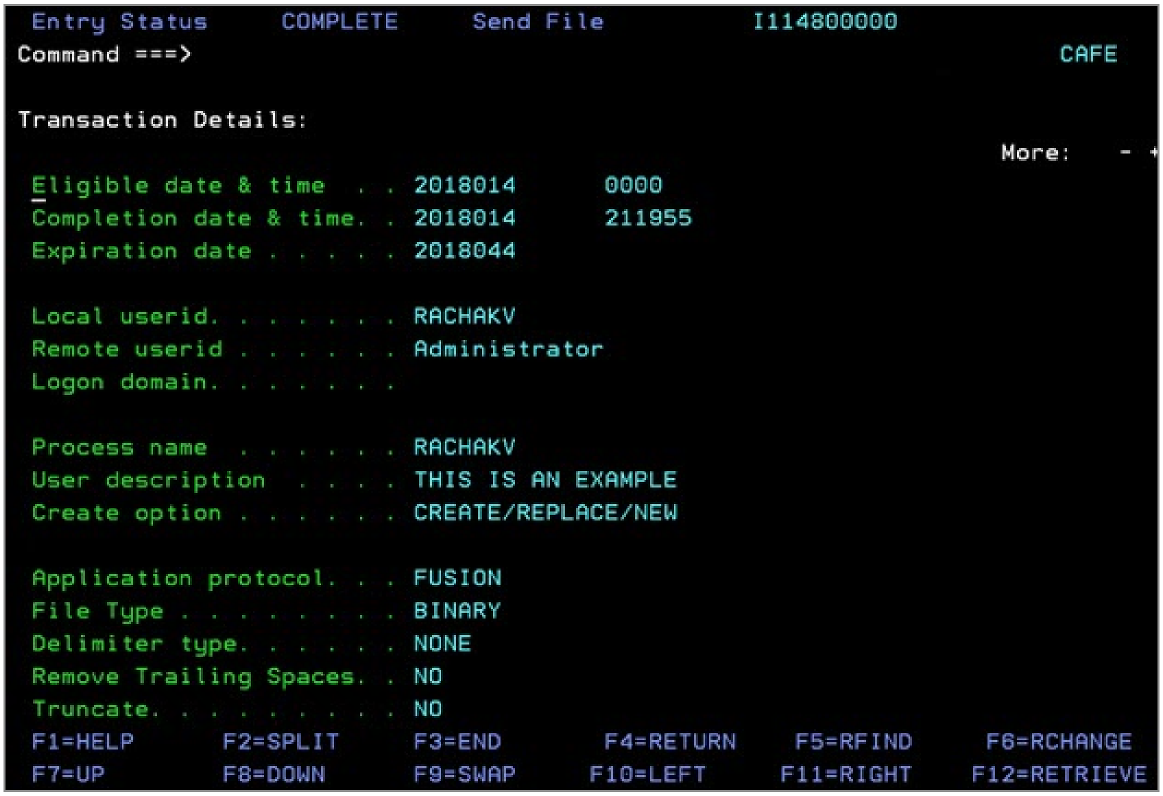
After the transaction selection results are displayed, you can use the S command to view the details of a transaction. You can check all the parameters settings and view information such as the number of bytes and records transferred (if the transfer was successful) and the time and date the transfer was completed.

Note: The S command is only available from a history search.

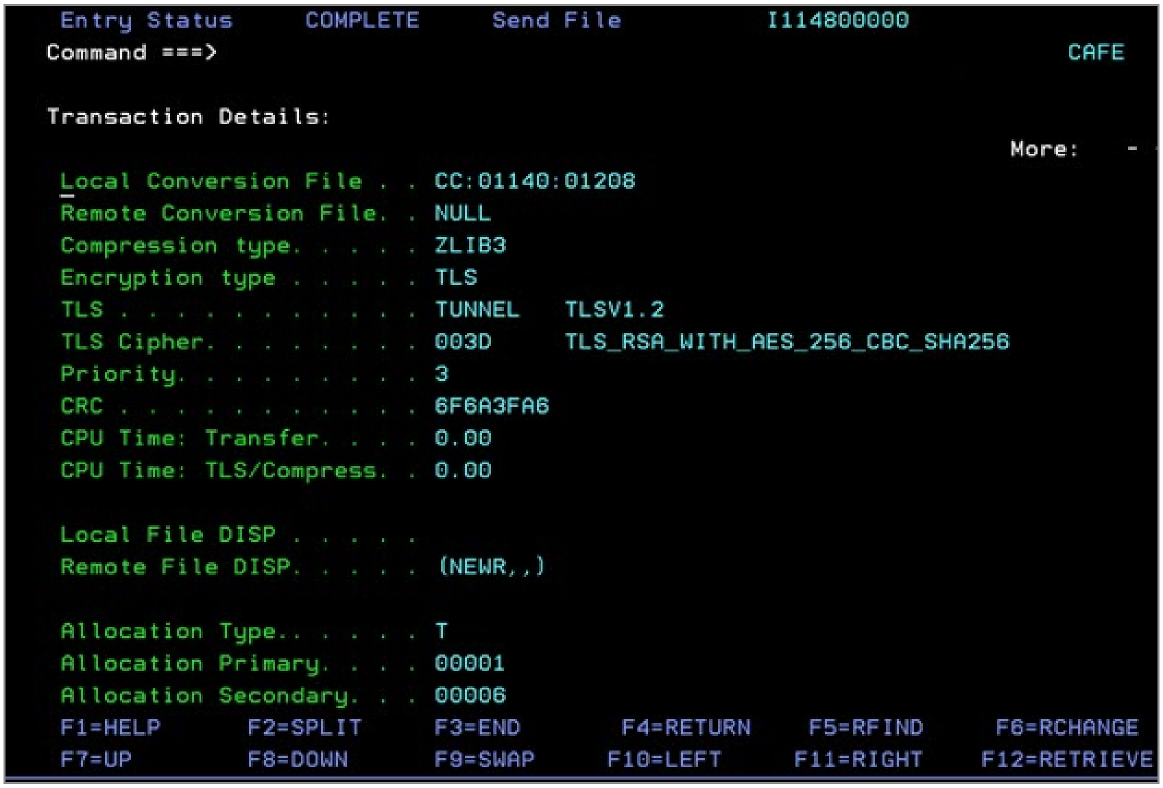
To view the details of a transaction, type S in the field to the far left of a transaction in the Transaction Selection Results panel and press Enter (see Manipulating Transaction Selection Results for the Transaction Selection Results panel). The details of this transaction are displayed as shown in the following figures. You can use the F8 and F7 keys to go to the next and previous pages of the transaction details.



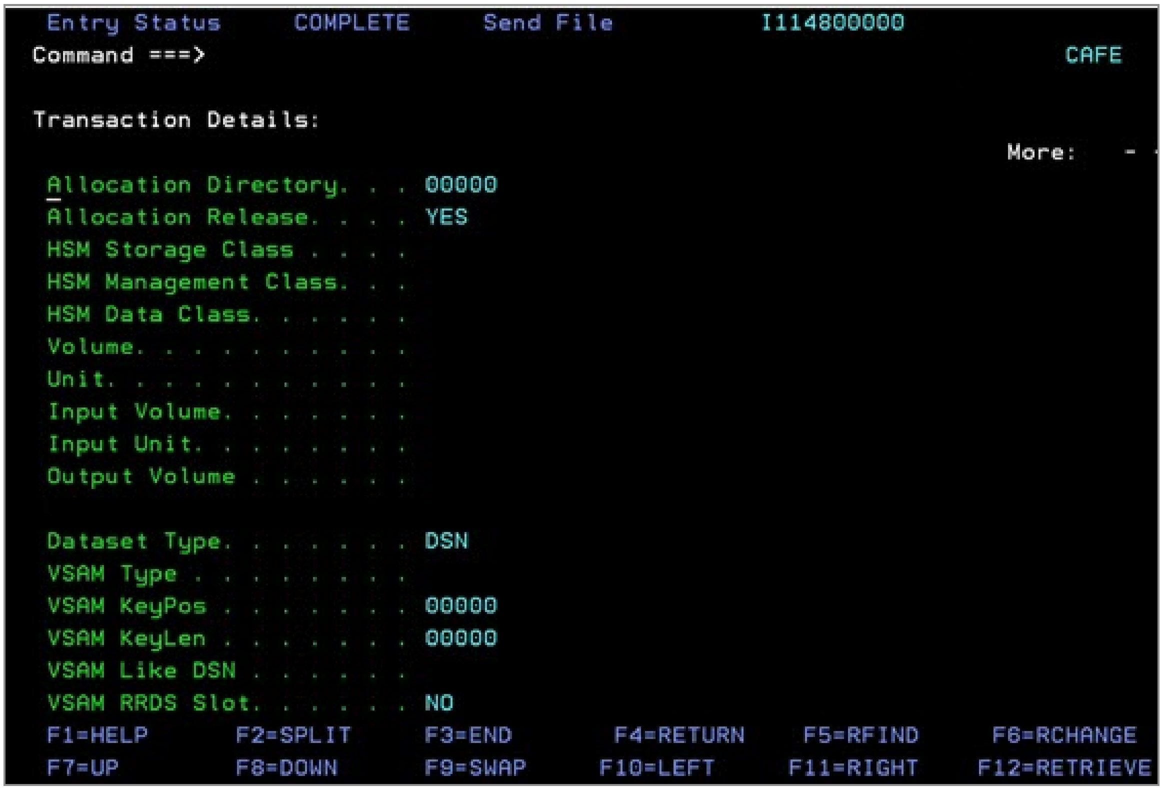
```
 Entry Status      COMPLETE      Send File           I114800000
Command ===> _                                                          CAFE

Transaction Details:
                                                                    More:    -
 Remote system name. . . . TCPNODE2
 Remote IP Port. . . . . . 46464
 Remote Trans Number . . . R114800000
 NodeClass . . . . . . . . 000
 NodeClassExec . . . . . . 000
 Class of Service. . . . . COSDFLT

 Record count. . . . . . . 000000000007
 Byte count. . . . . . . . 000000002800
 Compressed byte count . . 000000000148
 Compression Ratio . . . . 18.91

 Last message issued . . . PGTF3101I Activity I114800000 successfully
                           transferred 7 records with node TCPNODE2 28
                           00 BYTES

 Date & time started . . . 2018014      211955
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



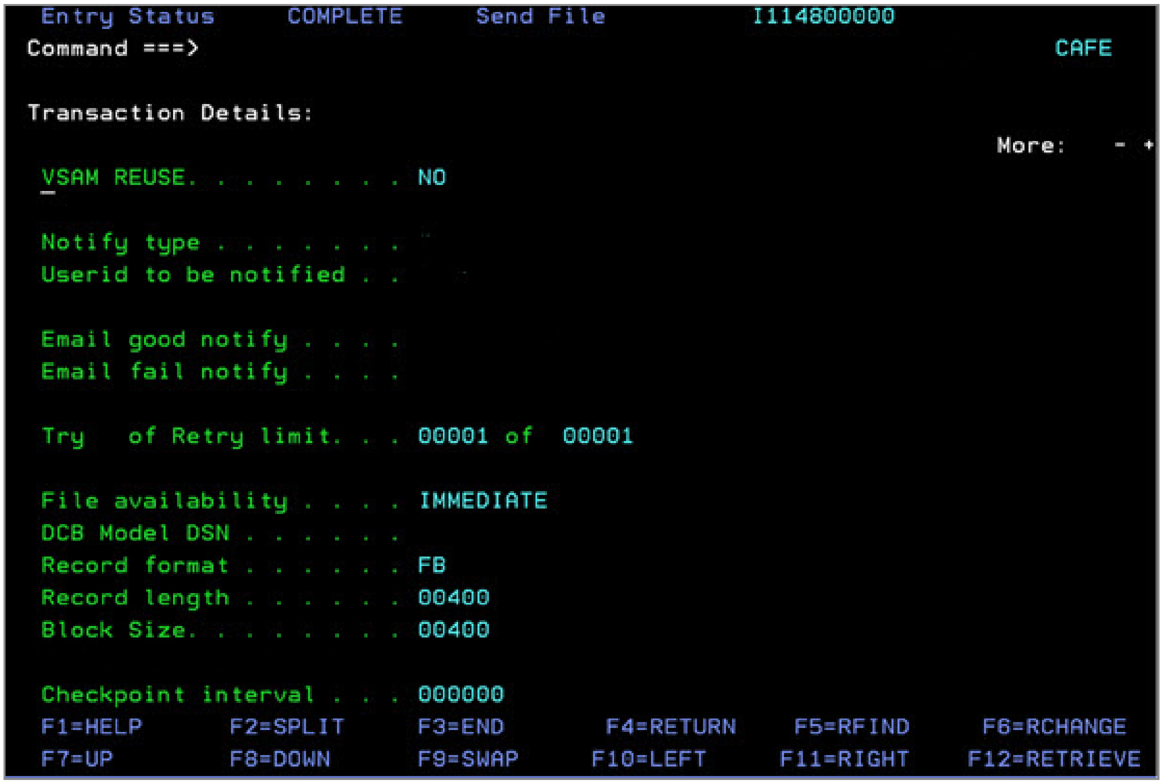
```
 Entry Status     COMPLETE     Send File           I114800000
Command ===>                                                        CAFE

Transaction Details:
                                                                 More:    - +
 Eligible date & time  . . 2018014      0000
 Completion date & time. . 2018014      211955
 Expiration date . . . . . 2018044

 Local userid. . . . . . . RACHAKV
 Remote userid . . . . . . Administrator
 Logon domain. . . . . . .

 Process name  . . . . . . RACHAKV
 User description  . . . . THIS IS AN EXAMPLE
 Create option . . . . . . CREATE/REPLACE/NEW

 Application protocol. . . FUSION
 File Type . . . . . . . . BINARY
 Delimiter type. . . . . . NONE
 Remove Trailing Spaces. . NO
 Truncate. . . . . . . . . NO
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



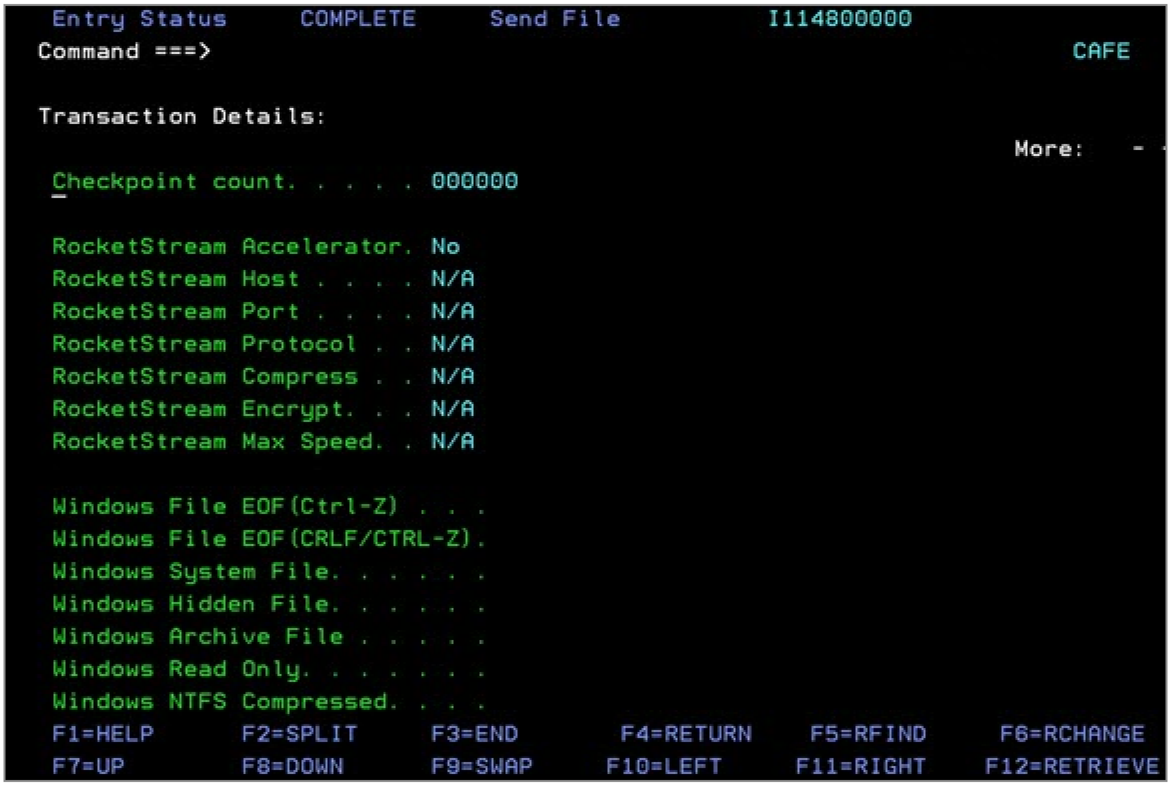
```
  Entry Status      COMPLETE      Send File          I114800000
 Command ===>                                                           CAFE

 Transaction Details:
                                                                    More:    -
  Local Conversion File . . CC:01140:01208
  Remote Conversion File. . NULL
  Compression type. . . . . ZLIB3
  Encryption type . . . . . TLS
  TLS . . . . . . . . . . . TUNNEL    TLSV1.2
  TLS Cipher. . . . . . . . 003D      TLS_RSA_WITH_AES_256_CBC_SHA256
  Priority. . . . . . . . . 3
  CRC . . . . . . . . . . . 6F6A3FA6
  CPU Time: Transfer. . . . 0.00
  CPU Time: TLS/Compress. . 0.00

  Local File DISP . . . . .
  Remote File DISP. . . . . (NEWR,,)

  Allocation Type.. . . . . T
  Allocation Primary. . . . 00001
  Allocation Secondary. . . 00006
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



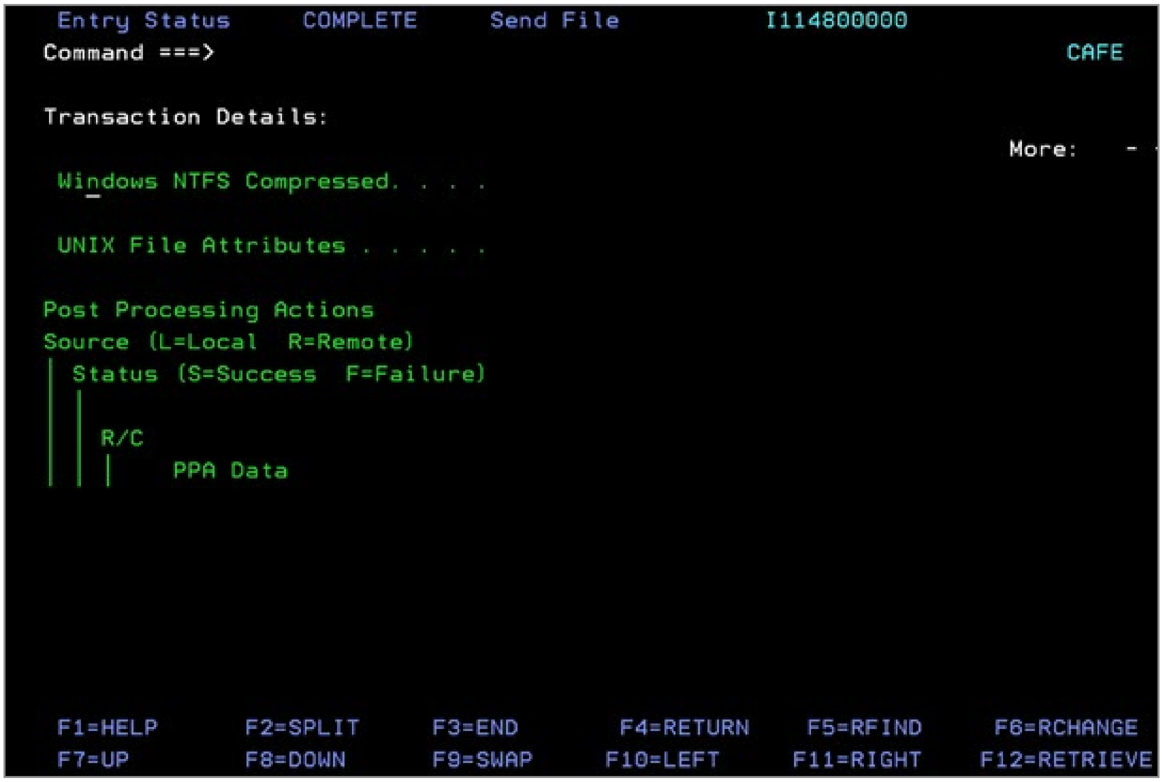
```
 Entry Status      COMPLETE      Send File          I114800000
Command ===>                                                           CAFE

Transaction Details:
                                                                   More:    -
 Allocation Directory. . . 00000
 Allocation Release. . . . YES
 HSM Storage Class . . . .
 HSM Management Class. . .
 HSM Data Class. . . . . .
 Volume. . . . . . . . . .
 Unit. . . . . . . . . . .
 Input Volume. . . . . . .
 Input Unit. . . . . . . .
 Output Volume . . . . . .

 Dataset Type. . . . . . . DSN
 VSAM Type . . . . . . . .
 VSAM KeyPos . . . . . . . 00000
 VSAM KeyLen . . . . . . . 00000
 VSAM Like DSN . . . . . .
 VSAM RRDS Slot. . . . . . NO
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
 Entry Status      COMPLETE      Send File           I114800000
Command ===>                                                           CAFE

Transaction Details:
                                                                   More:    - +
 VSAM REUSE. . . . . . . . . NO

 Notify type . . . . . . . .
 Userid to be notified . .

 Email good notify . . . .
 Email fail notify . . . .

 Try   of Retry limit. . . 00001 of  00001

 File availability . . . . IMMEDIATE
 DCB Model DSN . . . . . .
 Record format . . . . . . FB
 Record length . . . . . . 00400
 Block Size. . . . . . . . 00400

 Checkpoint interval . . . 000000
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
 Entry Status      COMPLETE      Send File           I114800000
Command ===>                                                           CAFE

Transaction Details:
                                                                   More:    -
 Checkpoint count. . . . . . 000000

 RocketStream Accelerator. No
 RocketStream Host . . . . N/A
 RocketStream Port . . . . N/A
 RocketStream Protocol . . N/A
 RocketStream Compress . . N/A
 RocketStream Encrypt. . . N/A
 RocketStream Max Speed. . N/A

 Windows File EOF(Ctrl-Z) . . .
 Windows File EOF(CRLF/CTRL-Z).
 Windows System File. . . . . .
 Windows Hidden File. . . . . .
 Windows Archive File . . . . .
 Windows Read Only. . . . . . .
 Windows NTFS Compressed. . . .
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



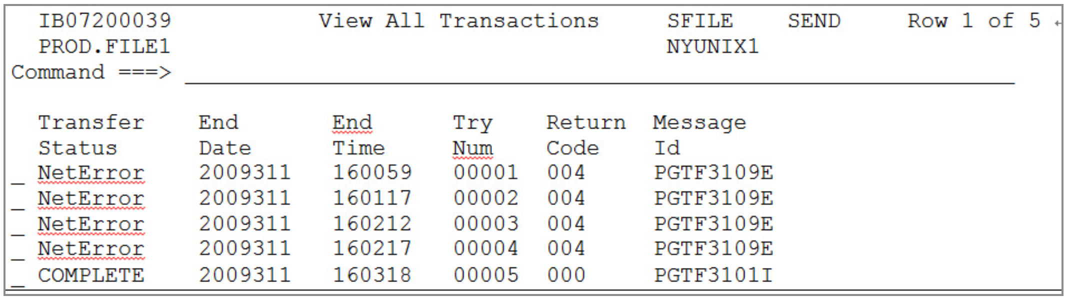
```
    Entry Status      COMPLETE      Send File           I114800000
  Command ===>                                                          CAFE

  Transaction Details:
                                                                    More:    -
    Windows NTFS Compressed. . . .

    UNIX File Attributes . . . . .

  Post Processing Actions
  Source (L=Local  R=Remote)
  | Status (S=Success  F=Failure)
  | |
  | | R/C
  | | |    PPA Data





    F1=HELP       F2=SPLIT      F3=END        F4=RETURN     F5=RFIND      F6=RCHANGE
    F7=UP         F8=DOWN       F9=SWAP      F10=LEFT      F11=RIGHT     F12=RETRIEVE
```

Viewing a Summary of Transaction Attempts

After the transaction selection results are displayed, you can view the summary of all transfer attempts of a transaction.

Note: The V command is only available from a history search.

To view the summary of all transfer attempts for a single transaction, type V on the line to the far left of a single transaction located on the Transaction Selection Results panel (see Manipulating Transaction Selection Results for the Transaction Selection Results panel) and press Enter. A summary screen is displayed with all the attempts of this transaction, as shown in the following figure. Each attempt is displayed on a single line.



```
  IB07200039           View All Transactions     SFILE    SEND      Row 1 of 5
  PROD.FILE1                                     NYUNIX1
Command ===> ______________________________________________________________

  Transfer    End        End        Try      Return   Message
  Status      Date       Time       Num      Code     Id
_ NetError    2009311    160059     00001    004      PGTF3109E
_ NetError    2009311    160117     00002    004      PGTF3109E
_ NetError    2009311    160212     00003    004      PGTF3109E
_ NetError    2009311    160217     00004    004      PGTF3109E
_ COMPLETE    2009311    160318     00005    000      PGTF3101I
```

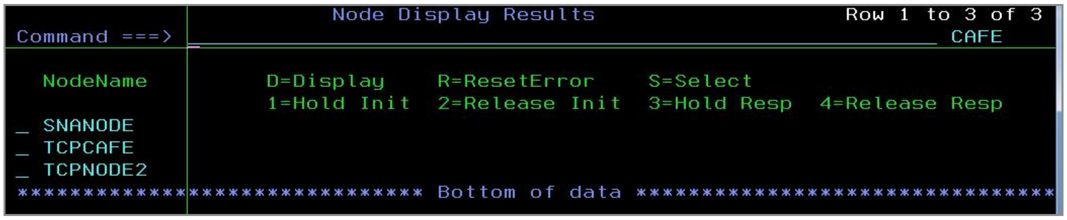
In this example, the transfer request was attempted 5 times. For the first 4 attempts, the request received a network error; while for the last attempt, the request was completed successfully.

Each attempt of the request displayed in this screen has an action field. You can type S on the input field to the left of each attempted transaction and press Enter to view the details of this attempted transaction. No other actions can be performed on this screen.

Managing Remote Nodes

You can use the **NODE info** option in the Platform Server primary window to display the remote systems that have been defined to the Platform Server.

After you select **NODE info** in the Platform Server primary window, the Node Display Results panel is displayed as shown in the following figure.

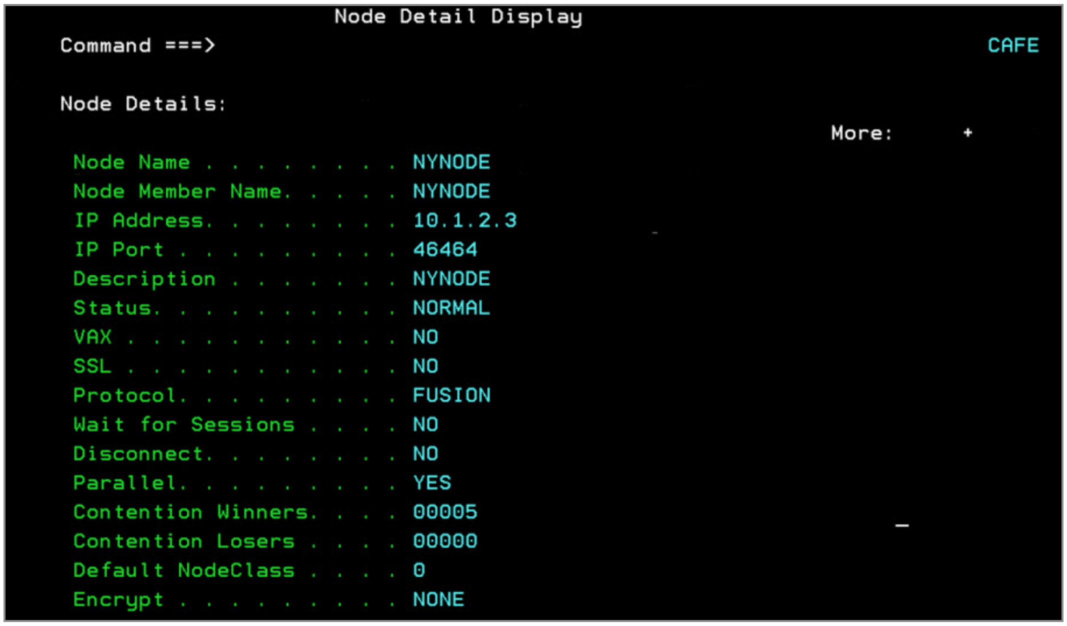


```
                     Node Display Results                        Row 1 to 3 of 3
Command ===> ___________________________________________________________ CAFE

   NodeName         D=Display    R=ResetError    S=Select
                    1=Hold Init  2=Release Init  3=Hold Resp  4=Release Resp
_  SNANODE
_  TCPCAFE
_  TCPNODE2
****************************** Bottom of data ********************************
```

You can view the remote systems defined to the Platform Server, and perform various operations by issuing the following commands in the input field to the far left of the node names and pressing Enter:

- D: Displays the details of a node in the Node Detail Display panel.
- R: Resets a node that is in error. When the node is reset, transfers using that node are retried again.
- S: Displays the details of a node.
- 1: Turns on the Initiator Hold flag.
- 2: Resets the Initiator Hold flag.
- 3: Turns on the Responder Hold flag.
- 4: Resets the Responder Hold flag.

The following figure shows an example of the details of node NYNODE:



```
                          Node Detail Display
Command ===>                                                            CAFE

Node Details:
                                                          More:       +
 Node Name . . . . . . . . NYNODE
 Node Member Name. . . . . NYNODE
 IP Address. . . . . . . . 10.1.2.3
 IP Port . . . . . . . . . 46464
 Description . . . . . . . NYNODE
 Status. . . . . . . . . . NORMAL
 VAX . . . . . . . . . . . NO
 SSL . . . . . . . . . . . NO
 Protocol. . . . . . . . . FUSION
 Wait for Sessions . . . . NO
 Disconnect. . . . . . . . NO
 Parallel. . . . . . . . . YES
 Contention Winners. . . . 00005
 Contention Losers . . . . 00000
 Default NodeClass . . . . 0
 Encrypt . . . . . . . . . NONE
```

Defining Send Requests

You can use the Send parameter in the Platform Server primary window to send a file to a file, job, or printer or to send a command to a remote system.

To initiate a send request, you can use one of the following two methods:

- Type the number 3 and a letter (A, B, C, or D) on the command line and press Enter to directly send a file to a file, job, or printer; or send a command to a remote system.

  For example, type 3A (or 3.A) on the command line.

- Type the number 3 and press Enter; a screen is displayed where you can select to type A, B, C, or D to send a file to a file, job, or printer; or send a command to a remote system.

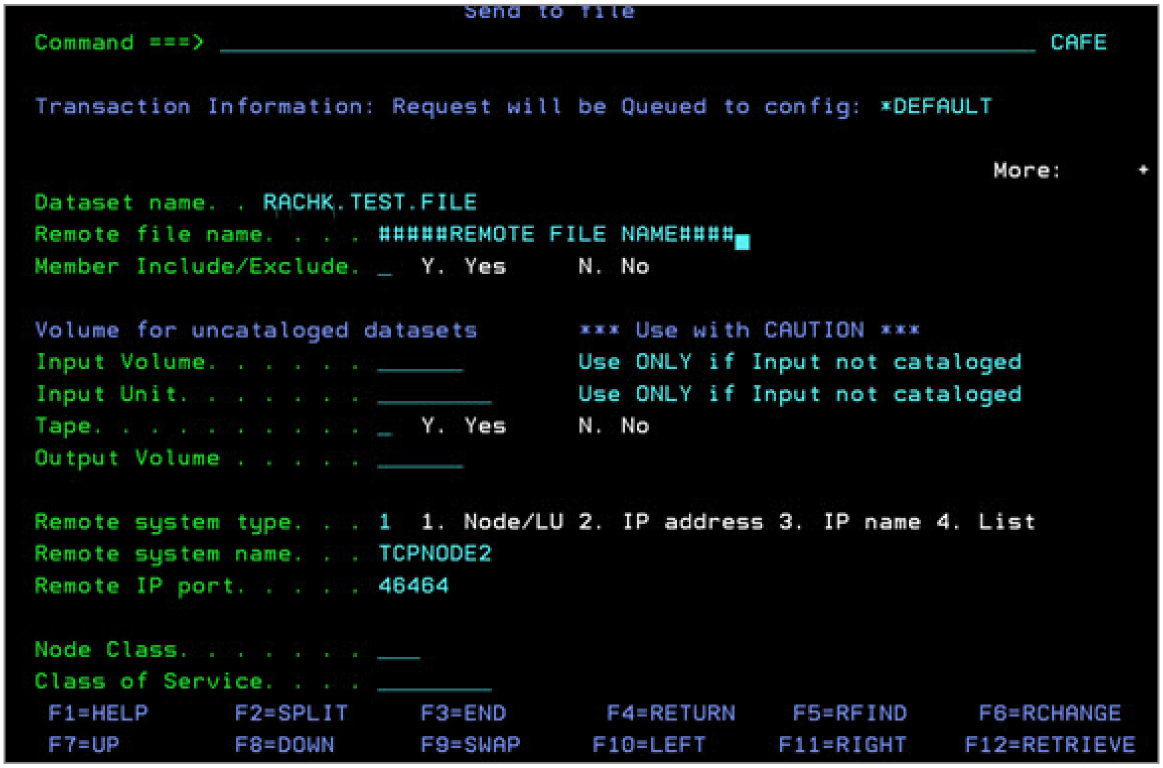
For more details about how to send a file and send a command, see the following introductions:

- Sending a File to a File
- Sending a File to a Job
- Sending a File to a Printer
- Sending a Command

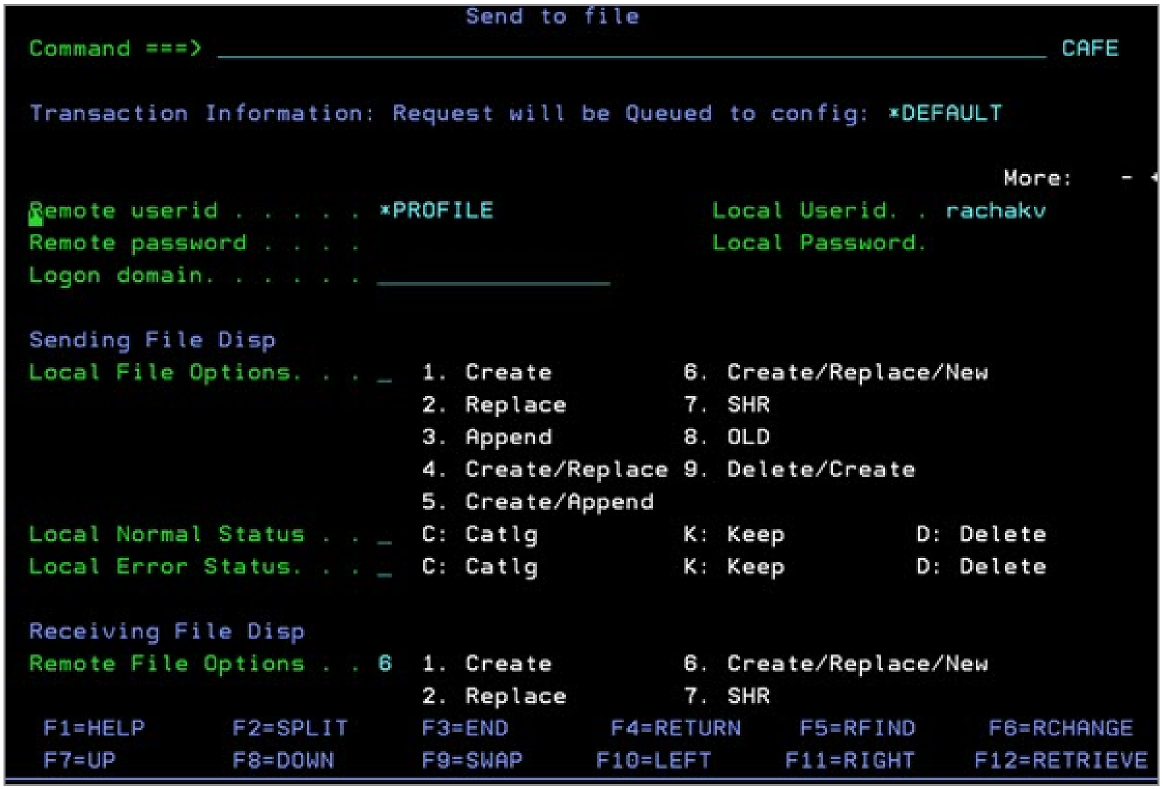
Sending a File to a File

By typing 3A (or 3.A) on the command line in the Platform Server primary window, you can send a file to a remote system, where the partner receives it to a file.

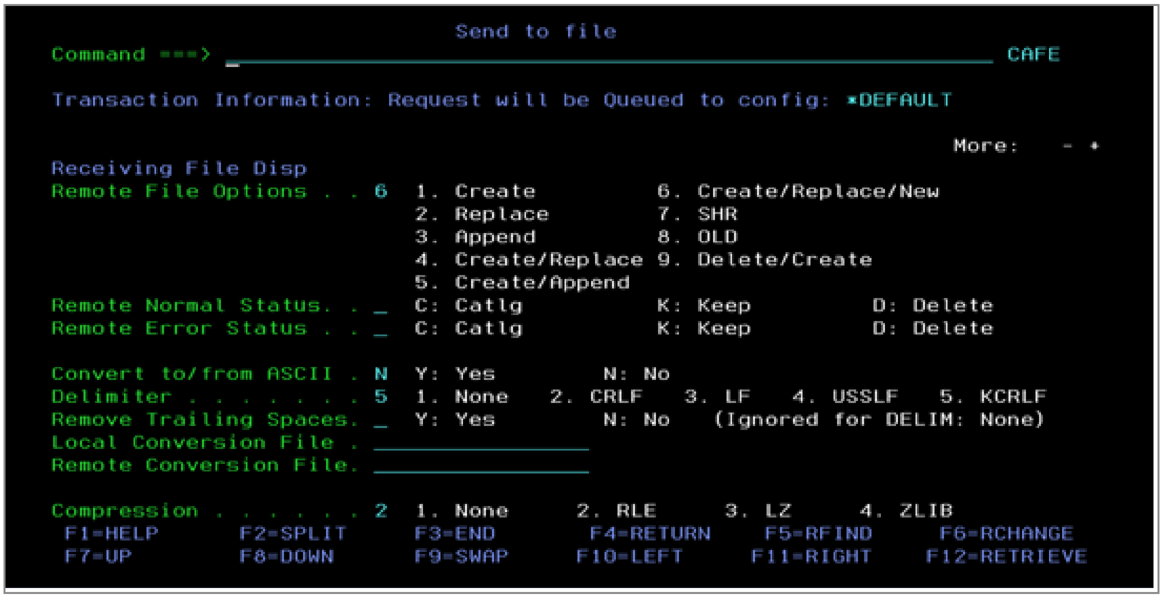
After you type 3A (or 3.A) on the command line, the "Send to file" panel is displayed, as shown in the following figures. The panel contains all the parameters you can configure to set up a transfer to send a file to a remote system. You can press the F8 and F7 keys to go to the next and previous pages of the "Send to file" panel.



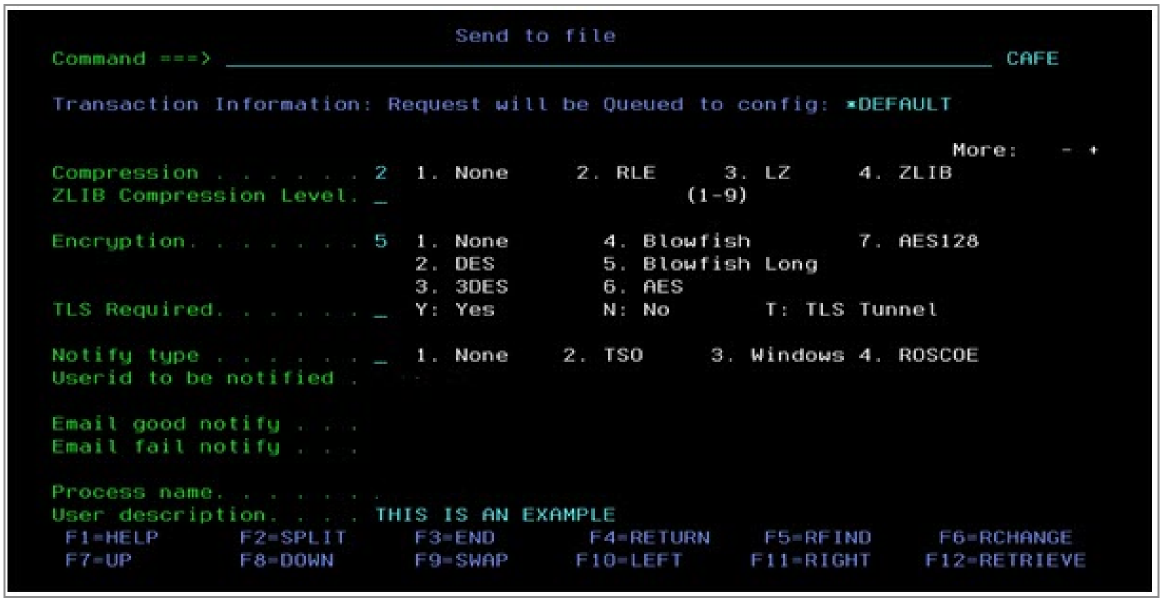
```
                              Send to file
 Command ===> ______________________________________________________ CAFE

 Transaction Information: Request will be Queued to config: *DEFAULT

                                                                More:     +
 Dataset name. . RACHK.TEST.FILE
 Remote file name. . . . #####REMOTE FILE NAME####
 Member Include/Exclude. _  Y. Yes     N. No

 Volume for uncataloged datasets       *** Use with CAUTION ***
 Input Volume. . . . . . ______        Use ONLY if Input not cataloged
 Input Unit. . . . . . . ________      Use ONLY if Input not cataloged
 Tape. . . . . . . . . . _  Y. Yes     N. No
 Output Volume . . . . . ______

 Remote system type. . . 1  1. Node/LU 2. IP address 3. IP name 4. List
 Remote system name. . . TCPNODE2
 Remote IP port. . . . . 46464

 Node Class. . . . . . . ___
 Class of Service. . . . ________
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



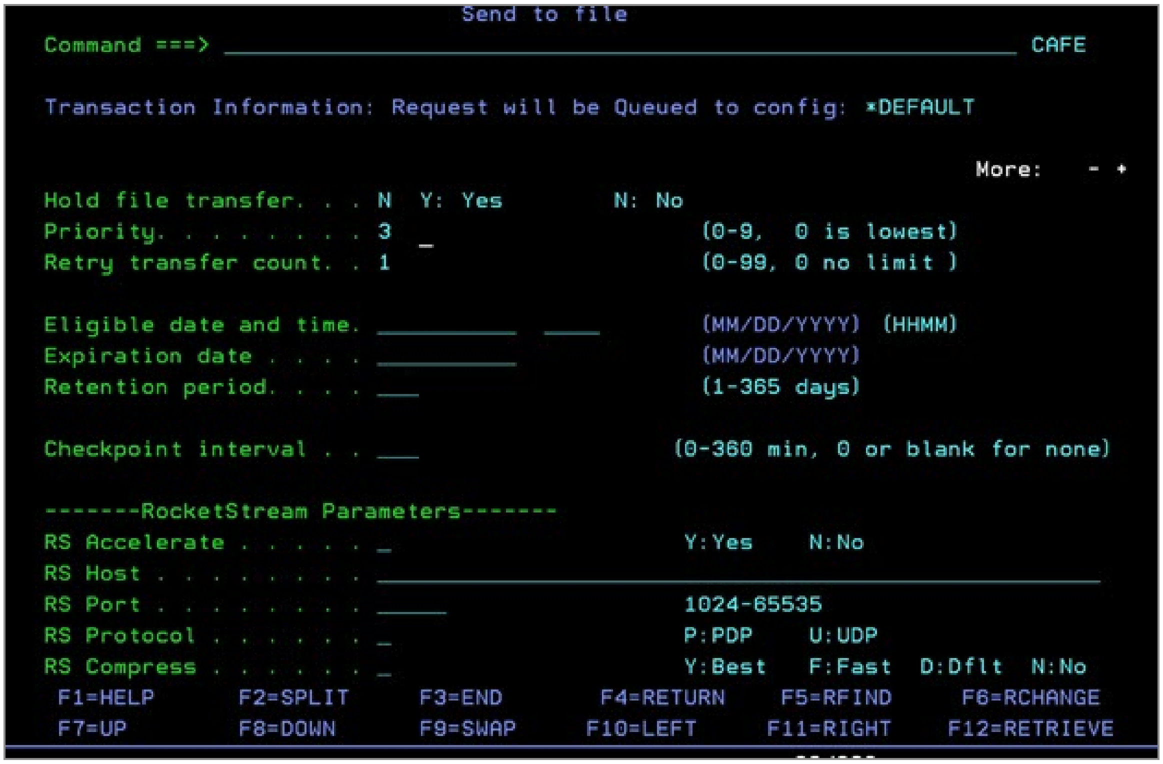
```
                              Send to file
 Command ===> ________________________________________________________ CAFE

 Transaction Information: Request will be Queued to config: *DEFAULT

                                                                 More:    - +
 Remote userid . . . . . *PROFILE              Local Userid. . rachakv
 Remote password . . . .                       Local Password.
 Logon domain. . . . . . ________________

 Sending File Disp
 Local File Options. . . _   1. Create           6. Create/Replace/New
                             2. Replace          7. SHR
                             3. Append           8. OLD
                             4. Create/Replace 9. Delete/Create
                             5. Create/Append
 Local Normal Status . . _   C: Catlg            K: Keep          D: Delete
 Local Error Status. . . _   C: Catlg            K: Keep          D: Delete

 Receiving File Disp
 Remote File Options . . 6   1. Create           6. Create/Replace/New
                             2. Replace          7. SHR
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                              Send to file
 Command ===> _________________________________________________________ CAFE

 Transaction Information: Request will be Queued to config: *DEFAULT
                                                                More:    - +
 Receiving File Disp
 Remote File Options . . 6   1. Create           6. Create/Replace/New
                             2. Replace          7. SHR
                             3. Append           8. OLD
                             4. Create/Replace 9. Delete/Create
                             5. Create/Append
 Remote Normal Status. . _   C: Catlg            K: Keep          D: Delete
 Remote Error Status . . _   C: Catlg            K: Keep          D: Delete

 Convert to/from ASCII . N   Y: Yes         N: No
 Delimiter . . . . . . . 5   1. None    2. CRLF    3. LF    4. USSLF    5. KCRLF
 Remove Trailing Spaces. _   Y: Yes         N: No    (Ignored for DELIM: None)
 Local Conversion File . ________________
 Remote Conversion File. ________________

 Compression . . . . . . 2   1. None      2. RLE      3. LZ      4. ZLIB
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                   Send to file
  Command ===> ______________________________________________________ CAFE

  Transaction Information: Request will be Queued to config: *DEFAULT

                                                                  More:    - +
  Compression . . . . . . . 2  1. None     2. RLE     3. LZ     4. ZLIB
  ZLIB Compression Level. _                          (1-9)

  Encryption. . . . . . . . 5  1. None       4. Blowfish        7. AES128
                               2. DES        5. Blowfish Long
                               3. 3DES       6. AES
  TLS Required. . . . . . . _  Y: Yes        N: No        T: TLS Tunnel

  Notify type . . . . . . . _  1. None    2. TSO     3. Windows 4. ROSCOE
  Userid to be notified .

  Email good notify . . .
  Email fail notify . . .

  Process name. . . . . . . .
  User description. . . . THIS IS AN EXAMPLE
   F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
   F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                 Send to file
  Command ===> ___________________________________________________________ CAFE

  Transaction Information: Request will be Queued to config: *DEFAULT

                                                                   More:    - +
  Hold file transfer. . . N  Y: Yes        N: No
  Priority. . . . . . . . 3  _                  (0-9,  0 is lowest)
  Retry transfer count. . 1                     (0-99, 0 no limit )

  Eligible date and time. __________ ____       (MM/DD/YYYY)  (HHMM)
  Expiration date . . . . __________            (MM/DD/YYYY)
  Retention period. . . . ___                   (1-365 days)

  Checkpoint interval . . ___                 (0-360 min, 0 or blank for none)

  -------RocketStream Parameters-------
  RS Accelerate . . . . . _                   Y:Yes    N:No
  RS Host . . . . . . . . ____________________________________________________
  RS Port . . . . . . . . _____               1024-65535
  RS Protocol . . . . . . _                   P:PDP    U:UDP
  RS Compress . . . . . . _                   Y:Best   F:Fast  D:Dflt  N:No
   F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
   F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

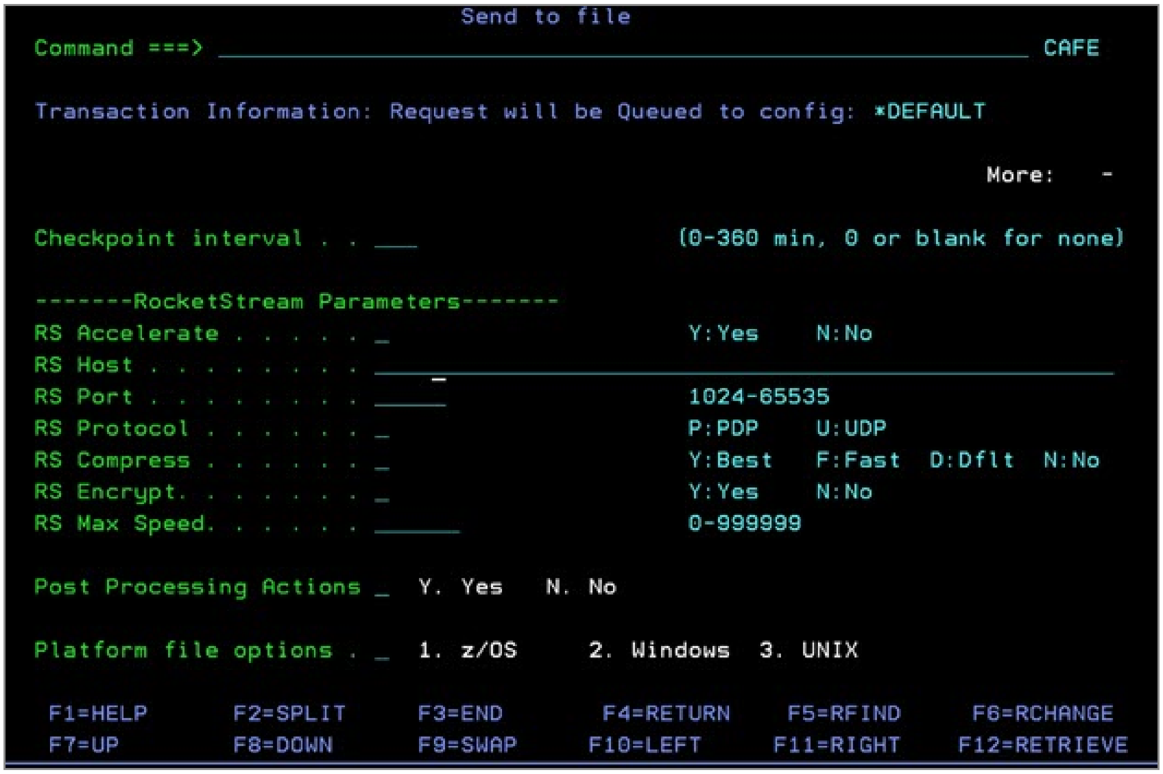
In the "Send to file" panel, some of the parameters are filled in with their default values. You can modify these parameter values at any time by typing values over the default values.

The values you set in the "Send to file" panel will be reserved. Therefore, the values displayed on initialization of the "Send to file" panel are generally based on the last values entered in the entry fields; in this way, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

For more details of the parameters in the "Send to file" panel, see Send Request Parameters.

When you enter all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next, as shown in the following figure.



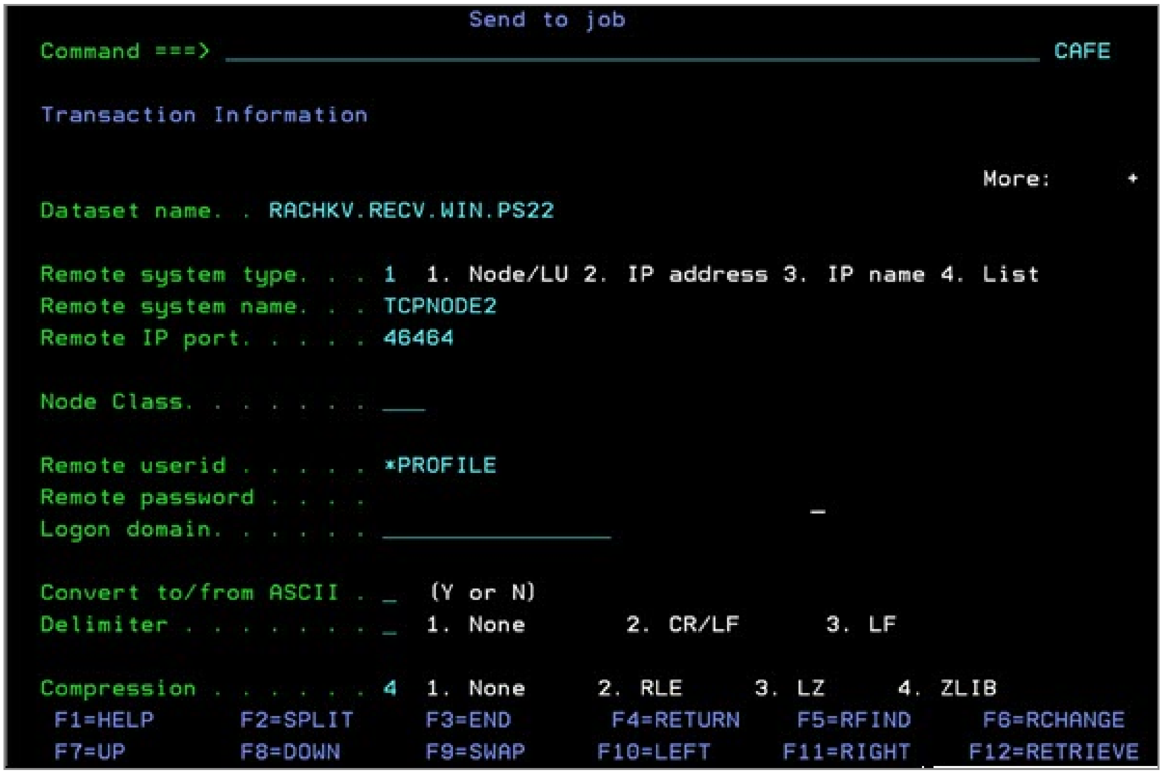
```
                              Send to file
 Command ===> ________________________________________________________ CAFE

 Transaction Information: Request will be Queued to config: *DEFAULT

                                                                 More:   -

 Checkpoint interval . . ___                  (0-360 min, 0 or blank for none)

 -------RocketStream Parameters-------
 RS Accelerate . . . . . . _                  Y:Yes     N:No
 RS Host . . . . . . . . . ____________________________________________________
 RS Port . . . . . . . . . _____              1024-65535
 RS Protocol . . . . . . . _                  P:PDP     U:UDP
 RS Compress . . . . . . . _                  Y:Best    F:Fast    D:Dflt   N:No
 RS Encrypt. . . . . . . . _                  Y:Yes     N:No
 RS Max Speed. . . . . . . ______             0-999999

 Post Processing Actions _  Y. Yes   N. No

 Platform file options . _  1. z/OS     2. Windows  3. UNIX

  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

You can select Y to schedule the transfer, T to schedule the transfer and track the progress, and N to cancel the transfer.

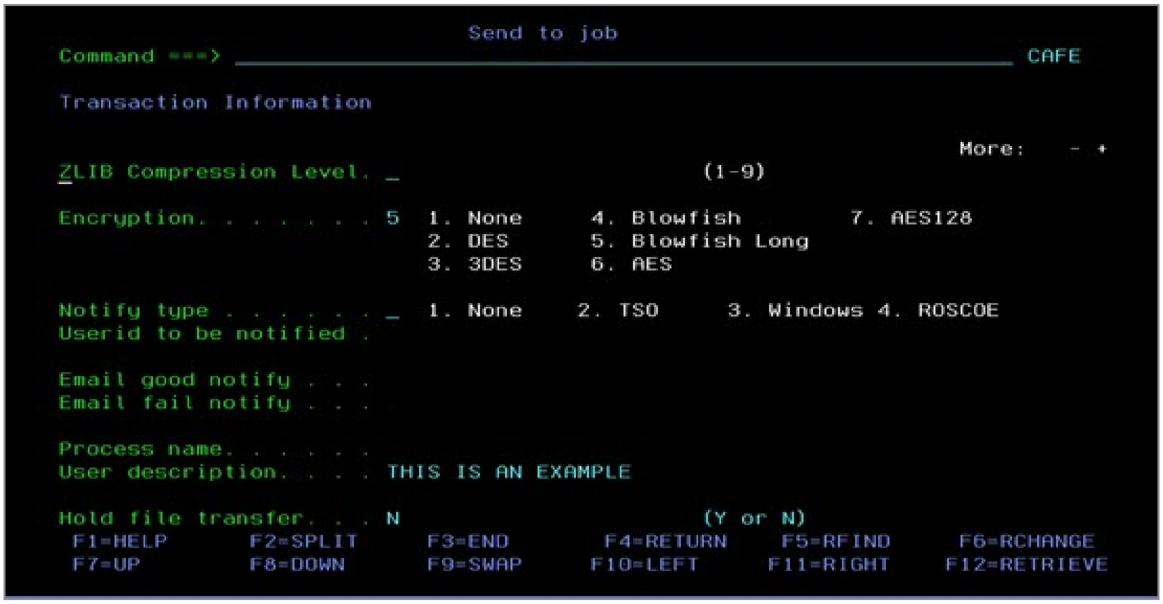
Sending a File to a Job

By typing 3B (or 3.B) on the command line in the Platform Server primary window, you can send a local file to a remote system where the partner executes it as a batch job.

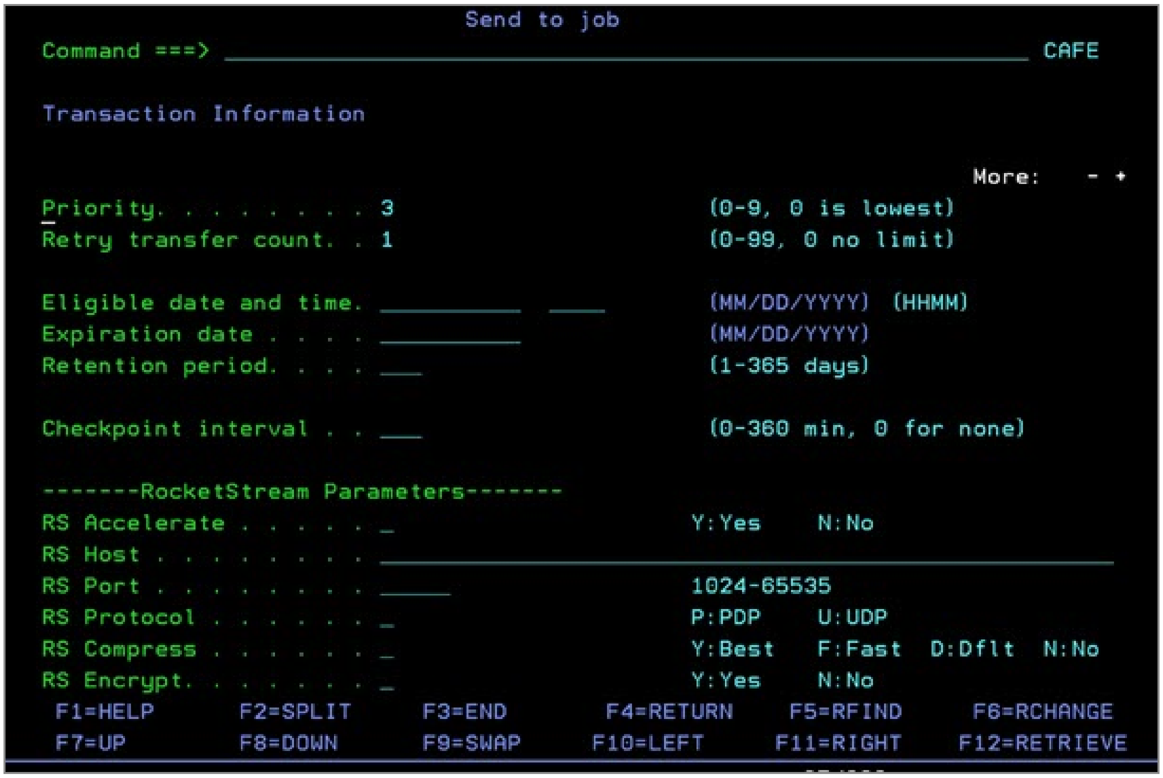
After you type 3B (or 3.B) on the command line, the "Send to job" panel is displayed, as shown in the following figures. The panel contains all the parameters you can configure. You can press the F8 and F7 keys to go to the next and previous pages of the "Send to job" panel.



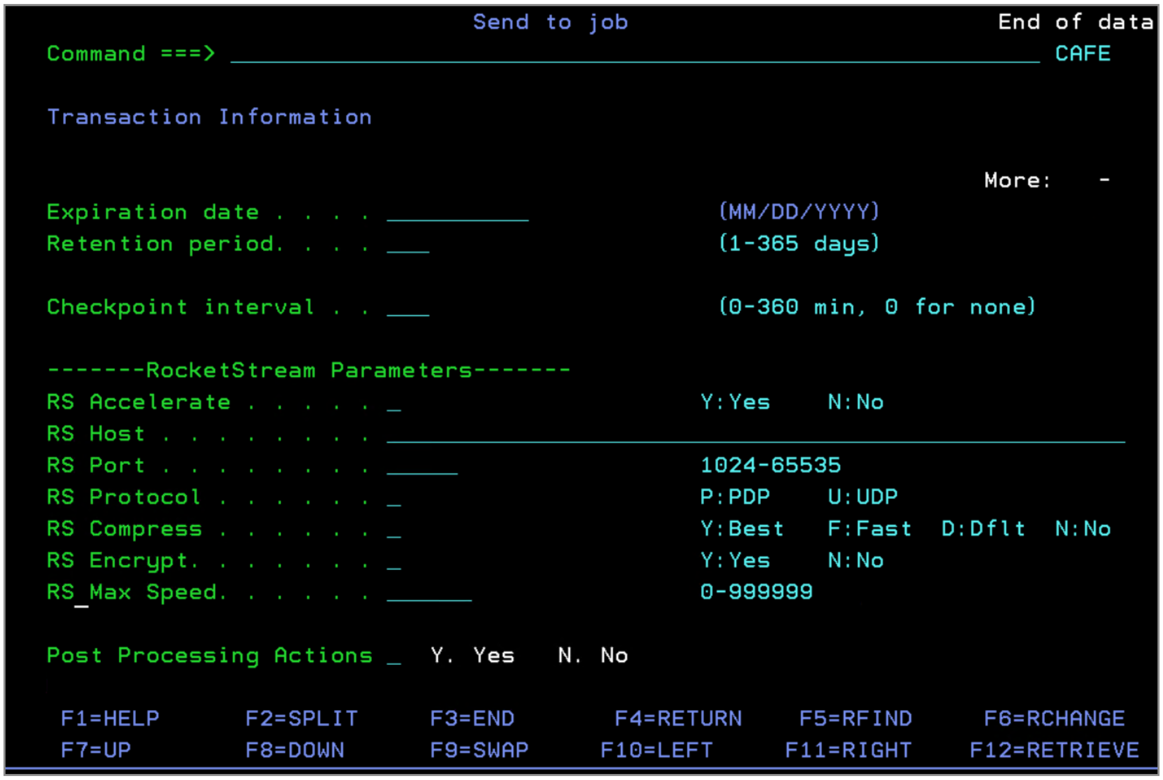
```
                               Send to job
 Command ===> _________________________________________________________ CAFE

 Transaction Information

                                                              More:      +
 Dataset name. . RACHKV.RECV.WIN.PS22

 Remote system type. . . 1  1. Node/LU 2. IP address 3. IP name 4. List
 Remote system name. . . TCPNODE2
 Remote IP port. . . . . 46464

 Node Class. . . . . . . ___

 Remote userid . . . . . *PROFILE
 Remote password . . . .
 Logon domain. . . . . . ________________

 Convert to/from ASCII . _  (Y or N)
 Delimiter . . . . . . . _  1. None      2. CR/LF      3. LF

 Compression . . . . . . 4  1. None     2. RLE     3. LZ     4. ZLIB
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                               Send to job
 Command ===> _______________________________________________________ CAFE

 Transaction Information

                                                             More:    - +
 ZLIB Compression Level. _                       (1-9)

 Encryption. . . . . . . 5  1. None     4. Blowfish         7. AES128
                            2. DES      5. Blowfish Long
                            3. 3DES     6. AES

 Notify type . . . . . . _  1. None    2. TSO     3. Windows 4. ROSCOE
 Userid to be notified .

 Email good notify . . .
 Email fail notify . . .

 Process name. . . . . .
 User description. . . . THIS IS AN EXAMPLE

 Hold file transfer. . . N                      (Y or N)
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                 Send to job
Command ===> ________________________________________________________ CAFE

Transaction Information

                                                              More:    - +
Priority. . . . . . . . 3                      (0-9, 0 is lowest)
Retry transfer count. . 1                      (0-99, 0 no limit)

Eligible date and time. __________ ____        (MM/DD/YYYY) (HHMM)
Expiration date . . . . __________             (MM/DD/YYYY)
Retention period. . . . ___                    (1-365 days)

Checkpoint interval . . ___                    (0-360 min, 0 for none)

-------RocketStream Parameters-------
RS Accelerate . . . . . _                     Y:Yes    N:No
RS Host . . . . . . . . ____________________________________________________
RS Port . . . . . . . . _____                 1024-65535
RS Protocol . . . . . . _                     P:PDP    U:UDP
RS Compress . . . . . . _                     Y:Best   F:Fast  D:Dflt  N:No
RS Encrypt. . . . . . . _                     Y:Yes    N:No
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                 Send to job                     End of data
Command ===> ________________________________________________________ CAFE

Transaction Information

                                                              More:    -
Expiration date . . . . __________             (MM/DD/YYYY)
Retention period. . . . ___                    (1-365 days)

Checkpoint interval . . ___                    (0-360 min, 0 for none)

-------RocketStream Parameters-------
RS Accelerate . . . . . _                     Y:Yes    N:No
RS Host . . . . . . . . ____________________________________________________
RS Port . . . . . . . . _____                 1024-65535
RS Protocol . . . . . . _                     P:PDP    U:UDP
RS Compress . . . . . . _                     Y:Best   F:Fast  D:Dflt  N:No
RS Encrypt. . . . . . . _                     Y:Yes    N:No
RS_Max Speed. . . . . . ______                0-999999

Post Processing Actions _  Y. Yes   N. No

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

In the "Send to job" panel, some of the parameters are filled in with their default values. You can modify these parameter values at any time by typing values over the default values.

The values you set in the "Send to job" panel will be reserved. Therefore, the values displayed on initialization of the "Send to job" panel are generally based on the last values entered in the entry fields; in this way, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

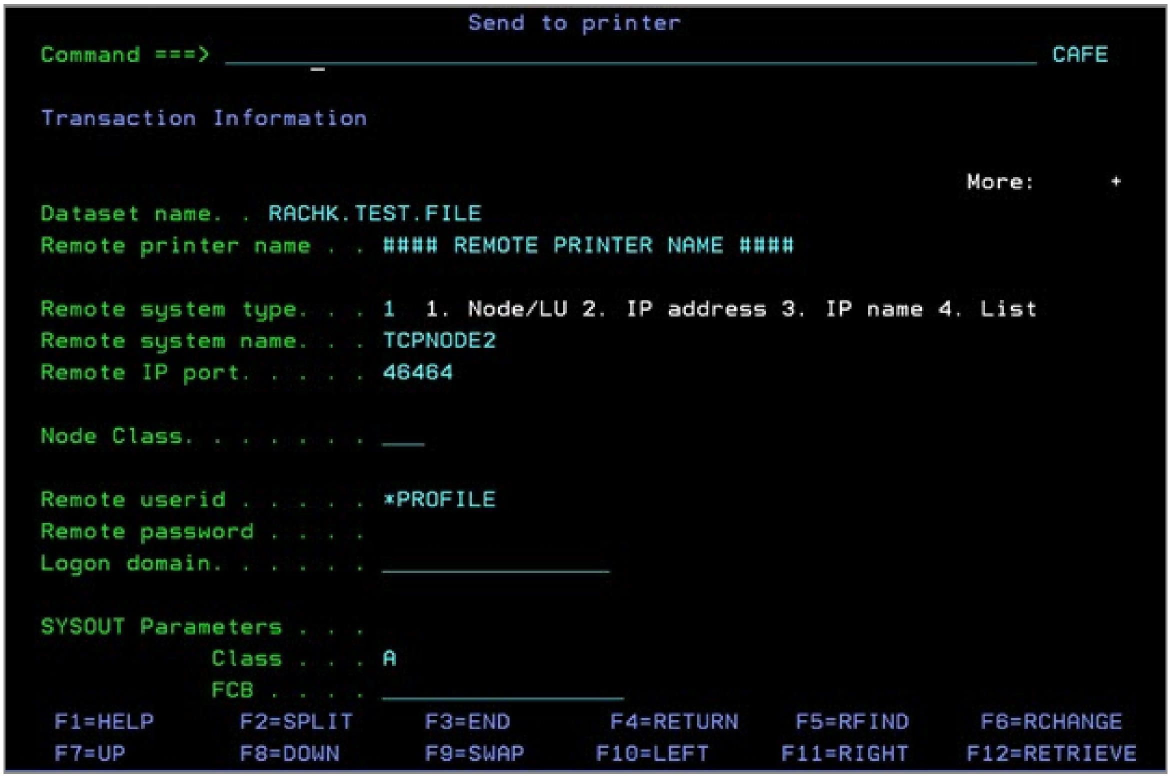
For information on the parameters in the "Send to job" panel see Send Request Parameters.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

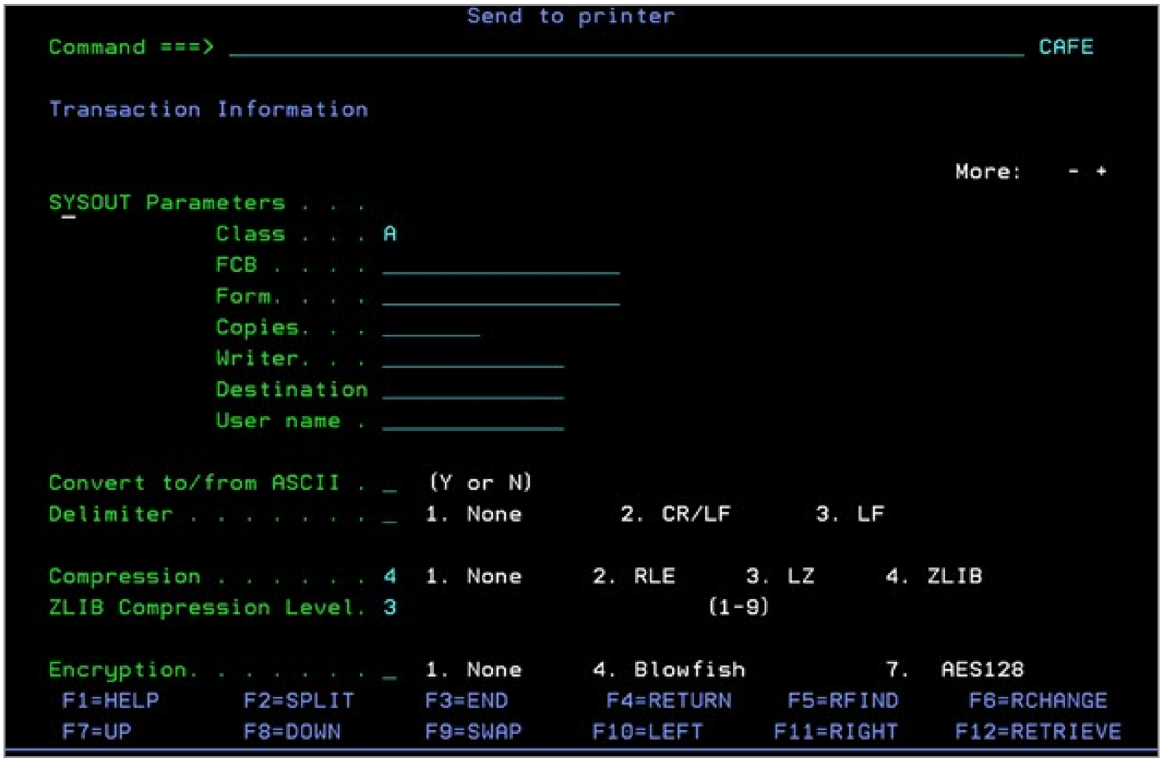
Sending a File to a Printer

By typing 3C (or 3.C) on the command line in the Platform Server primary window, you can send a file to a remote system where the partner executes it as a print job.

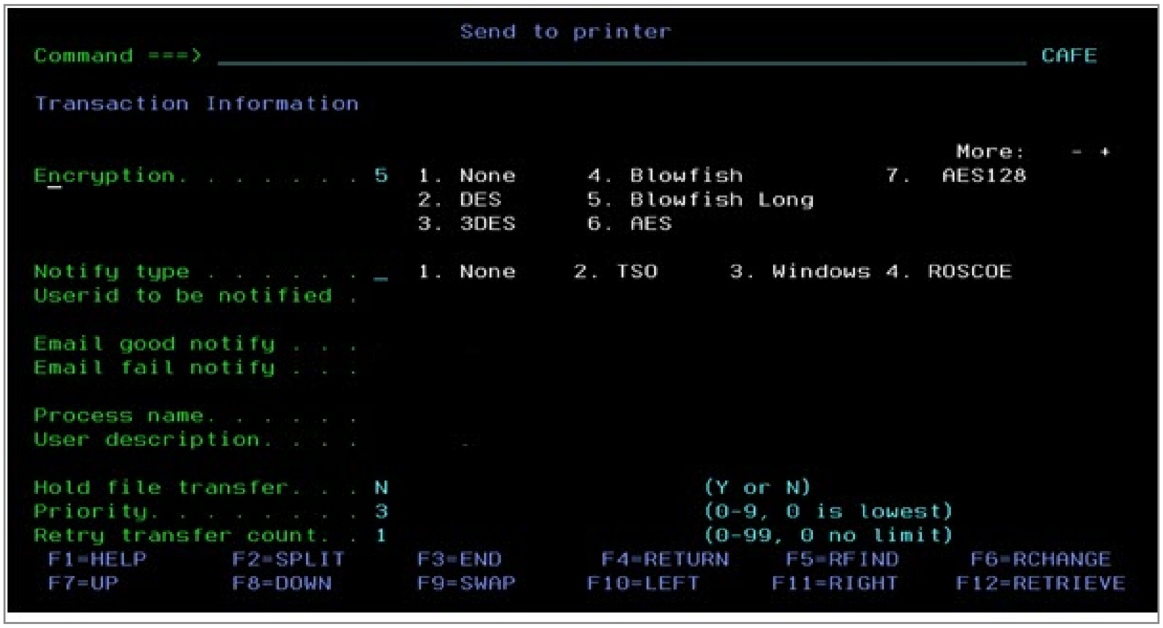
After you type 3C (or 3.C) on the command line, the "Send to printer" panel is displayed, as shown in the following figures. The panel contains all the parameters you can configure. You can press the F8 and F7 keys to go to the next and previous pages of the "Send to printer" panel.



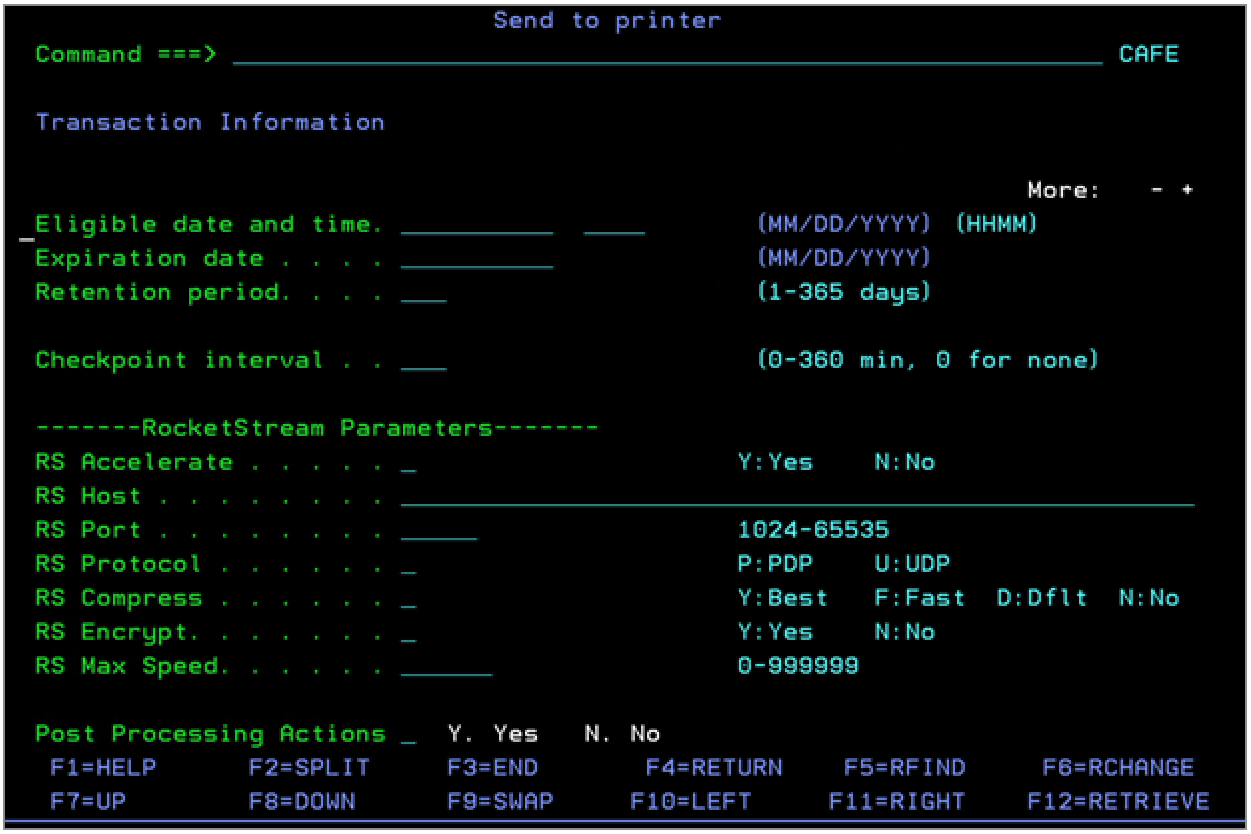
```
                            Send to printer
 Command ===> _______________________________________________________ CAFE

 Transaction Information

                                                            More:     +
 Dataset name. . RACHK.TEST.FILE
 Remote printer name . . #### REMOTE PRINTER NAME ####

 Remote system type. . . 1  1. Node/LU 2. IP address 3. IP name 4. List
 Remote system name. . . TCPNODE2
 Remote IP port. . . . . 46464

 Node Class. . . . . . . ___

 Remote userid . . . . . *PROFILE
 Remote password . . . .
 Logon domain. . . . . . ________________

 SYSOUT Parameters . . .
             Class . . . A
             FCB . . . . _________________
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                            Send to printer
 Command ===> _______________________________________________________ CAFE

 Transaction Information

                                                            More:    - +
 SYSOUT Parameters . . .
             Class . . . A
             FCB . . . . _________________
             Form. . . . _________________
             Copies. . . _______
             Writer. . . _____________
             Destination _____________
             User name . _____________

 Convert to/from ASCII . _  (Y or N)
 Delimiter . . . . . . . _  1. None       2. CR/LF      3. LF

 Compression . . . . . . 4  1. None      2. RLE      3. LZ      4. ZLIB
 ZLIB Compression Level. 3                          (1-9)

 Encryption. . . . . . . _  1. None      4. Blowfish          7.  AES128
  F1=HELP      F2=SPLIT     F3=END      F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP    F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                              Send to printer
Command ===> ____________________________________________________ CAFE

Transaction Information

                                                           More:    - +
Encryption. . . . . . . 5  1. None     4. Blowfish        7.  AES128
                           2. DES      5. Blowfish Long
                           3. 3DES     6. AES

Notify type . . . . . . _  1. None    2. TSO      3. Windows 4. ROSCOE
Userid to be notified .

Email good notify . . .
Email fail notify . . .

Process name. . . . . .
User description. . . .

Hold file transfer. . . N                    (Y or N)
Priority. . . . . . . . 3                    (0-9, 0 is lowest)
Retry transfer count. . 1                    (0-99, 0 no limit)
 F1=HELP     F2=SPLIT     F3=END      F4=RETURN     F5=RFIND     F6=RCHANGE
 F7=UP       F8=DOWN      F9=SWAP    F10=LEFT      F11=RIGHT    F12=RETRIEVE
```



```
                              Send to printer
Command ===> _________________________________________________________ CAFE

Transaction Information

                                                               More:    - +
_Eligible date and time. __________ ____        (MM/DD/YYYY) (HHMM)
 Expiration date . . . . __________             (MM/DD/YYYY)
 Retention period. . . . ___                    (1-365 days)

 Checkpoint interval . . ___                    (0-360 min, 0 for none)

 -------RocketStream Parameters-------
 RS Accelerate . . . . . _                     Y:Yes    N:No
 RS Host . . . . . . . . ____________________________________________________
 RS Port . . . . . . . . _____                 1024-65535
 RS Protocol . . . . . . _                     P:PDP    U:UDP
 RS Compress . . . . . . _                     Y:Best   F:Fast  D:Dflt  N:No
 RS Encrypt. . . . . . . _                     Y:Yes    N:No
 RS Max Speed. . . . . . ______                0-999999

 Post Processing Actions _  Y. Yes   N. No
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN     F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT      F11=RIGHT    F12=RETRIEVE
```

In the "Send to printer" panel, some of the parameters are filled in with their default values. You can modify these parameter values at any time by typing values over the default values.

The values you set in the "Send to printer" panel will be reserved. Therefore, the values displayed on initialization of the "Send to printer" panel are generally based on the last values entered in the entry fields; therefore, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

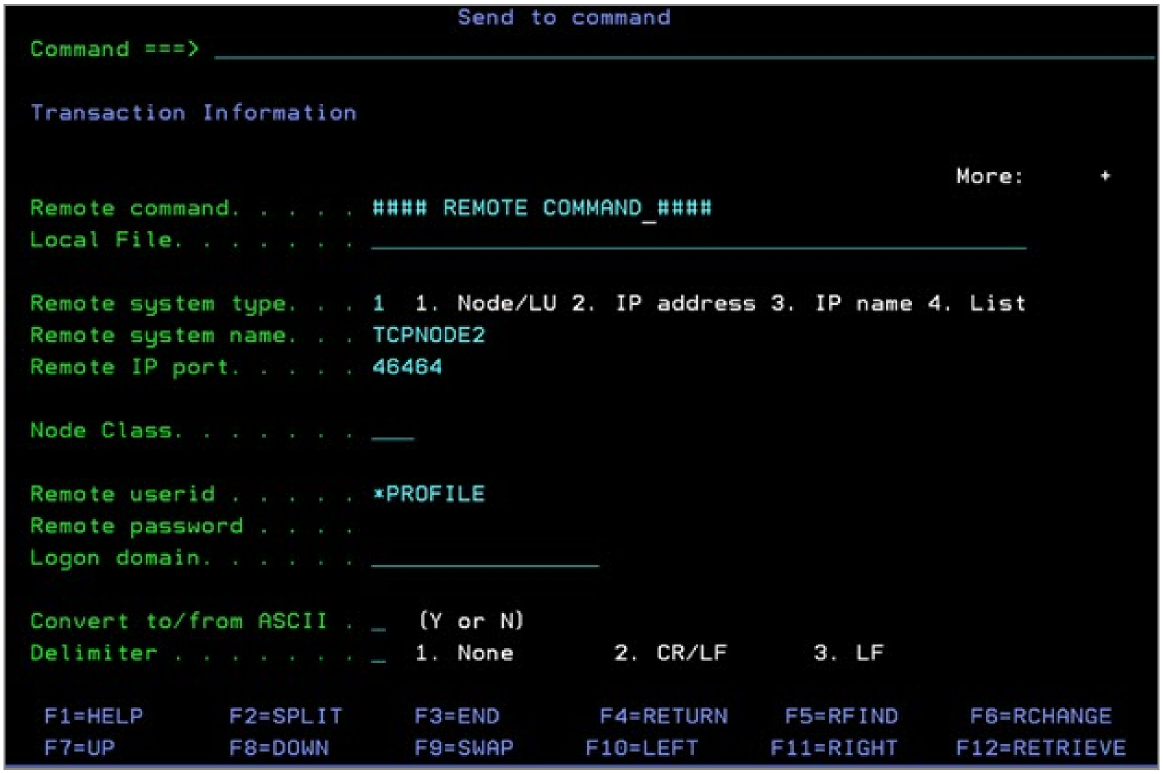
For information on the parameters in the "Send to printer" panel see Send Request Parameters.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

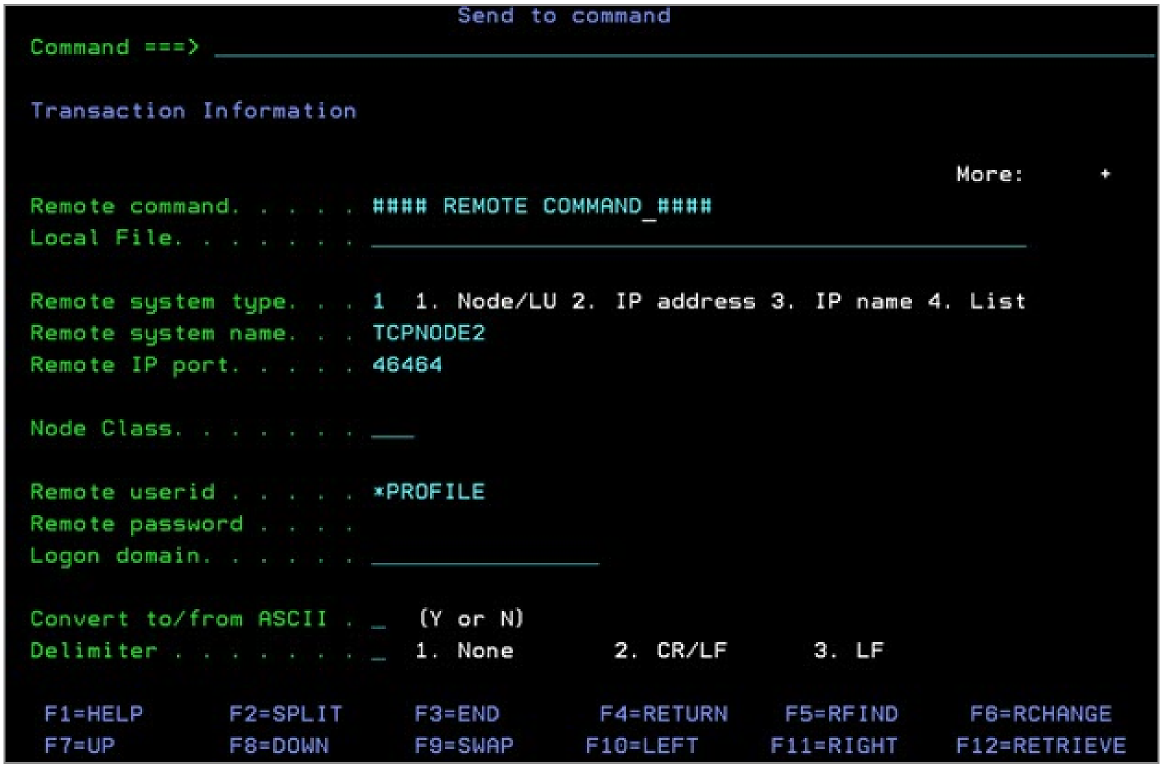
Sending a Command

By typing 3D (or 3.D) on the command line in the Platform Server primary window, you can send a command to the remote system where the partner executes it as a system command. When you send commands to a z/OS system, you cannot execute z/OS system commands; you can only execute scripts or REXX execs.

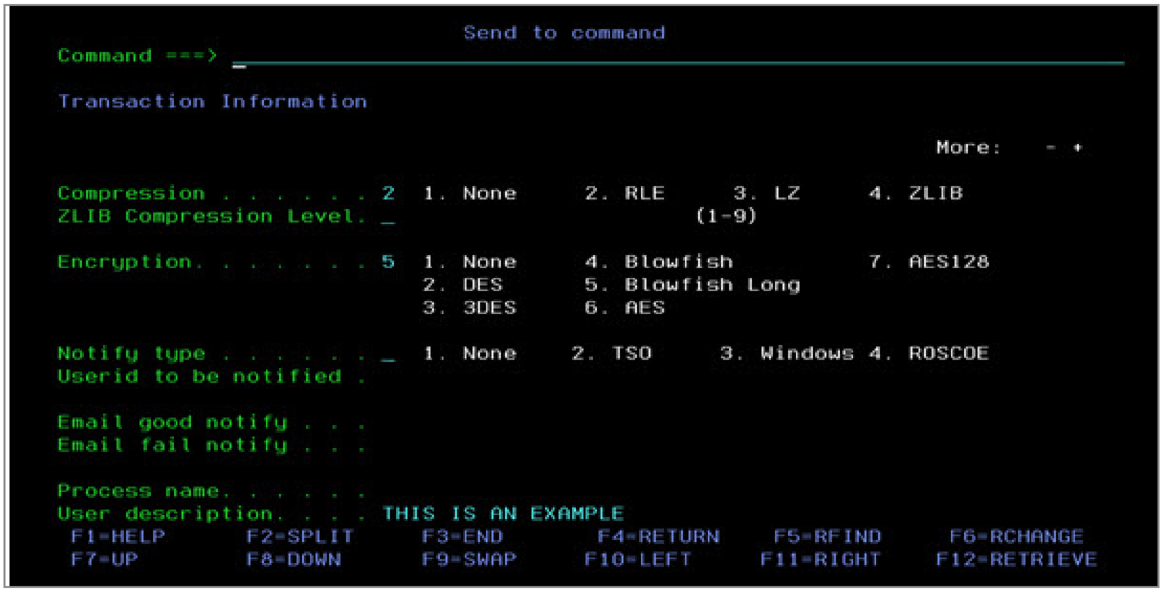
After you type 3D (or 3.D) on the command line, the "Send to command" panel is displayed, as shown in the following figures. The panel contains all the parameters you can configure. You can press the F8 and F7 keys to go to the next and previous pages of the "Send to command" panel.



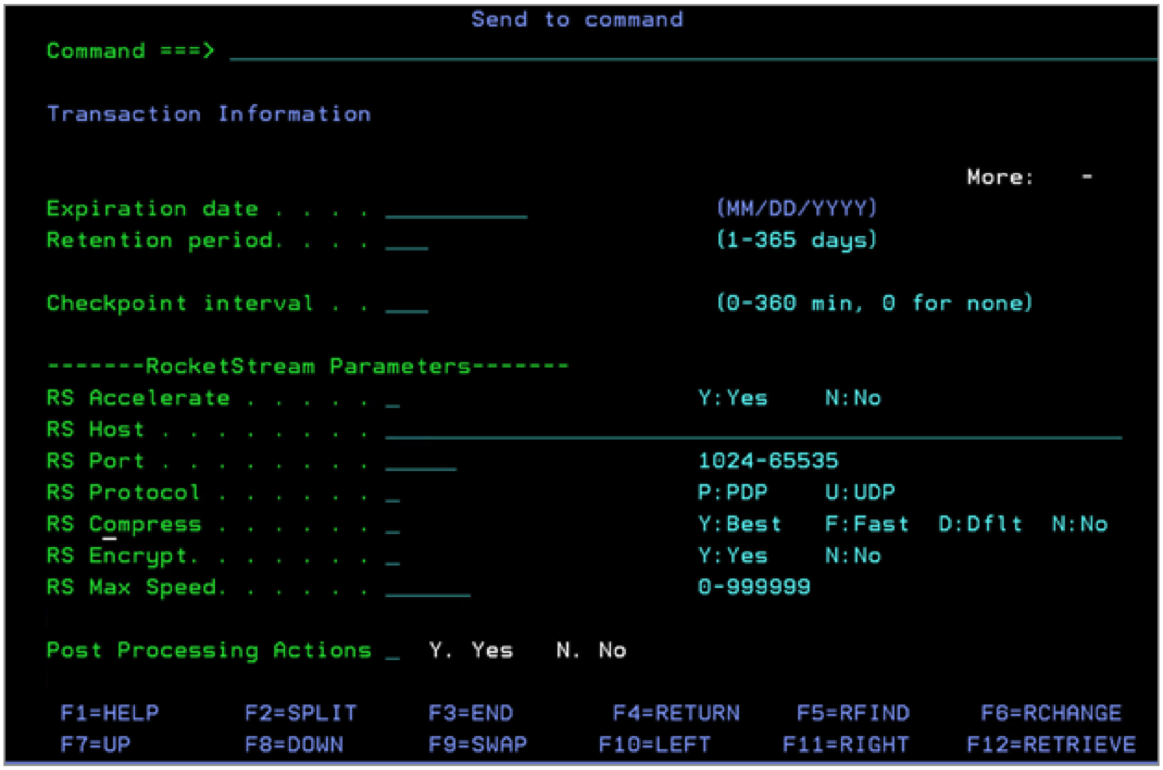
```
                               Send to command
Command ===> ________________________________________________________________

Transaction Information

                                                                More:      +
Remote command. . . . . #### REMOTE COMMAND_####
Local File. . . . . . . _____________________________________________

Remote system type. . . 1  1. Node/LU 2. IP address 3. IP name 4. List
Remote system name. . . TCPNODE2
Remote IP port. . . . . 46464

Node Class. . . . . . . ___

Remote userid . . . . . *PROFILE
Remote password . . . .
Logon domain. . . . . . ________________

Convert to/from ASCII . _  (Y or N)
Delimiter . . . . . . . _  1. None       2. CR/LF      3. LF

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                               Send to command
Command ===> ________________________________________________________________

Transaction Information

                                                                More:      +
Remote command. . . . . #### REMOTE COMMAND_####
Local File. . . . . . . _____________________________________________

Remote system type. . . 1  1. Node/LU 2. IP address 3. IP name 4. List
Remote system name. . . TCPNODE2
Remote IP port. . . . . 46464

Node Class. . . . . . . ___

Remote userid . . . . . *PROFILE
Remote password . . . .
Logon domain. . . . . . ________________

Convert to/from ASCII . _  (Y or N)
Delimiter . . . . . . . _  1. None       2. CR/LF      3. LF

 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                 Send to command
 Command ===> _

 Transaction Information

                                                                  More:    - +

 Compression . . . . . . . 2  1. None     2. RLE     3. LZ      4. ZLIB
 ZLIB Compression Level. _                        (1-9)

 Encryption. . . . . . . . 5  1. None     4. Blowfish          7. AES128
                              2. DES      5. Blowfish Long
                              3. 3DES     6. AES

 Notify type . . . . . . . _  1. None    2. TSO     3. Windows 4. ROSCOE
 Userid to be notified .

 Email good notify . . .
 Email fail notify . . .

 Process name. . . . . .
 User description. . . . THIS IS AN EXAMPLE
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```



```
                                 Send to command
 Command ===>

 Transaction Information

                                                               More:    -
 Expiration date . . . . __________            (MM/DD/YYYY)
 Retention period. . . . ___                   (1-365 days)

 Checkpoint interval . . ___                   (0-360 min, 0 for none)

 -------RocketStream Parameters-------
 RS Accelerate . . . . . _                    Y:Yes     N:No
 RS Host . . . . . . . . ____________________________________________________
 RS Port . . . . . . . . _____                1024-65535
 RS Protocol . . . . . . _                    P:PDP     U:UDP
 RS Compress . . . . . . _                    Y:Best    F:Fast   D:Dflt   N:No
 RS Encrypt. . . . . . . _                    Y:Yes     N:No
 RS Max Speed. . . . . . ______               0-999999

 Post Processing Actions _  Y. Yes   N. No

  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

In the "Send to command" panel, some of the parameters are filled in with their default values. You can modify these parameter values at any time by typing values over the default values.

The values you set in the "Send to command" panel will be reserved. Therefore, the values displayed on initialization of the "Send to command" panel are generally based on the last values entered in the entry fields; in this way, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

For information on the parameters in the "Send to command" panel see Send Request Parameters.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

Send Request Parameters

The following table lists the send request parameters:

Parameter	Description
`Checkpoint interval`	The Platform Server checkpoint feature uses an interval of time when determining when a checkpoint should be sent. Because the checkpointing is time-based, the checkpoint always occurs at a regular interval. Checkpoint interval is specified in minutes and has a valid range of 1 - 90 minutes. The default is 5 minutes.
`Class of Service`	This parameter defines the number of bytes transmitted by a node.
`Compression`	This parameter defines the compression algorithm used for this transfer. Valid values are: • `NONE`: Indicates that compression is not used for this transfer. It is a good practice to specify `COMPRESS=NO` on high-speed links because compression actually slows down file transfers. • `RLE`: Indicates that RLE compression is used for this

Parameter	Description
	transfer. RLE compression is more data-dependent. That is, the compression ratio might vary widely based upon the type of data being compressed. If network bandwidth is not a critical bottleneck for your network and you must save CPU cycles, choose `RLE`. • `LZ`: Indicates that LZ compression is used for this transfer. LZ compression provides better compression ratios than RLE compression. LZ compression also compresses a variety of data types better. If you need better compression ratios and you can spare CPU cycles, choose `LZ`. • `ZLIB`: Indicates that ZLIB compression is used for this transfer. Level 1 provides fast compression but low compression ratio. Levels 7 to 9 produce the best compression quality but are slower. Typically, ZLIB2 offers the best compromise between compression quality and speed. For more information on data compression of the Platform Server, see Data Compression.
`Convert to/from ASCII`	This parameter is used to convert data between ASCII and EBCDIC. Set this parameter to `Yes` when communicating with systems with defined data structures.
`Dataset name`	This parameter defines the name of the local dataset for this transaction. The dataset name must be a valid z/OS dataset and cannot exceed 255 characters in length. If the dataset provided is invalid, the interface prompts you for a valid dataset name. To enter a long file name, a plus sign (+) must be entered as the last character in the dataset name field; an additional panel is displayed in which you can enter a long file name. By default, the interface displays the last dataset entered for any transaction.

Parameter	Description
	The following file types are supported: - Sequential files - PDS': Transfer of entire PDS, members of a PDS based on wildcards, or individual members. - GDGs: When specifying Generation Data Groups (GDG), you can specify a generation dataset using either its true name (that is, with the `GOO–VOO` qualifier) or the relative generation number. The alias name is turned into the true name at the time the file transfer is scheduled. For example, if you specify generation number -1, the dataset transferred is the one that was -1 at queuing time, not necessarily at transmission time. - VSAM files (KSDS, RRDS, ESDS): This access is transparent to the end user.
`Delimiter`	Valid values are: - `NONE`: Indicates that no CRLF or LF translation should be performed during the transfer. - `CRLF`: Indicates that Carriage Return/Line Feed translation should be performed during the transfer. This parameter has no effect when sent with `DATA_TYPE=B` (Binary). - `LF`: Indicates that only Line Feed translation should be performed during the transfer.
`Eligible date and time`	This parameter defines the Julian date and the time of the day the transaction is first eligible for execution. The format of date is MM/DD/YYYY; for example, 01/10/2008 is January 10, 2008. The format of time is HHMM.
`Email fail notify`	This parameter defines the email address to be used when the transfer fails.

Parameter	Description
`Email good notify`	This parameter defines the email address to be used when the transfer ends successfully.
`Encryption`	This parameter defines the level of encryption used for this transaction only. If specified, this parameter overrides the `ENCRYPT` parameter specified in the GLOBAL or NODE definitions for this transfer. Valid values are as follows: • `NONE`: Indicates that encryption is not used for this transfer. • `AES`: Indicates that AES 256-bit encryption is used for this transfer. • `AES128`: Indicates that AES 128-bit encryption is used for this transfer. • `DES`: Indicates that DES encryption is used for this transfer. • `3DES`: Indicates that triple DES encryption is used for this transfer. • `BF|BLOWFISH`: Indicates that Blowfish 56-bit encryption is used for this transfer. • `BF|BLOWFISH_LONG`: Indicates that Blowfish 448-bit encryption is used for this transfer. • `AES`: Indicates that AES 256-bit encryption is used for this transfer. • `NEVER`: Indicates that no encryption is used for this node. Even if you override the `ENCRYPT` option in the batch interface, encryption is still turned off for this node.
`Expiration date`	This parameter defines the date at which the file transfer will be purged off of the work queue if the transfer is not yet completed.

Parameter	Description
	The default value is 30 days from now.
`Hold file transfer`	This parameter defines whether to place the file transfer in the queue on hold until such time that it is ready for transfer. To execute the transaction, you must release it from the queue.
`Input Unit`	This parameter defines the unit name for the input tape file. It can only be used when sending uncataloged tape files. When this parameter is specified, you must also specify the `Input Volume` parameter and `TAPE=YES`. **Note:** This parameter is ignored for all receive transfers or send transfers of which the input dataset is cataloged or on a disk.
`Input Volume`	This parameter defines the volume name that is used for the input dataset of a transfer. When this parameter is used, the catalog is not used to allocate the input dataset. The Platform Server verifies that the dataset exists on the `Input Volume` specified. This parameter is valid only when the Platform Server for z/OS is sending a file to a remote system, or is receiving a file from a Platform Server for z/OS. **Note:** Use this parameter with extreme care, because most sites use only cataloged datasets.
`Local Conversion File`	This parameter defines the name of the personalized conversion file to be used for ASCII/EBCDIC conversions when sending files to the defined remote system.
`Local Userid`	This parameter defines the local user ID of the user that is submitting the transaction.

Parameter	Description
`Local password`	This field holds the password of the local user ID.
`Logon domain`	This parameter defines the Windows domain to be used to authenticate a particular file transfer request that is initiated by the mainframe. This field is Windows-specific and is only valid when sending requests to a Windows machine.
`Member Include/Exclude`	By setting this parameter to `Y`, you can include or exclude PDS or PDSE file members in a send or receive transfer conducted between two z/OS systems. Up to 25 member names can be defined by this parameter. This parameter is ignored when the dataset names on both ends of the file transfer are not PDS or PDSE files. When you set this parameter to `Y` and press Enter, the Member Include/Exclude screen is displayed. See Member Include/Exclude for more details.
`Node Class`	This parameter defines the node class that is used for this file transfer request. This is only used when the NODE definition associated with the request defines the `DEFAULT_NODECLASS` parameter with a non-zero value. If the transfer request does not use a NODE definition, or if the NODE definition specifies `DEFAULT_NODECLASS=0`, `NODECLASS` processing will not be performed. If you define the `NODECLASS` parameter with a value higher than that of the `NODE WINNERS` parameter, the request will be flagged as an error and the request will be retried at the next error interval.
`Notify type`	This parameter defines the type of user ID to notify after a file transfer terminates. This is used in conjunction with the `User ID to be notified` parameter.

Parameter	Description
`Output Volume`	This parameter defines the volume name that is used for the output dataset of a transfer. If this parameter is used, the catalog is not used to allocate the output dataset. The Platform Server verifies that the dataset exists on the `Output Volume` specified. Use this parameter with extreme care, because most sites use only cataloged datasets. If you use this parameter with a cataloged dataset and specify a different volume than the one the cataloged dataset resides on, a duplicate uncataloged dataset will be created on the `Output Volume` volume. This parameter is valid only when the Platform Server for z/OS is receiving a file from a remote system, or is sending a file to a Platform Server for z/OS.
`Platform file options`	Based on the remote platform from which you are sending a file to or receiving a file, your file options differ. If you choose `z/OS`, the z/OS Options panel is displayed. See z/OS Options for details. If you choose `Windows`, the Windows Options panel is displayed. See Windows Options for details. If you choose `UNIX`, the UNIX Options panel is displayed. See UNIX Options for details.
`Post Processing Action (PPA)`	This parameter specifies the conditions when a postprocessing action should occur based on success or failure of the transfer, whether it should be executed locally or remotely, the type of command and the data that should be executed.
`Priority`	This parameter defines the execution priority assigned to the transaction when it is placed in the queue. Transactions with a higher priority are executed before those with a lower priority. The valid values are `0` - `9`; wherein, `0` represents the lowest priority.

Parameter	Description
`Process name`	This parameter defines an 8 character process name.
`RS Accelerator`	Setting this parameter to `Y` will force a transfer to be conducted through a Platform Server for Windows platform TIBCO Accelerator server. By using the TIBCO Accelerator technology, you can greatly improve data transfer speeds over IP networks with high latency. **Note:** You must be licensed for RSAccelerator (RSA) to use this technology.
`RS Compress`	When conducting file transfers through an RSAccelerator (RSA), you can configure the RSA server to compress the data being transferred. RSA uses a proprietary compression compatible with ZLIB. If you set the compression to `D` (Default), the file receives the greatest compression and might take slightly longer to transfer. If you set the compression to `F` (Fast), the file is less compressed but is sent out faster.
`RS Encrypt`	When conducting file transfers through an RSAccelerator (RSA), you can define the RSA server to encrypt the data with a 256-bit Blowfish encryption key by setting this parameter `Y` (Yes).
`RS Host`	This parameter defines the IP address or host name of the Platform Server for Windows platform TIBCO Accelerator server.
`RS Max Speed`	When conducting file transfers through an RSAccelerator (RSA), you can set the maximum speed in kilobytes per second to be used by the RSA server.
`RS Port`	This parameter defines the port number of the Platform Server for Windows platform TIBCO Accelerator server is listening on for transfers using the TIBCO Accelerator technology.
`RS Protocol`	When conducting file transfers through an RSAccelerator (RSA),

Parameter	Description
	you can define the RSA server to use its own enhanced version of User Datagram Protocol (UDP), TIBCO Accelerator's parallel implementation of TCP, called Parallel Delivery Protocol (PDP), or straight TCP.
`Receiving File Disp: Remote File Options`	Valid values are: • `Create`: Creates a file on the remote system with the attributes and characteristics as specified. If the file already exists on the remote system, the transaction fails. • `Replace`: Replaces the contents of the destination file with the contents of the source file. The characteristics and attributes of the remote file remain unchanged. If the file does not exist, the transaction fails. • `Append`: Appends the contents of the source file to the end of the destination file. The characteristics and attributes of the remote file remain unchanged. If the file does not exist, the transaction fails. • `Create/Replace`: If the file does not exist on the remote open system, creates the file; if the file does exist, replaces its contents with the contents of the source file. The characteristics and attributes of the remote file remain unchanged. • `Create/Append`: If the file does not exist on the remote system, creates the file; if the file does exist, appends the contents of the source file to the end of the destination file. The characteristics and attributes of the remote file remain unchanged. • `Create/Replace New`: Creates the remote file or replaces it with new attributes. Only specify this option when transferring files with ICL VME mainframes.
`Receiving File Disp:Remote Error Status`	This parameter defines the disposition of the remote file when the transfer ends in error.

Parameter	Description
`Receiving File Disp:Remote Normal Status`	This parameter defines the disposition of the remote file when the transfer ends successfully.
`Remote Conversion File`	This parameter defines the name of the remote conversion file to be used for ASCII/EBCDIC text conversions when sending a file to the defined remote system.
`Remote IP port`	If the remote system type is an IP name or IP address, you can enter the IP port on which the system is listening. If the remote system type is a node, LU name, or List, this field is ignored. The valid values are `0` - `65535`.
`Remote file name`	This parameter defines the name by which a file is known at the remote side. The file name can include up to 255 characters (including embedded spaces). If the remote file name contains embedded spaces, the file name must be enclosed in double quotation marks. To enter a long file name, a plus sign (+) must be entered as the last character in the file name field; after that, an additional panel is displayed in which you can enter a long file name. **Note:** By default, the interface displays the last file name entered for any transaction.
`Remote password`	This field holds the password of the remote user ID used to log on to the remote Platform Server. **Note:** This field is not displayed and has no default value. The remote password can be up to 20 characters in length. When the remote system is a Platform Server for z/OS, you can optionally change the password of the remote user ID by typing the old password followed by a slash and the new

Parameter	Description
	password. For example, `Remote Password. . . .oldpass/newpass`.
`Remote system name`	Based on what you configure for the `Remote system type` field, you can configure this field with either the node name being used for another PS remote system, an LU name for the remote Platform Server, the IP address or the IP name for a remote PS system. **Note:** Nodes are defined in members of the Platform Server resource definition library named CONFIG.
`Remote system type`	This parameter is required. It defines how the Platform Server communicates with the remote system. Valid values are: - Node/LU - IP address - IP name - List This field is used in conjunction with the `Remote system name` field. **Note:** If you want to use the DNS lookup feature (`IPNAME=...`), you must concatenate the IBM C Runtime libraries in the STEPLIB. Also, the SYSTCPD DD name must be in the Platform Server JCL and must point to the TCPIP DATA configuration file.
`Remote userid`	This parameter defines the user ID for the remote system or the name by which the issuer is known to the remote system. This field is not required by the Platform Server. This parameter defaults to the last user ID entered in this field

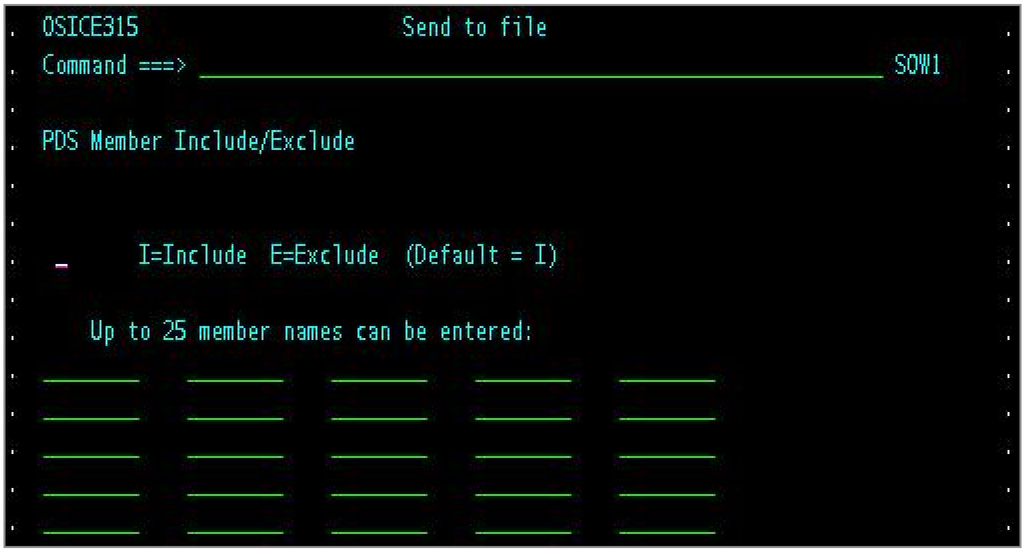
Parameter	Description
	and is restricted to 20 characters in length. If you specify `*PROFILE`, the Platform Server checks the PROFILE dataspace for a profile for this user ID that matches the NODE/IPNAME/IPADDR/LIST names.
`Remove Trailing Spaces`	When transmitting text files to remote systems, you can set this parameter to `Yes` to remove all trailing spaces and nulls before the file is transmitted. In this way, the transfer runs faster, and takes up less disk space on the remote system. This parameter is only in effect when communicating with Platform Server for Windows and Platform Server for UNIX systems. It is ignored when communicating with Platform Server for z/OS or for binary transfers.
`Retention period`	This parameter defines the number of days after the eligible date that the transaction expires and is purged from the queue. When this field is specified, its value overrides the value of the `Expiration date` parameter.
`Retry transfer count`	This parameter defines the number of attempts that the Platform Server makes to complete this transaction. The valid values are `0` - `99`; wherein, `0` means no limit.
`Sending File Disp: Local Error Status`	This parameter defines the disposition of the local file when the transfer ends in error.
`Sending File Disp: Local File Options`	Valid values are: • `Create`: Creates a file on the remote system with the attributes and characteristics as specified. If the file already exists on the remote system, the transaction fails. • `Replace`: Replaces the contents of the destination file

Parameter	Description
	with the contents of the source file. The characteristics and attributes of the remote file remain unchanged. If the file does not exist, the transaction fails. • `Append`: Appends the contents of the source file to the end of the destination file. The characteristics and attributes of the remote file remain unchanged. If the file does not exist, the transaction fails. • `Create/Replace`: If the file does not exist on the remote open system, creates the file; if the file does exist, replaces its contents with the contents of the source file. The characteristics and attributes of the remote file remain unchanged. • `Create/Append`: If the file does not exist on the remote system, creates the file; if the file does exist, appends the contents of the source file to the end of the destination file. The characteristics and attributes of the remote file remain unchanged. • `Create/Replace New`: Creates the remote file or replaces it with new attributes. Only specify this option when transferring files with ICL VME mainframes.
`Sending File Disp: Local Normal Status`	This parameter defines the disposition of the local file when the transfer ends successfully.
`TLS Required`	This parameter defines whether Secure Sockets is used for this request. **Note:** This definition overrides the GLOBAL or NODE definitions. If you specify `YES`, the Platform Server attempts to use SSL to authenticate both ends of the connection. When you specify `NO`, the Platform Server does not use SSL authentication. The default value is `NO`.

Parameter	Description
	By specifying TUNNEL, all transfer data will be sent over an encrypted TLS Tunnel connection.
`Tape`	This parameter notifies the Platform Server that the request is for a tape. It can only be used when sending uncataloged tape files. The default value is `NO`. When this parameter is specified, you must also specify the `Input Unit` and `Input Volume` parameters. **Note:** This parameter is ignored for all receive transfers or send transfers of which the input dataset is cataloged or on a disk.
`User description`	This parameter defines an optional 25-character description of the transaction being processed. This is descriptive data that might be supplied with the transaction.
`Userid to be notified`	This parameter defines the user ID to be notified with transaction information or error messages. The user ID is limited to 8 characters in length.

Member Include/Exclude

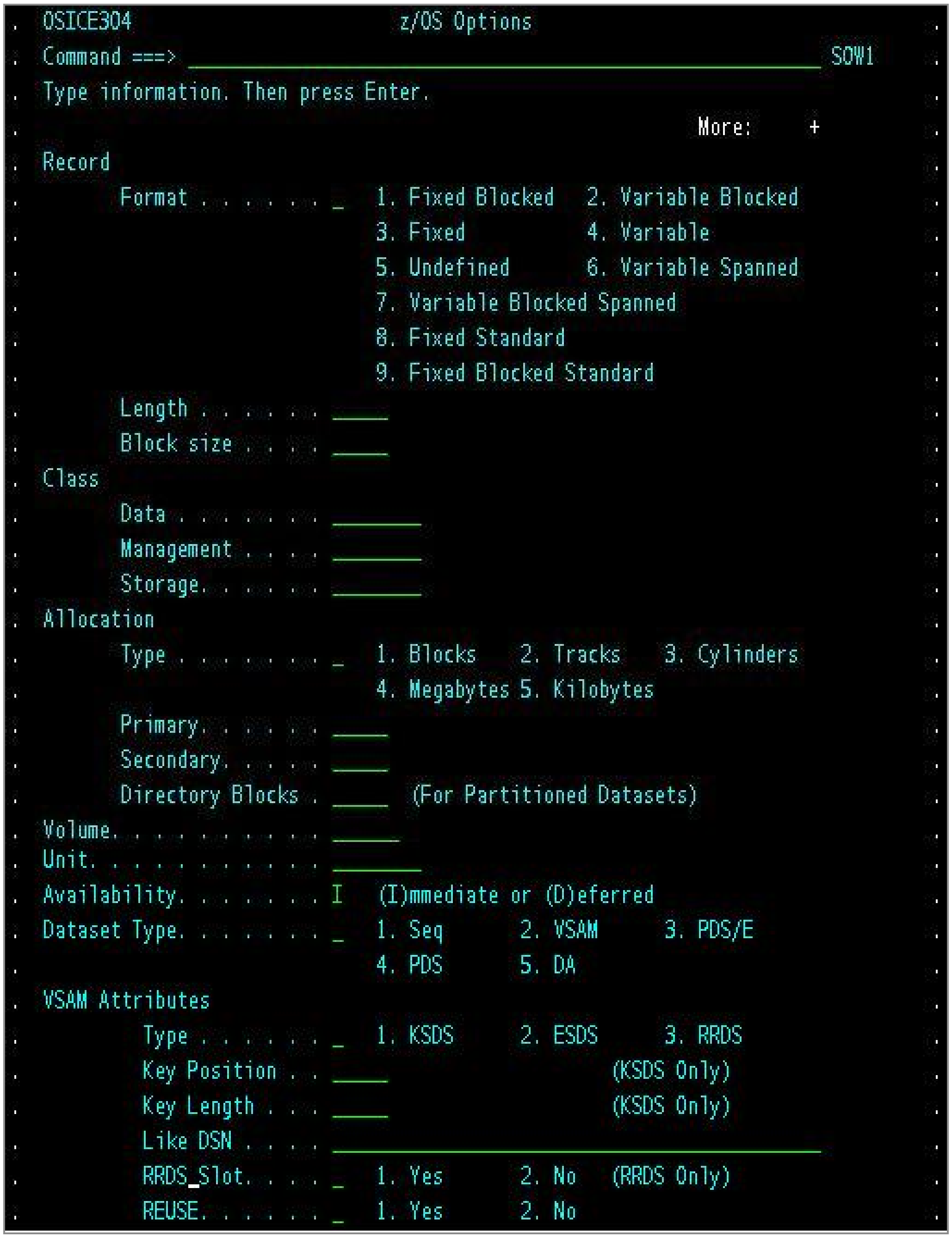
The following screen is displayed when you set the `Member Include/Exclude` parameter to `Y`. In this screen, you can define to include or exclude PDS or PDSE file members in a send or receive transfer conducted between two z/OS systems.



```
OSICE315                       Send to file
Command ===> ________________________________________________________ SOW1

PDS Member Include/Exclude


 _      I=Include  E=Exclude  (Default = I)

    Up to 25 member names can be entered:
________    ________    ________    ________    ________
________    ________    ________    ________    ________
________    ________    ________    ________    ________
________    ________    ________    ________    ________
________    ________    ________    ________    ________
```

z/OS Options

The following panel is displayed when you set the `Platform file options` parameter to `z/OS`.



```
 OSICE304                        z/OS Options
 Command ===> _________________________________________________________ SOW1
 Type information. Then press Enter.
                                                            More:     +
 Record
        Format . . . . . . _   1. Fixed Blocked    2. Variable Blocked
                               3. Fixed            4. Variable
                               5. Undefined        6. Variable Spanned
                               7. Variable Blocked Spanned
                               8. Fixed Standard
                               9. Fixed Blocked Standard
        Length . . . . . . _____
        Block size . . . . _____
 Class
        Data . . . . . . . ________
        Management . . . . ________
        Storage. . . . . . ________
 Allocation
        Type . . . . . . . _   1. Blocks    2. Tracks    3. Cylinders
                               4. Megabytes 5. Kilobytes
        Primary. . . . . . _____
        Secondary. . . . . _____
        Directory Blocks . _____  (For Partitioned Datasets)
 Volume. . . . . . . . . ______
 Unit. . . . . . . . . . ________
 Availability. . . . . . I   (I)mmediate or (D)eferred
 Dataset Type. . . . . . _   1. Seq       2. VSAM      3. PDS/E
                             4. PDS       5. DA
 VSAM Attributes
          Type . . . . . . _   1. KSDS      2. ESDS      3. RRDS
          Key Position . . _____                 (KSDS Only)
          Key Length . . . _____                 (KSDS Only)
          Like DSN . . . . ____________________________________________
          RRDS_Slot. . . . _   1. Yes       2. No    (RRDS Only)
          REUSE. . . . . . _   1. Yes       2. No
```

The following table lists the parameters in the z/OS Options panel:

Parameter	Description
`Record: Format`	This parameter defines the record format of the dataset being created. The default is `Variable`.

Parameter	Description
	Valid values are: - `Fixed Blocked`: All blocks and all logical records are fixed in size and each block includes one or more logical records. - `Vareach blockiable Blocked`: Blocks and logical record length can be of any size. Each block includes one or more logical records. - `Fixed`: Each record contains exactly the number of characters defined by the record length. - `Variable`: The length of each record is less than or equal to the record length. - `Undefined`: Blocks are of variable size. The blocks do not include any logical records. The logical record length is displayed as zero. You must also specify the `Block size` parameter if you specify `Undefined`.
`Record: Length`	This parameter defines the logical record length, in bytes, of the records to be stored in the dataset. The default value is `0`.
`Record: Block size`	This parameter defines the size of the block. For FB, the block size must be a multiple of the record length. For VB, the record length can be any size up to the block size minus four. The maximum value is `32760`. If you set this parameter to the default value, `0`, the operating system determines the optimum block size.
`Class: Data`	This parameter represents the z/OS data class as defined to the Data Facility/System Managed Storage. In addition, it is used to indirectly select file attributes such as record format and logical record length. This parameter can contain 1 - 8 characters, including numeric, alphabetic, or national characters ($, #, or @). The first character must be an alphabetic or national character.

Parameter	Description
`Class: Management`	This parameter represents the z/OS management class as defined to the Data Facility/System Managed Storage. This parameter can contain 1 - 8 characters, including numeric, alphabetic, or national characters ($, #, or @). The first character must be an alphabetic or national character.
`Class: Storage`	This parameter represents the z/OS storage class as defined to the Data Facility/System Managed Storage, which is used to indicate the media type for the host file and the installation backup, restore and archive policies. This parameter can contain 1 - 8 characters, including numeric, alphabetic, or national characters ($, #, or @). The first character must be an alphabetic or national character.
`Allocation: Type`	Valid values are: • `Tracks`: If dataset size is expressed in tracks • `Cylinders`: If dataset size is expressed in cylinders • `Megabytes`: If dataset size is expressed in megabytes • `Kilobytes`: If dataset size is expressed in kilobytes
`Allocation: Primary`	This parameter defines the primary allocation quantity used by the z/OS partner when creating datasets as the initial number of units of tracks, cylinders, and so on to allocate.
`Allocation: Secondary`	This parameter defines the secondary allocation quantity used by the z/OS partner when creating datasets as the next number of units of tracks, cylinders, and so on to allocate when the initial space in the dataset has been exhausted.
`Allocation: Directory Blocks`	Only specify this field when creating a PDS (Partitioned dataset). This parameter specifies the number of 256-byte directory blocks for a PDS.

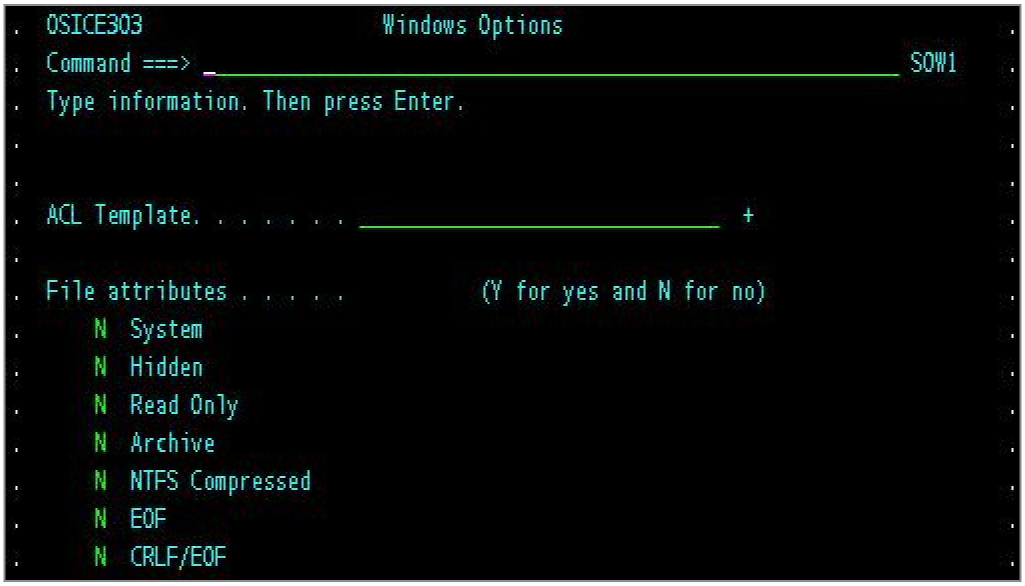
Parameter	Description
	Note: When creating a sequential dataset, leave this field blank.
`Volume`	This parameter defines the volume name of the disk drive on which the z/OS dataset is to be allocated. It can contain 1 - 6 characters.
`Unit`	This parameter defines the name of the type of unit where the host dataset is to be allocated. It can contain 1 - 8 characters.
`Availability`	This parameter defines the availability of the file on the target system. Valid values are: • `I` (Immediate): Immediate availability. • `D` (Deferred): Deferred availability.
`Dataset type`	This parameter defines the type of file to be created. By default, the file type of the remote file is used. Valid values are: • `Seq`: Sequential dataset • `VSAM`: VSAM dataset • `PDS/E`: Partitioned dataset extended • `PDS`: Partitioned dataset • `DA`: Direct access
`VSAM attributes: Type`	This parameter defines the type of VSAM file to be created. This field is used only when creating a VSAM file. If this parameter is not defined, the source file type is used when creating the file. Valid values are:

Parameter	Description
	• `KSDS`: Key sequenced dataset • `ESDS`: Entry sequenced dataset • `RRDS`: Relative record dataset
`VSAM attributes: Key Position`	This parameter defines the relative position of the key relative to zero. It is only used for VSAM KSDS files, and is ignored for other VSAM file types. If this field is not defined, the key position of the source file is used. The valid values are `0` - `32760`. **Note:** This field is a displacement. Therefore, for a key in the first byte of a record, the key position must be defined as `0`.
`VSAM attributes: Key Length`	This parameter defines the key length. This is only used for VSAM KSDS files, and is ignored for other VSAM file types. If this field is not defined, the key position of the source file is used. the valid values are `1` - `255`.
`VSAM attributes: Like DSN`	This parameter defines a file that is used as a model for creating VSAM files. This field is used only when creating a VSAM file. If this field is not defined, the attributes of the source file are used.
`VSAM attributes: RRDS Slot`	This parameter is used only when sending or receiving an RRDS file. When sending a record, it defines the Platform Server to insert the RRDS slot number in the first 4 bytes of the record read. When receiving a record, it informs the Platform Server that the RRDS slot number is in the first 4 bytes of the record to be written. This enables a RRDS to RRDS file transfer to maintain the RRDS slot numbers.
`VSAM attributes: REUSE`	This parameter defines whether the file should be opened with the REUSE option. The REUSE option overrides the file when it is used. It can be done under the following two circumstances:

Parameter	Description
	• When the `IDCAMS REUSE` parameter is defined • When the file is being loaded for the first time If `VSAM_REUSE` is specified under any other circumstances, VSAM returns an OPEN error, and the transfer is terminated.

Windows Options

The following panel is displayed when you set the `Platform file options` parameter to `Windows`.



```
OSICE303                    Windows Options
Command ===> _________________________________________________________ SOW1
Type information. Then press Enter.


ACL Template. . . . . . . ______________________________ +

File attributes . . . . .          (Y for yes and N for no)
    N  System
    N  Hidden
    N  Read Only
    N  Archive
    N  NTFS Compressed
    N  EOF
    N  CRLF/EOF
```

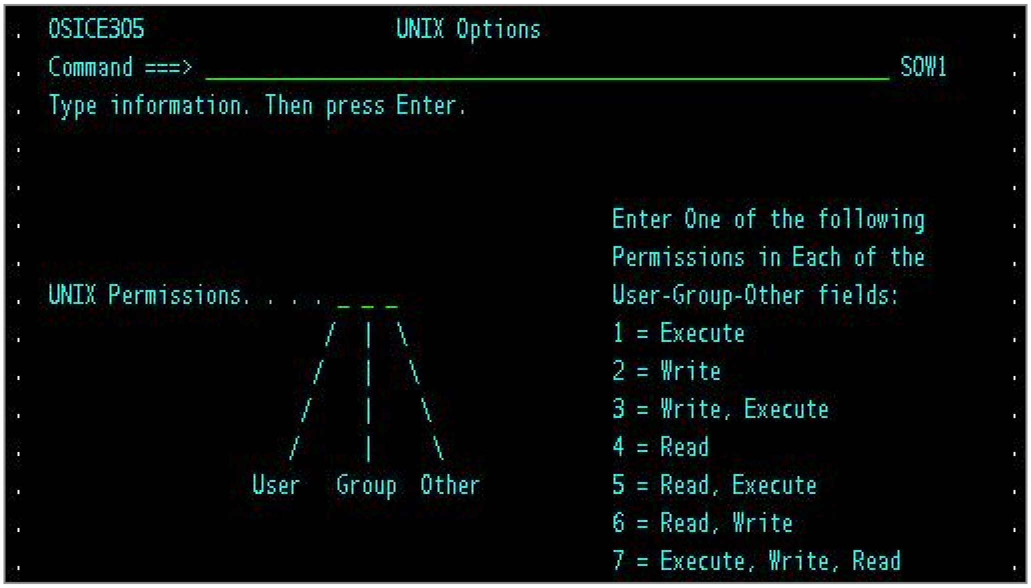
The following table lists the parameters in the Windows Options panel:

Parameter	Description
`ACL Template`	The receiving partner uses this file name as a template for its Access Control List (ACL). The ACL is a list that specifies users and groups and their access permissions to a file. The ACL of this file is copied to the ACL of the destination file. On Windows, the file specified must be readable by the partner that is receiving the file to file transfer.

Parameter	Description
	Note: This field is Windows-specific and is only valid when sending a file to a Windows machine.
`File Attributes`	You can define the following file attributes: - `System`: Indicates that the file is a system file that only the operating system, not the user, can view it. - `Hidden`: Indicates that the user cannot view this file. - `Read Only`: Indicates that the user can view this file but cannot modify it. - `Archive`: Marks a file that has changed since it was last backed up. - `NTFS Compressed`: Creates and compresses the file on the remote system (when this attribute is selected from the panel, batch interface, JCL, or TSO). This attribute is available on NTFS partitions. If the receiving file system is not NTFS, this parameter is ignored. - `EOF`: Appends an end of file marker to the end of the file. - `CRLF/EOF`: Appends a CR/LF (0x0d, 0x0a) to the end of the file, followed by the DOS End of File character, Control Z (0x1a). If a trailing Control Z or CR/LF is already present, it is not added again. This attribute is available only when Carriage Return/Line Feed processing is enabled.

UNIX Options

The following panel is displayed when you set the `Platform file options` parameter to `UNIX`.



```
.  OSICE305                    UNIX Options                                  .
.  Command ===> ___________________________________________________  SOW1     .
.  Type information. Then press Enter.                                       .
.                                                                            .
.                                                                            .
.                                              Enter One of the following    .
.                                              Permissions in Each of the    .
.  UNIX Permissions. . . . _ _ _               User-Group-Other fields:      .
.                         / | \                1 = Execute                   .
.                        /  |  \               2 = Write                     .
.                       /   |   \              3 = Write, Execute            .
.                      /    |    \             4 = Read                      .
.                   User  Group  Other         5 = Read, Execute             .
.                                              6 = Read, Write               .
.                                              7 = Execute, Write, Read      .
```

The following table lists the parameter in the UNIX Options panel:

Parameter	Description
`UNIX Permissions`	This parameter defines the permissions given to a file sent to a UNIX system. Select the options according to the authority that you want to give to the user, group, and other. This parameter can be any valid value of `000` - `777`.

Additional Parameters for Sending to Printer

The following table lists the two parameters that are specific to the "Send to printer" panel:

Parameter	Description
`Remote printer name`	This parameter defines the name of the printer to which the file is sent. By specifying this parameter, you can send the file that is being transferred directly to the print queue or spool on the remote or local side.
`SYSOUT Parameters`	If you have specified that a mainframe printer is the file destination, you can specify the SYSOUT parameters. These parameters are Windows-specific and are only valid when sending

Parameter	Description
	a file to a Windows machine. • `Class`: This parameter defines the class to which the JES output will be routed. This is a mandatory parameter. On mainframes, the printer queues are organized around a printer class rather than around a specific printer. The class has a one-character name (alphabetic or numeric character). Ask the mainframe staff for the class name. • `FCB`: This parameter defines the form control block for the JES output. This is a symbolic name on the mainframe that is essentially a "font profile". The FCB name is defined by an administrator (or systems programmer) on a mainframe and indicates character size and so on. **Note:** Do not specify a value for this unless necessary. • `Form`: This parameter defines the form name upon which this output should be printed. The host operator will receive a message to load the correct type of paper into the printer to print this report. For example, if you are printing shipping labels, the operator will be prompted to load labels into the printer before the printing starts. If you need to use this parameter, tell the operator at the mainframe printer site what paper form to load when he or she sees this name. • `Copies`: This parameter defines the number of times that the JES report will be printed on the host printer. The default value is `1`. • `Writer`: This parameter defines the external writer name that will be used to process this printer file on the mainframe. Essentially, this is the name of a "service" program on the mainframe, which will be given control when it is time to process this file from the printer queue. The "service" program, which is written by the customer, decides how it wants to process this print file.

Parameter	Description
	Note: Do not specify a value for this parameter unless you are directed to by the systems analyst on the mainframe. - Destination: This parameter defines the JES print destination name which can be a 1-character to 8-character symbolic name that identifies routing information for this print file on the mainframe. This "destination" might indicate the name of a branch office, geographic location, or specific printer to which the report will be sent. If this value is not supplied, most mainframe systems applies a default value of LOCAL.

Additional Parameters for Sending a Command

The following table lists the two parameters that are specific to the "Send to command" panel.

Parameter	Description
Remote command	This parameter defines the command to execute on the remote system. Commands cannot be sent to a target z/OS system.
Local file	If a local file name is defined, it will be used to store the output of the remote command if the remote system is Windows or UNIX (z/OS does not send back output).

Defining Receive Requests

You can use the Receive parameter in the Platform Server primary window to receive a file sent from a remote system to a file, job, or printer.

To perform a receive operation, you can use one of the following two methods:

- Type the number 4 and a letter (A, B, or C) on the command line and press Enter to directly receive a file sent from a remote system to a file, job, or printer.

For example, type 4A (or 4.A) on the command line.

- Type the number 4 and press Enter; a screen is displayed from which you can select to type A, B, or C to receive a file sent from a remote system to a file, job, or printer.

For more details about how to receive a file from the remote system to a file, job, and printer, see the following introductions:

- Receiving a File to a File
- Receiving a File to a Job
- Receiving a File to a Printer

Receiving a File to a File

By typing 4A (or 4.A) on the command line in the Platform Server primary window, you can receive a file from a remote system to a file.

After you type 4A (or 4.A) on the command line, the "Receive to file" panel is displayed. The panel contains all the parameters you can configure to set up a transfer to receive a file from a remote system. You can press the F8 and F7 keys to go to the next and previous pages of the "Receive to file" panel.

In the "Receive to file" panel, some of the parameters have been filled in with their default values. You can modify these parameter values at any time by typing values over the default values.

The values displayed on initialization of the "Receive to file" panel are generally based on the last values entered in the entry fields; therefore, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

The parameters in the "Receive to file" panel are the same as those in the "Send to file" panel. See Send Request Parameters for details of the parameters.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

Receiving a File to a Job

By typing 4B (or 4.B) on the command line in the Platform Server primary window, you can receive a file from a remote system and execute it as a job on the local system.

After you type 4B (or 4.B) on the command line, the "Receive to job" panel is displayed. The panel contains all the parameters you can configure. You can press the F8 and F7 keys to go to the next and previous pages of the "Receive to job" panel.

In the "Receive to job" panel, some of the parameters have been filled in with their default values. You can modify these parameter values at any time by typing values over the default values and press Enter.

The values displayed on initialization of the "Receive to job" panel are generally based on the last values entered in the entry fields; therefore, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

The parameters in the "Receive to job" panel are the same as those in the "Send to file" panel. See Send Request Parameters for details of the parameters.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

Receiving a File to a Printer

By typing 4C (or 4.C) on the command line in the Platform Server primary window, you can receive a file from a remote system and execute it as a print job on the local system.

After you type 4C (or 4.C) on the command line, the "Receive to printer" panel is displayed. The panel contains all the parameters you can configure. You can press the F8 and F7 keys to go to the next and previous pages of the "Receive to printer" panel.

In the "Receive to printer" panel, some of the parameters have been filled in with their default values. You can modify these parameter values at any time by typing values over the default values and press Enter.

The values displayed on initialization of the "Receive to printer" panel are generally based on the last values entered in the entry fields; therefore, you do not have to retype values that do not change each time.

The fields that are most commonly changed per transfer are placed at the top of the panel so that you can perform less scrolling up and down operations after the initial settings are entered.

The parameters in the "Receive to printer" panel are the same as those in the "Send to file" panel. See Send Request Parameters for details of the parameters. For parameters that are specific to the "Receive to printer" panel, see Additional Parameters for Receiving to Printer.

When you fill in all the necessary parameter values you want to use, press Enter. The Confirmation panel is displayed for you to set what the Platform Server does next. See Sending a File to a File for more details about the Confirmation panel.

Additional Parameters for Receiving to Printer

The following table lists the two parameters that are specific to the "Receive to printer" panel:

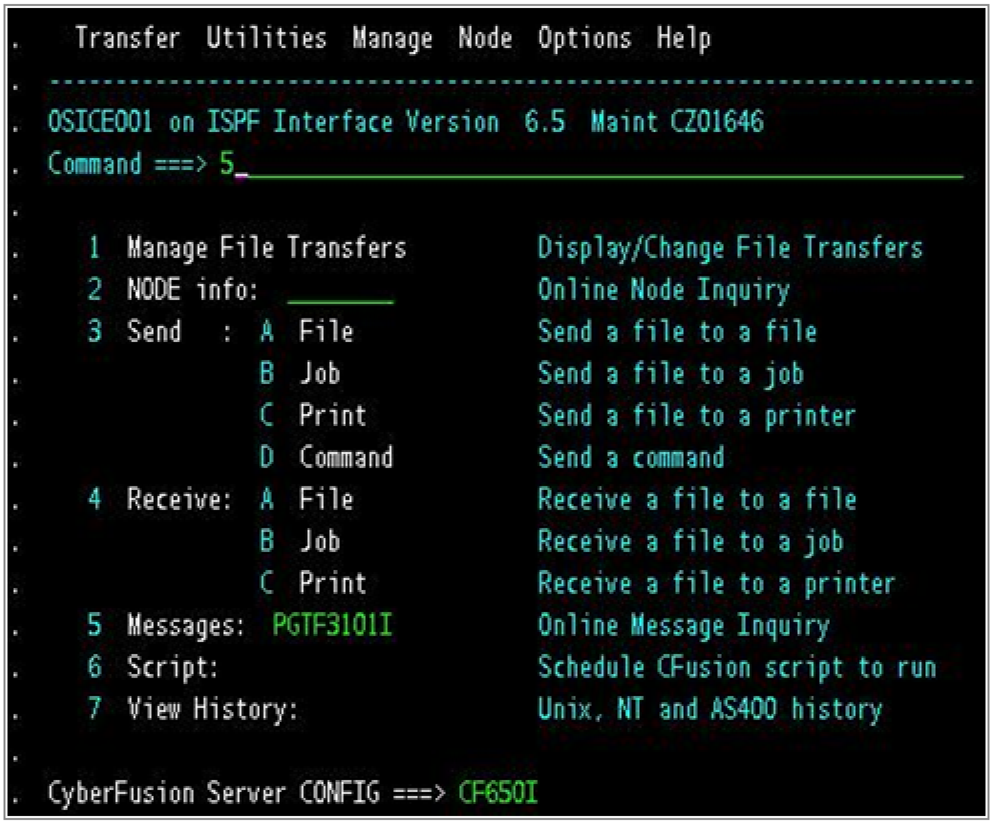
Parameter	Description
`Local printer name`	This parameter defines the name of the printer to which the file will be received. By specifying this parameter you can receive the remote file that is being transferred directly to the print queue or spool on the local side.
`SYSOUT Parameters`	If you specify that a mainframe printer is the file destination, you can specify the SYSOUT parameters. These parameters are Windows-specific and are only valid when sending a file to a Windows machine. • `Class`: This parameter defines the class to which the JES output is routed. This is a mandatory parameter. On mainframes, the printer queues are organized around a printer class rather than around a specific printer. The class has a one-character name (alphabetic or numeric character). Ask the mainframe staff for the class name. • `FCB`: This parameter defines the form control block for the JES output. This is a symbolic name on the mainframe that is essentially a "font profile". The FCB name is defined by an administrator (or systems programmer) on a

Parameter	Description
	mainframe and indicates character size and so on. **Note:** Do not specify a value for this parameter unless necessary. • `Form`: This parameter defines the form name upon which this output should be printed. The host operator will receive a message to load the correct type of paper into the printer to print this report. For example, if you are printing shipping labels, the operator will be prompted to load labels into the printer before the printing starts. If you need to use this parameter, tell the operator at the mainframe printer site what paper form to load when he or she sees this name. • `Copies`: This parameter defines the number of times that the JES report will be printed on the host printer. The default value is `1`. • `Writer`: This parameter defines the external writer name that will be used to process this printer file on the mainframe. Essentially, this is the name of a "service" program on the mainframe, which will be given control when it is time to process this file from the printer queue. The "service" program, which is written by the customer, decides how it wants to process this print file. **Note:** Do not specify a value for this parameter unless you are directed to by the systems analyst on the mainframe. • `Destination`: This parameter defines the JES print destination name which can be a 1-character to 8-character symbolic name that identifies routing information for this print file on the mainframe. This "destination" might indicate the name of a branch office, geographic location, or specific printer to which the report will be sent. If this value is not supplied, most mainframe systems applies a default value of `LOCAL`.

Viewing Platform Server Messages

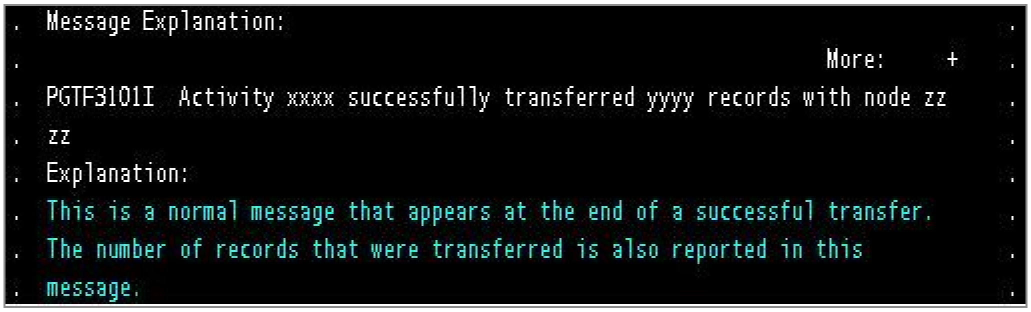
You can use the `Messages` option in the Platform Server primary window to display the details of the Platform Server messages.

For example, you received the Platform Server informational return code PGTF3101I for a transfer you attempted. To display the details of this message, input 5 at the command prompt and PGTF3101I on the input line next to the option `Messages` as shown in the following figure, and then press Enter.



```
.   Transfer  Utilities  Manage  Node  Options  Help
. ---------------------------------------------------------------------
. OSICE001 on ISPF Interface Version  6.5  Maint CZ01646
. Command ===> 5_
.
.     1  Manage File Transfers          Display/Change File Transfers
.     2  NODE info: ________            Online Node Inquiry
.     3  Send    :  A  File             Send a file to a file
.                   B  Job              Send a file to a job
.                   C  Print            Send a file to a printer
.                   D  Command          Send a command
.     4  Receive:   A  File             Receive a file to a file
.                   B  Job              Receive a file to a job
.                   C  Print            Receive a file to a printer
.     5  Messages:  PGTF3101I           Online Message Inquiry
.     6  Script:                        Schedule CFusion script to run
.     7  View History:                  Unix, NT and AS400 history
.
. CyberFusion Server CONFIG ===> CF650I
```

The following figure shows the details of the Platform Server message PGTF3101I:



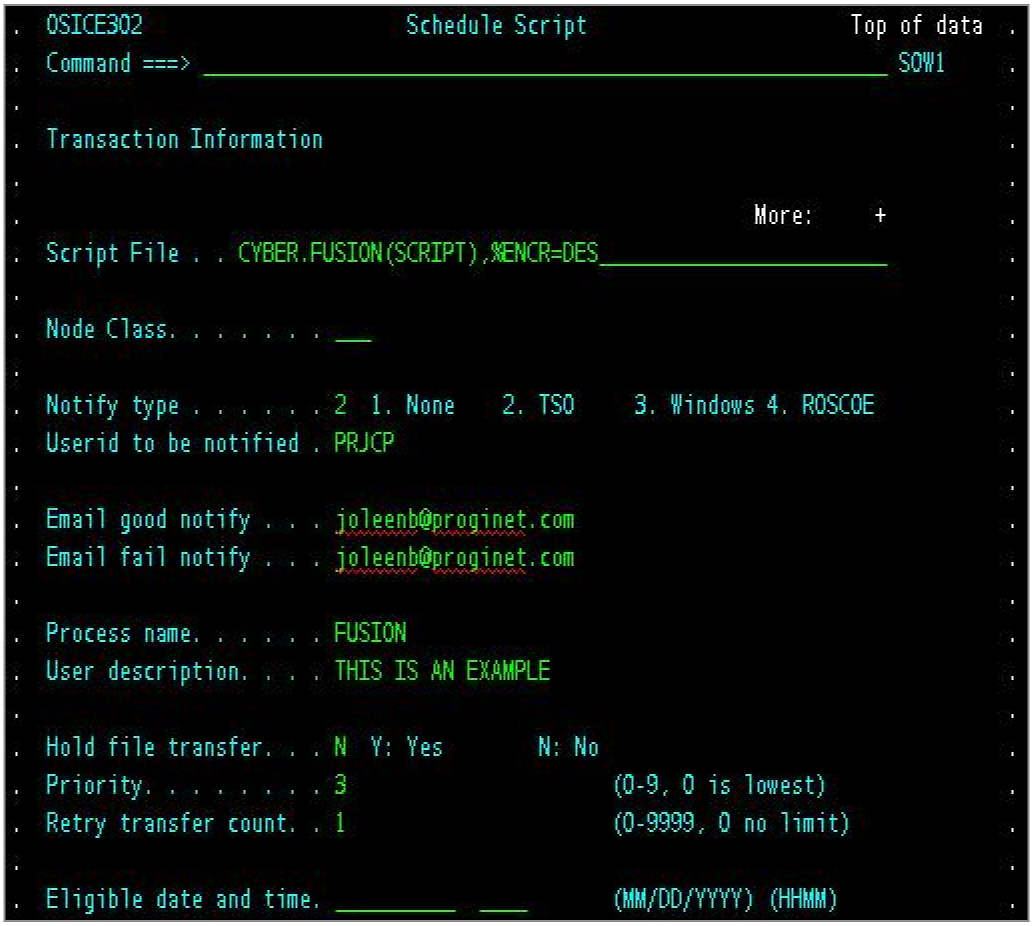
```
. Message Explanation:                                                    .
.                                                          More:     +    .
. PGTF3101I  Activity xxxx successfully transferred yyyy records with node zz .
. zz                                                                      .
. Explanation:                                                            .
. This is a normal message that appears at the end of a successful transfer. .
. The number of records that were transferred is also reported in this    .
. message.                                                                .
```

Scheduling Platform Server Scripts

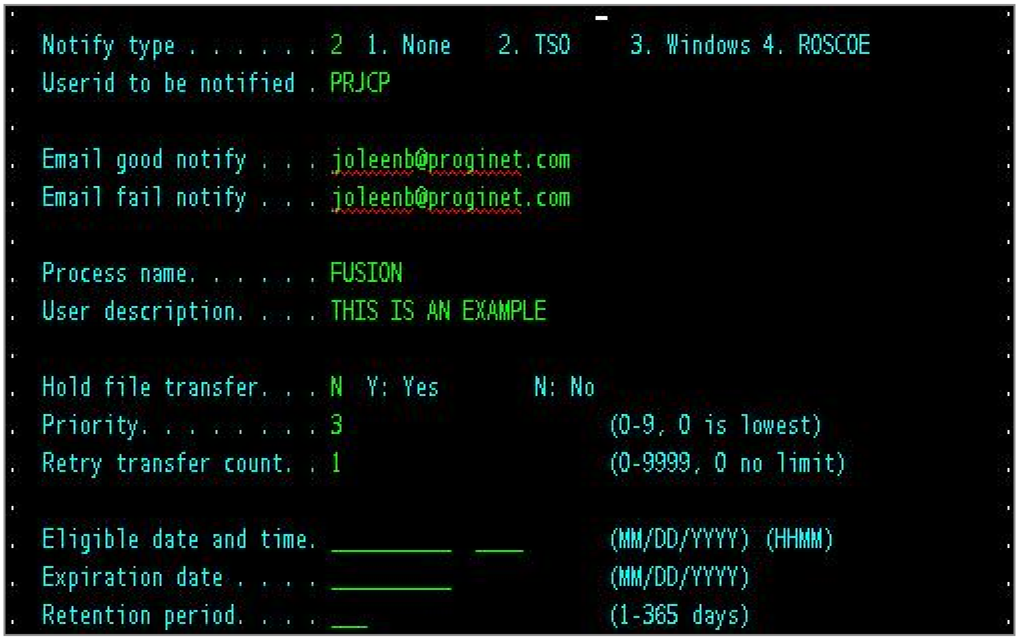
You can quickly set up a Platform Server script to be executed by selecting the `Script` option in the Platform Server primary window. You can also execute a Platform Server script through the "Send to file" panel.

Note: See Interactive Interface for details of the primary server primary window.

After you select the `Script` option in the Platform Server primary window, the Schedule Script panel is displayed as shown in the following figures.



```
.  OSICE302                       Schedule Script                   Top of data  .
.  Command ===> ____________________________________________________ SOW1         .
.                                                                                .
.  Transaction Information                                                       .
.                                                                                .
.                                                         More:     +            .
.  Script File . . CYBER.FUSION(SCRIPT),%ENCR=DES______________________          .
.                                                                                .
.  Node Class. . . . . . . ___                                                   .
.                                                                                .
.  Notify type . . . . . . 2  1. None    2. TSO     3. Windows 4. ROSCOE         .
.  Userid to be notified . PRJCP                                                 .
.                                                                                .
.  Email good notify . . . joleenb@proginet.com                                  .
.  Email fail notify . . . joleenb@proginet.com                                  .
.                                                                                .
.  Process name. . . . . . FUSION                                                .
.  User description. . . . THIS IS AN EXAMPLE                                    .
.                                                                                .
.  Hold file transfer. . . N  Y: Yes       N: No                                 .
.  Priority. . . . . . . . 3                    (0-9, 0 is lowest)               .
.  Retry transfer count. . 1                    (0-9999, 0 no limit)             .
.                                                                                .
.  Eligible date and time. __________  ____     (MM/DD/YYYY) (HHMM)              .
```



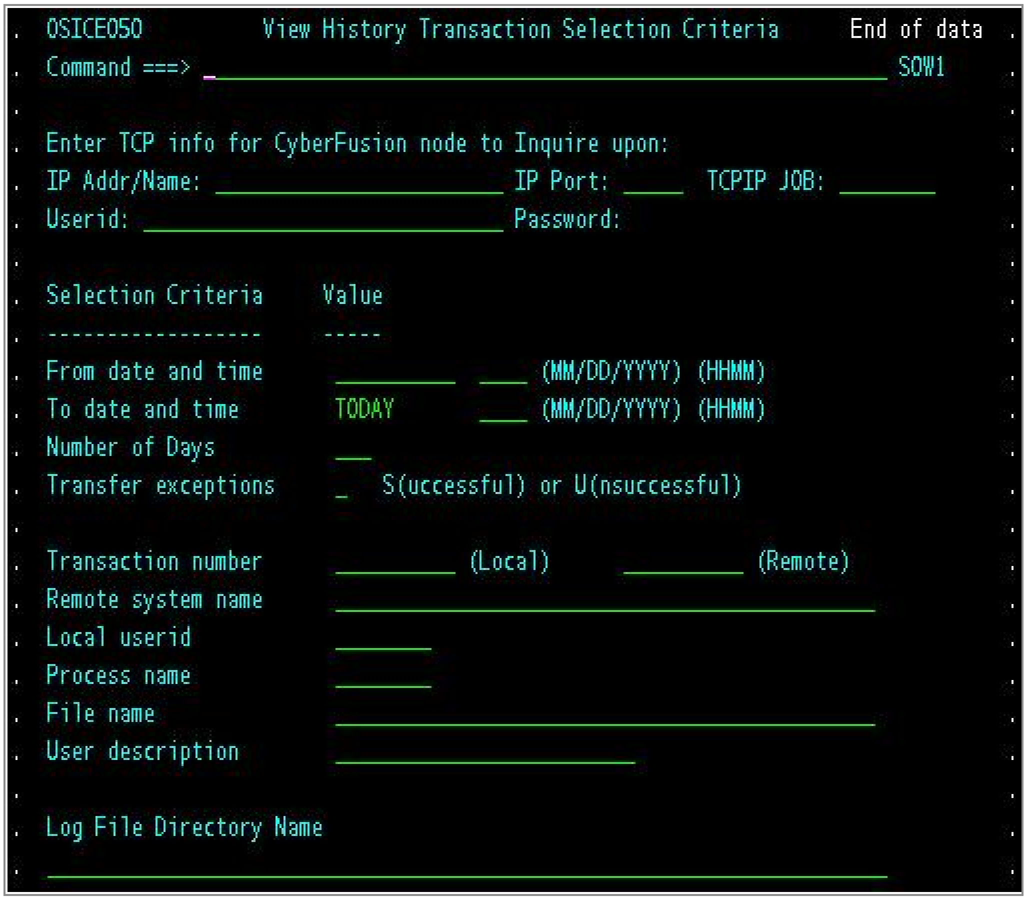
```
. Notify type . . . . . . . 2  1. None    2. TSO     3. Windows 4. ROSCOE   .
. Userid to be notified . PRJCP                                              .
.                                                                            .
. Email good notify . . . joleenb@proginet.com                               .
. Email fail notify . . . joleenb@proginet.com                               .
.                                                                            .
. Process name. . . . . . FUSION                                             .
. User description. . . . THIS IS AN EXAMPLE                                 .
.                                                                            .
. Hold file transfer. . . N  Y: Yes       N: No                              .
. Priority. . . . . . . . 3                      (0-9, 0 is lowest)          .
. Retry transfer count. . 1                      (0-9999, 0 no limit)        .
.                                                                            .
. Eligible date and time. __________  ____       (MM/DD/YYYY) (HHMM)         .
. Expiration date . . . . __________             (MM/DD/YYYY)                .
. Retention period. . . . ___                    (1-365 days)                .
```

If the script uses input parameters, you must enter the input parameters as well. The only parameter that is specific to the Schedule Script panel is `Script File` which holds the dataset name that contains the script to execute. For details of the other parameters in this panel, see Send Request Parameters.

Viewing Transaction History

You can view the history of a transaction by using the `View History option` in the Platform Server primary window.
For details of the primary server primary window, see Interactive Interface.

After you select the `View History option` in the Platform Server primary window, the View History Transaction Selection Criteria panel is displayed as shown in the following figure:



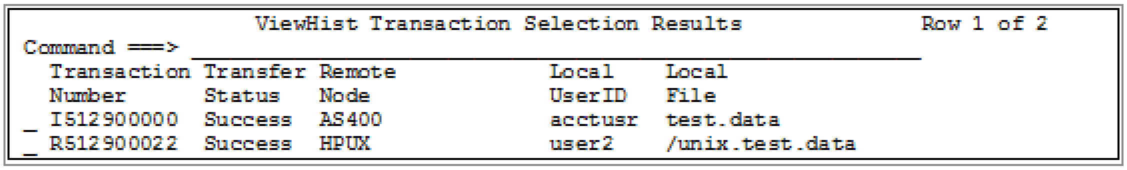
```
OSICE050         View History Transaction Selection Criteria      End of data
Command ===> ________________________________________________________ SOW1

Enter TCP info for CyberFusion node to Inquire upon:
IP Addr/Name: ________________________ IP Port: _____  TCPIP JOB: ________
Userid: ______________________________ Password:

Selection Criteria      Value
------------------      -----
From date and time       __________  ____ (MM/DD/YYYY) (HHMM)
To date and time         TODAY       ____ (MM/DD/YYYY) (HHMM)
Number of Days           ___
Transfer exceptions      _   S(uccessful) or U(nsuccessful)

Transaction number       __________ (Local)      __________ (Remote)
Remote system name       _____________________________________________
Local userid             ________
Process name             ________
File name                _____________________________________________
User description         _________________________

Log File Directory Name
______________________________________________________________________
```

Note: To view a transaction, you must specify the `IP Addr/Name`, `IP Port`, `Userid` (the user ID used to log in to the remote system), and `Password` fields.

The remote Platform Server must have a node defined for the z/OS system on it and that node needs to be configured to allow MFT Command Center MFTNQ requests in order for you to view the history information from a remote Platform Server.

After you enter the appropriate information and select the transfers, the output is displayed in the ViewHist Transaction Selection Results panel.

The following figure shows a sample output:



```
                    ViewHist Transaction Selection Results              Row 1 of 2
Command ===> ____________________________________________________________
  Transaction Transfer Remote            Local     Local
  Number      Status   Node              UserID    File
_ I512900000  Success  AS400             acctusr   test.data
_ R512900022  Success  HPUX              user2     /unix.test.data
```

To display the details of a transaction from the ViewHist Transaction Selection Results panel, enter S on the line to the far left of the particular transaction and press Enter. See Viewing Transaction Details for more details.

Action Bar

The **Action Bar** is the area at the top of the Platform Server primary window that contains keywords that you use to access available actions.

The **Action Bar** provides the following keywords:

- Transfer
- Utilities
- Manage
- Node
- Options
- Help

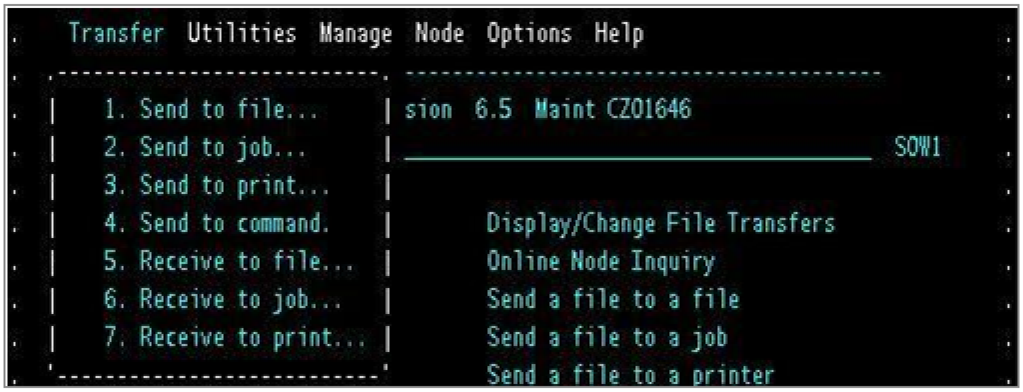
When you select a keyword from the **Action Bar**, a submenu is displayed. When you select an option from the submenu, a pop-up dialog is displayed from which you can complete the requested action. The pop-up dialog displays the default values of the entry fields, and you can change the default values by typing over them. Generally, the pop-up dialog displays the value last entered in an entry field.

Apart from operating from the **Action Bar**, you can also complete the most common actions directly from the main menu in the Platform Server primary window by typing the number or the number and letter of the corresponding action on the command line. For example, if you want to send a file to a remote system where the partner executes it as a job, type 3.B or 3B on the command line. The "Send to job" panel is displayed, and you can enter all the information about your transfer in the panel.

Transfer

Use **Transfer** to send a file to a file, job, or printer, or send a command to a remote system or receive a file from the remote system to a file, job, or printer.

To select **Transfer** from the **Action Bar**, press Tab until the cursor is before **Transfer** or use the arrow keys to place the cursor on **Transfer** and press Enter. A submenu is displayed as shown in the following figure:



```
.   Transfer  Utilities  Manage  Node  Options  Help                  .
.  .--------------------------.  -------------------------------------.
.  |    1. Send to file...    | sion  6.5  Maint CZ01646              .
.  |    2. Send to job...     | ____________________________________ SOW1 .
.  |    3. Send to print...   |                                       .
.  |    4. Send to command.   |       Display/Change File Transfers   .
.  |    5. Receive to file... |       Online Node Inquiry             .
.  |    6. Receive to job...  |       Send a file to a file           .
.  |    7. Receive to print...|       Send a file to a job            .
.  '--------------------------'       Send a file to a printer        .
```

Select the desired option in the submenu and press Enter. The results are the same as those when you choose one of these options from the main menu in the Platform Server primary window. For the parameters of these options, see the descriptions in Send Request Parameters and Additional Parameters for Receiving to Printer.

To exit, press F3.

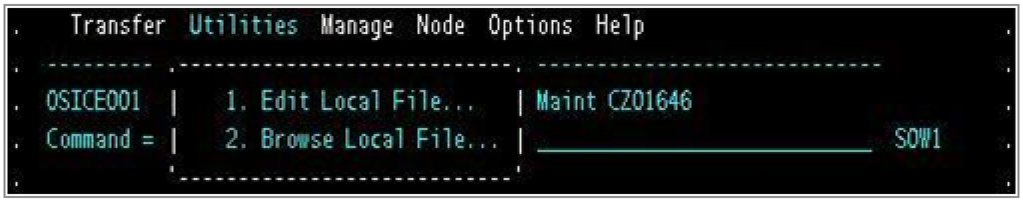
Utilities

Use **Utilities** to browse or edit local datasets and list remote directories.

When you select **Utilities** from the **Action Bar**, a submenu is displayed. The submenu contains the following two options:

- `Edit Local File`: By selecting this option, you can edit the local dataset name.
- `Browse Local File`: By selecting this option, you can browse the local dataset.

The following figure shows the submenu of **Utilities**:

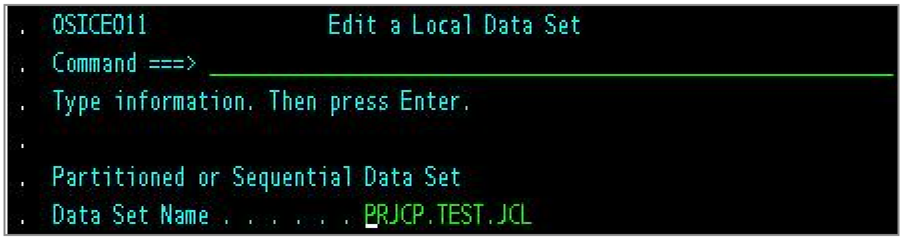


```
.   Transfer  Utilities  Manage  Node  Options  Help                  .
.  ---------  .---------------------------.  ----------------------------  .
.  OSICE001  |    1. Edit Local File...   | Maint CZ01646             .
.  Command = |    2. Browse Local File... | __________________________ SOW1 .
.            '---------------------------'                            .
```

Editing a Local Dataset File

You can edit a local dataset name by selecting the **Edit Local File** option from the submenu of **Utilities**.

After you select the **Edit Local File** option, the "Edit a Local Data Set" panel is displayed as shown in the following figure:



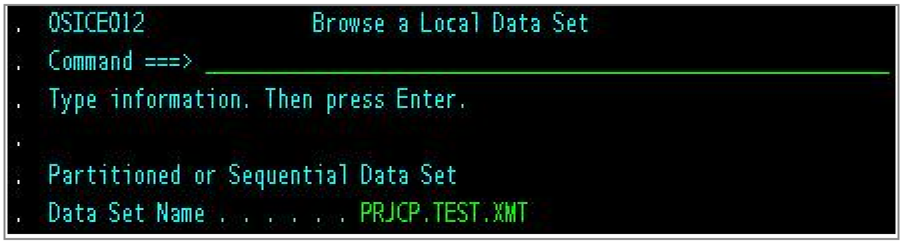
```
. OSICE011              Edit a Local Data Set
. Command ===> ____________________________________________
. Type information. Then press Enter.
.
. Partitioned or Sequential Data Set
. Data Set Name . . . . . . PRJCP.TEST.JCL
```

To modify the name of a dataset, type the name of the dataset you want to edit in the **Data Set Name** field and press Enter, then you can type a new name for this dataset.

Browsing a Local File

You can browse a local dataset by selecting the **Browse Local File** option from the submenu of **Utilities**.

When you select the **Browse Local File** option, the "Browse a Local Data Set" panel is displayed as shown in the following figure:



```
. OSICE012              Browse a Local Data Set
. Command ===> ____________________________________________
. Type information. Then press Enter.
.
. Partitioned or Sequential Data Set
. Data Set Name . . . . . . PRJCP.TEST.XMT
```

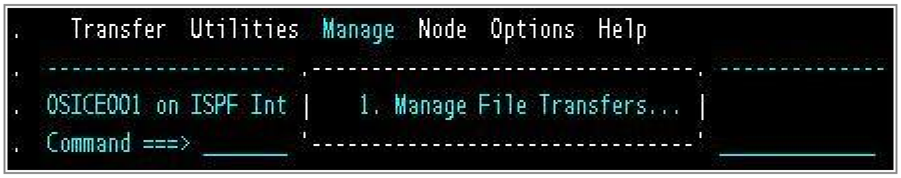
To view the file contents, type the name of the local dataset that you want to browse and press Enter.

Manage

Use **Manage** to open the **Manage File Transfers** panels from which you can review, release, or purge entries from the work queue as well as review past transactions on the audit file.

For details about how to manage file transfers, see Managing File Transfers.

The following figure shows the submenu of **Manage**:

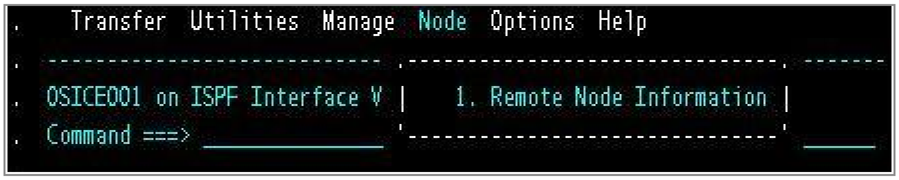


```
.    Transfer  Utilities  Manage  Node  Options  Help
.  -------------------- .------------------------------. -------------
.  OSICE001 on ISPF Int |    1. Manage File Transfers... |
.  Command ===> _______  '------------------------------'  ____________
```

Node

Use **Node** to display the remote systems that have been defined to the Platform Server. For details about node operations, see Managing Remote Nodes.

The following figure shows the submenu of **Node**:

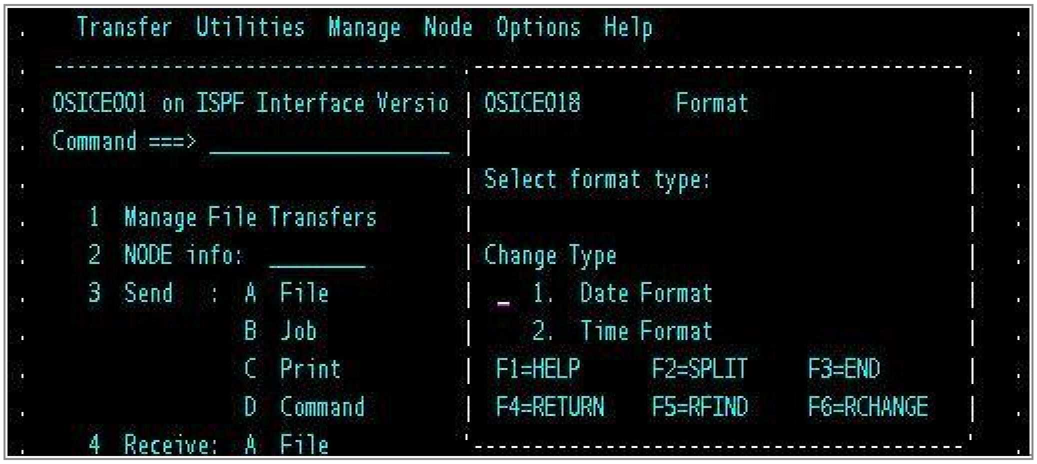


```
.    Transfer  Utilities  Manage  Node  Options  Help
.  ---------------------------- .-----------------------------. -------
.  OSICE001 on ISPF Interface V |    1. Remote Node Information |
.  Command ===> ______________  '-----------------------------'  _____
```

Options

Use **Options** to change the format of the date and time used in file transfers.

Select **Options**, then select **Format (1)** and press Enter. The submenu is displayed, see the following figure:



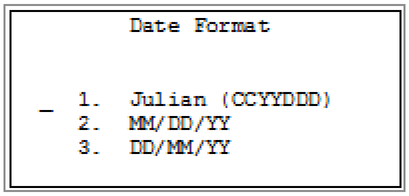
```
.    Transfer  Utilities  Manage  Node  Options  Help                  .
.  ------------------------------- .-----------------------------------.  .
.  OSICE001 on ISPF Interface Versio | OSICE018        Format            |  .
.  Command ===> ___________________ |                                   |  .
.                                   | Select format type:               |  .
.      1  Manage File Transfers     |                                   |  .
.      2  NODE info:  ________      | Change Type                       |  .
.      3  Send    :  A  File        | _  1.  Date Format                |  .
.                    B  Job         |    2.  Time Format                |  .
.                    C  Print       |  F1=HELP     F2=SPLIT    F3=END   |  .
.                    D  Command     |  F4=RETURN   F5=RFIND    F6=RCHANGE |  .
.      4  Receive:  A  File         '-----------------------------------'  .
```

Changing the Date and Time Formats

You can change the date and time formats used in file transfers by using the **Options** submenu.

To change the format of the date values, select **1** (Date Format) from the submenu.

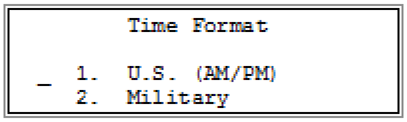
The following figure shows the valid values:



```
         Date Format


_   1.   Julian (CCYYDDD)
    2.   MM/DD/YY
    3.   DD/MM/YY
```

To change the format of the numeric time values, select **2** (Time Format) from the submenu.

The following figure shows the valid values:

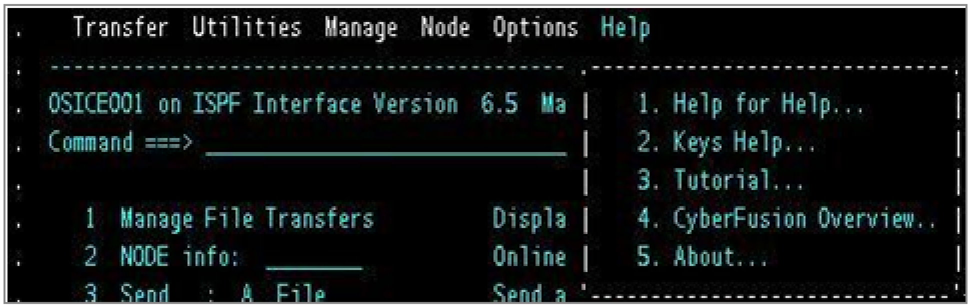


```
         Time Format

_   1.   U.S. (AM/PM)
    2.   Military
```

Help

Use **Help** to view the types of help available, descriptions about the function keys, actions, and functions available in the interactive interface, and an overview of the Platform Server product.

The following figure shows the submenu of **Help**:



```
.   Transfer  Utilities  Manage  Node  Options  Help
.  ---------------------------------------- .----------------------------.
.  OSICE001 on ISPF Interface Version  6.5  Ma |  1. Help for Help...       |
.  Command ===> ____________________________ |  2. Keys Help...          |
.                                             |  3. Tutorial...           |
.     1  Manage File Transfers        Displa |  4. CyberFusion Overview.. |
.     2  NODE info:  _______          Online |  5. About...               |
.     3  Send   :  A  File            Send a '----------------------------'
```

The following table lists the five choices in the submenu of **Help**:

Option	Description
Help for Help	Provides information about how to use the Help feature. It describes the types of help that are available, including Extended Help, Field Help, Keys Help, Tutorial Help, MFT Platform Server Overview, and diagnostic codes.
Keys Help	Provides information about the actions that are performed when the function keys are clicked.
Tutorial	Provides a list of all major actions and functions that are available in the Interactive interface and explanations about how to use them.
CyberFusion Overview	Provides an overview of the Platform Server product.
About	Shows the current release level of the Platform Server ISPF interface.

Script Interface

The Platform Server provides a batch script interface with which you can use Job Control Language (JCL) to queue transactions to the Platform Server for z/OS.

By using the control statements, you can perform a wide variety of functions to determine whether the requests are successful. You can perform a single file transfer or multiple file transfers within one job step. In addition, you can perform validation on each transfer to let the script determine the next step to perform.

The Platform Server batch script interface uses the Platform Server Batch interface to actually perform the file transfer requests. See Batch Interface for information on the statements required to perform a file transfer.

JCL Statements

You can use the JCL statements to perform batch jobs and to queue file transfers.

The Script interface JCL is identical to the Batch interface JCL with the following exceptions:

- The program executed is OSIUC000 instead of OSIUB000.
- The SYSTRACE DD statement is added.

Required JCL Statements

The following is a sample JCL that includes the required JCL statements:

```
//OSIUC000 EXEC PGM=OSIUC000,PARM='SERVER=FUSION'
//STEPLIB  DD DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD SYSOUT=*
//SYSTRACE DD SYSOUT=*
//SYSIN  DD *
//INCLUDE DD DSN=YOUR.INCLUDE.LIBRARY,DISP=SHR
```

The following table lists the required JCL statements:

Statement	Description
EXEC PGM=OSIUC000,PARM='data'	Defines the program that must be executed to run the Script interface. The `PARM` field defines the name of the Platform Server started task. You can also use the `PARM` parameter to override any `SYSIN` parameters. For more information, see Appendix G. Overriding JCL SYSIN Parameters. **Note:** The program name must be defined as `OSIUC000`.
STEPLIB	Defines the library that contains the Platform Server load modules. **Note:** You must include this statement in the JCL to identify the Platform Server load library. But it is not required if the Platform Server LOADLIB is located inside a LNKLST.
SYSPRINT	Defines the output report file that records the parameters that are used for file transfers. If the file is successfully queued, you can read the output report file and find out the transaction number assigned to the job.
SYSTRACE	Defines the output file that provides information on the script records which have been executed. It provides a log of all script records that have been executed and return code information for file transfer requests.
SYSIN	The input file that contains the script records and any PROCESS statements required.

Optional JCL Statements

The following table lists the optional JCL statements:

Statement	Description
SYSSAY	Defines the output file that provides information on the destination where all Script SAY commands are written If this DD statement is not defined, SAY data will be written to the SYSTRACE DD statement.
INCLUDE	Defines the library that should be searched when an INCLUDE statement is encountered. **Note:** This DD statement is required only when the INCLUDE statement is used. It can be left out in all other cases.

The PARM Field of the EXEC PGM=JCL Card

You can define the `PARM` field in a JCL statement. The information defined in the `PARM` field can be used to define the method of communication to the Platform Server started task and override the Platform Server PROCESS parameters.

Note: The `PARM` data specified in the Script interface is identical to that supported by the Batch interface. See The PARM Field of the EXEC PGM=JCL Card in Batch Interface for more information on the fields supported in the `PARM` field.

Defining the Communication Method

You can define the started task and the way in which TIBCO MFT Platform Server for z/OS clients communicate with the started task in the `PARM` field in a JCL statement.

The following three communication methods are provided:

- Cross Memory Services
- SNA LU6.2
- TCP

For more details of the three communication methods, see Communication Methods.

Communication Methods

TIBCO MFT Platform Server for z/OS clients communicate with the Platform Server started task in three ways.

Cross Memory Services

This method can be used when the Platform Server clients are running on the same CPU as the Platform Server started task.

- Required Parameter

  `SERVER`: defines the name of the Platform Server started task.

- Optional Parameter

  No optional parameter is required.

> **Note:** The script interface supports the `JCL EXEC PARMDD` parameter. The `PARMDD` parameter allows you to define a DD statement that is used to hold PARM data to be passed to the script program. The `EXEC PARMDD` parameter is mutually exclusive with the `PARM` parameter. While MVS supports up to 32767 characters in the `PARMDD` parameter, Platform Server for z/OS supports a maximum of 4096 characters.

SNA LU6.2

This method can be used when the Platform Server clients are not running on the same CPU as the Platform Server started task.

- Required Parameters
  - `SERVLUNAME`: defines the APPLID of the Platform Server.
  - `ACBNAME`: defines the name of the Platform Server client ACB defined to VTAM.
- Optional Parameter

  `MODENAME`: defines the name of the SNA mode with which the session is established.

TCP

This method can be used when the Platform Server clients are not running on the same CPU as the Platform Server started task.

- Required Parameters
  - `TCPIPJOBNAME`: defines the name of the TCP/IP started task on CPU where script is running.
  - `SERVIPNAME` or `SERVIPADDR`: defines the IP name or the IP address of the TCP where the started task is running.
- Optional Parameter

  `SERVIPPORT`: defines the IP port that is used by the started task to listen for requests.

Defining the CONFIG Parameter

You can define the `CONFIG` parameter in the `PARM` field in a JCL statement to specify the name of the CONFIG entry within the FUSCFG configuration entry.

The maximum length of this parameter is 24 characters. And its value must match the `CONFIG` parameter within the FUSCFG member of the FUSCFG DD statement.

The FUSCFG CONFIG entry can be defined to communicate by using TCP, SNA, or Cross Memory methods. The advantage of using these methods of communication is that all configuration data for communicating to the Platform Server started task is controlled within a single file. If the connectivity information for a started task changes, the FUSCFG entry can be changed so that individual batch jobs do not have to be changed.

Note: The `FUSCFG` file used by the batch interface and the script interface is the same as the FUSCFG information used by the ISPF and REXX interfaces. The same file can be used by the Batch interface as by the ISPF and REXX interfaces. For more information on the `FUSCFG` file, see the *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide.*

Example of Defining the CONFIG Parameter

```
//STEP0001 EXEC PGM=OSIUB000,
//PARM='CONFIG=CONFIGENTRYNAME'
```

Defining Script Interface Parameters

You can configure the script interface parameters by using either the **PARM** field of the EXEC PGM=JCL card or SYSIN DD statement. Due to limitations of the PARM field, most of the parameters should be define by the SYSIN DD statement. Parameters defined by the PARM will execute before the parameters defined by the SYSIN DD statement.

Symbolic Parameters

The Platform Server scripting supports up to 200 user-defined script variables. You do not have to predefine the variables.

The variables can be either numeric or character variables. The variable type depends on the way in which the values are initialized by the SET function. Only numeric variables can be used in addition or subtraction. You can specify a starting position and length for character variables by suffixing `(ss.ll)` to the variable name. For example, when `%FILENAME(10.5)` is specified, the Platform Server uses bytes 10 - 14 of variable `%FILENAME`. The starting position `ss` is relative to 1.

The following table lists the predefined variables:

Parameter	Description
`%DATE`	Today's date is saved in this field in the format: CCYYDDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%GDATE`	Today's date is saved in this field in the format: CCYYMMDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%GDATE1`	Today's date is saved in this field in the format: 1YYMMDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%GDATE2`	Today's date is saved in this field in the format: YYYYDDMM. You cannot set this field. If you attempt to use this field as the target

Parameter	Description
	of a SET function, an error occurs.
%GDATE3	Today's date is saved in this field in the format: MMDDYYYY. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
%GDATE4	Today's date is saved in this field in the format: DDMMYYYY. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
%IDCAMS1 - %IDCAMS40	Used by the CALLIDCAMS script function and cleared by the CLEARIDCAMS script function. When used by the CALLIDCAMS function, they define the exact data that is passed to the z/OS IDCAMS utility. **Note:** When calling IDCAMS, keep column 1 blank. It is a good practice to define IDCAMS data in columns 2 - 71.
%MAXEXEC	Defines the maximum number of script statements that can be executed before an error is detected. This parameter is used in loop detection to ensure that the script is not looping. The default value is `10000`, and the valid values are `0` - `99999999`. You can increase or decrease this parameter in a SET function.
%MAXLOOP	Defines the maximum number of script statements that can be executed between batch PROCESS statements before an error is detected. This parameter is used in loop detection to ensure that the script is not looping. The default value is `500`, and the valid values are `0` - `99999999`. This parameter is reset to `0` after each PROCESS statement. You can

Parameter	Description
	increase or decrease this parameter in a SET function.
`%MAXRC`	Defines the maximum return code encountered. You can change this parameter in a SET function. This parameter is set on `PROCESS` and `CALLPGM` requests when the return code exceeds the current value of this parameter.
`%PROC`	Defines the process under which the script is running. This parameter is only valid when the script is running within the Platform Server started task. If the script is not running within the Platform Server started task, this parameter defaults to the job name.
`%RC`	Defines the return code of the last event for which a return code is set. You can use this parameter in a SET function. This parameter is set on `PROCESS` and `CALLPGM` requests.
`%SERVER`	Defines the name of the Platform Server with which the script program is communicating. The value stored in this parameter depends on the parameters defined in the EXEC PARM statement. The following values can be stored in this variable: `CONFIG`, `IPADDR`, `IPNAME`, `LUNAME`, or `SERVER`. You cannot set this field. If you attempts to use this field as the target of a SET function, an error occurs.
`%TDAY`	Defines the current English day of week. The field must be `9` bytes long and space filled. The values are `Sunday` - `Saturday`.
`%TDAYOFWEEK`	Defines the numeric day of week. The values are `1` - `7`; wherein, `1` stands for Sunday and `7` stands for Saturday.

Parameter	Description
%TDD	Defines the numeric day of month. The values are 1 - 31.
%TDDD	Defines the numeric Julian day. The values are 1 - 366.
%TIME	Defines the current system time in the format of HHMMSS. You cannot set this field. If you attempts to use this field as the target of a SET function, an error occurs.
%TIME1	Defines the current system time in the format of HHMMSSTH. You cannot set this field. If you attempts to use this field as the target of a SET function, an error occurs.
%TIME2	Defines the current system time in the format of MMSSTH. You cannot set this field. If you attempts to use this field as the target of a SET function, an error occurs.
%TMM	Defines the numeric month. The values are 1 - 12; wherein, 1 stands for January and 12 stands for December.
%TMONTH	Defines the current English month. The field must be 9 bytes long and space filled. The values are January - December.
%TRN	Defines the 10-byte transaction number of the last Platform Server PROCESS request that was executed. Before the first PROCESS statement, the value is displayed as "***N/A****".
%TYYYY	Defines the numeric year.

Parameter	Description
	This parameter must be 4 bytes long.
`%USER`	Defines the z/OS user ID associated with the job that is executing the script. You cannot set this field. If you attempts to use this field as the target of a SET function, an error occurs.
`%WTORDATA`	Saves the information that is returned by the operator in response to the script WTOR function. Although you can change this variable in a SET command, it is typically referenced after a `WTOR` command to see the data that the console operator entered.
`%WTORLENGTH`	Used for the following two purposes by the script WTOR function: • When you set this parameter before the WTOR function, it defines the maximum number of characters that can be entered by the console operator in response to the WTOR message. • After the console operator replies to the WTOR message, the length of the information that is entered by the operator is saved in this field.
`%YDATE`	Yesterday's date is saved in this field in the format: CCYYDDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%YGDATE`	Yesterday's date is saved in this field in the format: CCYYMMDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%YGDATE1`	Yesterday's date is saved in this field in the format: 1YYMMDD. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.

Parameter	Description
`%YGDATE2`	Yesterday's date is saved in this field in the format: YYYYDDMM. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%YGDATE3`	Yesterday's date is saved in this field in the format: MMDDYYYY. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.
`%YGDATE4`	Yesterday's date is saved in this field in the format: DDMMYYYY. You cannot set this field. If you attempt to use this field as the target of a SET function, an error occurs.

Script Syntax Rules

All commands require double quotation marks (") around character variables with the exception of the `SAY` command. Routine names are not considered character variables and as such should not be enclosed in double quotation marks.

When using variables within a command, such as `WTO` or `WTOR`, that requires double quotation marks ("), add a space after the variable name. Additionally, you can terminate the variable name with a period (.). When a period is added after a variable name, it is removed from the statement.

The member SCRIPT within the Platform Server SAMPLIB contains an exec that shows how the various commands are utilized.

COMMENT

The format of this function is `*comment_data`. Wherein, the asterisk (*) denotes a comment. The asterisk (*) can be in any column, but must be placed before any characters. For example, `* this line is a comment`.

Note: A comment might also begin with REM. This mimics TCS, which uses this notation for comments.

:ROUTINE

This routine name can be used in GOTO or CALL requests which are the two function statements that transfer control to a routine name.

Note: Do not include the colon in a GOTO or CALL function that references the routine name.

The format of this function is `:`*`routine_name`*. The colon (:) in column 1 defines a routine name, and must be followed by one or more characters that make up the routine name. For example, `:SENDFILE`.

CALL

This function branches to the routine defined on the CALL statement. Control is given to the statement following the routine name. Execution returns to the line following the CALL function when the RETURN function is detected. Up to 16 levels of CALL are supported.

The format of this function is `CALL` *`routine_name`*. For example,

```
CALL SEND_ROUTINE
. . . . . . . . .
EXIT
*
:SEND_ROUTINE
. . . . . . . . .
RETURN
```

Note: The colon that is the first character of the routine name is not used in the CALL function.

CALLIDCAMS

This function calls the z/OS IDCAMS utility. It passes the parameter values defined by the *%IDCAMS1* - *%IDCAMS40* variables to the z/OS IDCAMS utility. The output of the IDCAMS utility is displayed in the scripting SYSTRACE DD statement. Special variable *%RC* is set with the code returned by the IDCAMS utility.

The format of this function is `CALLIDCAMS`. For example,

```
SET %IDCAMS1 = " /*   COMMENT LINE 1 */"
SET %IDCAMS2 = " LISTCAT ENT(S7T.MFTTCB.AUDIT) ALL"
SET %IDCAMS3 = " /*   COMMENT LINE 3 */"
CALLIDCAMS
SAY Return code from CALLIDCAMS=%RC
```

CALLJCL

You can use this function to call a user-written program directly from a script. Only one parameter can be passed to the program being called. The program being called must be defined either in LINKLIB or in STEPLIB of the job running the script. The parameter is passed to the user-written program by using the same linkage that is used when passing a JCL PARM statement to a program. Only character variables can be passed to the user program. Character data is passed as a string of characters.

The program called by the CALLJCL function, should not update any of the variables. Variables cannot be updated from the called program. The only data that can be passed back to the script file is the return code. Variable *%RC* is updated with the return code set by the called program.

If the program defined in the CALLJCL function is not found, an error is displayed and the script is terminated. If an abend occurs while in control of the user-written program, error messages are displayed and the script is terminated.

The format of this function is `CALLJCL` *`program_nameparameter_data`*. For example, `CALLJCL USERPGM %CHARDATA`. This command calls the USERPGM program and passes a parameter to the program. Assume that the variable is set in the following way: `%CHARDATA = "TESTDATA"`. When control is passed to USERPGM, R1 points to the address of an address list containing an address:

`R1 –> Address (parameter 1) –> X'0008' (the length of data passed), TESTDATA (the data passed)`

Note: USERPGM can be written in any language that supports z/OS linkage conventions.

CALLPGM

You can use this function to call a user-written program directly from a script. Many parameters can be passed to the program being called. The program being called must be defined either in LINKLIB or in STEPLIB of the job running the script. Parameters are passed to the user-written program by using standard z/OS linkage. Both numeric and character variables can be passed to the user program. Numeric variables are passed as a full word binary, while character data is passed as a string of characters.

> **Note:** Character strings passed point to the first byte of the data and are terminated by a single NULL character.

The program called by the CALLPGM function, should not update any of the variables. Variables cannot be updated from the called program. The only data that can be passed back to the script file is the return code. Variable *%RC* is updated with the return code set by the called program.

If the program defined in the CALLPGM function is not found, an error is displayed and the script is terminated. If an abend occurs while in control of the user-written program, error messages are displayed and the script is terminated.

The format of this function is `CALLPGM` *`program_nameparm1parm2parm3parm4parm5`* `. . .` *`parm n`*. For example, `CALLPGM USERPGM %A %B 5 %CHARDATA`. This command calls the USERPGM program and passes 4 parameters to the program. Assume that the variables are set in the following way: `%A = "12"`, `%B = 255`, and `%CHARDATA = "TESTDATA"` . When control is passed to USERPGM, R1 points to the address of an address list containing 4 addresses: R1 > Address (parameter 1), Address (parameter 2), Address (parameter 3), Address (parameter 4) > `c'12',x'00', x'000000FF', x'00000005', c'TESTDATA',x'00'`.

> **Note:** Because standard z/OS linkage conventions are used, USERPGM can be written in any language that supports z/OS linkage conventions.

CLEARIDCAMS

You can use this function to clear the contents of all `%IDCAMS` variables. This can be used if you need to execute CALLIDCAMS multiple times and pass different data to IDCAMS.

The format of this function is `CLEARIDCAMS`. For example, `CLEARIDCAMS`.

DEBUGOFF

You can use this function to turn off function statement debugging. It is a good practice to always leave function statement debugging turned off, unless directed to turn it on by TIBCO technical support. By default, function statement debugging is turned off at the start of script execution.

The format of this function is `DEBUGOFF`. For example, `DEBUGOFF`.

Note: No operands are allowed on this command.

DEBUGON

You can use this function to turn on function statement debugging. It is a good practice to always leave function statement debugging turned off, unless directed to turn it on by TIBCO technical support. By default, function statement debugging is turned off at the start of script execution.

The format of this function is `DEBUGON`. For example, `DEBUGON`.

Note: No operands are allowed on this command.

EXIT

You can use this function to terminate the execution of the script. An optional parameter defines the return code that should be passed back to the caller. If the return code is not specified on the EXIT function, the `%MAXRC` variable defines the return code passed back to the caller.

The format of this function is `EXIT` *`return_code`*. For example,

```
EXIT 16
```

The script execution terminates with the return code `16`.

```
SET %MAXRC = 32
EXIT
```

The script execution terminates with return code `32`.

GOTO

You can use this function to defines the script to branch to a routine defined within the script.

The format of this function is `GOTO` *routine_name*. For example,

```
GOTO SENDFILE
```

The routine name can contain 1 - 16 characters. The colon that is the first character of the routine name is not used in the GOTO function.

IF/THEN/ELSE/ENDIF

You can use this function to perform a comparison to see whether the condition tested is true or false. If the condition is true, the statements following the IF/THEN statement are executed until the ELSE of the ENDIF function is detected. If an ELSE is detected, the instructions following the ELSE are ignored until the ENDIF function is detected. If the condition is false, the statements following the IF/THEN function are ignored until an ELSE or ENDIF is detected. If ELSE is detected, the instructions following the ELSE are executed until the ENDIF is detected.

The format of this function is `IF operand1 condition operand2 THEN`. Both `operand1` and `operand2` must be of the same variable type (that is to say both are numeric or both are characters).

If the operand detected in an IF statement is a GOTO function, the IF clause is terminated in the same manner that the ENDIF terminates the IF function. Execution continues at the statement following the routine specified by the GOTO function.

The syntax rules of this function are as follows:

- The first line must contain the IF statement, condition to be tested, and the THEN statement.
- The ELSE statement must be on a line with no other statements.
- Valid conditions include greater than (>), less than (<), and equal to (=) .
- Operands can be tested for one or more conditions.

- The IF function is ended by the ENDIF statement.
- Character data specified in `operand1` or `operand2` must be enclosed in double quotation marks.

For example:

```
IF %RC > 0 THEN
       SET %FAIL = %FAIL + 1
       SAY Request failed RC=%RC
   ELSE
       SET %GOOD = %GOOD + 1
       SAY Request Successful
   ENDIF
IF %WTORREPLY <> “C” THEN
      EXIT 32
  ELSE
      NOOP
  ENDIF
```

ⓘ **Note:** Nested IF statements are not supported. If you have multiple levels of CALL, you can have one active IF statement within each CALL level.

INCLUDE

You can use this function to copy Platform Server statements from a predefined member. The INCLUDE DD statement must be defined in JCL. The member to be included must be in the PDS defined by the INCLUDE DD statement.

The format of this function is `INCLUDE` *`member`*. For example,

```
INCLUDE MEMBER1
```

NOOP

This is a dummy instruction that is typically used as the target of the IF/THEN/ELSE function. It has no effect on the execution of the script, and is used for cosmetic purposes only.

The format of this function is `NOOP`.

PROCESS

This function contains the parameter statements that will be passed to the batch interface program.

Unless the PROCESS statement `WAIT =YES` is defined, the PROCESS request terminates when the request is scheduled. By specifying `WAIT =YES`, the PROCESS request terminates only after the transfer request has been completed either successfully or unsuccessfully.

The `%RC` variable is updated with the return code set by the Batch interface program. See Batch Interface for more information on the supported PROCESS statements.

The format of this function is `PROCESS,`*`process_name`*`,TRANSFER,{SEND|RECEIVE}`.

QUOTE

You can use this function to define whether the script should include the double quotation marks when substituting data within PROCESS statements. The default value, `ON`, indicates that double quotation marks must be added when substituting variables within PROCESS statements. When you define the value as `OFF`, double quotation marks will not be added when substituting variables within PROCESS statements. This might be useful if you include multiple variables in a single PROCESS statement.

> **Note:** This parameter does not affect variable substitution within non-PROCESS statements. This parameter applies to ALLOC, DEALLOC, CALLJCL, and CALLPGM statements as well as Process statements.

The format of this function is `QUOTE ON|OFF`. For example,

```
QUOTE OFF
```

RETURN

You can use this function to terminate a routine executed by the CALL function. Execution returns to the statement following the CALL request. If a RETURN function is detected without a corresponding CALL function, an error is displayed and the script is terminated.

The format of this function is `RETURN`.

SAY

You can use this function to display information on the SYSTRACE DD statement. Variable substitution is performed for all variables in the message. More than one variable can be included in a single SAY function call.

The format of this function is `SAY` *`data_to_be_displayed`*. For example,

```
SET  %FILE = “TEST.ACCT.FILE”
PROCESS . . . .



       . . . . . . . . . . . . .
       WAIT=YES
SAY Request for file %FILE completed with Return Code=%RC
```

In this example, the output of the SAY command is Request for file "TEST.ACCT.FILE" completed with Return Code=0.

SET

You can use this function to change the value of data in a variable.

The format of this function is `SET` *`%target = operand2 { +- Operand3}`*.

The SET function can be expressed using various methods. All expression methods require a variable to be in the target field of the function. When a variable is used as `operand 2` or `operand3` of a SET function, the variable must have been initialized. Otherwise, an error occurs. Character data specified in `operand 2` or `operand3` must be enclosed in double quotation marks.

The following examples show the various expression methods of the SET function:

- `SET %A = 1`

  This command sets the variable *%A* to `1`.

- `SET %A = "TEST.DATA"`

  This command sets the variable *%A* to `TEST.DATA`.

- `SET %A = %A - 1`

  This command subtracts 1 from the numeric variable *%A*, and then assigns the result to the numeric variable *%A*.

- `SET %B = %A + 1`

This command adds 1 to the numeric variable *%A*, and then assigns the result to the numeric variable *%B*.

- `SET %A = %A + %B`

  This command adds the values of the variable *%A* and the variable *%B*, and then assigns the result to the variable *%A*.

  If two character fields are added together, the fields are concatenated. If numeric and character variables are mixed, the numeric field is converted to a character field and the resulting target field is a concatenation of the two fields.

- `SET %NUM =` *`number`*

  The script program OSIUC000 supports length subscript to be used in numeric script variables when performing a conversion to character variables. Before this, when converting a numeric variable to a character variable, leading zeroes are not included.

  For example, `SET %NUM = 1234`. It supports the following expression methods:

  - `SAY NUM = %NUM`

    The output of the SAY command is NUM = 1234.

  - `SAY NUM = %NUM(2)`

    The output of the SAY command is NUM = 34.

  - `SAY NUM = %NUM(6)`

    The output of the SAY command is NUM = 001234.

SETC

You can use this function to change the value of data in a variable. This function can be used to concatenate many variables into a single variable.

The format of the SETC function is `SETC` *`%target`* `= "operand2" { +- "Operand3"}`

The SETC function can be expressed using various methods. All expression methods require a variable to be in the target field of the function. When a variable is used as `operand2` or `operand3`, the variable must have been initialized. Otherwise, an error occurs. Character data or variables specified in `operand2` or `operand3` must be enclosed in double quotation marks.

The SETC function is similar to the SET function, with the following exceptions:

- It works for character variables only.

- You must enclose the operands in double quotation marks, even when the operand is a script variable.
- Variable substitution performed by the script program for `operand2` and `operand3` are not enclosed in double quotation marks. You must enclose the parameters in double quotation marks.

The following examples show the various expression methods of the SETC function:

- `SETC %A = "TEST.DATA"`

  This command sets the *%A* variable to `TEST.DATA`.

- `SETC %A = "%A.%B.%C.%D.%E"`

  This commands concatenates the contents of variables *%A*, *%B*, *%C*, *%D*, and *%E*, and then assigns the result to the variable *%A*.

  > ℹ **Note:** The period at the end of each variable is a termination character for the variable.

- `SETC %A = "%A..%B..%C..%D..%E"`

  This commands concatenates the contents of variables *%A*, *%B*, *%C*, *%D*, and *%E*, and then assigns the result to the variable *%A*.

  > ℹ **Note:** The first period between each variable terminates the variable. The second period between the variables is added to variable *%A*.

- `SETC %A = "%A" + "%B"`

  This command concatenates the contents of the variable *%A* to the contents of the variable *%B* and assigns the result to the variable *%A*.

TRACEOFF

You can use this function to turn off function statement tracing. It is a good practice to always leave function statement tracing turned on because this provides you with a history of the statements that have been executed. The only reason to turn off function statement tracing is that you do not want to display some statements on the SYSTRACE file. By default, Function statement tracing is turned on at the start of script execution.

The format of this function is `TRACEOFF`.

Note: No operands are allowed on this command.

TRACEON

You can use this function to turn on function statement tracing. It is a good practice to always leave function statement tracing turned on because this provides you with a history of the statements that have been executed. By default, function statement tracing is turned on at the start of script execution.

The format of this function is `TRACEON`.

Note: No operands are allowed on this command.

TRNWAIT

By using this function, you can wait for transfers to complete. You can wait for 1 to 10 transfers to complete by entering one or more transaction IDs. Optionally, you can wait for all transfers that have been queued by this script instance to complete by specifying `ALL`.

As transfers are queued, the scripting program keeps track of up to 200 transfers. `TRNWAIT ALL` waits for all transfers to compete. If all transfers are successful, the script variable `%TRNSTAT` is set to `SUCCESS`. If any transfer fails, the script variable `%TRNSTAT` is set to `FAILED`. When a transfer is not found, the status is neither `SUCCESS` nor `FAILED`, it is ignored.

Note: `%TRNSTAT` must include 8 bytes and spaces must be included.

When using the `TRNWAIT` parameter to track transfers scheduled by the Script interface, you must use SNA or TCP to communicate with the started task.

The format of this function is `TRNWAIT {ALL | TransId1 TransId2…TransId10}`.

UNDEF

You can use this function to define whether the script program should treat an undefined variable as an error or as a NULL variable.

The format of this function is `UNDEF ERROR|NULL`. For example, `UNDEF NULL`.

The default value, `ERROR`, indicates that the script program should treat undefined variables as errors which will terminate script operations. While, `NULL` indicates that the script program should treat undefined variables as a NULL string.

WAITSECS

You can use this function to put the script into a wait state for a predefined number of seconds. You can define the number of seconds in the wait command to `1 - 9999`. If the number of seconds is not specified, the default value of `1` second is used.

The format of this function is `WAITSECS` *`number_of_seconds`*. For example, `WAITSECS 20` indicates putting the script in a wait for 20 seconds.

WTO

You can use this function to display a message on the Operator Console. This can be combined with message automation to perform a task when a specific message is displayed.

The format of this function is `WTO "`*`message_to _be_display`*`"`. For example, `WTO "MESSAGE1 File transfer request to node %NODE failed"`.

The message to be displayed must be imbedded in double quotation marks. Variable substitution is performed for the message in double quotation marks.

WTOR

By using this function, you can display a message on the Operator Console and wait for a reply. The message to be displayed must be imbedded in double quotation marks. Variable substitution is performed for the message in double quotation marks.

The variable `%WTORLENGTH` defines the maximum number of bytes that the console operator can enter in response to the WTOR command. When the WTOR function is completed, the following fields are updated:

- `%WTORDATA`: Contains the data entered by the console operator.
- `%WTORLENGTH`: Contains the length of data entered by the console operator.

The format of this function is `WTOR "`*`message_to_be_displayed`*`"`. For example, use the following commands to display a message and wait for a one-byte response:

```
SET %WTORLENGTH = 1
WTOR “MESSAGE1 SEND to node %NODE failed: Reply Y to continue”
SAY Reply Length=%WTORLENGTH: Reply Data=%WTORDATA
```

Examples of Holding Long File Names

When you are working with long file names, you might experience a problem where your symbolic parameter goes over the 120-byte character limit. The following two examples show how to hold a long file name by using two variables:

```
SET %RF = "/data/b2b/JPB/SENDING/Final/JUNE/"
SET %RF = %RF + "PGPMFOUT.ENCRYPTED-FILENM.PGPO9156ARMPFN"
SET %RF2 = ".D" + %GDATE
SET %RF2 = %RF2 + ".T"
SET %RF2 = %RF2 + %TIME
QUOTE OFF
PROCESS ....
....
RFILE="%RF.%RF2"
...
```

Note: The `QUOTE OFF` parameter is important because it tells the script program not to include quotation marks when performing variable substitution.

```
SET %RF = "/data/b2b/JPB/SENDING/Final/JUNE/"
SET %RF = %RF +"PGPMFOUT.ENCRYPTED-FILENM.PGPO9156ARMPFN"
QUOTE OFF
PROCESS ....
....
RFILE="%RF..D%GDATE..T%TIME"
...
```

Note: For the ".." in the `RFILE` parameter, the first dot indicates the end of the variable, while the second dot is part of the file name.

Sample Script

The sample script takes a file, and sends it to the same remote system for ten times. Each time the remote file name is concatenated with a counter to make the names different. If the request fails, a message is displayed on the z/OS console, and the script asks the operator whether to continue. After 10 transfer requests, the script terminates.

The following statements set the three numeric variables to `0`:

```
SET %COUNT = 0
SET %FAIL  = 0
SET %GOOD  = 0   Sample
```

The following statements define a routine named SENDLOOP:

```
:SENDLOOP
   SET %LF = "TEST.FILE"
   SET %RF = "TEST.FILE" + %COUNT
   SET %IPADDR = "127.0.0.1"
   SET %IPPORT = 46464
  CALL SENDTEXT
*
  IF %RC = 0 THEN
     SET %GOOD = %GOOD + 1
     GOTO CONTINUE
   ELSE
     NOOP
   ENDIF
 CALLPGM TESTPGM %RC %LF %RF
 SET %FAIL = %FAIL + 1
 SET %WTORLENGTH = 1
*
```

The `:SENDLOOP` statement defines a routine name, `SENDLOOP`.

The second statement sets the character variable `%LF`.

The statement `SET %RF = "TEST.FILE" + %COUNT` sets the character variable `%RF` to the contents of the character variable `%LF` concatenated with a numeric variable. The first time this statement is executed, the variable `%RF` is assigned the value "`TEST.FILE0`". The second time this routine is executed (when `%COUNT` = `1`), the variable `%RF` is assigned the value "`TEST.FILE1`".

The following two statements set the character variables `%IPADDR` and `%IPPORT`.

The `CALL SENDTEXT` statement passes control to routine SENDTEXT.

The IF statement checks the current return code (`%RC`). If the current return code is zero, the variable `%GOOD` is incremented by 1, and the execution is transferred to routine CONTINUE. If the return code is not zero, the execution is transferred to the statement following ENDIF.

The CALLPGM statement defines the script program to call the TESTPGM program which passes three parameters. When the TESTPGM program is completed, the variable `%RC` is updated with the return code set by TESTPGM.

The next statement increments the variable `%FAIL` by 1.

The last statement sets the reserved variable `%WTORLENGTH` to `1`. This sets the maximum length of the data that an operator can respond to a WTOR request to `1`.

The following statements define a routine named WTORMSG:

```
:WTORMSG
 WTOR "ERROR SENDING FILE %RF REPLY Y=CONTINUE   X=CANCEL"
 SAY WTORLENGTH=%WTORLENGTH    WTORDATA=%WTORDATA
*
 IF %WTORDATA = "X" THEN
      EXIT
    ENDIF
 IF %WTORDATA = "Y" THEN
     GOTO CONTINUE
    ENDIF
 GOTO WTORMSG
*
```

The `:WTORMSG` statement defines a routine name, `WTORMSG`.

The WTOR function issues a z/OS WTOR request. It displays a message on the z/OS console, and waits for an operator reply. Because the `%WTORLENGTH` field is set to `1`, the reply can be a maximum of 1 byte long. The variable `%WTORLENGTH` returns the length of the data actually entered by the operator, while the variable `%WTORDATA` returns the data entered by the operator.

The SAY function displays the contents of the variables `%WTORLENGTH` and `%WTORDATA`.

The first IF function checks whether the operator responded to the WTOR function with the character X. If so, the script terminates because of the EXIT call. Because no parameter is specified by the EXIT call, the variable `%MAXRC` sets the termination return code. No ELSE function is specified because the IF function is terminated by the ENDIF function before an ELSE function is encountered.

The second IF function checks whether the operator responded to the WTOR function with the character Y. If so, the control is passed to the routine labeled CONTINUE. No ELSE function is specified because the IF function is terminated by the ENDIF function before an ELSE function is encountered.

The last function transfers control to the routine labeled WTORMSG. In this case, the routine is looping until the operator enters either Y or X in response to the WTOR function.

The following statements define a routine named CONTINUE:

```
:CONTINUE
  SET %COUNT = %COUNT + 1
  IF %COUNT >= 10 THEN
     GOTO DONE
    ENDIF
*
  WAITSECS 2
  GOTO SENDLOOP
*
```

The `:CONTINUE` statement defines a routine name, `CONTINUE`.

The first SET statement increments the variable `%COUNT` by 1.

The IF statement checks whether the variable `%COUNT` is greater than or equal to 10. If the condition is true (`%COUNT` is greater than or equal to 10), control is passed to routine DONE. If the condition is not true, control is passed to the statement following the ENDIF statement.

The WAITSECS function causes the script to go into a wait for 2 seconds.

The GOTO statement causes execution to pass to routine SENDLOOP.

The following statements define a routine named DONE:

```
:DONE
  SAY GOOD COUNT=%GOOD
  SAY FAIL COUNT=%FAIL
  EXIT
*
```

The `:DONE` statement defines a routine name, `DONE`.

The first SAY statement displays a message containing the variable `%GOOD`.

The second SAY statement displays a message containing the variable `%FAIL`.

The EXIT statement causes termination of the script. Because no parameter is supplied with the exit, the variable `%MAXRC` supplies the script return code.

The following statements define a routine named SENDTEXT:

```
:SENDTEXT     * PROCESS SEND FILE REQUEST
   PROCESS,SENDB,TRANSFER,SEND
        DSN=%LF
        REMOTE_FILE=%RF
        RUSER=*PROFILE
        TYPE=TEXT
        WAIT=YES
        EFFECT=R
        TRY=1
        IPADDR=%IPADDR
        IPPORT=%IPPORT
   WTO "RETURN CALL FROM SENDTEXT=%RC."
   RETURN
```

The `:SENDTEXT` statement defines a routine name, `SENDTEXT`.

The PROCESS statement and all statements up to and including `IPPORT` are input into the Platform Server Batch interface program. These statements cause a TEXT SEND request to be initiated to the system defined by the `IPADDR` parameter. The `EFFECT=R` statement indicates replacing the file if the file already exists. If the file does not exist, an error is generated. When the file transfer is completed, the variable `%RC` is updated with the return code set by the Batch interface program. Control is then passed to the statement following the last PROCESS parameter. The `WAIT=YES` statement indicates that control should not be passed back to the script until the transfer has been completed either successfully or unsuccessfully.

Note: Variable substitution is performed before control is passed to the Batch interface program.

The WTO function displays a message on the operator console. In this case, it displays a message with the variable `%RC`. The return code for the PROCESS request is substituted for the variable `%RC` in the WTO message.

The RETURN statement passes control to the statement following the statement that issued the CALL function to this routine.

Running Scripts within Platform Server Started Task

Apart from scheduling scripts requests in an address space separate from the Platform Server address space, you can also schedule a script request in the Platform Server address space.

Typically, scripts are executed in an address space separate from the Platform Server address space. You submit a job, and the job executes the script. The script job can then submit file transfer requests to the Platform Server to process, and gets the response from the Platform Server. When the script terminates, the job also terminates. Besides, you can also schedule a script request in the Platform Server address space. When a request is scheduled in the Platform Server address space, the batch job that schedules the request typically terminates when the script is scheduled. The script itself is executed within the Platform Server address space.

The following example shows a sample JCL that submits a script to run in the Platform Server address space:

```
//FUSJOB   JOB   ,CLASS=A,MSGCLASS=X
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB  DD DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
  PROCESS,RUNSCRPT,TRANSFER,SCRIPT
  SCRIPT="MY.SCRIPT.LIB(SCRIPT)"
//
```

The JCL schedules the script in `MY.SCRIPT.LIB(SCRIPT1)` in the Platform Server started task called FUSION. The file transfer requests also run in the FUSION started task.

You can run the script in one Platform Server started task, and the file transfers in another started task. Use the `SERVER` parameter to define the Platform Server started task that will run the file transfers.

In the following example, the JCL schedules a script to run in STC FUSION; while, the file transfers will run in STC FUSION1.

```
//FUSJOB   JOB   ,CLASS=A,MSGCLASS=X
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB  DD DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
  PROCESS,RUNSCRPT,TRANSFER,SCRIPT
  SCRIPT="MY.SCRIPT.LIB(SCRIPT1),SERVER=FUSION1"
//
```

Symbolic variables can also be passed to the script to be executed. The following JCL shows how symbolic variables can be passed to a script:

```
//FUSJOB   JOB   ,CLASS=A,MSGCLASS=X
//OSIUB000 EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
//STEPLIB  DD DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD SYSOUT=*
//SYSIN  DD *
  PROCESS,RUNSCRPT,TRANSFER,
  SCRIPT="MY.SCRIPT.LIB(SCRIPT1),SERVER=FUSION1,%LDSN=MY.DSN"
//
```

In this example, the JCL creates a symbolic parameter called `%LDSN`, and assigns the value of `MY.DSN` to it. This information is passed to the script when the script executes.

Script Execution in Platform Server Started Task

You can schedule a script request in the Platform Server address space.

When a script is executed in a stand-alone environment, the script interface program reads input from the SYSIN DD statement. It sends output to the SYSPRINT and SYSTRACE DD statements. To allow multiple copies of this program to work at the same time, the Platform Server allocates different DD statements for each of the input and output functions. This is done without any user intervention. The only difference is that the script output is no longer associated with a stand-alone job. All script output is associated with the Platform Server started task. The Platform Server allocates a DD statement for all output and sends the output to the JES spool. The Platform Server then displays a message indicating the DD statement to which the output is sent. The following example is a sample of the message:

PGTE4230I Activity I201200001 Script execution started: Output will be in ddname H0100001.

You can then look at DD statement H0100001 for the script output. All script output is deallocated after the file is closed, so that it can be deleted from the JES spool if necessary.

INDIRECT EXEC Statement

Because of the z/OS 256-byte limitation on scheduling scripts to run in the started task, to pass more than 256 bytes to a script running in the started task, for example passing several long dataset names, can be problematic. The Platform Server addresses these situations through the use of an indirection file that may contain one or more parameters and the dataset name of the script to execute that, in aggregate, exceed 256 bytes.

The Platform Server postprocessing actions (PPAs) can use an indirection file residing on the z/OS system where the file is to be executed (rather than an actual script file) by including the `INDIRECT=YES` statement.

The format of the statement is `INDIRECT=`{`YES` | `DELETE` | `NO`}.

Where:

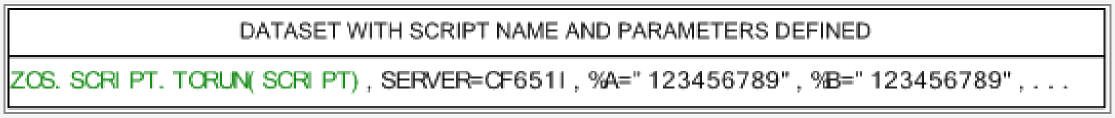
- `YES`: Indicates that the data in the defined dataset is not the actual script to run but is the dataset whose contents contain the name of the script to run and the required parameters.
- `DELETE`: Indicates that the data in the defined dataset is not the actual script to run but is the dataset whose contents contain the name of the script to run and the required parameters. When `DELETE` is specified, the script program deletes that same dataset when the script is completed.

  > **Note:** `DELETE` can only be defined when the dataset name is not contained in a DSN containing a member name.

- `NO`: indicates that the data in the defined dataset is the actual script to run.

The indirect dataset that contains the script name and necessary parameters is built into a single record separating each item with commas.

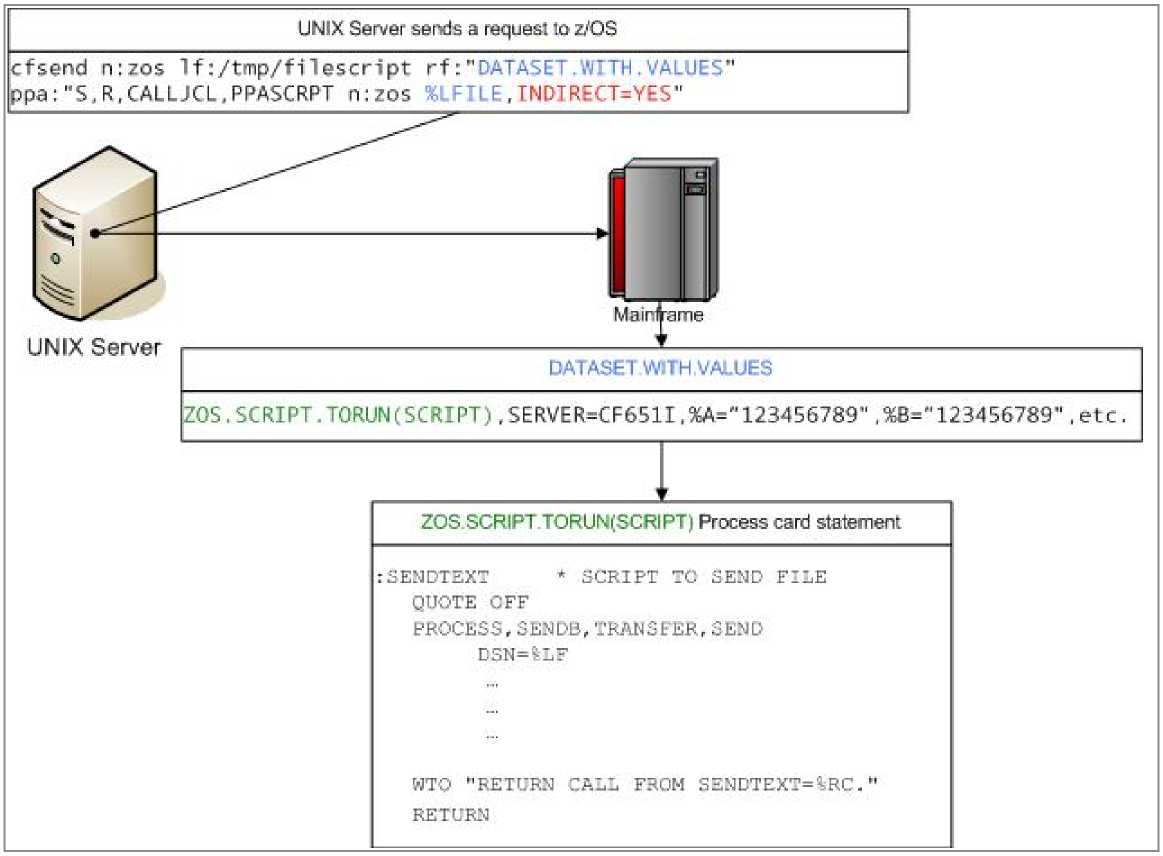
> **Note:** The fully-qualified dataset name of the script must be defined first, as shown in the following figure:



DATASET WITH SCRIPT NAME AND PARAMETERS DEFINED
ZOS.SCRIPT.TORUN(SCRIPT),SERVER=CF651I,%A="123456789",%B="123456789",...

Scripts are most frequently launched through remote execution statements such as a PPA (for details, see the descriptions of the `POST_ACTION` parameter in Batch Interface) where the INDIRECT parameter might be most useful. Another frequent occurrence is when you queue a script through the FUSSCRPT REXX exec.

The following figure shows an example transfer of a UNIX platform uploading a data file that contains the name of the script to be run with the various values to be passed to that script located on the remote z/OS platform:



UNIX Server sends a request to z/OS

```
cfsend n:zos lf:/tmp/filescript rf:"DATASET.WITH.VALUES"
ppa:"S,R,CALLJCL,PPASCRPT n:zos %LFILE,INDIRECT=YES"
```

DATASET.WITH.VALUES

```
ZOS.SCRIPT.TORUN(SCRIPT),SERVER=CF651I,%A="123456789",%B="123456789",etc.
```

ZOS.SCRIPT.TORUN(SCRIPT) Process card statement

```
:SENDTEXT      * SCRIPT TO SEND FILE
   QUOTE OFF
   PROCESS,SENDB,TRANSFER,SEND
        DSN=%LF
         ...
         ...
         ...

   WTO "RETURN CALL FROM SENDTEXT=%RC."
   RETURN
```

Note: The new PPA program, PPASCRPT, is used in this example. See PPA Program: PPASCRPT for more details about the program.

PPA Program: PPASCRPT

The PPASCRPT program runs as a PPA program in the Platform Server started task. It can accept two parameters from the PPA and schedule a script to be executed by the started task.

The following table lists the two parameters of the PPASCRPT program:

Parameter	Description
`Script Name`	Required. This parameter defines the fully qualified dataset name of a script that must be submitted.
`Node Name`	Optional. This parameter defines the name of the node to be associated with the script to be executed. If defined, the node name must be the first parameter entered. This parameter can be entered by using the following syntax: `N: | N= | NODE: | NODE=`. **Note:** This parameter is not case sensitive. The node name is always translated to uppercase.

The following examples are PPA statements that use the PPASCRPT program:

```
PPA=”S,R,CALLJCL,PPASCRPT n:NYNODE your.script.library(script1)”
PPA=”S,R,CALLJCL,PPASCRPT node=lanode your.script.library(script2),
%P1=123,%P2=abc”
PPA=”S,R,CALLJCL,PPASCRPT your.script.library(script3),INDIRECT=YES,
%P1=123,%P2=abc”
```

Note: You can override parameters in your indirect file or add a new parameter if needed to the indirect file by placing parameters with values after the `INDIRECT` parameter.

SAPI (SYSOUT API)

With the SAPI interface, the Platform Server supports extracting data from a JES Spool and sending this data to a remote node. One of the more useful features of the Platform Server is the ability of sending the data to a remote system as a file rather than as a report. You can specify substitutable parameters to make the remote file unique.

> **Note:** For more specific information on the SAPI parameters, review the parameters defined in the *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

When the Platform Server SAPI interface is enabled, the SAPI interface scans the NODE definition for SAPI selection criteria. If no selection criteria are defined, the next node is checked until all of the nodes are scanned. When a NODE definition is found with SAPI selection criteria defined, the Platform Server calls JES to see whether any output for the selection criteria exists. If output exists, the Platform Server queues a request to that node. Then, the request is processed and the data is sent to the remote system. When the request is completed, the spool data disposition depends on the NODE `SAPI_DISP` parameter.

> **Note:** Data is not staged in the Platform Server SAPI interface. Data is read directly from the JES spool. The data is multi-threaded so that SAPI requests can be active to multiple nodes at the same time. However, for a single node, only one SAPI request can be active at a time, while other non-SAPI requests can be active with the same node at the same time.

SAPI Security

When an SAPI request is executed, the local user ID is the user ID of the started task. The remote user ID depends on the NODE parameters defined.

If the NODE `DEFAULT_USERID` and `DEFAULT_PASSWORD` parameters are defined, the remote user ID and password is taken from the NODE definition. If the NODE definition is not defined, the remote user ID defaults to `*PROFILE`. In that case, a user Profile must be defined for the started task user ID and the node name.

Note: When sending data to a Windows system, you must specify the DEFAULT_LOGON_DOMAIN parameter to define the Windows domain where the user must be validated.

Starting SAPI Interface

To start the SAPI interface, set the GLOBAL SAPI_INTERFACE parameter to YES. The SAPI interface will start the next time that Platform Server is started.

Defining SAPI Selection Criteria

You can select data from the JES spool bases on a combination of four NODE parameters.

The Platform Server does not process held output. In addition, the Platform Server requests that JES processes the data in priority order, although that priority is up to the individual JES system.

The four NODE parameters are as follows:

- SAPI_CLASS: JES SYSOUT classes. You can define up to eight JES SYSOUT classes.
- SAPI_DEST: JES destination
- SAPI_FORM: JES form name
- SAPI_WRITER: JES writer name

These parameters are specified on individual NODE definitions. Each NODE definition that you define must have different selection criteria. Multiple selection parameters can be defined for a single NODE definition. The Platform Server only extracts data that meets all of the selection criteria.

Note: You cannot specify SAPI_CLASS as the sole selection criteria because all data for that class will be retrieved by the Platform Server.

It is a good practice to use SAPI_WRITER as selection criteria. The writer name can contain 1 - 8 characters. Most importantly, the writer name does not have to be predefined to the JES2 or JES3 systems. To send a dataset to a JES writer, you can specify the following parameter in the JCL that writes the output: `SYSOUT=(A,`*`writer_name`*`)`. Wherein, the parameter defined as `A` is a SYSOUT class. Any valid SYSOUT class can be defined.

ℹ **Note:** If the `SAPI_CLASS` parameter is defined, its class must match one of the classes defined.

SAPI Interval

The Platform Server checks the spool every 60 seconds for data in the spool.

ℹ **Note:** This interval is fixed and cannot be changed.

Defining SAPI Remote File

The Platform Server supports sending a spool file to a remote system as a file, rather than as a report.

You can use the NODE `SAPI_REMOTE_FILE` parameter to define the file name on the remote system. You can also define substitutable parameters to customize the remote file name, and make the file name unique.

ℹ **Note:** For more specific information of the SAPI parameters, review the parameters defined in the *TIBCO Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

SAPI File Transfer Parameters

You can set parameters to define the file transfers queued by the SAPI interface.

The following table lists the fixed parameters that can be used:

Parameter	Description
`TYPE=TEXT`	The records are processed as TEXT records. The system performs EBCDIC to ASCII conversion and adds the CRLF delimiter.
`PROCESS`	The process name is defaulted to `FUSNSAPI`.

Note: The `DEFAULT_COMPRESS` and `CHECKPOINT` parameters default to `NO`, unless the NODE `COMPRESS` and `CHECKPOINT` parameters are defined. If these parameters are defined, they will be taken from the NODE definition.

Defining SAPI File Disposition

You can use the NODE `SAPI_DISP` parameter of the Platform Server to specify the disposition of a dataset when a SPOOL request is completed.

You can set the following three dispositions:

- `KEEP`: Keeps the spool file.
- `HOLD`: Puts the spool file on hold, and tries the request again later.
- `DELETE`: Deletes the spool file.

The spool dispositions are used when a spool file request is completed and is removed from the Platform Server request queue. A request is removed from the request queue in the following three situations:

- Successful request: The request is successfully completed.
- Permanent network error: The request is ended with a network error and exceeds the try count defined.
- Permanent Error: The request terminates with an error other than a network error.

Note: When a request receives a network error and the request is eligible for a retry, the disposition is hard coded as `KEEP`. In that way, the Platform Server tries the request again when the errors are reset.

Example

This example shows how to enable the SAPI interface and how to define the SAPI parameters to send data to a remote system as a file.

To enable the SAPI interface, you must specify the following GLOBAL parameter:

`SAPI_INTERFACE=YES`.

The following example is a sample NODE definition:

```
**      SAPI Selection Criteria
*
SAPI_CLASS=A
SAPI_WRITER=ACCTDATA
*
**          SAPI Default parameters
*
SAPI_REMOTE_FILE=d:\printer\test.%JOBN..%JOBID..D%JDATE..T%TIME
SAPI_TRY=5
DEFAULT_COMPRESS=NO
CHECKPOINT=NO
SAPI_DISP=(DELETE,KEEP,HOLD)
*
**          SAPI Security Definitions
*
DEFAULT_USERID=ntuser
DEFAULT_PASSWORD=ntpass
DEFAULT_LOGON_DOMAIN=ACCTDOMAIN
*
**          TCP Connection Parameters
*
APPLICATION=FUSION
IPPORT=46464
IPADDR=192.168.0.100
PARALLEL=YES
WINNERS=5
```

In this example, the NODE definition uses SYSOUT class and writer as a filter. This node will only select output written to SYSOUT class A and SYSOUT writer ACCTDATA.

Any file transfers initiated as a result of data selected from the queue will be sent to the remote system as a file. The file name will be created based on the `SAPI_REMOTE_FILE` parameter, and substitutable parameters will be inserted into the file name to make it unique. The Platform Server will try the request for 5 times before the transfer fails with a permanent network error. If a network error occurs, the system will not compress the data, and requests will start from the beginning. When a request is completed successfully, the SYSOUT file will be deleted. When the request is completed with a permanent network error, the data will be kept so that the Platform Server can process it at a later time. When the request is completed with an error other than a network error, the data will be held.

When the data is sent to the remote system, the user ID will be ntuser and the password is ntpass. ACCTDOMAIN will be passed as the logon domain.

The TCP connection parameters define the IP address and IP port used for communicating with the remote node. The Fusion protocol will be used to communicate with this node, and a maximum of 5 concurrent initiator transfers are supported.

FUSUTIL

You can use the Platform Server utility, FUSUTIL, to rename or delete a file, or determine whether a file exists.

For details of the FUSUTIL parameters, see Parameter Description.

These commands are not meant to be utilized as stand-alone commands. They are built to be able to work with the following Platform Server features:

- SEND command: See Using FUSUTIL in SEND Command.
- Post Processing Actions (PPA): See Using FUSUTIL in Post Processing Actions (PPA).

The FUSUTIL utility can run on the Platform Server. It can also execute on MFT Internet Server V8.1 and above.

- TIBCO Managed File Transfer Platform Server for z/OS 5.5 and above
- TIBCO Managed File Transfer Platform Server for UNIX 5.5 and above
- TIBCO Managed File Transfer Platform Server for Windows 5.5 and above
- TIBCO Managed File Transfer Platform Server for AS/400 5.4.2 and above

Although utility programs on each of the platforms on which the system runs can do some of these functions, a single program that can use the same syntax to perform each of the above functions is preferred. The FUSUTIL utility is designed to have a single syntax that works across all platforms that support the Platform Server.

FUSUTIL Security

The FUSUTIL utility is run under the authorization of the user that initiated the Platform Server request.

The user must have authorization to execute the desired function. Otherwise, the request fails with an error indicating that the user is not authorized to perform the function.

FUSUTIL Programs

The FUSUTIL commands consist of two components: the command name and the data to be passed to the FUSUTIL program.

The case of the command name depends on the platform being used. On UNIX, because the FUSUTIL utility is stored as a lowercase executable program, the command name must be defined in lowercase. On other platforms, the command name is not case sensitive. Therefore, the command name can be entered in lowercase, uppercase, or a mixture of uppercase and lowercase.

FUSUTIL DELETE

You can use the FUSUTIL DELETE program to delete a file.

The program can be entered in one of the following two ways:

```
FUSUTIL DELETE name_of_file_to_be_deleted
```

```
FUSUTIL D name_of_file_to_be_deleted
```

Note: The DELETE function can be defined by the word DELETE or the character D.

At least one space must be added after the program name FUSUTIL and before the name of the file to be deleted. The file to be deleted must be defined; otherwise, the request fails and an error is displayed.

The file name must be the fully qualified file name with all directories and qualifiers defined. On systems such as UNIX where the file system is case sensitive, the file name must exactly match the file name in the file system.

For example, the following command deletes the file `mvsfile.acct.Y2004.JAN`:

```
FUSUTIL D mvsfile.acct.Y2004.JAN
```

FUSUTIL EXIST

You can use the FUSUTIL EXIST program to determine whether a file exists on a platform.

Note: On z/OS, it also checks whether the file is in exclusive use by another program.

The program can be entered in one of the following two ways:

```
FUSUTIL EXIST name_of_file_to_be_checked
```

```
FUSUTIL E name_of_file_to_be_checked
```

Note: The EXIST function can be defined by the word EXIST or the character E.

At least one space must be added after the program name FUSUTIL and before the name of the file to be checked. The file to be checked must be defined; otherwise, the request fails and an error is displayed.

The file name must be the fully qualified file name with all directories and qualifiers defined. On systems such as UNIX where the file system is case sensitive, the file name must exactly match the file name in the file system.

For example, the following commands check whether the file `mvsfile.acct.Y2004.JAN` exists. On z/OS, they also check whether an exclusive ENQ exist on the file. If yes, the request fails as if the file does not exist.

```
fusutil E mvsfile.acct.Y2004.JAN
```

```
fusutil EXIST mvsfile.acct.Y2004.JAN
```

FUSUTIL RENAME

You can use the FUSUTIL RENAME program to rename a file.

This program can be entered in one of the following two ways:

```
FUSUTIL RENAME old_file_namenew_file_name
FUSUTIL R old_file_namenew_file_name
```

Note: The RENAME function can be defined by the word RENAME or the character R.

At least one space must be added after the program name FUSUTIL and before the old file name, and at least one space must be added between the old file name and the new file name. The old file name and the new file name must be defined; otherwise, the request fails and an error is displayed. On systems such as UNIX where the file system is case sensitive, the file names must exactly match the file name in the file system.

For example, the following commands rename the file `/temp/acctfile` to `/temp/bkup/acctfile`.

```
fusutil R /temp/acctfile /temp/bkup/acctfile
fusutil RENAME /temp/acctfile /temp/bkup/acctfile
```

Using FUSUTIL in SEND Command

The FUSUTIL utility can be used within a SEND file transfer.

The FUSUTIL commands typically run as the second step of a file transfer request. You can use this method to get confirmation that the RENAME or DELETE function actually works. Additionally, you can use the EXIST function as part of a z/OS Platform Server script to check whether a file exists on a remote platform; or you can use it as part of a UNIX shell script to perform the same function as on z/OS Platform Server. If the file exists, you can then issue a file transfer request to receive the file from the remote system.

Example

If you are initiating a file transfer request on Windows, you can request the following PPAs to be performed after the file transfer is completed:

```
COMMAND="fusutil EXIST /acct/data/march.data"
```

This command checks whether the file `/acct/data/march.data` exists on the remote platform. A zero return code indicates that the file exists, while a non-zero return code indicates that the file cannot be found. A non-zero return code is also returned if a network error occurs, or if the syntax of the FUSUTIL command is invalid.

Using FUSUTIL in Post Processing Actions (PPA)

You can use the FUSUTIL utility in the Platform Server PPA functions to perform commands immediately after a successful or unsuccessful file transfer in a single request.

You can perform the following operations by using the FUSUTIL utility in the PPA functions:

- Use the RENAME function to archive a file after it is successfully sent.
- Use the DELETE function to delete a file after it is successfully sent.

Note: You can use the PPA substitutable parameters to simplify the functions.

Example

If you are initiating a file transfer SEND request on Windows, you can request the following PPAs to be performed after the file transfer is completed:

```
PPA="S,L,COMMAND,FUSUTIL R %LFILE %LFILE.D%GDATE.T%TIME"
```

This command renames the local file defined by the `%LFILE` substitutable parameter. It suffixes the date and time to the file name, effectively archiving the file.

Troubleshooting

This section provides possible solutions to some issues that you might encounter when using the Platform Server.

Restore Not Possible

Problem: on the RECEIVE command, the following error occurs when the installation dataset has been transformed to the mainframe with an `LRECL` that is not 80:

```
Restore not possible. Input data is not complete.
The first record is:
Received file appears not to be an Interactive Data Transmission
Facility file.
X'00FF0000'
dataset X.RECEIVE.INVALID.FILE from USERID on N1
Enter copy parameters or 'DELETE' or 'END' +
```

Solution: Make record length fixed at block 80.

Allocation Failure

Problem: The following error occurs on the `RECEIVE` command if the dataset received is already allocated.

```
RECEIVE INDS(prkevin.test.delete.loadlib)

dataset PROD.FT710.LOAD from PRKEVIN on AL
Enter restore parameters or 'DELETE' or 'END' +

RECEIVE command terminated. Output dataset unusable. +
Allocation failure for dataset 'PROD.STABLE.IND$FILE.LOADLIB'
dataset PROD.FT710.LOAD ALREADY IN USE, TRY LATER+
dataset IS ALLOCATED TO ANOTHER JOB OR USER
```

Solution: Use a new dataset for the `RECEIVE` command.

Rebuilding the Work Queue

Problem: Errors occur when you delete and redefine the work queue file while using the `DEFQUEUE` JCL to rebuild the work queue.

Solution: Shut down the server before you run the `DEFQUEUE` command.

Trouble with VSAM Files

Problem: Decoding VSAM errors (`RPLFDBWD`).

The Platform Server keeps a machine readable log to keep a record of all past activities. This log is a key sequenced (that is indexed) dataset. If errors occur to the audit file, the server prints a diagnostic field called `RPLFDBWD`.

Solution: You must know the following logic when analyzing the VSAM errors:

- The first byte identifies the specific IBM VSAM module (part of z/OS data facility product) which is reporting the error. This does not have meaning to you, but IBM technical staff can tell you what VSAM module this represents and what it is doing.
- The second byte identifies the type of error. The most common user caused error type is X'08' which means logic error.
- The third byte is not used.
- The fourth byte is the error reason code as documented in chapter 4 of the IBM manual: *OS/390/DFP Macro instructions for datasets*.

See the following examples on how to decode the RPL feedback word.

Example 1: `RPLFDBWD 47080048`

This is the VSAM request parameter list from the IBM supplied RPL mapping DSECT. It is interpreted as follows:

- The first byte identifies the specific IBM VSAM module that reports the error. In this example, X'47' identifies IDA019R8 which is a VSAM record management routine responsible for keyed I/O as per IBM technical support.
- The second byte, X'08', indicates that this is a logic error. If an LERAD routine has been specified in the VSAM ACB, it can be driven; however, VSAM LERADs is not used in the Platform Server for z/OS.
- The third byte is reserved.
- The fourth byte, '48', is the logical error reason code as documented in chapter 4 of the IBM manual: *OS/390 DFP 3.3 Macro instructions for datasets (SC26-4747)*. Essentially, a return code of X'48' indicates that the Platform Server tries to perform indexed I/O on a non-indexed dataset. For example, the Platform Server treats the dataset as KSDS, but it is actually ESDS or RRDS. This can be the result of an

installation error.

Example 2: `RPLFDBWD 0A080008`

- The first byte, X'0A', identifies the IBM VSAM Module IDA019RA, which detected the error; the module performs direct record locate.
- The second byte, X'08", indicates a logic error.
- The third byte is reserved.
- The fourth byte, X'08', indicates that the Platform Server tries to store a record with a duplicate key. This can be caused by the audit file and the work queue falling out of synchronization with each other.

Activating the VTAM Resources

Problem: VTAM resource does not become active.

Solution: Go through "Defining the VSAM Files" in *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide* and verify that the VTAM resources have been defined properly.

Remote Nodes Are Not Coming Active When the Product Is Started

Problem: You receive message PGTS1131E indicating that `MEMBER IS NOT ACTIVATED DUE TO ERRORS`.

Solution: Check the remote node configurations and verify that you have specified all of the correct information.

Transaction Does Not Start after It Is Queued.

Problem: Transaction is placed on the queue as inactive.

Solution: Check whether the remote node became active when you started the Platform Server. If not, check the remote node configurations and verify that you have specified all of the correct information. Check the eligible time and date to see whether they are the current time and date.

Repeated Error Message for the Same Transaction

Problem: You continually receive the same error message on the system console.

Solution: Check whether the retry limit is zero. If so, the Platform Server is having difficulty performing this transaction and is continually retrying. To avoid this, you can set the retry limit to a non-zero value.

Receiving `SENSE=087D0001`

Problem: VTAM cannot find the LU.

```
IST663I BFINIT REQUEST FROM NCP066 FAILED, SENSE=087D0001
IST664I REAL OLU=TIBCO.DANL1I2  REAL DLU=TIBCO.FUSNAPPL
IST889I SID=F2CF2303704E1476
IST894I ADJSSCPS TRIED FAILURE SENSE ADJSSCPS TRIED FAILURE SENSE
IST895I ISTAPNCP    08400007
IST314I END
```

Solution: The most likely cause of this is that the LU is not active. Check whether the LUs are all active.

Receiving SENSE=08570003

Problem: The session is not active.

```
IST663I BFINIT REQUEST FROM NCP066 FAILED, SENSE=08570003
IST664I REAL OLU=TIBCO.DANL1I2  REAL DLU=TIBCO.FUSNAPPL
IST889I SID=F2CF2303704E1708
IST1138I REQUIRED RESOURCE TIBCO.FUSNAPPL NOT ACTIVE
IST314I END
```

Solution: The probable cause is that the Platform Server is not up and running; therefore, the APPL LU is not active. Check whether the Platform Server address space is up and that the APPL LU is active.

Batch Job Has Been Submitted However It Has Not Been Placed on the Queue

Look in the output from the JCL for any error messages. The following table lists some of the messages that you might see:

Key Number	Error Message	Cause	User Response
PGTB4100E	`Parameter not recognized in next record`	Most likely the cause is a typo.	Verify that the specified value in the line after this message is correct.
PGTB4102E	`Invalid parameter value in next record`	The value specified for this parameter is not valid.	Check the valid range of values for this parameter.

Key Number	Error Message	Cause	User Response
PGTB4103E	`Dataset name not cataloged`	The dataset that is specified for this transaction is not cataloged.	Verify that the dataset name is specified correctly.
PGTB4105E	`Operand read before a PROCESS card`	The parsing program found parameters before your process card.	Verify that you specify the process card correctly and that all parameters follow the process card.
PGTB4106E	`Invalid number of operands in next record`	The number of operands is outside the valid number.	Verify that you specify this number correctly.
PGTB4107E	`No more room left in the server work Queue`	The maximum number of transactions are currently on the work queue.	You can either wait for some of the transactions that are on the queue to be completed and removed, or you can re-create the work queue and specify a larger VSAM file.
PGTB4108E	`Fusion server not available`	The server to which you specified this transaction to be queued is not currently up and running.	Start the server and re-queue the transaction.
PGTB4109E	`Security denied access`	You do not have authorization to one of the resources that you specified.	Verify that you have authorization to the resources specified in the batch job.

Key Number	Error Message	Cause	User Response
PGTB4115E	`No process card in the SYSIN file`	The required process card is not found by the parsing routine.	Verify that you correctly specified a process card for this batch job.
PGTB4116E	`Prior process not queued - Parameter error`	An error occurred to one of the parameters specified for the previous process.	Verify that the parameters specified are correct.
PGTB4117E	`Required parameter missing in next record`	The required parameter in the next record is missing.	Check that you have specified a value in the next parameter field.
PGTB4118E	`Required parameter invalid in next record`	The required parameter in the next record is invalid.	Check that the parameter is within the valid range.
PGTB4119I	`Invalid/Missing Server parm - Default used`	The server is not specified or is invalid.	The default server is used to queue this transaction. To queue this transaction to a different server, you can specify the server parameter with the correct server.
PGTB4120I	`Invalid Queue effect for activity`	Queue effect specified is invalid for this transaction type.	Check that you specified a valid value.

Appendix A. Sample SNA Mode Definition

The following example is a sample SNA mode definition that is used when working with the Platform Server.

Note: These definitions are not required if you are using the platform server TCP interface.

```
USERMODE MODETAB
*********************************************************************
*  MODE ENTRIES FOR CYBERFUSION PERFORMANCE BENCHMARKS
*********************************************************************
FUSN256 MODEENT LOGMODE=FUSN256, LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                        -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8585',         RUSIZE SEC/PRI  256/256           -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*********************************************************************
FUSN512 MODEENT LOGMODE=FUSN512, LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                        -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8686',         RUSIZE SEC/PRI  512/512           -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*********************************************************************
FUSN1K  MODEENT LOGMODE=FUSN1K,  LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                        -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8787',         RUSIZE SEC/PRI 1024/1024          -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*********************************************************************
FUSN2K  MODEENT LOGMODE=FUSN2K,  LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                                                  -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8888',         RUSIZE SEC/PRI 2048/2048          -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*********************************************************************
FUSN4K  MODEENT LOGMODE=FUSN4K,  LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                                                  -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8989',         RUSIZE SEC/PRI 4096/4096          -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*********************************************************************
FUSN8K  MODEENT LOGMODE=FUSN8K,  LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                                                  -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8A8A',         RUSIZE SEC/PRI 8192/8192          -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*
FUSN16K MODEENT LOGMODE=FUSN16K, LOGON MODE TABLE ENTRY NAME       -
        COS=FAST,                        -
        FMPROF=X'13',            FUNCTION MANAGER PROFILE          -
        TSPROF=X'07',            TRANSMISSION SERVICES PROFILE     -
        PRIPROT=X'B0',           PRIMARY  LOGICAL UNIT PROTOCOL    -
        SECPROT=X'B0',           SECONDARY LOGICAL UNIT PROTOCOL   -
        COMPROT=X'50B1',         COMMON  LOGICAL UNIT PROTOCOL     -
        RUSIZES=X'8B8B',         RUSIZE SEC/PRI 16K (8 X 2**11)    -
        PSNDPAC=5,               PRIMARY  SEND  PACING COUNT       -
        SRCVPAC=5,               SECONDARY RECEIVE PACING COUNT    -
        SSNDPAC=5,               SECONDARY SEND  PACING COUNT      -
        PSERVIC=X'060200000000000000002C00' PRESENTATION SRVCS
*
*
        MODEEND
        END
```

Appendix B. Running Traces

To solve a communication problem, TIBCO Support might ask you to take a trace of the activity in error.

You might be asked to take the following two types of traces:

- Platform server internal trace

  See Running the Platform Server Internal Trace for details.

- VTAM buffer trace

  See Running VTAM Buffer Traces for details.

Running Platform Server Internal Traces

The Platform Server can trace either VTAM or TCP communications.

You can use the following operator commands to activate the respective traces:

- VTAM trace: `F FUSION,TRON`
- TCP trace: `F FUSION,TCPTRON`

You can use the following operator commands to turned off the respective traces:

- VTAM trace: `F FUSION,TROFF`
- TCP trace: `F FUSION, TCPTROFF`

For more information, see "Operator Commands" in *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Example for Taking a TCP Trace

The following steps illustrate how to take a TCP trace:

1. Turn on the TCP trace.

   ```
   F FUSION,TCPTRON
   ```

2. Perform the activity that you want to trace.

3. Turn off the TCP trace.

   ```
   F FUSION,TCPTROFF
   ```

Locations for Trace Data

VTAM trace data is sent to DD statement VTAMTRAC defined in the Platform Server JCL. TCP trace data is sent to DD statement TCPTRAC defined in the Platform Server JCL.

For more information, see "The Startup JCL" in *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*.

Running VTAM Buffer Traces

In certain cases, TIBCO Support might request that you collect a buffer trace to assist them in debugging communication problems.

Use the following command to instruct VTAM to start tracing all activities that are occurring with a particular LU.

```
F vtam,TRACE,TYPE=BUF,ID=luname
```

Where, *vtam* is the name of the VTAM started task on the system that is running the Platform Server, and *luname* is the name of the LU for which you want to collect trace information.

To start collecting trace information, run the generalized tracing facility (GTF). The following sample PROC can be used to start the generalized trace facility:

```
//GTF  PROC
//*******************************************************************
//* GTFTP                                                           *
//* THIS IS THE STANDARD TRACE PROC FOR TRACING THINGS IN VTAM.     *
//* THE SYSIN FILE IS LOCATED IN USER.PARMLIB(GTFTP)                *
//*        ---------------------------------------------------------*
//*******************************************************************
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',       -
// TIME=1440,REGION=4000K
//IEFRDER DD DISP=SHR,DSN=SYS1.TRACE
//SYSLIB  DD DISP=SHR,DSNAME=USER.PARMLIB(GTFTP)
//*-----------END OF PROC "GTF"-------------------------------------*
```

The following `SYSIN` file is associated with the GTF PROC:

```
TRACE=RNIO,USR
```

After you have reproduced the problem that you are trying to debug, you must perform the following tasks:

- Stop the VTAM buffer trace using the following command:

  `F vtam,NOTRACE,TYPE=BUF,ID=luname`

- Stop GTF using the conventional IBM stop command from a console or SDSF.

After GTF has stopped collecting the trace information, you must format the information that GTF has collected. The following sample PROC can be used to format the buffer trace:

```
//TAPBUF PROC OUTC=X,D=LOCAL
//*--------------------------------------------------------*
//*-THIS PROCEDURE INVOKES THE TRACE ANALYSIS PROGRAM -----*
//*-IN ORDER TO PRINT A PREVIOUSLY TAKEN VTAM BUFFER-------*
//*-TRACE THAT IS HELD IN THE dataset "SYS1.TRACE"--------*
//*--------------------------------------------------------*
//TAPSTEP EXEC PGM=ACFTAP,REGION=3000K,PARM='READ'
//SYSTRACE DD DISP=SHR,DSN=SYS1.TRACE
//STEPLIB  DD DISP=SHR,DSN=NCP6.SSPLIB
//SYSIN    DD DISP=SHR,DSN=USER.PARMLIB(TAPBUF)
//SORTIN   DD DISP=(NEW,DELETE),DSN=&&SORTIN,SPACE=(CYL,(13,5)),
// DCB=(RECFM=F,LRECL=364,BLKSIZE=364),UNIT=SYSDA
//SORTOUT  DD DISP=(NEW,DELETE),DSN=&&SORTOUT,SPACE=(CYL,(13,5)),
// DCB=(RECFM=F,LRECL=364,BLKSIZE=364),UNIT=SYSDA
//SYSPRINT DD SYSOUT=&OUTC,DEST=&D
//SYSSDPRT DD SYSOUT=&OUTC,DEST=&D,FREE=CLOSE,SPIN=UNALLOC
//SYSSSPRT DD SYSOUT=&OUTC,DEST=&D
//SYSGSPRT DD SYSOUT=&OUTC,DEST=&D
//SYSDTPRT DD SYSOUT=&OUTC,DEST=&D
//SYSNEPRT DD SYSOUT=&OUTC,DEST=&D
//SYSCAPRT DD SYSOUT=&OUTC,DEST=&D
//SYSCSPRT DD SYSOUT=&OUTC,DEST=&D
//SYSTEMP1 DD DUMMY   FOR ETHERNET DATA
//SYSTEMP2 DD DUMMY   FOR ETHERNET DATA
//*-----------END OF PROC "TAPBUF"-------------------------*
```

The following example shows the `SYSIN` file associated with the TAPBUF PROC:

```
INPUT=BUFFER,
SOURCE=GTF,
LONGPIU=YES,
SUMMARY=NO,
LSPRT=NO,
LDPRT=NO,
SDPRT=YES,
SSPRT=YES,
DTPRT=YES,
NEPRT=YES,
GSPRT=YES,
VTPRT=NO,
IXPRT=NO,
NPPRT=NO,
NTPRT=NO,
LUPRT=NO,
GO
QUIT
```

Note: `LONGPIU=YES` is only supported by VTAM version 4 or higher. This parameter instructs VTAM to print the entire request unit instead of the default that is to print only the first 252 bytes. If you are running VTAM version 4 or higher, ensure that this parameter is set to `YES`.

You can get several different formats from running this sample TAPBUF PROC. The information that you send to TIBCO Support is the output generated with the DD statement SYSSDPRT.

Appendix C. Automated Operations

The Platform Server provides several standardized messages to realize automated operations for many organizations.

Many organizations want to automate file transfer processes. In some cases, the data that is transferred must be further processed at the remote side. Therefore, many software packages for the host can scan for certain Write To Operator (WTO) messages and perform different transactions based on the contents of the WTO message.

The need to automate file transfer processes requires that certain z/OS messages be standardized. To interface with these automation packages, the Platform Server has standardized several messages that are issued for every transaction.

Start of Transfer

The following message is issued at the start of every transfer.

Note: The information denoted by the brackets ({ }) are replaceable parameters.

```
PGTF3100 {Send/Receive} Activity {Transaction_Number} Started to Remote Node
{Remote_Node}
```

Where:

- `Send/Receive`: Indicates the direction of the dataset on the host system. For example, if you are sending a file from a Windows system to the host, the direction is `Receive`.
- *Transaction_Number*: Indicates the number that is assigned to the transfer.
- *Remote_Node*: Indicates the destination to which you are sending or from which you are receiving a transfer request.

Completion of Transfer

The message PGTF3108I is issued at the completion of all file transfer activity. This message can be used for automated processing or scheduling.

> **Note:** The information denoted by the brackets ({ }) are replaceable parameters.

Successful Completion

```
PGTF3108I Transfer {Activity_Number} successfully transferred {Record_Count}
records with node:
```

`{Remote_Node}{Byte_Count}(Bytes)` or `{Remote_Node}{Member_Count} {Members}`

Where:

- *Activity_Number*: Indicates the number that is assigned to the transfer.
- *Record_Count*: Indicates the number of records sent or received by the transaction.
- *Remote_Node*: Indicates the destination to which you are sending or from which you are receiving a transfer request. This can be displayed as an LU name, IP name, or IP address.
- *Byte_Count*: Indicates the number of bytes transferred (uncompressed) for a sequential file.
- *Member_Count*: Indicates the number of members processed for PDS/Library transfers.

```
PGTF3108I {Activity_Number} RC={Return_Code} {Remote_Node} {Local_DSN}
{Process} Record Count={Record_Count} {S/R} {User_Data}
```

Where:

- *Activity_Number*: Indicates the number that is assigned to the transfer.
- *Return_Code*: Indicates the number that is returned at the completion of a file transfer. A successful transfer has a return code of all zeroes. An unsuccessful file transfer issues a return code that helps you determine the specific nature of the problem.
- *Remote_Node*: Indicates the destination to which you are sending or from which you are receiving a transfer request. This can be displayed as an LU name, IP name, or IP address.

- *Local_DSN*: Indicates the name of the local dataset for the transaction. The dataset name must be a valid z/OS dataset and cannot exceed 54 characters in length. If the dataset provided is invalid, the interface prompts you for a valid name.
- `Process`: Indicates an eight-character description of the transaction being processed.
- *Record_Count*: indicates the number of records sent or received by the transaction.
- `S/R`: Indicates the direction of the dataset on the host system. For example, if you are sending a file from the Windows system to the host, the direction is `R`.
- *User_Data*: Indicates the data entered by the user on the PROCESS `DESCRIPTION` parameter.

Unsuccessful Completion

```
PGTF3109I Activity {Activity_Number} unsuccessful with Node {Remote_Node}
Return Code={Return_Code} error type ERROR
```

Where:

- *Activity_Number*: Indicates the number that is assigned to the transfer.
- *Remote_Node*: Indicates the destination to which you are sending or from which you are receiving a file transfer.
- *Return_Code*: Indicates the number that is returned at the completion of a file transfer. A successful transfer has a return code of all zeroes. An unsuccessful file transfer issues a return code that helps the user determine the specific nature of the problem.
- *Error_Type*: Indicates the type of error that caused the transfer to fail.

```
PGTF3108I {Activity_Number} RC={Return_Code} {Remote_Node} {Local_DSN}
{Process} Record Count={Record_Count} {S/R} {User_Data}
```

Where:

- *Activity_Number*: Indicates the number that is assigned to the transfer.
- *Return_Code*: Indicates the number that is returned at the completion of a file transfer. A successful transfer has a return code of all zeroes. An unsuccessful file transfer issues a return code that helps you determine the specific nature of the problem.
- *Remote_Node*: Indicates the destination to which you are sending or from which you are receiving a file transfer.

- *Local_DSN*: Indicates the name of the local dataset for the transaction. The dataset name must be a valid z/OS dataset and cannot exceed 54 characters in length. If the dataset provided is invalid, the interface prompts you for a valid name.
- `Process`: Indicates an eight-character description of the transaction being processed.
- *Record_Count*: Indicates the number of records sent or received by the transaction.
- `S/R`: Indicates the direction of the dataset on the host system. For example, if you are sending a file from the Windows system to the host, the direction is `R`.
- *User_Data*: Indicates the data entered by the user on the PROCESS `DESCRIPTION` parameter.

Appendix D. File Name Tokens

File name tokens are a feature of the Platform Server with which you can substitute generated values into file names when communicating between any combination of z/OS, Windows, and UNIX Platform Servers.

Given a file name that includes one or more tokens, the Platform Servers resolve the token or tokens (for example, date, time, and user) within the file name to the actual file name. This can be very useful when it is necessary for the file name to be unique or to follow naming conventions. Windows and UNIX resolve tokens on both the local and remote file names. While for requests initiated by z/OS, the tokens are resolved by z/OS for the local file name only. This means that if a token is specified in the remote file name, you must refer to the documentation for the remote system to get the list of supported tokens.

> **Note:** The Platform Server for z/OS performs token substitution to its local file name on both initiator and responder requests from another Platform Server for z/OS. While, when the Platform Server for z/OS is initiating a request with file name tokens in the remote file name to a non Platform Server for z/OS, check the documentation for the remote system for a complete list of valid tokens.

Instead of entering a standard file name, you enter a name that consists of tokens which are specified by a dollar sign ($) followed by the token name in parentheses. For example,

```
$(LocalFileName)
```

See the following topics for more information on file name tokens:

- File Name Tokens List
- Examples of Using the File Name Tokens
- Rules for Using the File Name Tokens

File Name Tokens List

The following table lists the file name tokens supported by the Platform Server for z/OS, their definitions, and generated values:

Token	Definition	Generated Value (Examples)
`CYear`	CCYY	2009
`Date`	YYYYMMDD	20090625
`Date1`	YYMMDD	090625
`Date2`	MMDDYY	062509
`Date3`	DDMMYY	250609
`DateUS`	MMDDYYYY	12312014
`Day`	DD	01 - 31
`Hour`	HH	00 - 23
`JDate`	YYYYDDD	2009176
`LocalUserId`	z/OS user ID associated with the transfer.	Local file: `$(LocalUserId).$(RemoteFileBase)` Remote file: `ACCT.TAX.Y2009` User ID: PRUSER1 The tokens are resolved to `PRUSER1.ACCT.TAX`.
`Member`	Member name associated with the transfer.	Local file: `$(RemoteFileBase).$(Member)` Remote file: `PROD.TEST.ACCT.TAX(Y2009)` The tokens are resolved to `PROD.TEST.ACCT.Y2009`.
`Minute`	MM	00 - 59
`Month`	MM	01 -12

Token	Definition	Generated Value (Examples)
RemoteFileBase	The remote file name only.	Local file: $(RemoteFileBase) Remote file: PROD.TEST.ACCT.TAX.Y2009 The token is resolved to PROD.TEST.ACCT.TAX.
RemoteFileExt	All file name data after the first qualifier.	Local file: $(RemoteFileExt) Remote file: PROD.TEST.ACCT.TAX.Y2009 The token is resolved to TEST.ACCT.TAX.Y2009.
RemoteFileName	The remote file name including the extension to be used.	Local file: $(RemoteFileName) Remote file: c:\source\directory\tstfile1.txt The token is resolved to tstfile1.txt.
RemoteFileNameFull	The remote file name including the member name to be used.	Local file: $(RemoteFileNameFull) Remote file: PROD.TEST.ACCT.TAX(Y2009) The token is resolved to PROD.TEST.ACCT.TAX.(Y2009).
RemoteTransactionNumber	Remote transaction number used in the file transfer.	Local file: /tmp/$(RemoteTransactionNumber).txt Remote file: c:\source\directory\testfile1.txt The token is resolved to d:\fn\I925600005.
RemoteUserId	Remote user ID used in the file transfer.	Remote user ID: TEST\cfuser1 Remote file: PROD.$(RemoteUserId).DATA The token is resolved to PROD.CFUSER1.DATA.

Token	Definition	Generated Value (Examples)
`Second`	SS	00 - 59
`SJOBNAME`	Job name used in JCL JOB statement of transfer.	LocalFileName: `TEST.$(SJobName)` is resolved to `TEST.JOBTEST` where `JOBTEST` is the job name.
`SJOBID`	Job ID generated when the transfer JCL is submitted.	LocalFileName: `TEST.$(SJobId)` is resolved to `TEST.JOBXXX` where `XXXX` is the job ID.
`Time`	HHMMSMSS	084513126
`Time1`	HHMMSS	084513
`Time2`	HHMMSST	0845138
`Year`	YY	00 - 99

Examples of Using File Name Tokens

When transferring a file, you can create the file name using file name tokens instead of a regular file name. The following examples show how to use file name tokens.

These examples use this sample system date and time: Wednesday, April 25, 2009 5:03:45.061 PM.

Example 1:

The user has entered a date token proceeded by the letter D when sending a file to a Platform Server for z/OS. The Platform Server on the z/OS system resolves the string into the `Date1`*YYMMDD* format.

File name: `PROD.TEST.DATA.SAMPLE.D$(Date1)`

Resolved file name: `PROD.TEST.DATA.SAMPLE.D090425`

Example 2:

When the Platform Server for z/OS receives files, the following substitution is performed for either token `LocalFileBase` (or `RemoteFileBase`) and `LocalFileExt` (or `RemoteFileExt`).

File name: `PROD.TEST.ACCT.TAX.Y2009`

`RemoteFileBase` will be resolved to: `PROD.TEST.ACCT.TAX`

`RemoteFileExt` will be resolved to: `TEST.ACCT.TAX.Y2009`

Example 3:

This example can be used when the Platform Server for z/OS is sending a file to a Platform Server for UNIX.

The user has entered a string of file name tokens when sending a file to a Platform Server for UNIX. The Platform Server on the UNIX system resolves the string into the directory name and file name.

File name: `/directory/filename.$(Date3).dat`

Resolved file name: `/directory/filename.030510.dat`

Example 4:

This example can be used when the Platform Server for z/OS is sending a file to a Platform Server for Windows:

The user has used the file name tokens to generate a resolved file name that has dashes between the date and time fields when sending a file to a Platform Server for Windows.

File name: `C:\directory\$(SDD)-$(SMON)-$(SYYYY)\$(SHH24)-$(SMI)-$(SSS).dat`

Resolved file name: `C:\directory\25-APR-2009\17-03-45.dat`

Rules for Using File Name Tokens

When creating a file name that uses file name tokens, you must follow the following rules:

- Substitution parameters are enclosed in `$(`*`token_name`*`)`: a dollar sign, followed by an open parenthesis, followed by the token name, followed by a close parenthesis.
- Each `$(`*`token_name`*`)` can only contain one token name.
- Any text in the local file name that is not a substitution parameter is embedded, as is, into the generated name.

- Tokens might be used anywhere within the local file name.
- Space permitting, any number of substitution parameters can be embedded within the file name.
- If the resolved file name length is greater than the maximum file name length (255 characters) supported by the Platform Server, the resolved file name is truncated.
- Tokens are case sensitive and will affect the output of the file name; therefore, you must use them exactly as they are documented in File Name Tokens List.
- If a formatted name contains an invalid substitution code, the transfer will fail with an error stating that a substitution code is invalid.

Appendix E. CA-7 Job Scheduler Interface

The Platform Server can trigger the execution of jobs on the z/OS system through the CA-7 Job Scheduler. This is achieved by utilizing the CA-7 interface module called U7SVC that is distributed as part of the CA-7 software product.

Module U7SVC is automatically loaded at the Platform Server initialization. It is invoked after a file transfer is successfully completed. For the Platform Server to load U7SVC, U7SVC must either be in the Platform Server load library or concatenated in the STEPLIB DD statement of the startup PROC. When a dataset is successfully created, the name of the dataset and the volume serial on which it is created are passed to the U7SVC module. The z/OS installation must use dataset triggering within CA-7 to schedule the execution of jobs.

Appendix F. Using the Process WAIT Parameter with MVS JCL IF/THEN/ELSE/ENDIF Parameters

By using the Platform Server `WAIT` parameter in conjunction with IBM IF/THEN/ELSE/ENDIF statement construct, you can conditionally execute additional job steps based on the outcome of the Platform Server file transfer.

Note: You can use IF/THEN/ELSE logic in the REXX interface or the Batch interface along with the JCL IF/THEN/ELSE parameters when performing Platform Server transfers.

The following sample illustrates the syntax that can be used:

```
//SYSIN DD *
   PROCESS,IFTHEN,TRANSFER,RECEIVE
       .
            .
      .
// [name] IF  [(]relational-expression[)] THEN
      .
            .                  action when expression is true
            .
// [name] ELSE
            .
            .                  action when relation-expression
            .
//[name] ENDIF
```

By using this syntax, you can execute job steps within a job based on certain conditions of the outcome of the Platform Server transfer.

The IF statement is always followed by a relational expression followed by a THEN clause. These three things are required when using the IF/THEN/ELSE/ENDIF statement construct. The THEN clause specifies those steps that the system processes upon the confirmation that the relational expression, which is evaluated at startup, is a true condition.

If the relational expression is a false condition, the ENDIF statement will follow. The ENDIF statement indicates the end of the IF/THEN/ELSE/ENDIF statement construct. It is mandatory that ENDIF be coded at the end of every construct. However, before the ENDIF statement, you can specify the ELSE clause. The ELSE clause specifies those steps that the system processes if the relational expression is false. This optional statement is followed by the ENDIF statement.

The following example shows how to use IF/THEN/ELSE/ENDIF within the Platform Server batch JCL.

Note: For more information on how to use Modal Logic, see the JCL reference manual.

```
//IFTHEN  JOB 555,'IF THEN JCL',MSGCLASS=X,MSGLEVEL=1,
00001004
//    REGION=4000K,TIME=(2),NOTIFY=&USERID
00002004
//****************************************************************
00030000
//STEP1  EXEC PGM=OSIUB000,PARM='SERVER=FUSION'
00030104
//STEPLIB DD DISP=SHR,DSN=FUSION.LOADLIB
00030203
//SYSUDUMP DD SYSOUT=*
00030303
//SYSPRINT DD SYSOUT=*
00030403
//SYSIN  DD *
00030503
  PROCESS,TEST1,TRANSFER,RECEIVE
00030603
    DSN=INPUT.DATATSET.NAME
00030703
    REMOTE_FILE=C:\TEMP\TEST.TXT
00030803
    NODE=NODE1
00030903
    TYPE=TEXT
00031003
    CRLF=YES
00031103
    EFFECT=CR
00031203
    HOLD=NO
00031303
    REMOTE_USER=USER1
00031403
    RPASS=USER1
```

```
00031503
    NOTIFY=USER1
00031603
    NOTIFY_TYPE=TSO
00031703
    WAIT=YES
00031804
    RETRY=2
00031904
//   NZERO IF (RC>5) THEN
00032003
//STEP2 EXEC PGM=IEBGENER
00033004
//SYSPRINT DD SYSOUT=X
00040000
//SYSUT1 DD DISP=SHR,DSN=FUSION.SAMPLIB(MEMBER1)
00050000
//SYSUT2 DD SYSOUT=X
00060000
//SYSIN DD DUMMY
00070000
//   ELSE
00081003
//STEP3 EXEC PGM=IEBGENER
00090000
//SYSPRINT DD SYSOUT=X
00100000
//SYSUT1 DD DISP=SHR,DSN=FUSION.SAMPLIB(MEMBER2)
00110000
//SYSUT2 DD SYSOUT=X
00120000
//SYSIN DD DUMMY
00130000
//ENDSTEP ENDIF
00420001
/*
00430001
```

Appendix G. Overriding JCL SYSIN Parameters

The parameters specified by the PARM string are processed after the SYSIN parameters and override any duplicate parameter values specified in the SYSIN statement. Therefore, you can specify SYSIN parameters by using the `PARM` field of the JCL EXEC statement. The parameters in the PARM string are also applied to every PROCESS statement (every transaction) queued in that particular step.

In the `PARM` field, the parameters describing the server (`SERVER`, `SERVLUNAME`, `SERVIPADDR`, and `SERVIPNAME`) and their values must be the first group of items specified. For more information on the required PARM parameters, see Batch Interface.

Following the server name, any valid SYSIN parameters can be specified in the parameter string. For more information on valid SYSIN parameters, see Batch Interface.

Each parameter and its corresponding value must be separated by an equal sign (=). A comma must be used to separate each parameter or value pair. The PARM statement might span multiple lines by typing up to column 71 and placing a continuation mark in column 72. The next line must begin in column 16. In the following example, the interpreted value of `DSN` is `DATASET.NAME`. Long file names with spaces are also supported by enclosing the remote file name in double quotation marks.

```
//STEPNAME EXEC PGM=OSIUB000,PARM=’SERVER=FUSION,NODE=RemNode,DSN=DATAS
//       ET.NAME,REMOTE_FILE=”c:\test file.tst”’
```

In the following batch example, some SYSIN parameters will be overwritten:

```
//BATCH1    JOB 555,MSGCLASS=X,REGION=4M
//STEP1     EXEC PGM=OSIUB000,PARM=’SERVER=FUSION,REMOTE_
FILE=C:\SalesRe-
//          port.DOC’
//STEPLIB   DD DISP=SHR,DSN=PROD.LOADLIB
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=X
//SYSIN     DD *
PROCESS,SALES1,TRANSFER,RECEIVE
        NODE=REMNOD1
        DSN=SALES.REPORT1
        REMOTE_FILE=C:\OVERRIDE
        EFFECT=CR
        ISSUER=USERID
        RPASS=PASSWD
PROCESS,SALES2,TRANSFER,RECEIVE
        NODE=REMNOD2
        DSN=SALES.REPORT2
        EFFECT=CR
        ISSUER=USERID
        RPASS=PASSWD
//STEP2     EXEC PGM=OSIUB000,PARM='SERVER=FUSION,DSN=DISTRIB.ANNUAL.RE-
//          PORT,REMOTE_FILE=C:\AnnualReport.Doc'
//STEPLIB   DD DISP=SHR,DSN=PROD.LOADLIB
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=X
//SYSIN     DD *
PROCESS,DISTRIB1,TRANSFER,SEND
        NODE=REMNOD1
        DSN=DATASET.OVERRIDE
        EFFECT=CR
        ISSUER=USERID
        RPASS=PASSWD
PROCESS,DISTRIB2,TRANSFER,SEND
        NODE=REMNOD2
        EFFECT=CR
        ISSUER=USERID
        RPASS=PASSWD
```

In this example, the SYSIN parameters in bold will be overridden and are therefore unnecessary to code. In the first step, processes SALES1 and SALES2 will receive a remote file called `C:\SalesReport.DOC` whether or not the parameter is specified in the SYSIN statement. In the second step, processes DISTRIB1 and DISTRIB2 will send the dataset named `DISTRIB.ANNUAL.REPORT` to the remote file named `C:\AnnualReport.Doc`. Both the `DSN` and the `REMOTE_FILE` parameters are ignored if specified in the SYSIN statement.

When a parameter is overridden, a warning message is written to the batch output file following the SYSIN parameters. The message contains the name of the overridden parameter and is displayed for every parameter specified in the PARM string. The following example is a sample warning message:

```
PGTB4124W PARM string parameter DSN will over write SYSIN value
```

Appendix H. PDS, HFS, VSAM, and MQ Support

The Platform Server supports transfers of PDS and HFS files, message queue, and creation of VSAM files.

PDS Support

The Platform Server supports transfers of libraries between z/OS systems.

The following functions for library transfers are supported:

- Transfer of individual members.
- Transfer of complete libraries.
- Transfer of members with wildcard.
- Renaming of members that are transferred.
- Transfer of members between PDS and PDS/E datasets.
- Transfer of LOADLIBs between PDS datasets.
- All member directory information is sent along with the member.
- Alias members are fully supported.
- dataset allocation attributes are sent to the receiving system.
- dataset allocation attributes can be overridden.

The two restrictions to Platform Server PDS support are as follows:

- The Platform Server does not write data to a PDS/E program library.
- The Platform Server does not send an alias alone without the actual member.

Note: When you attempt to send or receive a library and the remote system does not support library transfers, the Platform Server will end the transfer and display a message on both the local and remote systems which indicates that the remote system does not support library transfers. The transfer request is purged from the request queue afterwards.

Additional Batch Parameters for Library Support

The following two additional batch parameters are added for library support:

- `DATASET_TYPE=LIBRARY`

  Informs the receiver of the file that if a dataset is to be created, a PDS/E must be created. If this parameter is not set, a PDS is created at the remote end.

- `ALLOC_DIR=`*number_of_directory_blocks*

  Defines the number of directory blocks that will be allocated when a file is created. This parameter overrides the local dataset attributes that are sent to the system which creates the dataset.

Generic and Wildcard Characters for Members

The generic and wildcard support for members must conform to the following rules:

- The asterisk (*) denotes a generic member.
- If a PDS is specified without a member name, all members are sent.
- `FILE=MY.PDS` is the same as `MY.PDS(*)`.
- Question mark (?) is the wildcard character. It can be used with characters of a member and with the generic character (*).

Examples

The following examples are samples of generic and wildcard support.

Example 1:

```
FILE=MY.PDS(abc*)
REMOTE_FILE=YOUR.PDS(WXYZ*)
```

All files starting with ABC are sent to the remote system. When the files are written, the first 4 characters are replaced by WXYZ.

Example 2:

```
FILE=MY.PDS(???TEST?)
REMOTE_FILE=YOUR.PDS
```

Members in which characters 4 - 7 are TEST are sent. The member names on the remote system are the same as on the local system.

Alias Support

The alias support conforms to the following rules:

- When a full library or partial library transfer is specified, all members and aliases that match the selection criteria will be sent to the remote system.
- When a specific member is sent, that member, and all aliases of that member will be sent to the remote system.

Examples of Sending PDS Files

The following examples show how to send PDS files.

Example 1: Sending member `ABC` to node REMOTE member `ABC`

```
PROCESS SEND
               DSN=MY.LOCAL.PDS(ABC)
               REMOTE_FILE=YOUR.PDS
               NODE=REMOTE
```

Example 2: Sending member `ABC` to node REMOTE member `XYZ`

```
PROCESS SEND
               DSN=MY.LOCAL.PDS(ABC)
               REMOTE_FILE=YOUR.PDS(XYZ)
               NODE=REMOTE
```

Example 3: Sending all members of a PDS to a PDSE on node REMOTE

```
PROCESS SEND
               DSN=MY.LOCAL.PDS
               REMOTE_FILE=YOUR.PDS
               DATASET_TYPE=LIBRARY
               NODE=REMOTE
```

Example 4: Sending selected members of a PDS to a PDSE on node REMOTE and changing characters 3 - 6 from PROD to TEST when writing the members

```
PROCESS SEND
              DSN=MY.LOCAL.PDS(??PROD*)
              REMOTE_FILE=YOUR.PDS(??TEST)
              DATASET_TYPE=LIBRARY
              NODE=REMOTE
```

HFS File Support

The Platform Server supports the transfer of HFS files. Both send and receive operations are supported.

The restrictions for this support are as follows:

- The Platform Server started task user must be defined as a superuser.
- Users initiating a request for an HFS file must have the OMVS segment defined so that they are recognized as OMVS users.
- For a file create operation, the HFS directory must exist. The Platform Server for z/OS will not create a directory.
- The user associated with a file transfer request must have the necessary authority to read or update an existing file depending on whether a send or receive transfer is performed.
- For a file create operation, the user must have write authorization for the directory that the file is written to.

The Platform Server supports file names of up to 256 characters. The Platform Server creates the security environment for a transfer request by validating the user ID and password through RACF. After the security environment is established, standard OMVS security validation is performed.

During text record processing, line feed (LF) is inserted at the end of each record. LF is not inserted on binary transfers. All text transfers store data in EBCDIC.

MQ Support

The Platform Server provides support for message queues.

The Platform Server supports message queues in the following ways:

- Data can be read from or written to/from message queues instead of files.
- The Platform Server DNI processing can be performed on message queues.
- The Platform Server can notify message queues when a transfer request is completed.

MQ File Name Requirements

The Platform Server recognizes that a file transfer is for a message queue by the format of the file name.

The format of a Platform Server message queue name is as follows:

```
$MQ:XXXX:mmmmmmmmmmmmm
```

Where, `$MQ` is a fixed literal and must be the first 3 bytes of the dataset name, *XXXX* defines the queue manager name, and *mmmmmmmm* defines the queue name.

For example,

```
LF=$MQ:MQ1:FUSION.MESSAGE.QUEUE
```

This example defines the queue manager name as MQ1 and the queue name as FUSION.MESSAGE.QUEUE.

MQ Send and Receive Processing

The Platform Server can read records from a message queue or write records to a message queue.

When the Platform Server processes a message queue, the message queue journals the request. The Platform Server does not support SYNCPOINT processing; therefore, large amounts of data cannot be sent to or read from the message queue. When a file transfer request fails, the Platform Server issues an MQ BACKOUT request, which causes the queue contents to revert to the state before the file transfer starts.

The Platform Server can send or receive data from message queues. Likewise, the Platform Server can initiate a request to another Platform Server for z/OS to process data from a message queue.

Note: Only Platform Servers for z/OS support message queues. If you send an MQ request to a Platform Server that does not support message queue, the results are unpredictable.

Examples

The following examples are sample Platform Server send and receive processing:

Example 1:

```
PROCESS MQSEND,TRANSFER,SEND
       LF=$MQ:MQ1:FUSION.MESSAGE.QUEUE1
       RF=C:\temp\mqdata
```

This example sends the contents of the message queue defined by the `LF` parameter to a remote node where the data is saved as a file.

Example 2:

```
PROCESS MQRECV,TRANSFER,RECEIVE
       LF=$MQ:MQ1:FUSION.MESSAGE.QUEUE1
       RF=C:\temp\mqdata
```

This example receives the remote file `c:\temp\mqdata`, and writes the data to the message queue defined by the `LF` parameter.

Example 3:

```
PROCESS MQSEND,TRANSFER,SEND
       LF=PROD.ZOS.DATA
 RF=$MQ:MQ1:FUSION.MESSAGE.QUEUE1
```

This example sends the contents of the `PROD.ZOS.DATA` file to the message queue on the remote node defined by the `RF` parameter.

Note: In this case, the remote node must be a Platform Server for z/OS with MQ support.

Example 4:

```
PROCESS MQRECV,TRANSFER,RECEIVE
       LF=PROD.ZOS.DATA
 RF=$MQ:MQ1:FUSION.MESSAGE.QUEUE1
```

This example receives message queue data from the remote node, and saves the data in the `PROD.ZOS.DATA` file.

Note: In this case, the remote node must be a Platform Server for z/OS with MQ support.

VSAM File Support

Platform Server for z/OS supports the creation of VSAM files.

Platform Server uses the following parameters documented in the Platform Server Batch interface to create VSAM files:

- `VSAM_KEYPOS`
- `VSAM_KEYLEN`
- `VSAM_LIKE`
- `VSAM_TYPE`
- `VSAM_LRECL`
- `VSAM_RRSLOT`
- `VSAM_REUSE`

Note: If these parameters are not defined, the Platform Server uses the attributes of the VSAM file being sent to create the new VSAM file.

Additionally, VSAM files can be created by using the Platform Server `DATACLASS` parameter. This requires an SMS administrator to properly set up the SMS DATACLASS.

Appendix I. Extended Attribute Support

When a transfer is initiated on a non-z/OS system and a file is to be created on z/OS, the z/OS dataset attributes are set by specifying extended attributes on the KNFUSION command line using the `-e` sub-parameter.

The following parameters are supported:

- `RECFM={FB | F | U | V | VB}`
- `LRECL=`*record_length*
- `BLKSIZE=`*block_size*
- `VOLSER=`*volume*
- `ALCUNIT={CYL | TRK | BLK}`
- `UNIT=`*unit_name*
- `PRIMARY=`*primary_allocation_number*
- `SECONDARY=`*secondary_allocation_number*
- `DIRBLK=`*directory_blocks_number_to_allocate*
- `DATACLASS=`*SMS_data_class*
- `MGTCLASS=`*SMS_management_class*
- `STORCLASS=`*SMS_storage_class*

Example

```
KNSNDF  -eRECFM=FB,LRECL=80,BLKSIZE=3200,ALCUNIT=CYL,PRIMARY=3,
SECONDARY=1
```

This example creates a file with a 3-cylinder primary allocation and a 1-cylinder secondary allocation with the following DCB characteristics:

- `RECFM=FB`
- `LRECL=80`
- `BLKSIZE=3200`

If you do not supply these parameters, the Platform Server uses the following default attributes to allocate the dataset:

- RECFM=VB
- LRECL=20000
- BLKSIZE=20004
- ALCUNIT=value from the GLOBAL SPACE parameter
- PRIMARY=value from the GLOBAL PRIMARY_SPACE parameter
- SECONDARY=value from the SECONDARY_SPACE parameter
- DIRBLK=0 (the default is to allocate a sequential dataset and not a file)
- VOLSER=value from the GLOBAL VOLUME parameter
- UNIT=value from the GLOBAL UNIT parameter
- DATACLASS=the system default
- MGTCLASS=the system default
- STORCLASS=the system default

Appendix J. Netview Support

In addition to support within TSO and batch environments, the Platform Server REXX execs are fully functional under Netview.

The execs operate under Netview the same way as they operate under TSO with the following exceptions:

- The SAY keyword on the `FUSSEND` and `FUSRECV` commands is ignored. Status records will not be displayed for the Platform Server file transfers.
- You cannot terminate a wait that has been defined by the `WAIT` parameter of the `FUSSEND` or `FUSRECV` command.
- Because of the restrictions placed upon FUSFTP REXX execs by Netview, it might be difficult to use FUSFTP REXX execs under Netview. After executing FUSFTP, the exec will ask you for more information. To enter this information, you must enter GO *xxxxxxxx*; where, *xxxxxxxx* is the information to be returned to FUSFTP.

Activating Platform Server REXX Execs under Netview

To activate the Platform Server REXX execs under Netview, additions must be made to the Netview startup JCL.

The following additions must be made:

- The Platform Server `LOADLIB` must be added to the STEPLIB DD statement. For example:

  ```
  //STEPLIB  DD   DSN=&SQ1..CNMLINK,DISP=SHR
  //         DD   DSN=FUSION.LOADLIB,DISP=SHR
  ```

- The Platform Server execs library must be added to the DSICLD DD statement. For example:

  ```
  //DSICLD   DD   DSN=&SQ1..CNMCLST,DISP=SHR
  //         DD   DSN=FUSION.EXECS,DISP=SHR
  ```

After these changes have been made, Netview must be restarted. Afterwards, the Platform Server REXX execs can be used within the Netview environment.

Appendix K. XCOM Interface

The Platform Server provides the XCOM interface which can facilitate the migration from CA-XCOM to the Platform Server. The XCOM interface can accept XCOM batch parameters and create a Platform Server request instead of an XCOM request.

With this program, users who are familiar with XCOM batch parameters can get immediately productive using the Platform Server. In addition, this program eases the transition from XCOM to the Platform Server; both products can be run simultaneously.

The Platform Server and XCOM are different products, and as such perform the same tasks differently. The Platform Server for z/OS XCOM interface does not perform all of the functions that XCOM performs, just like XCOM does not perform all of the Platform Server functions. The Platform Server XCOM interface is designed to support enough XCOM functionality so that 90 - 95 percent of a user's request can be executed without any changes. When the Platform Server XCOM interface determines that it cannot perform a particular function, the step terminates with a return code of `0x28` (decimal 40).

The XCOM batch interface program is called XCOMJOB. The following components of XCOMJOB are supported by the Platform Server XCOM interface:

- `TYPE=EXECUTE`: Executes a request and waits for completion.
- `TYPE=SCHEDULE`: Schedules a request with the started task.
- `TYPE=INQUIRE`: Inquires on one or more scheduled requests.

Installing and Configuring the XCOM Interface

The Platform Server XCOM interface is installed when the Platform Server is installed.

You must perform a single step to assemble the XCOMDFLT options table. This is the same XCOMDFLT table that is assembled when XCOM is installed at an installation.

To create the Platform Server XCOMDFLT table, complete the following steps:

Procedure

1. Copy the XCOMDFLT member into the Platform Server SAMPLIB.

If you want, you can use the Platform Server default XCOMDFLT table that already exists in the FUSION SAMPLIB.

> **Note:** Because most XCOM parameters are ignored by the Platform Server, it might be easier to update the supplied XCOMDFLT member.

2. Make any changes to the XCOMDFLT table to configure it for the Platform Server.

   The parameters that require changes are as follows:

   - JOBACB: Points to the name of the VTAM ACBs defined to the Platform Server.
   - ACBNAME: Points to ACBNAME of the Platform Server started task.

   Add the following Platform Server parameter to #DFLTAB:

   FUSION_STCOVER={ YES | NO}

   - If this parameter is specified as YES, the ACBNAME parameter overrides the STCAPPL parameter normally defined in the PARM JCL statement. The STCAPPL statement is ignored when FUSION.STCOVER=YES is defined. This is done so that a user can submit the same JCL, except that the STEPLIB will point to the Platform Server library, and the Platform Server request is routed to the Platform Server started task, not to the XCOM started task.
   - If this parameter is specified as NO, the STCAPPL statement on the PARM JCL statement overrides the ACBNAME parameter defined to the XCOMDFLT table.

3. Assemble the XCOMDFLT table.

   You can use any assembly JCL. The following changes must be made to the JCL to assemble the Platform Server XCOMDFLT module:

   - Ensure that the SYSIN DD statement points to the XCOMDFLT in the Platform Server SAMPLIB.
   - Ensure the SYSLIB DD statement of the Assembler step points to the Platform Server SAMPLIB dataset.
   - Ensure that the XCOMDFLT module is linked into the Platform Server LOADLIB dataset with the name XCOMDFLT.

   At this point, the Platform Server XCOM interface is ready to use. The Platform Server XCOM interface program is called FUSXJOB. This program has an alias of XCOMJOB.

   To run the Platform Server XCOM interface, you must change the STEPLIB of the program to use the Platform Server loadlib. That way, when the XCOMJOB program is executed, the Platform Server program is executed instead of the XCOM program.

As stated before, the Platform Server XCOM interface accepts XCOM parameters and creates a Platform Server transfer request. Therefore, after the transfer parameters are read, only Platform Server protocols are used, which means the following things:

- The remote system must have a version of Platform Server installed. The Platform Server does not communicate with the XCOM started task.
- The messages displayed are Platform Server messages, not XCOM messages.

Fields Ignored by the XCOM Interface

The XCOM interface ignores some Platform Server fields.

The PARM fields that are ignored by the Platform Server XCOM interface are `COMPNEG`, `DISPALG`, `DUMPCL`, `EDESC`, `EROUT`, `GROUP`, `IDESC`, `IROUT`, `LOG`, `LOGCLASS`, `LOGDEST`, `LOGMODE`, and `LU`.

The SYSIN01 fields that are ignored by the Platform Server XCOM interface are `DISP`, `DROPSESS`, `EPRTY`, `LPASS`, `LUSER`, `REPORT`, `REPORTHOLD`, `RESTART`, `RNOTIFY`, `RNOTIFYNAME`, `SPRTY`, `TRUNCATE`, `USER`, and `PACK`.

The SYSIN01 fields that cause the Platform Server XCOM interface to terminate with a return code 40 (`0x28`) are `DEN`, `EXPDT`, `LABEL`, `LLABEL`, `LUNITCT`, `LVOLCT`, `LVOLSQ`, `RETPD`, `UNITCT`, `VOLCT`, `VOLSQ`, `CHARS`, `SPOOL`, `HOLDCOUNT`, `XTCERRDECR`, `XTCERRINCR`, `XTCERRREL`, `XTCERRPURGE`, `XTCGOODDECR`, `XTCGOODINCR`, `XTCGOODREL`, `XTCGOODPURGE`, `XTCJOB`, and `XTCNET`.

Example

The following example shows the JCL required to run the Platform Server XCOM interface:

```
//jobname JOB  ,'C-Fusion’,MSGCLASS=X,REGION=5M,CLASS=A
//S1SCH    EXEC PGM=XCOMJOB,
//  PARM='TYPE=SCHEDULE,ACBNAME=FUSN,GROUP=NYACCT'
//STEPLIB  DD  DISP=SHR,DSN=FUSION.LOADLIB
//XCOMLOG  DD  SYSOUT=*
//XCOMINQ  DD  DSN=yourdsn,DISP=SHR
//SYSIN01  DD  *
```

```
TYPE=SEND
CODE=TEXT
RECSEP=YES
LFILE=LOCAL.TEST.DATA
FILE=REMOTE.TEST.DATA
USERID=ABC123
PASSWORD=XYZ
//
```

Though you are using the Platform Server XCOM interface, you can still use some Platform Server features. For example, instead of the `USERID` and `PASSWORD` parameters defined in this example, you can use the Platform Server user profile feature. The `USERID` and `PASSWORD` parameters are replaced by the following parameter:

```
USERID=*PROFILE
```

That way, the clear text password does not have to be displayed in the JCL.

The following example shows the output that is displayed after executing the previous job:

```
TYPE=SEND

CODE=TEXT

RECSEP=YES

LFILE=LOCAL.TEST.DATA
FILE=REMOTE.TEST.DATA

USERID=ABC123
PASSWORD=XXXXXXXXXXXXXXXXXXXXXXXXX

PGTX4512I Activity I414000000 Queued to MFT Platform Server started task

PGTX4510I Successful Requests=1  Failed Requests=0  Timeout Requests=0

PGTX4511I FUSXJOB ending with Return code 0
```

If `TYPE=EXECUTE` is requested instead of `TYPE=SCHEDULE`, the following messages are displayed in the XCOMLOG dataset:

```
TYPE=SEND
CODE=TEXT
```

```
RECSEP=YES

LFILE=LOCAL.TEST.DATA
FILE=REMOTE.TEST.DATA

USERID=ABC123
PASSWORD=XXXXXXXXXXXXXXXXXXXXXXXXX
PGTX4512I Activity I414000002 Queued to MFT Platform Server started task
PGTX4521E Activity=I414000002  Status=ACTIVE    Record Count=0  Byte
Count=0
PGTX4521E Activity=I414000002  Status=COMPLETE  Record Count=3200 Byte
Count=259200
PGTX4510I Successful Requests=1  Failed Requests=0  Timeout Requests=0
PGTX4511I FUSXJOB ending with Return code 0
```

Appendix L. Preprocessing Actions

Preprocessing actions are executed before a transfer starts. Preprocessing actions can be executed on the initiator or the responder.

The format of preprocessing actions is the same as the format for postprocessing actions and the same tokens can be used. A maximum of 4 PPA (preprocessing and postprocessing) parameters can be defined for each transfer.

Sample Preprocessing Actions:

```
PPA="P,L,CALLPGM,mypgm tokens or data to be passed"
PPA="P,R,COMMAND,unixcmd command parameters"
PPA="P,L,CALLPGM,FUSPRE %QUEUE"
```

Transfer Retry Considerations for Preprocessing

When a transfer is retried after a recoverable error or after the preprocessing action terminates the transfer with a retryable error, preprocessing actions may be executed or may be bypassed. These criteria define when preprocessing actions are executed for a retried file transfer.

Initiator Transfer Retry

- When a transfer retries because a preprocessing action sets the return code to a recoverable error (2, 3, or 4), all preprocessing actions are executed on a transfer retry.
- When all preprocessing actions have executed with return code 0 or 1, the processing actions are not executed on the transfer retry.

Responder Transfer Retry

- When a transfer performs a checkpoint restart, preprocessing actions are not executed.
- When a transfer is retried and checkpoint restart is not performed, all preprocessing actions are executed.

Preprocessing Return Codes

Based on the return code from the Preprocessing program or command, Platform Server performs the following actions:

Return Code	Action
0	Continue with the transfer.
1	Override transfer parameters and continue with the transfer. This value is only used when using the CALLPGM action with the %QUEUE token. When not using the CALLPGM action with the %QUEUE token, this return code is treated the same as Return Code 4 - retryable error.
>1 & <=4	Terminate the transfer with a retryable error.
>4	Terminate the transfer with a permanent error.

Overriding of Transfer Parameters Using Preprocessing

PPA Preprocessing allows you to override file transfer parameters.

In order to override transfer parameter, you must follow these steps:

1. Only PPA CALLPGM allows you to override the transfer parameters.
2. Use the special token %QUEUE. This token recommends the use of special linkage when communicating to the PPA program. When the %QUEUE token is used, all other tokens and any parameter data is ignored.
3. SAMPLIB program FUSPRE defines how to code a preprocessing program that overrides transfer parameters. Any program name can be used.
4. Add the Preprocessing load module to the Platform Server STC STEPLIB.
5. When the return code is set to 1, transfer parameters can be overridden and the return code will be reset to 0.

The following transfer parameters can be overridden:

- `Local File Name`
- `Remote File Name`
- `Process Name`
- `User Data`
- `Compression`
- `Encryption`

SAMPLIB Member FUSPRE shows how the Preprocessing program can be coded to override transfer parameters. Here is an example of the PPA parameter that allows you to change transfer parameters:

```
PPA="P,L,CALLPGM,FUSPRE01 %QUEUE"
```

Appendix M. Post Processing Action (PPA) Substitutable Parameters

One thing that is common to most of the postprocessing actions is that you want to process the file that has just been sent or received. You can easily specify postprocessing actions by using the substitutable parameters. You can also define other parameters that can be useful when performing postprocessing actions.

When performing Platform Server postprocessing actions (`POST_ACTION` or `PPA`), you can define actions to perform when a transfer is completed. Typically, this action is to process a file when the transfer is completed successfully. Additional actions that you might want to take includes deleting or renaming a file when a send transfer is completed successfully or calling a scheduler, such as U7SVC, when a transfer is completed successfully.

The Platform Server runs on different platforms, not all of the parameters are valid on all systems. The substitutable parameters are relative to the system where they are running on. The parameters are substituted only after execution is completed and just before the postprocessing action is performed. Therefore, the parameter `%LFILE` on the initiator is different than the parameter `%LFILE` on the responder.

> **Note:** When used in the script program (OSIUC000), PPA substitutable parameters must be defined with two % parameters. This is because the script variables also use %. %% indicates to the scripting program that this is a PPA substitutable parameter and not a scripting variable.
>
> Example: %%DIR

The following table lists the substitutable parameters of postprocessing actions:

Parameter	Supported System	Description
`%ACB`	z/OS	Defines the VTAM ACB name.
`%DIR`	UNIX\Windows	Defines the directory name without the file name and drive. For example, if the file name is

Parameter	Supported System	Description
		`c:\a\b\c\d\config.txt`, `%DIR` is `a\b\c\d`.
`%DRIVE`	Windows	Defines the drive name without the file name, directory, and colon. For example, if the file name is `c:\a\b\c\d\config.txt`, `%DRIVE` is `c`.
`%FILE`	UNIX\Windows	Defines the file name without the drive or directory. For example, if the file name is `c:\a\b\c\d\config.txt`, `%FILE` is `config.txt`.
`%GDATE`	UNIX\Windows\z/OS	Defines the Gregorian date without the century. The date is in the format of YYMMDD.
`%GDATEC`	UNIX\Windows\z/OS	Defines the Gregorian date with the century. The date is in the format of CCYYMMDD.
`%HDIR`	UNIX\Windows	Defines the highest level sub-directory. For example, if the file name is `c:\a\b\c\d\config.txt`, `%HDIR` is `a`.
`%HLQ`	UNIX\Windows	Defines the high-level qualifier of a file. For example, if the file name is `c:\a\b\c\d\config.txt`, `%HLQ` is `config`.
`%JDATE`	UNIX\Windows\z/OS	Defines the Julian date without the century. The date is in the format of YYDDD.

Parameter	Supported System	Description
`%JDATEC`	UNIX\Windows\z/OS	Defines the Julian date with the century. The date is in the format of CCYYDDD.
`%JOBN`	z/OS	Defines the name of the Platform Server started task. This field can include 1 - 8 characters.
`%LFILE`	UNIX\Windows\z/OS	Defines the fully qualified local file name. For example, if the file name is `c:\a\b\c\d\config.txt`, `%LFILE` is `c:\a\b\c\d\config.txt`; if the file name is `NYTAX.ACCT.DATA.Y2001`, `%FILE` is `NYTAX.ACCT.DATA.Y2001`.
`%LLQ`	UNIX\Windows\z/OS	Defines the low-level qualifier of a file. For example, if the file name is `c:\a\b\c\d\config.txt`, `%LLQ` is `txt`; if the file name is `NYTAX.ACCT.DATA.Y2001`, `%LLQ` is `Y2001`.
`%NODRIVE`	Windows	Defines the file name without the drive. For example, if the file name is `c:\a\b\c\d\config.txt`, `%NODRIVE` is `a\b\c\d\config.txt`.
`%NOHDIR`	UNIX\Windows	Defines the directory name without the highest level sub-directory. For example, if the file name is `c:\a\b\c\d\config.txt`, `%NOHDIR` is `b\c\d`.
`%NOHLQ`	z/OS	Defines the file name without the high-level qualifier. For example, if the file name is

Parameter	Supported System	Description
		NYTAX.ACCT.DATA.Y2001, %NOHLQ is ACCT.DATA.Y2001.
%NOLLQ	z/OS	Defines the file name without the low-level qualifier. For example, if the file name is NYTAX.ACCT.DATA.Y2001, %NOLLQ is NYTAX.ACCT.DATA.
%NOSDIR	UNIX\Windows	Defines the directory name without the lowest level sub-directory. For example, if the file name is c:\a\b\c\d\config.txt, %NOSDIR is a\b\c.
%PROC	UNIX\Windows\z/OS	Defines the process name associated with the file transfer. The process name can include 1 - 8 characters.
%Q01 - %Q22	z/OS	Breaks up the z/OS file name into individual qualifiers. With this parameter, you can extract a particular qualifier. For example, if a file name is NYTAX.ACCT.DATA.Y2008, %Q01 is NYDAT, %Q02 is ACCT, %Q03 is DATA, %Q04 is Y2008, and %Q05 is not defined and therefore defaults to Q05.
%QUEUE	UNIX\Windows	Denotes that the preprocessing routine is called and can override some of the transfer parameters. SAMPLIB module FUSPRE shows a sample assembler program that can be used for processing.

Parameter	Supported System	Description
		This parameter is only allowed for preprocessing when the action is CALLPGM. For more information, see the "Preprocessing Actions" section.
`%SDIR`	UNIX\Windows	Defines the lowest level sub-directory. For example, if the file name is `c:\a\b\c\d\config.txt`, `%SDIR` is `d`.
`%SJOBID`	z/OS	Defines the job ID generated when the JCL transfer is submitted.
`%SJOBNAME`	z/OS	Defines the job name used in the JCL JOB statement of transfer.
`%SYSID`	z/OS	Defines the SMF system ID for the z/OS system where the Platform Server is running. This field can include 1 - 4 characters.
`%TIME`	UNIX\Windows\z/OS	Defines the time of the day. The time is in the format of HHMMSS.
`%TRN`	UNIX\Windows\z/OS	Defines the 10-digit Platform Server transaction number. **Note:** When used on the initiator, the transaction number of the initiator is used; while on the responder, the transaction number of the responder is used.
`%UDATA`	UNIX\Windows\z/OS	Defines the user data associated with the

Parameter	Supported System	Description
		file transfer. The user data can include 1 - 25 characters.

Note: The leading slash (backslash (\) for Windows and forward slash (/) for UNIX) is typically not included in the substitutable parameters. If the substitutable parameters are not defined (for example `%DRIVE` on UNIX or z/OS), the parameters are assigned to the parameter names without the leading percent sign (%). Therefore, if you use the parameter `%DRIVE` on z/OS, the value assigned is `DRIVE`.

Example

In this example, a file is sent from a UNIX system to a Windows system. The UNIX file is `/a/b/c/test.data`. The Windows file is `c:\temp\test.txt`.

The following sample shows how the postprocessing actions (PPA) can be coded if you want to rename the UNIX file and process the Windows file when the transfer is completed successfully.

The following command renames a file in the initiator if the request is completed successfully:

```
PPA="S,I,COMMAND,mv %LFILE /%NOSDIR/bkup/%FILE.%GDATEC.%TIME"
```

This executes the following UNIX command: `mv /a/b/c/test.data /a/b/bkup/test.data.20081231.141516`.

The following command processes a file in the responder if the request is completed successfully:

```
PPA="S,R,COMMAND,procfile %LFILE"
```

This executes the following Windows command: `procfile c:\temp\test.txt`.

Note: In the command, `procfile` is either an executable file or a batch file.

Appendix N. Directory Transfers

The Platform Server supports directory transfers for both send and receive requests.

The following types of files support directory transfer:

- z/OS sequential datasets
- z/OS libraries (specifically PDS/PDSE)
- z/OS UNIX file system

The Platform Server determines whether a file transfer request is a directory request based on the local and remote file names. The Platform Server supports both send and receive directory transfers.

For a send transfer, the Platform Server scans the local file name for the wildcard character "*". If the wildcard character is present in the local file name, the transfer is a directory transfer. Then, the Platform Server scans through the z/OS catalog, HFS file system, or PDS/PDSE, and initiates a file transfer request for any transfer that meets the selection criteria. After that, the Platform Server updates the remote file name with information from the local file extracted.

For a receive transfer, the Platform Server scans the remote file name for the wildcard character "*". If the wildcard character is present in the remote file name, the transfer is a directory transfer. Then, the Platform Server sends a request to the remote Platform Server system to extract files that match the selection criteria, and initiates a file transfer request for any transfer that meets the selection criteria. After that, the Platform Server updates the local file name with information from the file extracted from the remote Platform Server system.

Note: To preserve system resources, the Platform Server for z/OS limits the amount of storage used to hold file names for directory transfers. For example, if the average file name length is 64 bytes, the Platform Server for z/OS can send or receive approximately 2,000 files.

Directory Transfer Parameters

You can define the following two parameters for directory transfers: `CONTINUE` and `SUBDIR`.

By specifying CONTINUE=YES, the Platform Server will attempt to perform transfer requests, even if a transfer request fails. By specifying CONTINUE=NO, the Platform Server will stop processing when a transfer request fails.

The SUBDIR parameter defines whether subdirectories are processed. The Platform Server supports directory transfers for three different file types, and the SUBDIR parameter has a different meaning for all three file types. The following descriptions shows how the SUBDIR parameter is used for the three file types:

- z/OS sequential datasets

  z/OS datasets cannot have subdirectories. But because of the way that you extract data from the catalog, a mix of sequential and PDS/PDSE datasets might be returned from a catalog request. By specifying SUBDIR=NO, the Platform Server does not process any PDS/PDSE files extracted from the catalog. By specifying SUBDIR=YES, the Platform Server processes PDS/PDSE files in addition to the sequential files processed.

- z/OS libraries (specifically PDS/PDSE)

  PDS and PDSE files cannot have subdirectories. Therefore, the SUBDIR parameter is ignored for PDS and PDSE files.

- z/OS UNIX file system

  The z/OS UNIX file system is a directory based system. When you specify SUBDIR=NO, the Platform Server only looks for files in the defined base directory, and ignores any files in the subdirectories. When you specify SUBDIR=YES, the Platform Server looks for files in the base directory as well as files in the subdirectories.

Supported Wildcard Characters

Directory transfers support local file name and remote file name wildcard characters.

The following local file name wildcard characters are supported:

- Percent sign (%): Matches on current character
- Question mark (?): Matches on current character
- Asterisk (*): Matches on current qualifier
- Double asterisks (**): Matches on current qualifier and all subsequent qualifiers

The following remote file name wildcard character is supported:

- Asterisk (*): Matches on current qualifier

Examples

Example 1: Sending datasets to another z/OS system

In this example, the following files are defined to the z/OS system:

`TEST.DATA.ACCT1`

`TEST.DATA.ACCT2008`

`TEST.DATA.ACCTPDS` (contains members A, B, and C)

Specify the following parameters on a send file request:

`LFILE=TEST.DATA.*`

`RFILE=/tmp/data/`

`SUBDIR=YES`

Five file transfer requests are queued with the following local and remote files:

```
LF=TEST.DATA.ACCT1
RF=/tmp/data/ACCT1
```

```
LF=TEST.DATA.ACCT2008
RF=/tmp/data/ACCT2008
```

```
LF=TEST.DATA.ACCTPDS(A)
RF=/tmp/data/ACCTPDS.A
```

```
LF=TEST.DATA.ACCTPDS(B)
RF=/tmp/data/ACCTPDS.B
```

```
LF=TEST.DATA.ACCTPDS(C)
RF=/tmp/data/ACCTPDS.C
```

Example 2: Sending UNIX files to remote UNIX system

In this example, the following files are defined to the z/OS system:

`/test/data/acct1`

/test/data/acct2004

/test/data/acctpds/a

/test/data/acctpds/b

/test/data/acctpds/c

Specify the following parameters on a send file request:

LFILE=/test/data/*

RFILE=/tmp/data/

SUBDIR=NO.

Two file transfer requests are queued with the following local and remote files:

```
LF=/test/data/acct1
RF=/tmp/data/acct1
```

```
LF=/test/data/acct2008
RF=/tmp/data/acct2008
```

Example 3: Sending PDS members to remote UNIX system

In this example, the following members are in a PDS or PDSE file:

TEST.PDS(A1)

TEST.PDS(A2)

TEST.PDS(ABCD)

TEST.PDS(B1)

TEST.PDS(B2)

TEST.PDS(BCDE)

Specify the following parameters on a send file request:

LFILE=TEST.PDS(A*)

RFILE=/tmp/data/

SUBDIR=NO

Three file transfer requests are queued with the following local and remote files:

```
LF=TEST.PDS(A1)
RF=/tmp/data/A1
```

```
LF=TEST.PDS(A2)
RF=/tmp/data/A2
```

```
LF=TEST.PDS(ABCD)
RF=/tmp/data/ABCD
```

Example 4: Receiving files from remote UNIX system

In this example, the following files are in the remote UNIX system:

`/tmp/data/acct1`

`/tmp/data/acct2007`

`/tmp/data/acct2008`

`/tmp/data/acct2009`

`/tmp/data/auditing/abc`

`/tmp/data/auditing/def`

Specify the following parameters on a receive file request:

`LFILE=TEST.PDS( )`

`RFILE=/tmp/data/*`

`SUBDIR=NO`

Four file transfer requests are queued with the following local and remote files:

```
LF=TEST.PDS(ACCT1)
RF=/tmp/data/acct1
LF=TEST.PDS(ACCT2007)
RF=/tmp/data/acct2007
LF=TEST.PDS(ACCT2008)
RF=/tmp/data/acct2008
LF=TEST.PDS(ACCT2009)
RF=/tmp/data/acct2009
```

Sample Node Substitution

A user can have approximately 1000 remote locations. Often, you have files that need to be sent to each of the 1000 remote locations. Node substitution provides a way in which you can execute a single process to send files to different remote locations without updating the PROCESS statements whenever a new remote location is added or deleted.

Note: To use node substitution, you must define the node where the file is to be sent.

The following table lists the token parameters that can be used in node substitution:

Parameter	Description
`%LLQ`	Low-level qualifier
`%LFILE`	Local file name
`%NOHLQ`	Local file name without the high-level qualifier
`%Q01` - `%Q22`	Individual z/OS file qualifiers

The `NODE` parameter can use any of these token parameters. However, the token is restricted to 8 bytes or less. If the token contains more than 8 characters, the first 8 characters are substituted.

The `RFILE` parameter can also use any of these token parameters. The length of the remote file cannot exceed 255 characters.

User ID and Password Processing

In node substitution, you can generate the user ID and password in one of the following ways:

- `RUSER` and `RPASS` parameters in the PROCESS statement
- `RUSER=*PROFILE`
  - With a profile for each node definition
  - `Profile NODE=*ALLNODE` to set the user ID and password the same for all nodes

`WAIT` Parameter Processing

The following explanation shows how the `WAIT` parameter works for send directory transfers:

- When the `NODE` parameter does not have a token:

  Single-thread the transfers.

  - `WAIT=YES`: The batch process is completed after all transfers are completed.
  - `WAIT=NO`: The batch process is completed after the send directory transfer is queued.
- When the `NODE` parameter has a token:
  - `WAIT=YES`: Single-thread the transfers (Queue Transfers with `WAIT=YES`). The batch process is completed after all transfers are completed.
  - `WAIT=NO`: Submit the transfer without waiting for the transfer to complete. (Queue Transfers with `WAIT=NO`). The batch process is completed after the send directory transfer is queued.

Examples

The following two examples show sample commands of node substitution.

Example 1:

```
//REMLOC JOB  NOTIFY=&SYSUID
//*
//*
//EXEC0000 EXEC PGM=OSIUB000,
//         PARM='SERVER=FUSION'
//*
//STEPLIB  DD   DISP=SHR,DSN=FUSION.LOADLIB
//SYSPRINT DD   SYSOUT=*
//SYSTRACE DD   DUMMY
//SYSIN    DD   *
           PROCESS,REMLOC,TRANSFER,SEND
           TRANS_TYPE=FILE
           DSN=COMPANY.ACCT.S*
           RFILE=C:\COMPANY\INCOMING\%NOHLQ.TXT
           EFFECT=CN
           NODE=%LLQ
           RUSER=*PROFILE
           WAIT=NO
/*
```

Example 2:

Assume that the following files exist:

```
PROD.ACCT.NYNODE.TAX2008.DATA

PROD.ACCT.CHNODE.TAX2008.DATA

PROD.ACCT.LANODE.TAX2008.DATA

PROD.ACCT.ATNODE.TAX2008.DATA

PROD.ACCT.PHNODE.TAX2008.DATA
```

Issue the following PROCESS statements:

```
NODE=%Q03
LFILE=PROD.ACCT.**
RFILE=/acct/data/%Q04.%Q05
```

The following five transfers will be scheduled:

```
NODE=NYNODE
LFILE=PROD.ACCT.NYNODE.TAX2008.DATA
RFILE=/acct/data/TAX2008.DATA
```

```
NODE=CHNODE
LFILE=PROD.ACCT.CHNODE.TAX2008.DATA
RFILE=/acct/data/TAX2008.DATA
```

```
NODE=LANODE
LFILE=PROD.ACCT.LANODE.TAX2008.DATA
RFILE=/acct/data/TAX2008.DATA
```

```
NODE=ATNODE
LFILE=PROD.ACCT.ATNODE.TAX2008.DATA
RFILE=/acct/data/TAX2008.DATA
```

```
NODE=PHNODE
LFILE=PROD.ACCT.PHNODE.TAX2008.DATA
RFILE=/acct/data/TAX2008.DATA
```

Appendix O. Assembler API

FUSCFAPI is an API program with which you can execute the Platform Server platform transfers, and get the status of a file transfer. This API is designed to be called from an Assembler program, although it can be called by any programming language that can create the necessary control blocks and provide the z/OS standard linkage.

With this API, you can communicate with the Platform Server z/OS started task through one of following four methods. You can select a method by the settings in APIDSECT (see for more details).

- Cross memory through a PC call
- SNA communication
- TCP communication through an IP address
- TCP communication through an IP name

This API supports the following three functions:

- Execute Transfer: With this function, you can execute a file transfer by using the same parameter input as that is used by the Platform Server Batch interface. Transfers can be queued for execution (`WAIT=NO`) or executed immediately (`WAIT=YES`). When a transfer is queued for execution, control is returned to the API immediately after the request is queued. The return code defines whether the request is queued, and the caller is not notified when the transfer is completed. When a transfer is executed immediately, control is returned to the API only after the transfer is completed successfully or the transfer exhausts all of its retries. The return code defines the actual status of the file transfer.
- Transfer Status: This function passes a transfer ID to the Platform Server, and the Platform Server returns the status of that transfer.
- Ping: With this function, you can check the status of the Platform Server started task.

JCL Requirements

STEPLIB must include the load library that contains the Platform Server load modules:

Statement	Description
VTAMTRAC	An optional DD statement. You must define this statement when using SNA to communicate with the Platform Server started task and `API_SERVER_TRACE_FLAG` is set to `API_SERVER_TRACE_YES`. Output from this DD is typically sent to SYSOUT.
TCPTRAC	An optional DD statement. You must define this statement when using TCP (`IPADDR` or `IP Name`) to communicate with the Platform Server started task and `API_SERVER_TRACE_FLAG` is set to `API_SERVER_TRACE_YES`. Output from this DD is typically sent to SYSOUT.

SYSPRINT	DD statement.
	The name used for this DD statement can be any valid DD name. However, the API calling program must have an OPEN DCB for this DD name with the following characteristics:
	`RECFM=FBA`
	`LRECL=133`
	`MACRF=PM`

AUDPATH1/2	Two DD statements. These two DD Statements are required under the following circumstances: • The server type is PCCALL. • The server function is Transfer Status.

Register Settings

You must set the registers at entry to and upon return from FUSCFAPI.

At entry to FUSCFAPI, you must set the following registers:

- R1: Address of the address of APIDSECT
- R13: Save area address
- R14: Return address
- R15: Address of FUSCFAPI entry point

Upon return from FUSCFAPI, you must set the following registers:

- R0 - R14: Unchanged
- R15: Return code of the request

APIDSECT Fields

The following table lists the APIDSECT fields:

Field	Description
`API_SERVER_TYPE`	Defines the method of communicating with the Platform Server for z/OS started task. This field is required. You must enter one of the following values: • `API_SERVER_TYPE_PCCALL` Communicate through cross memory by using a PC call. **Note:** The API program must be executing on the same LPAR for PCCALL communications to work. • `API_SERVER_TYPE_SNA` Communicate through SNA. When communicating through SNA, the client program can execute on any machine that has SNA connectivity between the client program and the Platform Server started task. • `API_SERVER_TYPE_IPADDR` Communicate through TCP by using an IP address.

Field	Description
	When communicating through IP address, the client program can execute on any machine that has TCP connectivity between the client program and the Platform Server started task. • `API_SERVER_TYPE_IPNAME` Communicate through TCP by using an IP name. When communicating through IP name, the client program can execute on any machine that has TCP connectivity between the client program and the Platform Server started task.
`API_SERVER_TRACE_FLAG`	Defines whether to trace the communication with the Platform Server started task. This field is optional. If not entered, not tracing is used. If tracing is set to `API_SERVER_TRACE_YES` for SNA communication, the VTAMTRAC DD statement must be defined. If tracing is set to `API_SERVER_TRACE_YES` for IPADDR or IPNAME communication, the TCPTRAC DD statement must be defined. **Note:** This option is ignored for PCCALL communication.

`API_OUTPUT_DCB_ADDR`	Defines the address of an open DCB that is to be used by the API program to write output about the request.
	This field is required.
	The DCB must have the following attributes:
	`RECFM=FBA`
	`LRECL=133`
	`MACRF=PM`
	Note: The calling application is responsible for opening and closing this DCB.

Field	Description
	Note: If API_OUTPUT_DCB_ADDR is NULL, the API program does not write any data to it. Error messages are displayed on the console. While this flag works for any request, it should be used only on a PING request.
API_SERVER_FLAG_SILENT	When this flag is set, the API program does not write any output. It also does not display any error messages on the console. While this flag works for any request, it should be used only on a PING request.
API_OUTPUT_DDNAME	When API_OUTPUT_DDNAME_ADDR is set to 0x00000000, this field is ignored. When API_OUTPUT_DCB_ADDR is set to a value other than 0x00000000, the output data is written to the DD name pointed to by this parameter. This parameter is only used when API_OUTPUT_DCB_ADDR is set to 0x00000000.

You must specify the following field when API_SERVER_TYPE_PCCALL is defined:

Field	Description
API_SERVER_STCNAME	Defines the name of the Platform Server z/OS started task. **Note:** When using API_SERVER_TYPE_PCCALL, the Platform Server started task must be executing on the same LPAR as the API program.

You can specify the following fields when API_SERVER_TYPE_SNA is defined:

Field	Description
API_SERVER_LUNAME	Defines the name of the Platform Server for z/OS started task VTAM APPLID. This field is required.

Field	Description
`API_SERVER_ACBNAME`	Defines the name of the VTAM ACB that is defined for use by the Platform Server API programs. This field is required. If this ACB name includes 6 characters or less, the Platform Server attempts to add a 2-digit numeric suffix to the ACB name. In this way, multiple API requests can use the same `API_SERVER_ACBNAME` value. A typical installation creates ACB definitions for the Platform Server clients with a name such as `FUSN00` - `FUSN09`, thus supporting 10 concurrent connections.
`API_SERVER_MODENAME`	Defines the VTAM mode name that is to be used for the connection. This field is optional. If not defined, the mode name associated with the VTAM definition is used.

You can specify the following fields when `API_SERVER_TYPE_IPADDR` is defined:

Field	Description
`API_SERVER_IPADDR`	Defines the IP address in EBCDIC dotted decimal format of the machine where the Platform Server started task is executing. This field is required.
`API_SERVER_TCIPJOBNAME`	Defines the name of the TCP/IP started task. This field is optional. If not defined, the value `TCPIP` is used.
`API_SERVER_IPPORT`	Defines the 5-digit EBCDIC decimal IP port that is to be used to connect to the Platform Server started task. This field is optional. If not defined, the default Platform Server port value of `46464` is used.

You can specify the following fields when API_SERVER_TYPE_IPNAME is defined:

Field	Description
API_SERVER_IPNAME	Defines the 64-digit IP name of the machine where the Platform Server started task is executing. This field is required.
API_SERVER_TCIPJOBNAME	Defines the name of the TCP/IP started task. This field is optional. If not defined, the value TCPIP is used.
API_SERVER_IPPORT	Defines the 5-digit EBCDIC decimal IP port that is to be used to connect to the Platform Server started task. This field is optional. If not defined, the default Platform Server port value of 46464 is used.

Execute Transfer Parameters

The following input parameter and output parameter are used when executing the Platform Server transfers:

Field	Description
API_TRANSFER_PARM_ADDR	Defines the parameter cards that is to be passed to the Platform Server API program. The parameters are identical to the parameters that are accepted by the Platform Server Batch interface. Comments can be placed in the parameter list by starting the parameter with an asterisk (*). However, comments are not allowed in the first parameter. The parameters are passed to the API using the following format: llll, which indicates length in bytes of the parameter passed. The length does not include the 2-byte length field. The first parameter passed must be one of the following parameters:

Field	Description
	- `PROCESS,,TRANSFER,SEND`: To send a file. - `PROCESS,,TRANSFER,RECEIVE`: To receive a file. - `PROCESS,,PING`: To receive a status. The parameter stream ends with a parameter that has a length of 0x0000. **Note:** Parameters are passed directly to the Platform Server Batch Interface program. Therefore, the API has the same restrictions as the Batch interface regarding continuation. Data is accepted in columns 1 - 71. Column 72 is used for continuation. Columns 73 and above are ignored. In addition to using column 72 as a continuation column, you can use the special Batch continuation characters: plus sing (+) and double plus signs (++).
`API_TRANSFER_ID`	This parameter is returned by the started task when a transfer is queued successfully or executed either successfully or unsuccessfully. The transfer ID is not returned under the following circumstances: - An APIDSECT parameter is filled in incorrectly. - A transfer parameter is invalid. - The API program cannot connect to the Platform Server started task. - A send transfer is requested, but the file does not exist. - You request a user profile, but the profile does not exist.

Transfer Status Parameters

The following input parameter and output parameter are used when executing the Platform Server transfers:

Field	Description
API_TRANSFER_PARM_ADDR	Defines the parameter cards that is to be passed to the Platform Server API program. The parameters are passed to the API using the following format: llll, which indicates length in bytes of the parameter passed. The length does not include the 2-byte length field. The following two parameters are required: • PROCESS,,OPERATE • TRNSTATUS=*tttttttttt* *tttttttttt* is the transfer ID of a file transfer. The parameter stream ends with a parameter that has a length of 0x0000.
API_TRANSFER_STATUS	This field is returned by the Platform Server and defines the transfer status of the transfer specified in the TRNSTATUS parameter. This is an 8-byte character field left aligned and padded with spaces.

Sample Program

A Sample program and a macro to map the required parameters are distributed in the Platform Server SAMPLIB dataset.

Parameter	Description
APIDSECT	Defines the parameters that are to be passed between the calling program and FUSCFAPI.
APIPROG	Shows a sample program that calls FUSCFAPI to perform the following functions: • Execute a transfer. • Get the status of a transfer.

Establish addressability to the APIDSECT work area:

```
LA    R10,WORK_APIDSECT_DATA        POINT to APIDSECT
     USING APIDSECT,R10
. . . . . . .
. . . . . . .
WORK_APIDSECT_DATA  DS  CL(APIDSECT_LENGTH)
. . . . . . .
. . . . . . .
     COPY APIDSECT
```

Initialization functions:

```
*    Setup Server Parameters
         XC    WORK_APIDSECT_DATA,WORK_APIDSECT_DATA
         MVI   API_SERVER_TRACE_FLAG,API_SERVER_TRACE_NO
*     Save DCB Address (DCB Must be open)
         LA    R15,WORK_OUTPUT_DCB
         ST    R15,API_OUTPUT_DCB_ADDR
```

Set PCCALL (Cross Memory Call) information:

```
         MVI   API_SERVER_TYPE,API_SERVER_TYPE_PCCALL   Indicate PC
         MVC   API_SERVER_STCNAME,=CL8'FUSION'      Started Task Name
```

Or, set SNA information:

```
         MVI   API_SERVER_TYPE,API_SERVER_TYPE_SNA    Indicate SNA
         MVC   API_SERVER_LUNAME,=CL8'FUSNAPPL'       Server APPLID
         MVC   API_SERVER_ACBNAME,=CL8'FUSN'          MY ACBNAME
         MVC   API_SERVER_MODENAME,=CL8'#BATCH'       Mode Name
```

Or, set TCP (IP Address) information:

```
         MVI   API_SERVER_TYPE,API_SERVER_TYPE_IPADDR  Indicate IPADDR
         MVC   API_SERVER_IPADDR,=CL15'127.0.0.1'      Server IPADDR
         MVC   API_SERVER_IPPORT,=CL5'46464'           Server IP Port
         MVC   API_SERVER_JOBNAME,=CL8'TCPIP'          TCPIP STC Name
```

Or, set TCP (IP Name) information:

```
         MVI   API_SERVER_TYPE,API_SERVER_TYPE_IPNAME  Indicate IPNAME
         MVC   API_SERVER_IPNAME,=CL24'server.ip.name' Server IPName
         MVC   API_SERVER_IPPORT,=CL5'46464'           Server IP Port
         MVC   API_SERVER_JOBNAME,=CL8'TCPIP'          TCPIP STC Name
```

Create file transfer parameter addressability:

```
*     Save Transfer Parameters address
         LA    R15,WORK_API_PARAMETERS        Point to Parameter Area
         ST    R15,API_TRANSFER_PARM_ADDR     Save in APIDSECT
WORK_API_PARAMETERS  DS  100CL82
```

Set transfer parameters:

```
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'PROCESS,,TRANSFER,SEND' Set PROCESS
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'TYPE=TEXT'     Set TYPE=
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'LF=LOCAL.FILE.NAME' Set FROM Dataset
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'RF=REMOTE.FILE.NAME' Set TO File name
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'NODE=ZOSSYSA'  Set TO NODE
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'WAIT=YES'      WAIT for transfer
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'TRY=1'         Only Try 1 time
         LA    R15,82(R15)                    Point to next parameter
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'PROCNAME=APISEND'  Process Name
         LA    R15,82(R15)                    Point to next parameter
```

```
*
         MVC   0(2,R15),=AL2(80)              Set the Length
         MVC   2(80,R15),=CL80'DESCRIPTION=TestAPI'  Description
         LA    R15,82(R15)                    Point to next parameter
```

Terminate transfer parameters:

```
         MVC   0(2,R15),=H'0'                 Indicate no more parms
```

Appendix P. TIBCO Accelerator

The Platform Server uses the TIBCO Accelerator technology to provide a faster way to send files to remote destinations, where latency problems normally exist in long distance connections.

TIBCO Accelerator provides greatly improved data transfer speeds over high bandwidth and high latency IP networks. With TIBCO Accelerator, transfers are completed up to 10 to 100 times faster than FTP, overcoming the slowness because of latency problems.

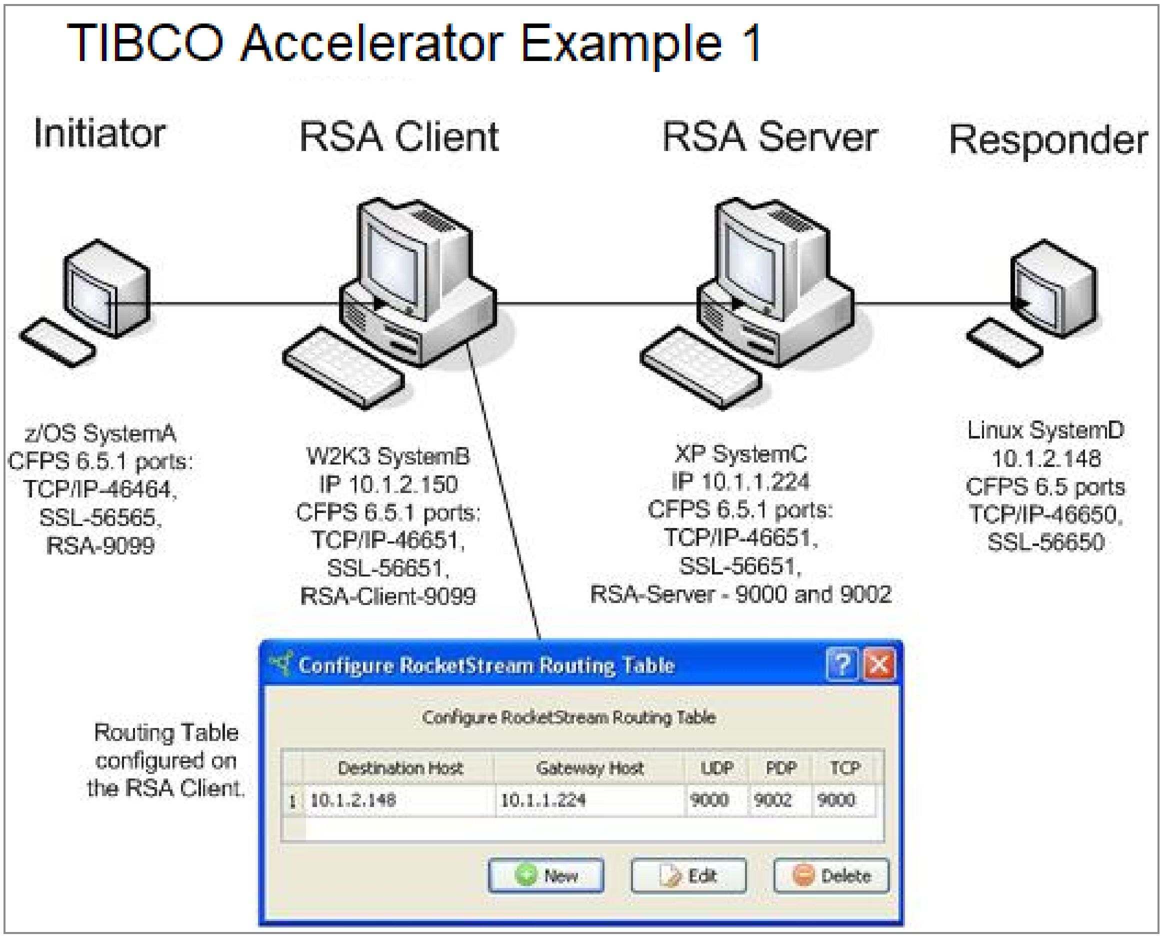
TIBCO Accelerator uses its own version of User Datagram Protocol (UDP) and parallel implementation of TCP, called Parallel Delivery Protocol (PDP).

Note: Ports 9000, 9002, and 9100 - 9199 must be opened in the firewall to allow the TIBCO client to access the TIBCO server. If requests are initiated from an external computer, these ports must be opened on the firewall for incoming traffic. If requests are initiated from an internal computer, these ports must be opened on the firewall for outgoing traffic.

Using TIBCO Accelerator within Platform Server for z/OS

Currently TIBCO Accelerator is available using the Platform Server for Windows. A Platform Server for Windows can act both as a TIBCO Accelerator i client, a TIBCO Accelerator i server, or both. It is possible to send files to and receive files from z/OS platforms, but only when the files pass through the Platform Servers for Windows running the TIBCO Accelerator i service (`RsTunnel.exe`).

The following figure shows an example of using TIBCO Accelerator i within a Platform Server for z/OS:



Configure RocketStream Routing Table

	Destination Host	Gateway Host	UDP	PDP	TCP
1	10.1.2.148	10.1.1.224	9000	9002	9000

This figure demonstrates sending a file from a Platform Server for z/OS (system A) to a Platform Server for Linux (system D). These two servers do not contain the TIBCO Accelerator technology; therefore, they must forward the data to a Platform Server for Windows that runs the TIBCO Accelerator service. See *TIBCO® Managed File Transfer Platform Server for Windows User's Guide* for more information on configuring TIBCO Accelerator on a Windows server.

The following sample JCL shows how to set a z/OS transfer to be passed to the TIBCO Accelerator client:

```
PROCESS,SEND1,TRANSFER,SEND
     DSN=HLQ.FILE.NAME
     REMOTE_FILE=/home/Linux/SEND1
     IPADDR=10.1.2.148
     IPPORT=46464
     TYPE=TEXT
     CRLF=Y
```

```
          EFFECT=CR
          HOLD=NO
          REMOTE_USER=RMTUSER
          RPASS=PASSWORD
          RSA=Y
          RSHOST=10.1.2.150
          RSPORT=9099
          RSP=PDP
```

The JCL is set to send a file to system D. However, the last four parameters are added to send this transfer to the TIBCO Accelerator client. By setting the RSA (or RSACCELERATER) parameter to Y, you define the Platform Server to send this file by using TIBCO Accelerator. In this example, you define the TIBCO Accelerator client (RSHOST) which receives the transfer request, and define what port (RSPORT) your TIBCO Accelerator client listens on and what TIBCO Accelerator protocol you want to use (RSP).

The following table lists the TIBCO Accelerator parameters supported by z/OS. See Batch Interface for detailed description of each parameter.

Parameter	Shortcut	Description
RSACCELERATOR	RSA	Defines whether transfers go to an TIBCO Accelerator host.
RSCOMPRESS	RSC	Defines whether TIBCO Accelerator uses compression.
RSENCRYPT	RSE	Defines whether TIBCO Accelerator uses encryption.
RSHOST	RSH	Defines the TIBCO Accelerator host name or IP address.
RSPORT		Defines the TIBCO Accelerator port number.
RSMAXSPEED	RSMAX	Defines the maximum speed that TIBCO Accelerator uses.
RSPROTOCOL	RSP	Defines the protocol that TIBCO Accelerator uses.

You can also configure RocketStream transfers through the Platform Server Interactive interface. See Interactive Interface for details.

No configuration change is required on the responder for this type of transfer. A TIBCO server can send a file to any Platform Server responder version 6.5.1 and below, including Platform Servers for Windows, UNIX, z/OS, and AS/400 (System i).

Appendix Q. SMF Data Format

The following table defines the data that the Platform Server writes to SMF. The Platform Server writes the standard SMF 18-byte header and the following table defines the data after the header. An assembler macro that defines this work area is located in: FUSION.SAMPLIB(OSIQUEUE)

Note: The hexadecimal displacement and decimal displacement are relative to 0.

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
0	0	8	Binary	**QUEUE_BYTE_COUNT**	Number of bytes transferred
24	36	4	Binary	**QUEUE_CHECKPOINT_COUNT**	Number of checkpoints that have been taken
28	40	4	Binary	**QUEUE_CHECKPOINT_INERVAL**	Number of minutes between checkpoints
2C	44	4	Binary	**QUEUE_RECORD_COUNT**	Number of records transferred
44	68	1	Hex	**QUEUE_WAIT_FLAG**	Wait Flag: initiators only 0x80: wait for transfer to complete
50	80	2	Binary	**QUEUE_TRY_COUNT**	Number of times the transfer has been attempted

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
52	84	2	Binary	**QUEUE_RETRY_MAX**	Maximum number of times a request can be tried
56	86	2	Binary	**QUEUE_LRECL**	MVS file LRECL
58	88	8	Char	**QUEUE_PROC**	Transfer process name
60	96	10	Char	**QUEUE_TRANSNUM**	Local transaction number
6D	109	8	Char\Hex	**QUEUE_REMOTE_SYS** **QUEUE_IPADDR** **QUEUE_NODENAME**	Remote system information: • If 0xffffffffffffffff, this is an IP name. Wherein, `QUEUE_IPNAME` has bytes 1 - 24, and `QUEUE_IPNAME2` has bytes 25 - 64 • If 2nd 4 bytes = 0xffffffff, this is an IP address. Wherein, the first 4 bytes have IP address in hexadecimal. • Otherwise, this

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					is a node name. Wherein, bytes 1 - 8 hold the node name, and `QUEUE_NODENAME_EXT` has bytes 9 - 32.
79	121	256	Char	**QUEUE_LOCAL_FILE**	z/OS local file name
17A	378	1	Char	**QUEUE_AFFECT**	Output file disposition • C: create • R: replace • A: append • X: create and replace • Y: create and append • Z: create and replace new
17B	379	25	Char	**QUEUE_USER_DATA**	User data associated with the transfer
194	404	1	Char	**QUEUE_TRANSFER_TYPE**	Type of transfer: • S: send • R: receive

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- U: submit - E: Rexx exec - I: script
195	405	8	Char	**QUEUE_LOCAL_USER**	Name of local MVS user
1B9	441	1	Char	**QUEUE_NOTIFY_USER_TYPE**	T: notify TSO user ID R: notify Roscoe user ID
1BA	442	20	Char	**QUEUE_REMOTE_USER**	Name of remote user ID
1CE	462	7	Char	**QUEUE_DATE_ELIGIBLE**	Date transfer is eligible to run (YYYYDDD)
1D5	469	6	Char	**QUEUE_TIME_ELIGIBLE**	Time transfer is eligible to run (HHMM00)
1DB	475	7	Char	**QUEUE_DATE_STARTED**	Date transfer execution started (YYYYDDD)
1E2	482	6	Char	**QUEUE_TIME_STARTED**	Time transfer execution started (HHMM00)
1E8	488	7	Char	**QUEUE_DATE_INTRUP**	Date transfer completed (YYYYDDD)
1EF	495	6	Char	**QUEUE_TIME_**	Time transfer

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
				INTRUP	completed (HHMM00)
1F5	501	7	Char	**QUEUE_DATE_ EXPIRE**	Date when transfer will expire and be purged (YYYYDDD)
1FC	508	1	Hex	**QUEUE_EMAIL_ FLAG**	Email notification flag • 0x80: email on successful transfer • 0x40: email on unsuccessful transfer • 0x08: MQ notify on successful transfer • 0x04: MQ notify on unsuccessful transfer
1FD	509	1	Char	**QUEUE_FTAM_ DOC_TYPE**	Document type: • 1 or 0x01: text • Any other value: binary
204	516	1	Char	**QUEUE_RECFM**	File MVS record format: • X: `RECFM=F` • A: `RECFM=FS`

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- F: RECFM=FB - B: RECFM=FBS - Y: RECFM=V - W: RECFM=VS - V: RECFM=VB - Z: RECFM=VBS - U: RECFM=U
205	517	1	Char	**QUEUE_ CHECKPOINT**	Checkpoint flag Y: use checkpoints
20B	523	1	Char	**QUEUE_CRLF**	Record delimiters (ignored for z/OS to z/OS): - Y: records are CRLF delimited. - N: records have no delimiters. - L: records are LF delimited.
20F	527	3	Char	**QUEUE_VERSION**	Version of local Platform Server system (VRM)
212	530	1	Char	**QUEUE_ TRANSFER_ WORK**	Type of file transfer: - F: file to file - P: file to print

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- J: file to job - C: send command
250	592	120	Char	**QUEUE_LAST_ MSG**	Last message issued regarding the transfer
340	832	256	Char	**QUEUE_REMOTE_ FILE**	Name of remote file
448	1096	64	Char	**QUEUE_EMAIL_ GOOD**	Email address for successful notification
4C8	1224	64	Char	**QUEUE_EMAIL_ FAIL**	Email address for unsuccessful notification
4CC	1228	6	Char	**QUEUE_VOLSER**	MVS volser for output file
4D3	1235	1	Hex	**QUEUE_ INITIATION_ FLAGS**	Flag that defines how transfer was initiated: - 0x80 bit: transfer is initiator - 0x40 bit: transfer is responder
4E0	1248	10	Char	**QUEUE_REMOTE_ TRANSNUM**	Remote transaction number

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
4EA	1258	4	Binary	**QUEUE_ALLOC_PRI**	Primary file allocation
4EE	1262	4	Binary	**QUEUE_ALLOC_SEC**	Secondary file allocation
4F2	1266	1	Hex	**QUEUE_COMPRESSION**	Compression flag: • 0x11: LZ compression • 0x12: RLE compression • 0x14: ZLIB compression (See **QUEUE_ZLIB_TYPE**)
4F3	1267	1	Hex	**QUEUE_ENCRYPT_FLAG**	Encryption flag (bit settings): • 0x80: no encryption • 0x40: DES • 0x20: 3DES • 0x10: Blowfish • 0x08: Blowfish long (448 bit) • 0x04: AES (for example, AES) 256 bit
4F4	1268	1	Hex	**QUEUE_SSL_FLAG**	SSL flag: • 0x80: SSL is on.

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- Any other value: SSL is off.
538	1336	8	Binary	**QUEUE_ COMPRESS_ BYTES**	Number of compressed bytes sent
545	1349	16	Char	**QUEUE_DOMAIN_ NAME**	Windows domain name
555	1365	14	Char	**QUEUE_IPNAME** **QUEUE_ NODENAME_EXT**	Either IP name or node name. See **QUEUE_REMOTE_ SYS** field (0x6D)
56D	1389	1	Char	**QUEUE_ERROR_ TYPE**	Type of error that occurred: - L: local - R: remote - N: network
56E	1390	2	Binary	**QUEUE_IPPORT**	IP port associated with initiator requests
58C	1420	4	Binary	**QUEUE_ALLOC_ DIR**	PDS directory block allocation
590	1424	2	Binary	**QUEUE_ MEMBER_COUNT**	Count of members transferred (z/OS to z/OS only)
597	1431	1	Hex	**QUEUE_RETURN_**	Transfer return code:

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
				CODE	- 0: transfer succeeds. - <> 0: transfer fails.
598	1432	8	Char	**QUEUE_DSN_ UNIT**	Unit name for output file
5A0	1440	8	Char	**QUEUE_ DATACLASS**	SMS data class for this transfer
5A8	1448	8	Char	**QUEUE_ MGMTCLASS**	SMS management class for this transfer
5B0	1456	8	Char	**QUEUE_ STORAGECLASS**	SMS storage class for this transfer
5B8	1464	8	Char	**QUEUE_JOBID**	The JES job ID when the transfer type is JOB.
5C0	1472	1	Char	**QUEUE_ALLOC_ TYPE**	Allocation type: - C: cylinders - T: tracks - B: blocks - K: kilobytes - M: megabytes
5C8	1480	2	Binary	**QUEUE_BLKSIZE**	The MVS block size of a file
562	1506	1	Hex	**QUEUE_LDISP_ DISP**	Local file status:

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- 0x80: `DISP=SHR` - 0x40: `DISP=MOD` - 0x20: `DISP=OLD` - 0x10: `DISP=NEW` - 0x30: `DISP=NEWR` - 0x50: `DISP=NEWA` - 0x31: `DISP=NEWN`
5E3	1507	1	Hex	**QUEUE_LDISP_ STATUS**	Local file normal, failure disposition: - 0x80: `DISP=(,KEEP)`, normal disposition - 0x40: `DISP=(,CATLG)` - 0x20: `DISP=(,DELETE)`, failure disposition - 0x08: `DISP=(,,KEEP)` - 0x04: `DISP=(,,CATLG)` - 0x02: `DISP=(,,DELETE)`
5E4	1508	1	Hex	**QUEUE_RDISP_**	Remote file status:

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
				DISP	- 0x80: `DISP=SHR` - 0x40: `DISP=MOD` - 0x20: `DISP=OLD` - 0x10: `DISP=NEW` - 0x30: `DISP=NEWR` - 0x50: `DISP=NEWA` - 0x31: `DISP=NEWN`
5E5	1509	1	Hex	**QUEUE_RDISP_ STATUS**	Remote file normal, failure disposition: - 0x80: `DISP=(,KEEP)`, normal disposition - 0x40 `DISP=(,CATLG)` - 0x20 `DISP=(,DELETE)` - 0x08 `DISP=(,,KEEP)`, failure disposition - 0x04 `DISP=(,,CATLG)` - 0x02 `DISP=(,,DELETE)`
5E8	1512	1	Char	**QUEUE_**	Command type (for

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
				COMMAND_TYPE	TRANS_ TYPE=COMMAND): • C: command • R: REXX eXEC • J: CALLJCL • P: CALLPGM • J: submit job
5FA	1530	6	Char	**QUEUE_IVOLUME**	Input volume for uncataloged input files
600	1536	3	Char	**QUEUE_UNIX_ FILE_ PERMISSIONS**	UNIX permissions: • Byte 1: user permissions (0 - 7) • Byte 2: group permissions (0 - 7) • Byte 3: other permissions (0-7)
606	1542	1	Char	**QUEUE_VSAM_ FILE_TYPE**	VSAM file type: • K: KSDS • E: ESDS • R: RRDS
60D	1549	44	Char	**QUEUE_DCB_ LIKE**	Model file for sequential or VSAM files

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
				QUEUE_VSAM_ LIKE	
63A	1594	1	Hex	**QUEUE_ACTION_ FLAG1**	Postprocessing action flag 1: • 0x80: success • 0x40: initiator
63B	1595	1	Hex	**QUEUE_ACTION_ TYPE1**	Postprocessing action type 1: • 0x01: CALLPGM • 0x02: COMMAND • 0x03: CALLJCL • 0x04: SUBMIT
63C	1596	64	Char	**QUEUE_ACTION_ DATA1**	Postprocessing action data 1 See **QUEUE_ ACTION_DATA1_EXT** for additional 192 bytes.
67C	1660	1	Hex	**QUEUE_ACTION_ FLAG2**	Postprocessing action flag 2: • 0x80: success • 0x40: initiator
67D	1661	1	Hex	**QUEUE_ACTION_ TYPE2**	Postprocessing action type 2: • 0x01: CALLPGM

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					- 0x02: COMMAND - 0x03: CALLJCL - 0x04: SUBMIT
67E	1662	4	Char	**QUEUE_ACTION_ DATA2**	Postprocessing action data 2 See **QUEUE_ ACTION_DATA1_EXT** for additional 192 bytes.
6BE	1726	1	Hex	**QUEUE_ACTION_ FLAG3**	Postprocessing action flag 3: - 0x80: success - 0x40: initiator
6BF	1727	1	Hex	**QUEUE_ACTION_ TYPE3**	Postprocessing action type 3: - 0x01: CALLPGM - 0x02: COMMAND - 0x03: CALLJCL - 0x04: SUBMIT
6C0	1728	64	Char	**QUEUE_ACTION_ DATA3**	Postprocessing action data 3 See **QUEUE_ ACTION_DATA1_EXT** for additional 192 bytes.

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
700	1792	1	Hex	**QUEUE_ACTION_ FLAG4**	Postprocessing action flag 4: • 0x80: success • 0x40: initiator
701	1793	1	Hex	**QUEUE_ACTION_ TYPE4**	Postprocessing action type 4: • 0x01: CALLPGM • 0x02: COMMAND • 0x03: CALLJCL • 0x04: SUBMIT
702	1794	64	Char	**QUEUE_ACTION_ DATA4**	Postprocessing action data 4 See **QUEUE_ ACTION_DATA1_EXT** for additional 192 bytes.
742	1858	4	Binary	**QUEUE_ACTION_ RETCODE1**	Postprocessing return code 1
746	1862	4	Binary	**QUEUE_ACTION_ RETCODE2**	Postprocessing return code 2
74A	1866	4	Binary	**QUEUE_ACTION_ RETCODE3**	Postprocessing return code 3
74E	1870	4	Binary	**QUEUE_ACTION_ RETCODE4**	Postprocessing return code 4

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
764	1892	16	Char	**QUEUE_LOCAL_CONVTBL_FILE**	Local conversion table file name
774	1908	16	Char	**QUEUE_REMOTE_CONVTBL_FILE**	Remote conversion table file name
791	1937	1	Char	**QUEUE_ZLIB_TYPE**	ZLIB compression type (1 - 9) See **QUEUE_COMPRESS** for details.
79C	1948	40	Char	**QUEUE_IPNAME2**	IP name extension See **QUEUE_REMOTE_SYS** for details.
7C4	1988	8	Char	**QUEUE_IUNIT**	Input file unit name
7CC	1996	4	Binary	**QUEUE_MQ_WAIT_INTERVAL**	Interval waited before EOF on MQ transfer
800	2048	192	Char	**QUEUE_ACTION_DATA1_EXT**	PPA Data1 extension
8C0	2240	192	Char	**QUEUE_ACTION_DATA2_EXT**	PPA Data2 extension
980	2432	192	Char	**QUEUE_ACTION_DATA3_EXT**	PPA Data3 extension
A40	2624	192	Char	**QUEUE_ACTION_DATA4_EXT**	PPA Data4 extension

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
B00	2816	64	Char	**QUEUE_RSHOST**	RocketStream host name
B40	2880	4	Binary	**QUEUE_ RSMAXSPEED**	RocketStream maximum speed (0 = unlimited)
B44	2884	2	Binary	**QUEUE_RSPORT**	RocketStream IP port number
B46	2886	1	Hex	**QUEUE_RSFLAG**	RocketStream Acceleration flag: • 0x80: use RocketStream Accelerator • Any other value: do not use RocketStream
B47	2887	1	Hex	**QUEUE_ RSPROTOCOL**	RocketStream protocol: • 0x00: TCP • 0x01: UDP • 0x02: PDP
B48	2888	1	Hex	**QUEUE_ RSENCRYPT**	RocketStream encryption: • 0x00: no encryption • 0x01: Blowfish 256 encryption

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
B49	2889	1	Hex	**QUEUE_ RSCOMPRESS**	RocketStream compression: • 0x00: no compression • 0x01: use the best compression • 0x02: use the default compression • 0x03: use the fastest compression
000B4A	2890	6	Char	**QUEUE_OUTPUT_ TAPE_VOL1**	Output volume #1
000B50	2896	6	Char	**QUEUE_OUTPUT_ TAPE_VOL2**	Output volume #2
000B56	2902	6	Char	**QUEUE_OUTPUT_ TAPE_VOL3**	Output volume #3
000B5C	2908	6	Char	**QUEUE_OUTPUT_ TAPE_VOL4**	Output volume #4
000B62	2914	6	Char	**QUEUE_OUTPUT_ TAPE_VOL5**	Output volume #5
000B68	2920	4	Binary	**QUEUE_BLKSIZE_ LBI**	LBI block size
000BBC	3004	2	Binary	**QUEUE_LOCAL_ LABEL_SEQ**	Local label seq #

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
000BBE	3006	2	Binary	**QUEUE_REMOTE_LABEL_SEQ**	Remote label seq #
000BC0	3008	1	Hex	**QUEUE_LOCAL_LABEL_TYPE**	Local label type
000BC1	3009	1	Hex	**QUEUE_REMOTE_LABEL_TYPE**	Remote label type
000BC2	3010	1	Hex	**QUEUE_CRC_FLAG**	CRC flag: • 0x80: Initiator supports CRC • 0x40: Responder supports CRC
000BC3	3011	1	Char	**QUEUE_TLS_SECTYPE**	TLS security type: • 0: TLSv1 • 1: TLSv1.1 • 2: TLSv1.2
000BC4	3012	4	Hex	**QUEUE_CRC**	CRC computed
000BC8	3016	8	Binary	**QUEUE_TIMEUSED**	Time used by the transfer task
000BD0	3024	4	Char	**QUEUE_TLS_CIPHERUSED**	Cipher used in SSL or tunnel mode
000BD4	3028	1	Hex	**QUEUE_FILE_OPTIONS2**	Truncate flag: 0x80 Truncate data 0x40 Wrap truncated

Hex Displ.	Decimal Displ.	Field Length	Field Type	Field Name	Description
					data
000BD5	3029	1	Hex	**QUEUE_CCSID_ BOM_FLAG**	BOM flag for ICONV conversion: • 0x80: Add BOM • 0x40: Remove BOM
000BD6	3030	1	Hex	**QUEUE_ PREPOSTPROC_ FLAG**	Directory transfer preprocessing and postprocessing flag
000BD8	3032	8	Binary	**QUEUE_ TIMEUSED_TLS**	Time used by SSL/Tunnel and ZLIB compression task
BE0	3040	8	Binary	**QUEUE_FILE_ ERROR_TRY_MAX**	File error try max count
BE2	3042	8	Binary	**QUEUE_FILE_ ERROR_TRY_INT**	File error try interval
BE4	3044	8	Binary	**QUEUE_ STCNAME**	Name of STC that executed this transfer.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for TIBCO® Managed File Transfer Platform Server for z/OS is available on the TIBCO® Managed File Transfer Platform Server for z/OS Product Documentation page.

- *TIBCO® Managed File Transfer Platform Server for z/OS Release Notes*
- *TIBCO® Managed File Transfer Platform Server for z/OS Managed File Transfer Overview*
- *TIBCO® Managed File Transfer Platform Server for z/OS Installation and Operation Guide*
- *TIBCO® Managed File Transfer Platform Server for z/OS Security Guide*
- *TIBCO® Managed File Transfer Platform Server for z/OS User's Guide*
- *TIBCO® Managed File Transfer Platform Server for z/OS Message Manual*

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (https://www.tibco.com/patents) for details.