iSoft Commerce Suite

Command Reference

Document Edition 1.0
February 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

About This Document

Audience xv
How to Print the Document xvi
Contacting iSoft xvi
Documentation Conventions xviii

1. Commerce Suite Command Reference

Understanding Command Syntax 1-4
addkey 1-5
- Syntax 1-5
- Required Parameters 1-5
- Command Example 1-7

addpair 1-8
- Syntax 1-8
- Required Parameters 1-8
- Optional Parameters 1-9
- Expanded Parameters 1-10
- Command Example 1-11

addpeer 1-12
- Syntax 1-12
- Required Parameters 1-12
- Command Example 1-13

addroute 1-14
- Syntax 1-14
- Required Parameters 1-14
- Command Example 1-15

addserver 1-16

Syntax 1-16
Required Parameters 1-16
Command Example 1-17
batch 1-18
Syntax 1-18
Required Parameters 1-18
Command Example 1-18
dbconnect 1-19
Syntax 1-19
Required Parameters 1-19
Command Example 1-19
dbdisconnect 1-20
Syntax 1-20
Required Parameters 1-20
Command Example 1-20
deletekey 1-21
Syntax 1-21
Required Parameters 1-21
Command Example 1-22
deletepair 1-23
Syntax 1-23
Required Parameters 1-23
Command Example 1-23
deleteroute 1-24
Syntax 1-24
Required Parameters 1-24
Command Example 1-24
deleteserver 1-25
Syntax 1-25
Required Parameters 1-25
Command Example 1-25
exportkey 1-26
Syntax 1-26
Required Parameters 1-26
Command Example 1-27

getkeys 1-28
Syntax 1-28
Required Parameters 1-28
Command Example 1-28
getpairs 1-29
Syntax 1-29
Required Parameters 1-29
Command Example 1-29
getroutes 1-30
Syntax 1-30
Required Parameters 1-30
Command Example 1-30
getservers 1-31
Syntax 1-31
Required Parameters 1-31
Command Example 1-31
getworkorders 1-32
Syntax 1-32
Required Parameters 1-32
Command Example 1-32
history 1-33
Syntax 1-33
Required Parameters 1-33
Command Example 1-33
importkey 1-34
Syntax 1-34
Required Parameters 1-34
Optional Parameters 1-35
Command Example 1-35
insertpair 1-36
Syntax 1-36
Required Parameters 1-36
Optional Parameters 1-36
Command Example 1-37
insertroute 1-38

Syntax 1-38
Required Parameters 1-38
Command Example 1-39
insertserver 1-40
Syntax 1-40
Required Parameters 1-40
Command Example 1-41
listinbound 1-42
Syntax 1-42
Required Parameters 1-42
Command Example 1-42
listkeys 1-43
Syntax 1-43
Optional Parameters 1-43
Command Examples 1-43
listpairs 1-44
Syntax 1-44
Optional Parameters 1-44
Command Examples 1-44
listpeers 1-45
Syntax 1-45
Required Parameters 1-45
Command Example 1-45
listroute 1-46
Syntax 1-46
Optional Parameters 1-46
Command Examples 1-46
listservers 1-47
Syntax 1-47
Required Parameters 1-47
Command Example 1-47
r 1-48
Syntax 1-48
Required Parameters 1-49
Command Example 1-49

receivefile 1-50
Syntax 1-50
Required Parameters 1-50
Command Example 1-50
remotekey 1-51
Syntax 1-51
Required Parameters 1-51
Example 1-52
remotepair 1-53
Syntax 1-53
Required Parameters 1-53
Optional Parameters 1-54
Example 1-54
remoteroute 1-55
Syntax 1-55
Required Parameters 1-55
Command Example 1-56
remotesend 1-57
Syntax 1-57
Required Parameters 1-57
Optional Parameters 1-57
remoteserver 1-59
Syntax 1-59
Required Parameters 1-59
Example 1-59
removekey 1-60
Syntax 1-60
Required Parameters 1-60
Example 1-61
removepair 1-62
Syntax 1-62
Required Parameters 1-62
Example 1-62
removeroute 1-63
Syntax 1-63

Required Parameters 1-63
Example 1-63
removeserver 1-64
Syntax 1-64
Required Parameters 1-64
Example 1-64
replicatekeys 1-65
Syntax 1-65
Required Parameters 1-65
Example 1-65
replicatepairs 1-66
Syntax 1-66
Required Parameters 1-66
Example 1-66
replicateroutes 1-67
Syntax 1-67
Required Parameters 1-67
Example 1-67
replicateservers 1-68
Syntax 1-68
Required Parameters 1-68
Example 1-68
requestcert 1-69
Syntax 1-69
Required Parameters 1-69
Optional Parameters 1-69
send 1-71
Syntax 1-71
send Command Quick Reference Table 1-71
Required Parameters 1-74
Optional Parameters 1-74
Expanded Parameters 1-78
Command Example 1-79
sendcert 1-80
Syntax 1-80

Required Parameters 1-80
Command Example 1-81
set 1-82
Syntax 1-82
set Command Quick Reference Table 1-82
Command Example 1-101
shutdown 1-102
Syntax 1-102
Required Parameters 1-102
Command Example 1-102
start 1-103
Syntax 1-103
Required Parameters 1-103
Command Example 1-103
startdatabse 1-104
Syntax 1-104
Required Parameters 1-104
Command Example 1-104
status 1-105
Syntax 1-105
Required Parameters 1-105
Example 1-105
stop 1-106
Syntax 1-106
Optional Parameters 1-106
Example 1-106
stopdatabse 1-108
Syntax 1-108
Required Parameters 1-108
Command Example 1-108
version 1-109
Syntax 1-109
Required Parameters 1-109
Example 1-109

Glossary

About This Document

This document introduces and outlines the iSoft Commerce Suite Commerce Suite features, services, and architecture.

- Chapter 1, "Commerce Suite Command Reference," provides information about the Commerce Suite administration commands.
- Glossary, provides a listing of commonly used terms found in this document.

Audience

This document is intended primarily for use by iSoft Commerce Suite data administration personnel responsible for configuration, maintenance, and use of the Commerce Suite system.

This document has been written with the assumption that iSoft Commerce Suite administrators and users have a general understanding of the following concepts and technologies:

- Your business application software and business practices
- Electronic Data Interchange over the Internet (EDI-INT)
- E-Commerce
- Uniform Code Council (UCC)
- Data types
- Transport protocols

- Security standards
- The Internet
- Windows operating systems
- UNIX operating system

How to Print the Document

You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at http://www.adobe.com.

Contacting iSoft

Refer to the following section for instructions on contacting various iSoft support departments.

For Customer Support

If you have any questions about this version of the iSoft Commerce Suite software, or if you have any problems installing and running iSoft Commerce Suite, contact iSoft Technical Support at **support@iSoft.com**.

When contacting Customer Support, be prepared to provide the following information:

- Your name, email address, phone number, and fax number.

- Your computer type, manufacturer, model, CPU, memory, disk space, and network environment.
- The name and version of the product you are using.
- A description of the problem and the content of any error messages.

If you are unable to resolve your problem, call us at 214-890-9988

For Sales Support

For Sales support, contact us at the following phone numbers and Email address:

- Phone: 214-890-9988
- Fax: 214-890-7686
- Email: **sales@iSoft.com**

For Documentation Support

Your feedback on the iSoft Commerce Suite documentation is important to us. Send us email at **docsupport@isoft.com** if you have questions or comments. Your comments will be reviewed directly by iSoft professionals who create and update the iSoft Commerce Suite documentation.

In your email message, please indicate that you are using the documentation for the iSoft Commerce Suite.

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Meaning
bold	Names of fields, buttons, icons, and check boxes.
Italics	Words or phrases that you type. Options that you select.
{ }	Indicates a set of choices from which you must choose one.
`monospace text`	Code samples, commands and their options, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
`monospace italic text`	Variables in code Example: `String Customername;`
|	Separates two mutually exclusive choices in a syntax line. Type one of the choices, not the symbol.
parameters	Specifies a variable name or other information you must provide.
[ ]	Indicates optional parameters. You typically type only the information within the brackets, not the brackets.
...	Indicates that a parameter can be repeated several times in a command line. You enter only the information, not the ellipsis (...).

1 Commerce Suite Command Reference

The following sections provide information about the Commerce Suite administration commands.

Table 1-1 presents an overview of the Commerce Suite administration commands. The following sections describe command syntax and parameters, and provide an example of each command.

Table 1-1 Administrative Commands Overview

Command	Task
addkey	Create key-material for data encryption and authentication purposes.
addpair	Define a new trading partner relationship.
addpeer	Define a supported Transport Agent.
addroute	Define a filter or routing rule.
addserver	Define a new server/protocol combination.
batch	Process a file containing one or more console commands.
dbconnect	Connect Commerce Suite to a database.
dbdisconnect	Disconnect Commerce Suite from a database.
deletekey	Remove a public-key pair definition from the database.
deletepair	Remove a trading partner relationship from the database.

Command	Task
deleteroute	Remove a routing rule from a database.
deleteserver	Remove a server/protocol from the database.
exportkey	Export a public-key certificate and private-key to files.
getkeys	Read key-pair information from the database.
getpairs	Read trading partner relationship data from the database.
getroute	Read routing rules from the database.
getservers	Read server/protocol settings from the database.
getworkorders	Read a pending work order from the database.
history	Display recent command history.
importkey	Import an X.509 certificate and corresponding private-key.
insertpair	Insert a trading-relationship pair into the database.
insertroute	Insert a routing rule into a database.
insertserver	Insert a server-protocol into the database.
listinbound	Display active inbound services.
listkeys	Display active public-key pairs.
listpairs	Display active trading partner relationships.
listpeers	Display active Router agents
listroute	Display a list of routing rules.
listservers	Display Commerce Suite protocol servers.
r	Repeat a command from the command history.
receivefile	Process a file as a received message.
remotekey	Replicate a public-key pair to the host.

Command	Task
remotepair	Replicate a public-key pair to a remote host.
remoteserver	Start a server-protocol on a remote host.
removekey	Removes a public-key pair from memory
removepair	Remove a trading-relationship pair from memory.
removeroute	Remove a routing rule from memory.
removeserver	Remove a server-protocol from memory.
send	Send data to a remote host computer.
sendcert	Send a certificate to a trading partner.
set	Set application parameters.
shutdown	Terminate the application.
start	Start an application service.
startdatabase	Connect Commerce Suite to a database.
status	Display current application status information.
stop	Stop an application service.
stopdatabase	Disconnect Commerce Suite from a database.
version	Display program version information.

Understanding Command Syntax

Syntax is the order in which you must type command-line commands and options that follow it. You must type elements that appear in bold in the syntax line exactly as they appear. Elements that appear in italic are placeholders representing information you must provide.

The following sample syntax line illustrates the form used in this document:

command parameters [options] [...]

The following table describes each command syntax element:

Element	Meaning
command	Specifies the name of the command.
{ }	Indicates a set of choices from which you must choose one.
|	Separates two mutually exclusive choices in a syntax line. Type one of the choices, not the symbol.
parameters	Specifies a variable name or other information you must provide.
[ ]	Indicates optional parameters. You typically type only the information within the brackets, not the brackets.
...	Indicates that a parameter can be repeated several times in a command line. You enter only the information, not the ellipsis (...).

addkey

`addkey` creates key-material for data encryption and authentication purposes. The key-material consists of a public-key and a private-key. The public-key is exportable to an X.509 digital-certificate format. The private-key is exportable to a PKS#1 RSA Private Key format. Both public and private key data may be stored in the database. If the `set -df` option has been entered, key material generated by the `addkey` command is automatically stored in the database.

Key-material is produced for a specific use by a specific trading relationship. For this reason, the `from`, `to`, and `usage` parameters must be entered when creating the key-material.

Syntax

```
addkey <from> <to> <usage> <key-bits> <issuer> <subject>
```

Required Parameters

The following parameters are required when executing the `addkey` command:

from
specifies the sending trading partner name.

to
specifies the receiving trading partner name.

usage = A | B | D | E | H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

A
specifies to sign only

B
: specifies to encrypt only

D
: specifies to decrypt only.

E
: specifies to sign and decrypt.

H
: specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

key-bits = 512 | 1024 | 2048
: specifies the length of the RSA modules in bits. (512, 1024, or 2048)

issuer = self | CA
: specifies how the public-key data is created.

self
: specifies that the public-key data will be created as a self-signed CA root certificate.

CA
: specifies that the public-key data will be created as an end-user certificate signed with the private-key associated with the signing relationship.

subject = C | S | L | O | OU | CN | E | T
: specifies a subject for the public-key certificate in a delimited string of keyword-value pairs. Each keyword-value pair is delimited with a semi-colon.

C
: specifies the country.

S
: specifies the state or province.

L
: specifies the locality (city or town)

O
: specifies the organization.

OU
: specifies the organization unit.

CN
: specifies the common name.

E
: specifies an email address.

T
: specifies the title.

Command Example

```
addkey sender receiver E 1024 self C=US;S=TX;L=Dallas;O=iSoft;
CN=Test
```

addpair

`addpair` defines a new trading partner relationship. A trading partner relationship is a set of data describing how data may be transferred from one trading partner entity to another trading partner entity. Each trading partner entity is defined by an alphanumeric sequence of characters or with a quoted identifier.

The `addpair` command always stores trading partner relationships in memory. If the `set -df` parameter has been issued, enabling the automatic write of memory updates to the database, then the `addpair` statement will have the result of inserting a record into the database.

Syntax

```
addpair <from> <to> <to-url> <rcpt-url> <notify-name> <inbox>
[in|out][<send-parameters>]
```

Required Parameters

The following parameters are required when executing the `addpair` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

to-url
specifies the destination URL address.

rcpt-url
specifies the receipt URL for outbound asynchronous receipts.

notify-name
specifies the intended outbound recipient of the receipt.

For Connect:Enterprise Interfacing: When using the Connect:Enterprise mailbox system, the mailbox receiving and storing the data must be specifies using the `notify-name` parameter.

inbox

specifies the inbound storage location or URL address.

When Commerce Suite is used in conjunction with an external mailboxing system, the relationships can be configured so that some inbound data is directed toward one maiboxing system and other inbound data is directed toward another system, or the underlying file system. To support this capability, the `inbox` parameter must be specified in a URL format if the repository is not the underlying file system.

For Connect:Enterprise Interfacing: When using the Connect:Enterprise mailbox system, the URL scheme must be `ce://` and be followed by the hostname and port of the Connect:Enterprise system into which the data is to be stored.

Multiple and different Connect:Enterprise mailboxing systems can be concurrently supported for inbound data delivery in this manner.

Example:

```
addpair yourco myco http://myco:8080 http://yourco8080 A2B
ce://mailbox:8000 in -sC -e -r1
```

Optional Parameters

The following parameters are optional when executing the `addpair` command.

send-parameters

specifies default `send` parameters (optional; outbound only)

direction = in | out

specifies inbound or outbound data flow. Omitting the directional parameter results in an outbound data-flow definition.

Using the `in` parameter in conjunction with a databse results in the relationship table protocol field being altered to make a value distinction between inbound and outbound relationships.

Expanded Parameters

The following parameters have been expanded to include MQSeries-specific definitions.

Traditional Syntax

```
addpair <from> <to> <to-URL> <rcpt-URL> <notify-name>
<inbox> [in|out][<send-parameters>]
```

Example

```
addpair HG012 MERCFDI http://hg.com/
mailto:edi@hg.com HG012 in/hg -sC
```

MQSeries Syntax

```
addpair<scheme>//<queue-manager-name>/<queue-name>
```

Outbound Example

```
<command>addpair A B http://as2.b.com
mailto:as2@A.com A_B mq://QM_polaris/A_B
out</command>
```

Inbound Example

```
<command>addpair B A http://as2.A.com
mailto:as2@B.com B_A mq://QM_polaris/B_A
in</command>
```

The `addpair` statements illustrate how the `mq://` scheme may be applied to the `inbox` parameter to denote the queue to receive inbound data for a relationship.

Note that an optional positional parameter (`out|in`) is now supported by the `addpair` command to distinguish between inbound and outbound relationships. This parameter is optional, but should be used for implementations that apply the same notifying parameter (`A_B`, `B_A` in the above example) for both inbound and outbound relationships.

Command Example

```
addpair HG012 MERCFDI http://hg.com/ mailto:edi@hg.com HG012 in/hg
-sC
```

addpeer

`addpeer` defines a supported Transport Agent. The `addpeer` command adds a Commerce Suite definition that communicates with the current Commerce Suite instance. The existing Commerce Suite instance may communicate with the new Commerce Suite instance as either a Router Agent, forwarding inbound connections, or as an Administrative Agent, forwarding remote-send commands.

The `addpeer` command is only necessary when dynamic clustering is not available or desired. Dynamic clustering is the technology used by the Commerce Suite to automatically distribute service-level information to a Router or Administrative Agent. When this information is not automatically distributed, the `addpeer` command can be used to statically define the Agents participating in an Agent cluster.

Syntax

```
addpeer <name> <url>
```

Required Parameters

The following parameters are required when executing the `addpeer` command.

name

specifies a symbolic name assigned to the new Commerce Suite.

url

specifies the address of the Commerce Suite agent's control agent and port.

Command Example

```
addpeer TRANSPORT1 http://192.168.0.12:2081
```

addroute

`addroute` defines a filter or routing rule.

Syntax

`addroute <from> <to> <condition> <filter> <url>`

Required Parameters

The following parameters are required when executing the `addroute` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

condition
specifies the routing condition. The only currently supported routing condition is `ct=<n>` where `<n>` is a content type code.

filter
this parameter is reserved. The value must be `none`.

url
specifies the outbound location for the route. The url may use the following schemas:

`file://`

`mq://`

Command Example

```
addroute me you ct=E none mq://QM_polaris/edi
```

addserver

`addserver` defines a new server/protocol combination. The `addserver` commands adds a definition for a Transport or Router Agent inbound service. The `addserver` command also implicitly defines Transport or Router servers that are to be remotely configured by an Administrative Agent.

The `addserver` command is only used by Administrative Agents to define services that must be started remotely by the Administrative Agent and those Transport or Router Agents that must receive configuration data from the Administrative Agent. When a `addserver` command is processed by an Administrative Agent, one or more command/control messages may be sent to a Transport or Router Agent instructing that Agent to start a service.

Syntax

```
addserver <name> <group> <role> <url> <control-url>
```

Required Parameters

The following parameters are required when executing the `addserver` command:

name
: specifies a user-defined name to identify a Commerce Suite process.

group
: The peer-group number to which this service belongs.

role
: specifies the remote Agent role. (T=Transport, R=Router)

url

specifies the URL of the service to be started.

control-url

The command/control URL of the remote Agent.

Command Example

```
addserver TN13 2 R http://edi.hg.com:4080 p2p://192.168.0.1:2080
```

batch

`batch` processes a file containing one or more console commands. The `batch` command instructs the Commerce Suite to open and execute a file containing a formatted configuration or work order file.

Syntax

```
batch <filename>
```

Required Parameters

The following parameters are required when executing the `batch` command.

filename
specifies an absolute or relative path and filename.

Command Example

```
batch p2pagent.cfg
batch /workorder/sendedi.txt
```

dbconnect

`dbconnect` connects Commerce Suite to a database. before the `dbconnect` command can be effective, parameters must be setup to tell the Commerce Suite how to connect. For ODBC, the driver-manager, host, database-name, user, and password must be known, since Commerce Suite uses dsn-less connections. For ESQL, only the database, user, and password must be known.

Syntax

```
dbconnect
```

Required Parameters

There are no required or optional parameters necessary when executing the `dbconnect` command.

Command Example

ODBC Example

```
set -ddSQL~Server -dhWEBHUB2 -dnp2pagent -dusa -dwpassword
dbconnect
```

ESQL/C Example

```
set -dnp2pagent -dusa -dwpassword
dbconnect
```

dbdisconnect

`dbdisconnect` disconnects the Commerce Suite from the database.

Syntax

```
dbdisconnect
```

Required Parameters

There are no required or optional parameters necessary when executing the `dbdisconnect` command.

Command Example

```
dbdisconnect
```

deletekey

`deletekey` removes a public-key pair definition from the database.

Syntax

```
deletekey <from> <to> <keyusage>
```

Required Parameters

The following parameters are required when executing the `deletekey` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

keyusage = A | B | D | E | H | J | O
specifies which key is to be deleted.

A
specifies to sign only

B
specifies to encrypt only

D
specifies to decrypt only.

E
specifies to sign and decrypt.

H
specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

Command Example

```
deletekey myco yourco E
```

deletepair

`deletepair` removes a trading partner relationship from the database.

Syntax

`deletepair <from> <to> <protocol>`

Required Parameters

The following parameters are required when executing the `deletepair` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

protocol = HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

Command Example

`deletepair sender receiver http`

deleteroute

`deleteroute` removes a routing rule from a database.

Syntax

```
deleteroute <from> <to> <condition>
```

Required Parameters

The following parameters are required when executing the `deleteroute` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

condition
specifies the routing condition.

Command Example

```
deleteroute me you ct=E
```

deleteserver

`deleteserver` removes a server/protocol from the database.

Syntax

`deleteserver <name>`

Required Parameters

The following parameters are required when executing the `deleteserver` command:

name
: specifies a user-defined name to identify a Commerce Suite process.

Command Example

`deleteserver servername`

exportkey

`exportkey` exports a public-key certificate and private-key to files.

Syntax

`exportkey <from> <to> <usage> <certificate-file> <private-key-file>`

Required Parameters

The following parameters are required when executing the `exportkey` command:

from
specifies the sending trading partner name.

to
specifies the receiving trading partner name.

usage = A | B | D | E |H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

A
specifies to sign only

B
specifies to encrypt only

D
specifies to decrypt only.

E
specifies to sign and decrypt.

H
specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

certificate-file
: specifies the file-system name for the public-key certificate.

private-key-file
: specifies the file-system name for the public-key certificate.

Command Example

```
exportkey sender receiver E signkey.cer signkey.prv
```

getkeys

`getkeys` reads public key information from the database. The `getkeys` command retrieves all certificate and key material information from the database and populates the Commerce Suite memory with the security material needed to process message transfers.

The `getkeys` command is only functional if the database parameters have been defined with the `set -d?` commands and the `dbconnect` command has been issued.

Syntax

```
getkeys
```

Required Parameters

There are no required or optional parameters necessary when executing the `getkeys` command.

Command Example

```
getkeys
```

getpairs

`getpairs` reads trading partner relationship data from the database. The `getpairs` command retrieves all trading partner relationship information from the database and populates the Commerce Suite memory with the configuration material needed to process message transfers.

The `getpairs` command is only functional if the database parameters have been defined with the `set -d*` command and the `dbconnect` command has been issued.

Syntax

```
getpairs
```

Required Parameters

There are no required or optional parameters necessary when executing the `getpairs` command.

Command Example

```
getpairs
```

getroutes

`getroutes` reads routing rules from the database.

Syntax

```
getroutes
```

Required Parameters

There are no required or optional parameters necessary when executing the `getroutes` command.

Command Example

```
getroutes
```

getservers

`getservers` reads server/protocol settings from the database. The `getservers` command retrieves all remote service and Agent information from the database and populates the Commerce Suite memory with the material needed to remotely configure Agents and issue remote commands.

The `getservers` command is only functional if the database parameters have been defined with the `set -d?` commands and the `dbconnect` command has been issued.

Syntax

```
getservers
```

Required Parameters

There are no required or optional parameters necessary when executing the `getservers` command.

Command Example

```
getservers
```

getworkorders

`getworkorders` reads a pending work order from the database. The `getworkorders` command retrieves pending work order records from the database and appends them to the internal Commerce Suite command queue for processing.

This command is typically issued automatically, internally within the Administrative Agent. Under normal circumstances, there is no need to issue this command at the console.

Syntax

```
getworkorders
```

Required Parameters

There are no required or optional parameters necessary when executing the `getworkorders` command.

Command Example

```
getworkorders
```

history

`history` displays recent command history. The `history` command displays a list of the most recently-entered commands stored in the command history queue. The `history` command will only display commands if the `set -ic<val>` command has been entered, enabling command history retention. The commands listed are assigned a sequential number that can be used by the `r` command to recall and re-issue a stored command.

Syntax

```
history
```

Required Parameters

There are no required or optional parameters necessary when executing the `history` command.

Command Example

```
history
```

importkey

`importkey` imports an X.509 certificate and corresponding private-key. The `importkey` command imports an X.509 public-key certificate and, optionally, an associated private-key data file into memory. The imported key material must be associated, in this command, with a defined trading partner relationship and usage code.

Syntax

```
importkey <from> <to> <usage> [<option> [...] ]
```

Required Parameters

The following parameters are required when executing the `importkey` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

usage = A | B | D | E | J | O
specifies the uses for the key-material when assigning the key-material to the trading-relationship.

A
specifies to sign only

B
specifies to encrypt only

D
specifies to decrypt only.

E
: specifies to sign and decrypt.

H
: specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

Optional Parameters

The following parameters are optional when executing the `importkey` command:

<*option*> [...]
: specifies one or more optional parameters.

-bC<*byte-stream*>
: specifies to base64-encode a certificate.

-bK<*byte-stream*>
: specifies to base64-encode a private-key.

-fC<*filename*>
: specifies a certificate file name.

-fK<*filename*>
: specifies a private-key file name.

Command Example

```
importkey HG012 FCNEDI2 E -fCpki/HG012.cer -fKpki/HG012.prv
```

insertpair

`insertpair` insert a trading relationship pair into the database.

Syntax

```
insertpair <from> <to> <url> <rcpt-url> <inbox> [<send-params>]
```

Required Parameters

The following parameters are required when executing the `insertpair` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

url
specifies the address of the service to be started.

rcpt-url
specifies the receipt address for outbound asynchronous receipts.

inbox
specifies the inbound storage location or URL address.

Optional Parameters

The following parameters are required when executing the `insertpair` command:

send-parameters
specifies default `send` parameters (optional; outbound only)

Command Example

```
insertpair myco yourco
http://www.yourco.com:8080/mailto:as2@myco.com inbox/yourco
```

insertroute

`insertroute` inserts a routing rule into a database.

Syntax

```
addroute <from> <to> <condition> <filter> <url>
```

Required Parameters

The following parameters are required when executing the `insertroute` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

condition
specifies the routing condition. The only currently supported routing condition is `ct=<n>` where *<n>* is a content type code.

filter
this parameter is reserved. The value must be `none`.

url
specifies the outbound location for the route. The url may use the following schemas:

`file://`

`mq://`

Command Example

```
insertroute you me ct=E none file://inbox/edi
```

insertserver

`insertserver` inserts a server-protocol into the database.

Syntax

```
insertserver <name> <group> <role> <url> <ctrlurl>
```

Required Parameters

The following parameters are required when executing the `insertserver` command:

name
specifies the user-defined name to identify a Commerce Suite process.

group
The peer-group number to which this service belongs.

role
specifies whether you want the Commerce Suite to assume an Administration, Router, or Transport role.

url
specifies the address on which the server will listen for data of a particular protocol.

ctrlurl
specifies the address on which the server will listen for control messages of a particular protocol.

Command Example

```
insertserver AT1 1 T http://127.0.0.1:4080 p2p://127.0.0.1:3502
```

listinbound

`listinbound` displays active inbound listeners.

Syntax

```
listinbound
```

Required Parameters

There are no required or optional parameters necessary when executing the `listinbound` command.

Command Example

```
listinbound
```

listkeys

`listkeys` displays active public-key pairs.

Syntax

```
listkeys [<from>|*[<to>|*[<max>]]]
```

Optional Parameters

The following parameters are optional when executing the `listkeys` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

max
specifies the maximum number of rows to display.

Command Examples

```
listkeys

listkeys sender

listkeys * receiver

listkeys sender * 200
```

listpairs

`listpairs` displays active trading partner relationships.

Syntax

```
listpairs [<from>|*[<to>|*]]
```

Optional Parameters

The following parameters are optional when executing the `listpairs` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

Command Examples

```
listpairs

listpairs isoft *

listpairs * isoft
```

listpeers

`listpeers` displays the active router agents.

Syntax

```
listpeers
```

Required Parameters

There are no required or optional parameters necessary when executing the `listpeers` command.

Command Example

```
listpeers
```

listroute

`listroute` displays a list of routing rules.

Syntax

```
listroute [<from>|*[<to>|*]]
```

Optional Parameters

The following parameters are optional when executing the `listroute` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

Command Examples

```
listpairs

listpairs isoft *

listpairs * isoft
```

listservers

`listservers` displays all active P2P protocol servers.

Syntax

```
listservers
```

Required Parameters

There are no required or optional parameters necessary when executing the `listservers` command.

Command Example

```
listservers
```

r

`r` repeats a command from the command history. The `r` command reissues a command stored in the command history queue. The Commerce Suite may be configured to store the last *N* commands, where *N* is the number provided as the parameter in the `set -ic<count>` command.

The typical sequence of events that must occur to enable the use of the `r` command follows:

1. The `p2pagent.cfg` file includes a `set -ic<N>` command to enable the retention of *N* commands. For example, the command `set -ic20` instructs the Commerce Suite to remember the previous 20 commands entered. As new commands are entered at the console, Commerce Suite will always retain the most recent N commands entered.
2. You may enter the history command to display the commands currently stored in the Commerce Suite command-history queue. When displayed, each command is assigned a number. So, for example, a history command may produce a list similar to the following:

   ```
   2 - "status"
   1 - "set -tr1200s"
   ```

3. Entering the `r1` command would instruct the Commerce Suite to re-issue the `set -tr1200s` command. The `r2` command would instruct the Commerce Suite to re-issue the `status` command.

Syntax

```
r [<nbr>]
```

Required Parameters

There are no required or optional parameters necessary when executing the `r` command.

Command Example

```
r

r3
```

receivefile

`receivefile` processes a file received as a received message. The `receivefile` command instructs the Commerce Suite to process a given filename as though it has just been received across the Internet. The `receivefile` command is useful if file decryption and authentication mechanisms need to be applied to a file that may have been received through an alternative transfer mechanism, or if the file needs to be reprocessed manually.

Syntax

```
receivefile <filename>
```

Required Parameters

The following parameters are required when executing the `receivefile` command:

filename
specifies the complete or relative path and filename to process.

Command Example

```
receivefile edix12.dat
receivefile /inbound/data/4589012.dat
```

remotekey

`remotekey` replicates a public-key pair to a remote host.

Syntax

`remotekey <from> <to> <usage>`

Required Parameters

The following parameters are required when executing the `remotekey` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

usage = A | B | D | E | H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

A
specifies to sign only

B
specifies to encrypt only

D
specifies to decrypt only.

E
specifies to sign and decrypt.

H
specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

Example

```
remotekey sender receiver J
```

remotepair

`remotepair` replicates a public-key pair to a remote host.

Syntax

```
remotepair <agentname> <from> <to> <URL> <rcpt-url> <notify>
<inbox> <params>
```

Required Parameters

The following parameters are required when executing the `remotepair` command:

agentname
specifies the agent receiving the `remotepair` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

url
specifies the destination address used when sending data to the trading partner.

rcpt-url
specifies the address to which asynchronous receipts should be sent.

notify
a symbolic name representing this relationship. This name appears on the disposition-notification-to header in an AS2 message.

inbox

specifies an inbox location in which received messages should be stored.

Optional Parameters

The following parameters are optional when executing the `remotepair` command:

Example

```
remotepair sender receiver http://127.0.0.1:4080
p2p://127.0.0.1:3502 mailto:as2@myco.com inbox/yourco
```

remoteroute

`remoteroute` forwards a route definition to an agent.

Syntax

```
remoteroute <agentname> <from> <to> <condition> <filter> <url>
```

Required Parameters

The following parameters are required when executing the `remoteroute` command.

agentname
specifies the agent receiving the `remoteroute` command.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

condition
specifies the routing condition. The only currently supported routing condition is `ct=<n>` where `<n>` is a content type code.

filter
this parameter is reserved. The value must be `none`.

url
specifies the outbound location for the route. The url may use the following schemas:

`file://`

`mq://`

Command Example

```
remoteroute agent1 me you ct=E none mq://QM_polaris/edi
```

remotesend

`remotesend` forwards a send command to another agent. The agent sending the command must have one or more servers defined.

Syntax

```
remotesend  <agentname> <protocol> <from> <to> [<option>] [...]
```

Required Parameters

The following parameters are required when executing the `remotesend` command:

agentname
specifies the agent receiving the `send` command.

protocol= HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

Optional Parameters

The following parameters are optional when executing the `remotesend` command:

option
specifies one or more `send` command options delimited by one or more spaces. The options parameter must at least specify the

source file to be sent. Each option in the `option` parameter must begin with a single dash and one or more characters identifying the option. Refer to the `send` command reference page for more information.

remoteserver

`remoteserver` starts a server-protocol on a remote host.

Syntax

```
remoteserver <name> <listener-url>
```

Required Parameters

The following parameters are required when executing the `remoteserver` command:

name
specifies the name of the remote server.

listener-url
specifies the address on which the server will listen for incoming messages.

Example

```
remoteserver AT1 http://127.0.0.1:4080
```

removekey

`removekey` removes a public-key pair from memory.

Syntax

```
removekey <from> <to> <usage>
```

Required Parameters

The following parameters are required when executing the `removekey` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

usage = A | B | D | E | H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

A
specifies to sign only

B
specifies to encrypt only

D
specifies to decrypt only.

E
specifies to sign and decrypt.

H
specifies to verify signatures only.

J
: specifies to encrypt and verify signatures

O
: specifies to sign, encrypt, decrypt, and verify signatures.

Example

```
removekey sender receiver E
```

removepair

`removepair` removes a trading partner relationship from memory.

Syntax

```
removepair <from> <to> <protocol>
```

Required Parameters

The following parameters are required when executing the `removepair` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

protocol = HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

Example

```
removepair sender receiver http
```

removeroute

`removeroute` removes a routing rule from memory.

Syntax

```
removeroute <from> <to> <condition>
```

Required Parameters

The following parameters are required when executing the `removeroute` command:

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

condition
specifies the routing condition.

Example

```
removeroute sender receiver ct=E
```

removeserver

`removeserver` removes a server-protocol from memory.

Syntax

```
removeserver <name>
```

Required Parameters

The following parameters are required when executing the `removeserver` command.

name
specifies the server name

Example

```
removeserver AT2
```

replicatekeys

`replicatekeys` generates a `importkey` command for each server defined for the current agent and sends all PKI definitions to all defined administrative agents

Syntax

```
replicatekeys
```

Required Parameters

There are no required or optional parameters necessary when executing the `replicatekeys` command.

Example

```
replicatekeys
```

replicatepairs

`replicatepairs` generates a `remotepair` command for each server defined for the current agent and sends all relationship definitions to all defined administrative agents.

Syntax

```
replicatepairs
```

Required Parameters

There are no required or optional parameters necessary when executing the `replicatepairs` command.

Example

```
replicatepairs
```

`replicatepairs` generates a `remoteserver` command for each server defined for the current agent and sends start commands to all the defined servers to start their listeners.

replicateroutes

`replicateroutes` generates an `addroutes` command for each server defined for the current agent and sends all route definitions to all defined administrative agents.

Syntax

```
replicateroutes
```

Required Parameters

There are no required or optional parameters necessary when executing the `replicateroutes` command.

Example

```
replicateroutes
```

replicateservers

`replicateservers` generates a `remoteserver` command for each server defined for the current agent and sends `start` commands to all the defined servers to start their listeners.

Syntax

```
replicateservers
```

Required Parameters

There are no required or optional parameters necessary when executing the `replicateservers` command.

Example

```
replicateservers
```

requestcert

requestcert generates a PKS10 request for a certificate.

Syntax

```
requestcert <from> <to> <usage> [<option>] [...]
```

Required Parameters

protocol= HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

usage = A | B | D | E | H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

E
specifies to sign and decrypt.

J
specifies to encrypt and verify signatures

Optional Parameters

The following parameters are optional when executing the `requestcert` command:

-form=DER| PEM

specifies the format type of the certificate request.

`DER` specifies a binary file.

`PEM` specifies privacy enhanced mail.

-type=NEW | RENEW

-f

specifies the file input.

send

`send` initiates an outbound transmission of data. This command may be issued from the console or by storing a work order in the database.

Syntax

```
send <protocol> <from> <to> [<option>] [...]
```

send Command Quick Reference Table

Table 1-2 presents an overview of the `set` command parameters:

Table 1-2 send Command Parameters

Parameter	Task
`protocol`	Specifies the communication protocol.
`from`	Specifies the trading partner sending the data.
`to`	Specifies the trading partner receiving the data.
`-a<address>`	Specifies an alternate URI address.
`-c<name>`	Specifies one of the following content-type MIME headers: (E=edi-x.12, F=edifact, M= MDN, T=plain text, X=XML).
`-de`	Specifies that the data-access type is external.
`-dh<hostname>`	Specifies the data-source hostname.
`-dp<password>`	Specifies the data-source password.

Parameter	Task
-ds<*params*>	Specifies additional external mailbox parameters, mailbox IDs, and batch IDs.
-du<*user*>	Specifies the data-access user name.
-e	Specifies to encrypt the message using Triple DES.
-e2	Specifies to encrypt the message using RC2.
-eB	Specifies to Base64-encode the Triple DES encrypted data.
-fE<*ext*>	Specifies to append the given file extension after sending the data.
-fN<*file-name*>	Specifies the file name to send.
-fP<*path*>	Specifies the file path.
-fS<*spec*>	Specifies to search for the given file specification and send the first match.
-hF	Specifies to fold headers.
-hno, -hno-	Specifies to enable/disable the preservation of outbound file names.
-hQ	Specifies to quote EDI names.
-i<*identifier*>	Specifies a sender message-identifier.
-n<*count*>	Specifies the maximum number of send attempts.
-oZ	Specifies to ZLIB compress the data payload.
-pB	Specifies to Base64-encode the message payload.

Parameter	Task
`-r`	Specifies to request an unsigned MDN receipt.
`-r1`	Specifies to request a SHA-1 signed MDN receipt.
`-r5`	Specifies to request a MD5 signed MDN receipt.
`-rA`	Specifies to request an asynchronous receipt.
`-r1A`	Specifies to request a SHA-1 asynchronous signed receipt.
`-r5A`	Specifies to request a MD5 asynchronous signed receipt.
`-s`	Specifies to include a SHA-1 digital signature.
`-sC`	Specifies to include a certificate with a signature.
`-s5, -sC5`	Specifies to include a MD5 signature.
`-sB`	Specifies to Base64-encode and include a SHA-1 signature.
`-sB5`	Specifies to Base64-encode and include a MD5 signature.
`-tS<`*time*`>`	Specifies the time to send the message.
`-tE<`*time*`>`	Specifies to send the message until the specified time.
`-tC<`*interval*`>`	Specifies the seconds between send attempts.
`-u<`*subject*`>`	Specifies the sender message subject.
`-x`	Specifies to delete the file after sending.

Required Parameters

The following parameters are required when executing the `send` command:

protocol= HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

Optional Parameters

The following parameters are optional when executing the `send` command:

option
specifies one or more options delimited by one or more spaces. The options parameter must at least specify the source file to be sent. Each option in the `option` parameter must begin with a single dash and one or more characters identifying the option.

-a<*address*>
specifies to override the *`to-url`* option in the trading partner relationship configuration using the IP address and port specified. This option allows you to use a different destination address for the send rather than the address specified in the relationship record.

-cE
sets the content type MIME header to *application/edi-x.12*.

-cF
sets the content type MIME header to *application/edifact*. The default is `octet-stream`.

-cM
sets the content type MIME header for a message disposition notification. The default is `octet-stream`.

-cT
sets the content type MIME header for plain text. The default is `octet-stream`.

-cX
sets the content type MIME header for XML. The default is `octet-stream`.

-de
specifies the data access-type is EXTERNAL.

-dh<*hostname*>
specifies the data-source host name.

-dp<*password*>
specifies the data-source password.

-ds<*params*>
specifies additional external mailbox parameters, mailbox IDs, and batch IDs. Use this parameter only when accessing data from an external mailbox system.

-du<*user*>
specifies the data-access user name.

-e
specifies to encrypt the message using the default block cipher: DES_EDE_CBC (also known as Triple DES).

-e2
specifies to encrypt the message using RC2.

-eB
specifies to Base64-encode the Triple-DES encrypted data.

-fE<*extension*>
specifies to append the given extension after sending the data.

-fN<*file-name*>
specifies to send the file matching the given file name. This filename cannot contain any spaces.

-fP<*path*>
specifies a path to a specific directory. The directory path cannot contain any spaces.

-fS<file-*spec*>
specifies to search for files matching the given file-spec and send the first match.

-hF
specifies to fold (wrap) MIME headers. (AS1 only)

-hno, -hno-
specifies to enable/disable the preserving of outbound file names.

-hQ
specifies to quote EDI names in MIME headers.

-i<*identifier*>
specifies a user-supplied message-identifier.

-n<*count*>
specifies the maximum number of send attempts.

-oZ
specifies to ZLIB compress the data before sending.

-pB
specifies to Base64-encode the message payload.

-r
requests an unsigned MDN receipt.

-r1
requests a SHA-1 hashed-signed receipt.

-r5
requests a MD5 hashed-signed receipt.

-rA
requests an asynchronous receipt.

-r1A
requests an SHA1-hashed asynchronous signed receipt.

-r5A
requests an MD5-hashed asynchronous signed receipt.

-s
specifies to sign the message using the default SHA-1 hash algorithm.

-sC
specifies to include a certificate with the digital signature.

-s5, -sC5
specifies to sign the message using the MD5 hash algorithm.

-sB
specifies to SHA-1 sign and Base64-encode the signature.

-s5B
specifies to MD5 sign and Base64-encode the signature.

-tS<date-*time*>
specifies to send a message on or after the entered time. Format: (*YYYYMMDDHHMMSS*)Year, Month, Day, Hour, Minute, Second.

-tE<*date-time*>
specifies to send the message until the entered time. Format: (*YYYYMMDDHHMMSS*) Year, Month, Day, Hour, Minute, Second.

-tC<*interval*>
specifies the interval, in seconds, between send attempts.

-u<*subject*>
specifies the sender's message subject.

-x
specifies to delete the local file after sending it.

Expanded Parameters

The `send` command parameters have been expanded to recognize MQSeries as a supported external mailboxing system the same way as Connect:Enterprise is recognized.

When sending from a MQSeries mailbox (or queue), no *`filename`* or *`path`* variables are defined. They are replaced by the `mailbox` keyword placeholder. These placeholders are reserved for mailbox identifiers.

The following code examples show the difference between the traditional `send` command syntax, and the syntax used to interface with a MQSeries mailbox.

Traditional Syntax

```
send <protocol> <from> <to> [<option>] [...]
```

Example

```
send http sndr rcvr -fNoutbox\edix12.txt -cE -oZ -sC
```

MQSeries Syntax

```
send <protocol> <mailbox> <mailbox> [<options>] [...]
```

Example

```
send http MAILBOX1 MAILBOX2 -de -ds MAILBOXID=isoft
```

Note that in this MQSeries example that both the sender and receiver names are set to the keyword `MAILBOX`. Also note two data parameters, `-de` and `-ds`. These indicate, respectively, that data is to be sent from an external data store and that the mailbox (queue) to be accessed is `isoft`. The batch identifiers not presently used with the Commerce Suite/MQ interface and may be set to zero.

When Commerce Suite/MQ processes a send command, it will resolve the mailbox name (or queue name) to the relationship established in the `addpair` command. From this command, Commerce Suite/MQ determines the destination address and default send parameters. Additional send parameters can be appended to the send command itself, enhancing or overriding the default parameters from the `addpair` command.

One-Time Sends

The `send` command supports one-time sends to the operating system file system as well as indirect sends to MQSeries queues using a mailbox identifier.

The following code provides an example of an indirect send using mailbox identifiers.

```
send http MAILBOX1 MAILBOX2 -de -ds MAILBOXID=isoft
```

Persistent Sends

The `send` command also supports persistent sends that will poll a a mailbox (or queue) at regular intervals listening for messages. When messages are encountered, Commerce Suite/MQ will send the first message in the mailbox (or queue).

The following code provides an example of an indirect persistent `send` using mailbox identifiers.

```
send http MAILBOX MAILBOX -de -dsMAILBOXID=user1_ibm -tC60s
-tE20030101000000
```

In this code example, the `send` command will repeat from mailbox name `user1` every 60 seconds until January 1, 2003.

Command Example

```
send http sndr rcvr -fNoutbox\edix12.txt -cE -oZ -sC -e -r1

send http sndr rcvr -fPout -fSout -fE.sent -tC60s -tE20040101000000
```

sendcert

`sendcert` sends an X.509 certificate to a trading partner.

Syntax

```
sendcert <protocol> <from> <to> <usage>
```

Required Parameters

The following parameters are required when executing the `sendcert` command:

protocol = HTTP | HTTPS | SMTP
specifies either the HTTP, HTTPS, or SMTP protocol.

from
specifies the trading partner name sending the data.

to
specifies the trading partner name receiving the data.

usage = A | B | D | E | H | J | O
specifies the uses for the key-material in the context of the Commerce Suite application.

A
specifies to sign only.

B
specifies to encrypt only.

D
specifies to decrypt only.

E
specifies to sign and decrypt.

H
: specifies to verify signatures only.

J
: specifies to encrypt and verify signatures.

O
: specifies to sign, encrypt, decrypt, and verify signatures.

Command Example

```
sendcert http myco yourco E
```

set

`set` sets the value of a program variable. Program variables determine the behavior of many program functions, including directory locations, timeouts, and messaging levels.

`set` parameters are of the following two types:

- **Value-Parameters:** This type of parameter carries an assigned value that may have a default value assigned at program start-up. The values assigned to value-parameters are given immediately after the parameter name
- **Switch-Parameters:** This type of parameter is toggled on or off and has a default state. Switch-parameters are set to the on state by specifying the switch name alone. Switch-parameters are set to the off state by specifying the switch name followed immediately by a dash (-).

Syntax

```
set <parameter> [...]
```

set Command Quick Reference Table

Table 1-3 presents an overview of the `set` command parameters:

Table 1-3 set Command Parameters

Parameter	Task
-ba<*ip-address*>	Specifies a mailbox IP address.
-be<*mailbox-system-name*>	Specifies the symbolic name of the external mailbox system.

Parameter	Task
-bh<*hostname*>	Specifies a mailbox IP address.
-bp<*ip-port*>	Specifies a mailbox IP port number.
-bu<*username*>	Specifies the mailbox userid.
-bw<*password*>	Specifies the mailbox password.
-ca<*ip-address*>	Specifies the administrative agent control address.
-cn<*name*>	Specifies the control agent name.
-cp<*ip-port*>	Specifies the administrative agent control port.
-dd<*drivername*>	Specifies the ODBC driver name.
-df, -df-	Specifies to enable/disable the immediate replication of configuration updates to the database.
-dh<*hostaname*>, -dh-	Specifies to enable/disable the database hostname.
-dn<*database*>, -dn-	Specifies to enable/disable the database name.
-du<*user-name*>, -du-	Specifies to enable/disable the database username.
-dw<*password*>, -dw-	Specifies to enable/disable the database password.
-ef, -ef-	Specifies to enable/disable the writing of files to an error path.
-ep<*path*>, -ep-	Specifies to enable/disable the error file path.
-f-	Specifies to bypass the default `p2pagent.cfg` configuration file upon startup.
-fi<*ext*>, -fi-	Specifies to enable/disable the default inbound file extension.

Parameter	Task
`-fo<ext>, -fo-`	Specifies to enable/disable the default outbound file extension.
`-fs, -fs-`	Specifies to enable/disable the use of short file names.
`-ft, -ft-`	Specifies to enable/disable the saving of persistent temporary files.
`-ga, -ga-`	Specifies to enable/disable the server as the primary administration agent.
`-gb, -gb-`	Specifies to enable/disable the server as the backup administration agent.
`-gp<group-number>, -gp-`	Specifies to enable/disable the peer group number.
`-gr, -gr-`	Specifies to enable/disable the server as a router agent.
`-gt, -gt-`	Specifies to enable/disable the server as a transport agent.
`-hni, -hni-`	Specifies to enable/disable the preserving of inbound file names.
`-hno, -hno-`	Specifies to enable/disable the preserving of outbound file names.
`-hu, -hu-`	Specifies to enable/disable forcing upper-case EDI names.
`-ic<count>, -ic-`	Specifies to enable/disable the storing of console commands for recall by the `r` command.
`-lc, -lc-`	Specifies to enable/disable the closing of log-files after each write.
`-lf, -lf-`	Specifies to enable/disable the writing of log information to a daily file.
`-lp<path>, -lp-`	Specifies to enable/disable the log file path.

Parameter	Task
-m<*messaging-level*>	Specifies one of the following message levels determining the quality of data written to a daily log file: (0=Silent, 1=Errors, 2=Warnings, 3=Informational, 4=Summary, 5=Detail, 6=Debug).
-nd, -nd-	Specifies to enable/disable the writing of notice records to a database.
-nf, -nf-	Specifies to enable/disable the writing of notice records to a file.
-np<*path*>, -np-	Specifies to enable/disable the notice file path.
-ns<*n*>, -ns-	Specifies to enable/disable the maximum number of send attempts.
-oc<*code*>	Specifies the work order class code for pending work orders.
-of, -of-	Specifies to enable/disable the fetching or work order files.
-on<*count*>	Specifies the number of work orders to fetch.
-op<*path*>, -op-	Specifies to enable/disable the work order path.
-oq<*code*>	Specifies the work order class code for queued work orders.
-os<*spec*>, -os-	Specifies to enable/disable the file specification criteria when searching for work orders.
-pd, -pd-	Specifies to enable/disable the storing of PKI data in a database.
-pp<*path*>, -pp-	Specifies to enable/disable the PKI path.
-rp<*path*>, -rp-	Specifies to enable/disable the asynchronous receipt path.

Parameter	Task
-sh<*hostname*>, -sh-	Specifies to enable/disable the SMTP host name.
-su<*username*>, -su-	Specifies to enable/disable the SMTP user name.
-sw<*password*>, -sw-	Specifies to enable/disable the SMTP password.
-tb<*timeout*{[*s*|*ms*]}>	Specifies the beacon interval.
-tc<*timeout*{[*s*|*ms*]}>	Specifies the connection attempt timeout.
-te<*timeout*{[*s*|*ms*]}>	Specifies the retry send interval.
-to<*timeout*{[*s*|*ms*]}>	Specifies the work order search interval.
-tp<*timeout*{[*s*|*ms*]}>	Specifies the thread-shutdown interval.
-tr<*timeout*{[*s*|*ms*]}>	Specifies the first-receive timeout.
-ts<*timeout*{[*s*|*ms*]}>	Specifies the first-send timeout.
-tu<*timeout*{[*s*|*ms*]}>	Specifies the next-receive timeout.
-tv<*timeout*{[*s*|*ms*]}>	Specifies the next-send timeout.
-ua<*address*>	Specifies the remote user-interface address.
-up<*port*>	Specifies the remote user-interface port.
-yf, -yf-	Specifies to enable/disable the recycling of output for retry.
-yp<*path*>, -yp-	Specifies to enable/disable the recycle path.

Required Parameters

The following parameters are required when executing the `set` command:

-ba<*IP-address*>
specifies the dot-notated Internet Protocol (IP) address of the mailbox server. If the Commerce Suite is configured to connect to a TCP/IP-based mailboxing system.

Example: `set -ba127.0.0.1`

-be<mailbox-system-name>
specifies the symbolic name of the external mailbox system, if the Commerce Suite is configured to connect to an external mailbox system.

Example: `set -beISOFTMBX`

For Connect:Enterprise Interfacing - This parameter now supports a Connect:Enterprise scheme identifier, `ce://`, that denotes a Connect:Enterprise mailbox symbolic name.

Syntax: `set -be<symbolicname>`

Example: `set -be symbolicname`

-bh<*hostname*>
specifies the mailbox IP address and port number. This parameter may be specified as an alternate to using the `-ba` and `-bp` parameters, if the mailbox server's IP address and port can be resolved using the Domain Name Service (DNS).

Example: `set -bhMBXSVR08:8000`

For Connect:Enterprise Interfacing - This parameter now supports a Connect:Enterprise scheme identifier, `ce://`, that denotes a Connect:Enterprise mailbox hostname and port.

Syntax: `set -bh[<scheme>://]<hostname>:<port>`

Example: `set -bhce://myhost:8080`

For MQSeries Interfacing - The external mailbox hostname parameter can be edited to include an MQSeries Queue Manager hostname within the parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that data will be written to the following path identifying a Queue Manager and Queue Name.

Syntax: `set -bh[<scheme>://]<hostname>:<port>`

Example: `set -bhmq://QM_polaris:8080`

-bp<*IP-port*>

specifies a mailbox IP port number if Commerce Suite is configured to connect to a TCP/IP-based maiboxing system.

Example: `set -bp4080`

-bu<*username*>

specifies a mailbox userid. This parameter must be specified if the mailbox server requires user name authentication.

Example: `set -Admin`

For Connect:Enterprise Interfacing - This parameter now supports a Connect:Enterprise scheme identifier, `ce://`, that denotes a Connect:Enterprise mailbox user name.

Syntax: `set -bu<login>`

Example: `set -bu<login>`

-bw<*password*>

specifies a mailbox password. This parameter must be specified if the mailbox server requires password authentication.

Example: `set -bwHX9WJ@4B`

For Connect:Enterprise Interfacing - This parameter now supports a Connect:Enterprise scheme identifier, `ce://`, that denotes a Connect:Enterprise mailbox password.

Syntax: `set -bw<password>`

Example: `set -bw<password>`

-ca<*IP-address*>

specifies the administrative agent control address. Control messages may be sent to a transport or router agent to change run-time settings.

Example: `set -ca127.0.0.1`

-cn<*name*>

specifies the control agent name. The name specified in this parameter is included in notice records and other audit information. Supplying a recognizable name using this parameter can provide a useful association between audit information and Commerce Suite instances.

Example: `set -cn<TRANSPORT1>`

-cp<IP-*port*>

specifies the administrative agent control port. Control messages may be sent to a transport or router agent to change run-time settings.

Example: `set -cp3501`

-dd<*ODBC-driver-name*>

specifies the name of the ODBC driver Manager to be used when the Commerce Suite must access data from a database using ODBC API. If spaces appear in the name of the Driver Manager, these must be replaced with the TILDE (~) as shown in the example below.

Example: `set -ddSQL~Server`

-df, -df-

specifies to enable/disable the immediate replication of configuration updates to the database. Setting the `-df` switch

will cause subsequent `add` and `remove` commands to replicate their effects to the database.

Example: `set -df`

-dh<*hostname*>, -dh-
specifies to enable/disable the database hostname.

Note: Do not use the `-dh` parameter to reference an external mailbox. Refer to the `-dn` parameter for referencing external mailbox systems.

Example: `set -dhDBSRVR04`

-dn<*database*>, -dn-
specifies to enable/disable the database name.

Example: `set -dnp2pagent`

-du<*login*>, -du-
specifies to enable/disable the database username.

Example: `set -dup2pdbuser`

-dw<*password*>, -dw-
specifies to enable/disable the database login password.

Example: `set -dwHB4V9J$L`

-ef, -ef-
specifies to enable/disable the writing of files to an error path in the event a data transfer cannot be completed. Use this parameter in conjunction with the `-ep` parameter described later.

Example: `set -ef`

-ep<*path*>, -ep-
specifies to enable/disable the error file path to be used to store inbound and outbound data packages that cannot be completely delivered.

Example: `set -eperror`

For MQSeries Interfacing - The error-path parameter can be set to include a URL, allowing for MQSeries notice queues to be defined within the notice path parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that the following path identifies a Queue Manager and Queue Name into data that will be written.

Syntax: `set -ep<`*`url|path`*`>`

Example: `set -epmq://QM_polaris/notices -ef`

-f-

specifies to bypass the default configuration file.

If this option is enabled, the Commerce Suite application bypasses the default configuration file, `p2pagent.cfg`, upon startup.

Example: `-f-`

-fi<*ext*>, -fi-

specifies to enable/disable the default inbound file extension.

Example: `set -fi<.doc>`

-fo<*ext*>, -fo-

specifies to enable/disable the default outbound file extension.

Example: `set -fo<.doc>`

-fs, -fs-

specifies to enable/disable the use of short file names consisting of a date-time stamp and unique message number. By default, incoming messages are stored using a long-name convention that include the names of the sending and receiving partners.

Example: `set -fs`

-ft, -ft-

specifies to enable/disable the saving of persisten temporary files typically used for troubleshooting purposes. During the course of message processing, Commerce Suite generates temporary files to hold messages at various stages of preparation. Normally, these files are not preserved.

Example: `set -ft`

-ga, -ga-
specifies to enable/disable the server as the primary administration agent for a Commerce Suite clustered configuration.

This option enables those services that provide administrative functions, such as polling the database for work orders and configuring and polling administered Transport and Router agents.

Example: `set -ga`

-gb, -gb-
specifies to enable/disable the server as the backup agent for a Commerce Suite clustered configuration.

Example: `set -gb`

-gp<*group-number*>, -gp-
specifies to enable/disable the Peer Group number to which the Commerce Suite belongs.

For Transport Agents, the group number indicates the group of Agents serviced by a Router Agent or Administrative Agent.

For Router Agents, the number indicates the group to be included in the load-balancing Transport Group.

For Administrative Agents, the number indicates the agents to be included in the outbound load-balancing Transport Group. The specified number must be in the range: 1 through 9. Specifying zero as the Peer Group specifies that the Commerce Suite is not to participate in any group.

Example: `set -gp5`

-gr, -gr-
specifies to enable/disable the server to act as a load-balancing router for incoming data transfers. Setting this switch enables the Commerce Suite to collect UDP broadcast messages from Transport Agents for it's group.

This option enables those services that provide routing functions, such as the collection of UDP packets that identify active Transport agents on the local network segment.

Example: `set -gr`

-gt, -gt-

specifies to enable/disable the server to act as a Transport Agent. Setting this switch enables the Commerce Suite to collect configuration data from an Administrative Agent, or directly from a database, and processing message transfers.

Example: `set -gt`

-hni, -hni-

specifies to enable/disable the preserving of inbound file names. This parameter must be used in conjunction with the `-hno` parameter on the outbound side of the data transfer in order for the file name to be preserved. If a file of the same name exists on the inbound system, it will be overwritten by the new file with the same name.

Example: `set -hni`

-hno, -hno-

specifies to enable/disable the preserving of outbound file names. This parameter must be used in conjunction with the `-hni` parameter on the inbound side of the data transfer in order for the file name to be preserved. If a file of the same name exists on the outbound system, it will be overwritten by the new file with the same name.

Example: `set -hno`

-hu, -hu-

specifies to enable/disable forcing upper-case names when composing MIME headers. Enabling this switch causes trading partner identifiers to be case-insensitive.

Example: `set -hu`

-ic<*count*>, -ic-

specifies to enable/disable the storing of console commands for recall using the `r` command. By default, the Commerce Suite

does not store console commands for recall. Setting this parameter causes the Commerce Suite to allocate storage to save commands that can be displayed using the `history` command or recalled using the `r` command.

Example: `set -ic20`

-lc, -lc-

specifies to enable/disable the closing of daily log files (.log) after each write (default).

-lf, -lf-

specifies to enable/disable the writing of log information to a daily file.

This option enables the writing of trace-log records to a file in the current application directory. Trace-log records indicate current application operation or the completion of an application function. Trace-log files can be used to record the operation of the application for audit purposes. The names of the trace-log files are determined by the Commerce Suite application. The file names begin with a P2P prefix and are followed by a series of numerals indicating the current date and a .log extension. The file-system directory location in which to store the files is specified with the `-lp<`*`path`*`>` parameter.

Example: `set -lf`

-lp<*path*>, -lp-

specifies to enable/disable the log file path.

Example: `set -lp/opt/p2pagent/logs`

-m<*messaging-level*>

specifies the message level for the daily log files. The message level determines the quantity of data to be written to the log files. The message level is specified with a number between 0 and 6, inclusive. The message levels are described as:

Level 0: Silent. No messages are written to the log.

Level 1: Errors. Only error messages are written to the log.

Level 2: Warnings. Errors and warnings are written to the log.

Level 3: Info. Errors, warnings, and general information messages are written to the log.

Level 4: Summary. Level 3 messages plus function completion messages are written to the log.

Level 5: Detail. Level 4 messages plus function initiation messages are written to the log.

Level 6: Debug. Level 5 messages plus variable content messages are written to the log.

Example: `-m3`

-nd, -nd-

specifies to enable/disable the writing of notice records to the database. notice records are written to summarize the status of a message transfer attempt. Notice records include statistical information regarding the transfer attempt.

Example: `set -nd`

-nf, -nf-

specifies to enable/disable the writing of notice records to a file. Notice records are written to summarize the status of a message transfer attempt. Notice records include statistical information regarding the transfer work order attempt.

Example: `set -nf`

-np<*path*>, -np-

specifies to enable/disable the file-system location where Commerce Suite stores notice files. The `-np` command may be used to specify the current working directory.

Example: `set -np/opt/p2pagent/notices`

For MQSeries Interfacing - The notice-path parameter can be set to include a URL, allowing for MQSeries notice queues to be defined within the notice path parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that the following path identifies a Queue Manager and Queue Name into data that will be written.

Syntax: `set -np<url|path>`

Example: `set -npmq://QM_polaris/notices -nf`

The `-nf` switch enables/disables notice storage to the notice *path* for MQSeries interfacing.

-ns<*number*>, -ns-
specifies to enable/disable the maximum number of send attempts.

Example: `set -ns20`

-oc
specifies the work order class code for pending work orders. `P` is the default.

Example: `set -ocP`

You do not need to issue this parameter unless you want to use a value other than the default.

-of, -of-
specifies to enable/disable the fetching of work order files.

Example: `set -of`

-on<*count*>
specifies the number of work orders to fetch.

Example: `set -on5`

-op<*path*>, -op-
specifies to enable/disable the file-system location where Commerce Suite searches for work orders. Work Orders may be stored in files or in the database. When they are stored as files, the Commerce Suite searches the given directory for work order files. The `-op-` command may be used to specify the current working directory.

Example: `set -op/opt/p2pagent/workorders`

For MQSeries Interfacing - The work order path parameter can be edited to include a URL, allowing for MQSeries work order

queues to be defined within the work order path parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that the following path identifies a Queue Manager and Queue Name into data that will be written.

Syntax: `set -op<url|path>`

Example: `set -opmq://QM_polaris/workorders`

If no scheme is defined within the notice path parameter, then the value is interpreted as a relative path.

-oq

specifies the work order class code for queued work orders. `Q` is the default.

Example: set -oqQ

You do not need to issue this parameter unles you want to use a value other than the default.

-os<spec>, -os-

specifies to enable/disable the file specification criteria when searching for work orders. Files that end with the given specification are considered to be work order files.

Example: `set -oswo`

-pd, -pd-

specifies to enable/disable the storing of PKI data in a database.

Example: `set -pd`

-pp<*path*>, -pp-

specifies to enable/disable the PKI directory path.

Example: `set -pppki`

-rp<*path*>, -rp-

specifies to enable/disable the asynchronous receipt path.

Example: `set -rp/opt/p2pagent/receipts`

For Connect:Enterprise Interfacing - The receipt path parameter can be edited to include a URL, allowing for Connect:Enterprise mailboxes to be defined within the receipt path parameter syntax.

Syntax: `set -rpce://<mailboxname>:<port>`

Example: `set rpce://mymailbox:8080`

For MQSeries Interfacing - The receipt path parameter can be edited to include a URL, allowing for MQSeries work order queues to be defined within the receipt path parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that the following path identifies a Queue Manager and Queue Name into data that will be written.

Syntax: `set -rp<url><path>`

Example: `set -rpmq://QM_polaris/receipts</command>`

If no scheme is defined within the notice path parameter, then the value is interpreted as a relative path.

-sh<*hostname*>, -sh-
specifies to enable/disable the SMTP host name used by Commerce Suite to deliver asynchronous receipts.

Example: `set -shmailsrvr`

-su<*username*>, -su-
specifies to enable/disable the SMTP host name.

Example: `set -sup2pagent@company.com`

-sw<*password*>, -sw-
specifies to enable/disable the SMTP password.

Example: `set -sw7YQG5#LB`

-tb<*timeout*>*[s | ms]*
specifies the interval, in seconds or milliseconds, that the Commerce Suite should wait between sending UDP beacons. This parameter is only effective if the start beacon command has been issued and the Commerce Suite is actively listening on

one or more ports for incoming data or is accepting `remotesend` commands from an Administrative Agent.

Example: `set -tb60s`

-tc<*timeout*>[*s* | *ms*]
specifies the interval, in seconds or milliseconds, that the Commerce Suite will wait for a connection attempt to a remote host to complete.

Example: `set -tc300s`

-te<*timeout*>[*s* | *ms*]
specifies the interval, in seconds or milliseconds, that the Commerce Suite will wait before retrying a failed attempt to send data.

Example: `set -te300s`

-to<*timeout*>[s|ms]
specifies the interval, in seconds or milliseconds, that the Commerce Suite will wait between searches for work orders.

Example: `set -to300s`

-tp<*timeout*>[*-s* | *ms*]
specifies the interval, in seconds or milliseconds, that the Commerce Suite will wait before terminating a thread during application shutdown. This time period allows active services to complete inbound and outbound message deliveries before terminating the application.

Example: `set -tp300ms`

-tr<*timeout*>[s|ms]
specifies the first-receive timeout, in seconds or milliseconds, that the Commerce Suite will wait for the first byte of incoming data on an inbound connection before considering the transfer to have failed.

Example: `set -tr3000s`

-ts<*timeout*>[s|ms]
specifies the first-send timeout, in seconds or milliseconds, that the Commerce Suite will wait for the first packet of data to be

sent on an outbound connection before considering the transfer to have failed.

Example: `set -ts300s`

-tu<*timeout*>[s|ms]
specifies the next-receive timeout, in seconds or milliseconds, that the Commerce Suite will wait for follow-on packets of data (not the first packet) on an inbound connection before considering the transfer to have failed.

Example: `set -tu300s`

-tv<*timeout*>[s|ms]
specifies the next-send timeout, in seconds or milliseconds, that the Commerce Suite will wait for follow-on packets of data to be sent (not the first packet) on outbound connections before considering the transfer to have failed.

Example: `set -tv300ms`

-ua<IP-*address*>
specifies the remote user interface address. The user interface service must be started with the `start gui` command to activate the Commerce Suite to respond to user interface requests.

Example: `set -ua127.0.0.1`

-up<IP-*port*>
specifies the remote user interface port. The user interface service must be started with the `start gui` command to activate the Commerce Suite to respond to user interface requests.

Example: `set -up8080`

-yf, -yf-
specifies to enable/disable the use of the recycle location to the given path.

Example: `set -yf`

-yp<*path*>, -yp-
specifies to enable/disable the file system directory location that should be used by Commerce Suite as a Recycle output location.

Example: `set-yp/opt/p2pagent/recycle`

For MQSeries Interfacing - The recycle path parameter can be edited to include a URL, allowing for MQSeries work order queues to be defined within the receipt path parameter syntax.

This parameter now supports an MQSeries scheme identifier `mq://`, that denotes that the following path identifies a Queue Manager and Queue Name into data that will be written.

Syntax: `set -yp<path>`

Example: `set -ypmq://QM_polaris/recycle</command>`

Command Example

```
set -m6
set -ft
set -lc-
```

shutdown

`shutdown` terminates the application. When the shutdown command is entered, the main inbound and outbound processes are signalled to stop accepting incoming transactions and to stop initiating outbound transactions. Existing transactions are given up to THREAD-TIMEOUT to complete their current task. THREAD-TIMEOUT is configured using the `set -te<val>[s|ms]` command.

Syntax

```
shutdown
```

Required Parameters

There are no required or optional parameters necessary when executing the `shutdown` command.

Command Example

```
shutdown
```

start

`start` starts a program service. Program services include protocol listeners and other services that can be started and stopped during program execution. The program service may be one of several internal services, identified by keyword, or URL.

Syntax

```
start <service>
<service> = beacon | control | gui | router | <url>
```

Required Parameters

The following parameters are required when executing the `start` command:

url
specifies the address for the application you want to start.

Note: You must specify either a service name or url when executing the `start` command.

Command Example

```
start gui
start http://127.0.0.1:4080
```

startdatabse

`startdatabase` connects Commerce Suite to a database. before the `startdatabase` command can be effective, parameters must be setup to tell the Commerce Suite how to connect. For ODBC, the driver-manager, host, database-name, user, and password must be known, since Commerce Suite uses dsn-less connections. For ESQL, only the database, user, and password must be known.

Syntax

```
startdatabase
```

Required Parameters

There are no required or optional parameters necessary when executing the `startdatabase` command.

Command Example

ODBC Example

```
set -ddSQL~Server -dhWEBHUB2 -dnp2pagent -dusa -dwpassword
startdatabase
```

ESQL/C Example

```
set -dnp2pagent -dusa -dwpassword
startdatabase
```

status

`status` displays to the console a summary of current program variables that indicate program status, including the state of control flags and counters, directory settings, and timeout values.

Syntax

```
status
```

Required Parameters

There are no required or optional parameters necessary when executing the `status` command.

Example

```
status
```

stop

`stop` stops a program service. Program services include protocol listeners and other services that can be stopped and restarted during program execution. The program service may be one of several internal services, identified by keyword, or combination of schema, IP-address, and IP-port, referring to an active listener. The address and port parameters are optional. However, if port is specified, address must also be specified.

Syntax

```
stop <service> {<address> <port>}
<service> = beacon | control | gui | router
```

Optional Parameters

The following parameters are optional when executing the `stop` command:

protocol = HTTP | HTTPS | SMTP
specifies the communication protocol.

address
specifies the application address.

port
specifies the port associated with the application address.

Example

```
stop router

stop https
```

```
stop http 127.0.0.1 8080
```

stopdatabse

`stopdatabase` connects Commerce Suite to a database. before the `stopdatabase` command can be effective, parameters must be setup to tell the Commerce Suite how to connect. For ODBC, the driver-manager, host, database-name, user, and password must be known, since Commerce Suite uses dsn-less connections. For ESQL, only the database, user, and password must be known.

Syntax

```
stopdatabase
```

Required Parameters

There are no required or optional parameters necessary when executing the `stopdatabase` command.

Command Example

ODBC Example

```
set -ddSQL~Server -dhWEBHUB2 -dnp2pagent -dusa -dwpassword
stopdatabase
```

ESQL/C Example

```
set -dnp2pagent -dusa -dwpassword
stopdatabase
```

version

`version` generates a log-file message that contains the current application build number. The build number is a sequence of characters that indicate the date that the application was prepared. This information can be useful in determining which version of the application is running and which updates have been applied.

Syntax

```
version
```

Required Parameters

There are no required or optional parameters necessary when executing the `version` command.

Example

```
version
ok
2002.08.20 14:54:08.616    VERS OK Build 3.1.2002.8.5.1
```

Glossary

A

Agent

An instance of the Peer-to-Peer Version 3 (P2P Agent) application configured to provide services to a particular role, i.e. Administrator, Transport, or Router.

Administrator Agent

An instance of the P2P Agent application configured to provide administrative services including the remote configuration of Transport and Router Agents and access to centrally located configuration data.

Application Service

See Service.

AS1

A draft specification first published in the Internet Engineering Task Force (IETF) standard's track. AS stands for Applicability Statement and is a specification about how to transport data, not how to validate or process data. AS1 provides an Internet solution for securely exchanging EDI and XML over the Internet using SMTP.

AS2

A draft specification first published in the Internet Engineering Task Force (IETF) standard's track. AS stands for Applicability Statement and is a specification about how to transport data, not how to validate or process data. AS2 specifies the means to connect, deliver, validate, and reply to (receipt) data in a secure and reliable way. AS2 provides an Internet solution for securely exchanging EDI over

the Internet using the hypertext transmission protocol (HTTP) instead of the simple mail transport protocol (SMTP) as the transport protocol.

Authentication

Ensures the accurate identification of both the sender and the receiver. Authentication is accomplished using digital signatures.

C

Cipher

A key-selected transformation between plaintext and ciphertext. An algorithm for putting a message into code by transposition and/or substitution of symbols.

Compression

The ability to represent data in forms that take less storage than the original. The limit to this is the amount of uniqueness in the data. It is not possible to compress everything down to a single byte, because a byte can only select 256 different results. Data compression is either "lossy," in which some information is lost, or "lossless," in which all of the original information can be completely recovered.

Configuration File

A text file containing one or more Console Command statements. A Configuration File can be processed automatically by the P2P Agent application upon startup if it is named p2pagent.cfg and stored in the same directory location as the P2P Agent executable program. A Configuration File can also be processed if the -f parameter is entered as a run-time program argument or as a console command.

Control Address

The IP address portion of the IP Address and Port used by the P2P Agent Transport and Router Agents to listen for incoming control messages from a supervising Administrative Agent; configured using the -ca Set Option.

Control Port

The IP Port portion of the IP Address and Port used by the P2P Agent Transport and Router Agents to listen for incoming control messages from a supervising Administrative Agent. Configured using the -cp Set Option.

Control Service

The set of application tasks which execute within the context of a thread of execution to process incoming commands being sent by an Administrator Agent. The Control Service is required by P2P Agent Agents acting in the Transport or Router Role, if the Agent is being remotely configured.

Cypher Text

Data that has been transformed from a plaintext form into encrypted text (an unreadable form) using an encryption process.

D

DEFLATE

Specifies the DEFLATE compression algorithm used to reduce the file transfer overhead. The DEFLATE compression algorithm is a lossless compressed data format that compresses data using a combination of the LZ77 algorithm and Huffman coding.

Delivery Notification

A message formatted according to (AS2) that is sent to a sending host computer to indicate the disposition of a received message. The format of Delivery Notifications used by P2P Agent is the Message Delivery Notification. or MDN, as defined in MDN.

Digital Certificate

A document that contains name, serial number, expiration dates and a copy of the owner's public key; used to encrypt data and validate signatures.

Digital Signature

A digital code that can be attached to an electronically transmitted message that uniquely identifies the sender. Like a written signature, the purpose of a digital signature is to guarantee that the

individual sending the message really is who he or she claims to be. Digital signatures are especially important for electronic commerce and are a key component of most authentication schemes. To be effective, digital signatures must be unforgeable. There are a number of different encryption techniques to guarantee this level of security.

Document Digest

A unique "fingerprint" summary (128 or 160 bits long) of an input file. It is used to create a digital signature and to ensure that the file has not been altered. It is also called a hash and is produced by a checksum program that processes a file.

DSS

Specifies the Digital Signature Algorithm (DSA) for digital signature generation and verification. The DSA is used by a signatory to generate a digital signature on data and by a verifier to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process.

E

EDI

Short for Electronic Data Interchange, the transfer of data between different companies using networks, such as the Internet. As more and more companies get connected to the Internet, EDI is becoming increasingly important as an easy mechanism for companies to buy, sell, and trade information. ANSI has approved a set of EDI standards known as the X12 standards.

EDIINT

EDI Over the Internet Working Group - a working group of the IETF that developed the AS1 and AS2 proposed standards.

Encryption

A process that uses a mathematical algorithm and a key to transform data into an unreadable format (called cyphertext). A receiver can then use a key to restore the data to its original content.

F

FIPS

Federal Information Processing Standard.

G

Graphical User Interface

A GUI (usually pronounced GOO-ee) is a graphical user interface that takes advantage of the computer's graphics capabilities to make the program easier to use. Well-designed graphical user interfaces can free the user from learning complex command languages. On the other hand, many users find that they work more effectively with a command-driven interface, especially if they already know the command language.

GZIP

Specifies a lossless compressed data format that is compatible with the widely used GZIP utility. This format includes a cyclic redundancy check value for detecting data corruption.

H

Hash

A hash value (or simply *hash*) is a number generated from a string of text. The hash is substantially smaller than the text itself, and is generated by a formula in such a way that it is extremely unlikely that some other text will produce the same hash value. Hashes play a role in security systems where they're used to ensure that transmitted messages have not been tampered with. The sender generates a hash of the message, encrypts it, and sends it with the message itself. The recipient then decrypts both the message and the hash, produces another hash from the received message, and compares the two hashes. If they're the same, there is a very high probability that the message was transmitted intact.

HTTP

See Hypertext Transfer Protocol.

Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) is the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

I

IETF

Internet Engineering Task Force - The Internet Engoineering Task Force is a large, open, international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet.

In-Beacon Service

The set of application tasks that execute within the context of a single application thread to receive UDP packets sent by one or more Transport Agents. P2P Agent Agents configured for the Router Role (Router Agents) use the In-Beacon Service to collect these UDP packets to maintain current information about active Transfer Agents on the local network segment.

Inbound Service

One or more sets of application tasks that execute within the context of one or more threads of execution to process incoming data being sent by a remote host computer. The Inbound Service consists of, at least, one inbound thread listening for incoming TCP/IP connections on a particular protocol (HTTP or HTTPS). The Inbound Service creates an Inbound Session thread for each separate incoming connection. Each discrete protocol is serviced by a separate Inbound Main thread, which is assigned a unique IP address and port on which to listen for incoming connections.

Integrity

Ensures that data is not tampered with or corrupted in transit. Integrity is accomplished using document digests and digital signatures.

K

Key Encryption

The translation of data into a secret code. Encryption is the most effective way to achieve data security. To read an encrypted file, you must have access to a secret key or password that enables you to decrypt it. Unencrypted data is called plain text; encrypted data is referred to as cipher text. There are two main types of encryption: asymmetric encryption (also called public-key encryption) and symmetric encryption.

M

MD5

Specifies the Message Digest Algorithm used to verify a file's integrity. The MD-5 is a one-way algorithm that takes any length of data and produces a 128-bit "fingerprint" or "message digest". This fingerprint is "non-reversible", meaning that the data cannot be determined based on its MD-5 fingerprint.

Message Disposition Notification (MDN)

A Message Disposition Notification (MDN) message is a response message defined to ensure the secure reliable delivery of messages for AS1 and AS2 protocols.

MIME

Multipurpose Internet Mail Extension - MIME is a specification for enhancing the capabilities of standard Internet electronic mail. It offers a simple standardized way to represent and encode a wide variety of media types for transmission using Internet mail.

N

NIST

National Institute of Standards and Technology. A part of the U.S. Department of Commerce, formerly called the National Bureau of Standards, that defines standards for voice, data, and video transmissions, encryption, and other kinds of technology.

Non-repudiation of Receipt

Confirms that the intended party received the data. This is accomplished using digital signatures and signed MDNs.

O

Out-Beacon Service

The set of application tasks that execute within the context of a single thread of execution to periodically transmit a small packet of data identifying the Transport Agent to one or more Router Agents. The Out-Beacon Service emits a UDP packet containing the IP Addresses and Ports on which the Agent is currently listening. Router Agents collect these packets to dynamically build a current list of Transport Agents to which inbound data can be routed for processing.

Outbound Service

The set of application tasks that execute within the context of one or more threads of execution to process requests for outgoing message delivery. The Outbound service consists of, at least, the main outbound thread that processes send transactions from the Outbound Queue. The main outbound thread creates an Outbound Session thread for each separate send request.

P

PKI Service

The set of application tasks that execute within the context of a single thread of execution to proactively search the configuration database for public-key certificates which are nearing their expiration date. The PKI Service implements the iSoft Zero-Administration PKI architecture, to facilitate the automated renewal of public-key certificates.

Plain Text

Unencrypted data.

Port

A specific communications end-point to a logical connection and the way a client program specifies a specific server program on a computer in a network.

Privacy

Ensures that only the intended receiver can view the data. This is accomplished using a combination of encryption algorithms and message packaging.

Private Key

A value known only to the owner, used to create a signature and decrypt data encrypted by its corresponding public key.

Public Key

A value, known by everyone to whom the certifcate has been distributed, used to encrypt data and validate a digital signature. Although mathematically related to the private key, it is astronomically difficult to derive from the public key.

Public Key Infrastructure

Public Key Infrastructure is a system of digital certificates, Certificate Authorities, and other registration authorities that verify and authenticate the validity of each party involved in an Internet transaction. PKIs are currently evolving and there is no single PKI or even a single agreed-upon standard for setting up a PKI.

P2P Agent

iSoft Peer-to-Peer Agent, Version 3.X.

R

RC2

Specifies the Rivest's Cipher encryption algorithm used to encrypt and decrypt messages. RC-2 is a conventional (secret key) block encryption algorithm and has a block size of 64-bits with a variable key size from one byte up to 128 bytes.

Role

The set of P2P Agent Application Services operating within a single instance of the P2P Agent application (a process) which, taken together, comprise a logical functional unit in an iSoft P2P network. The Roles supported by P2P Agent are Administrator, Router, and Transport.

Router Agent

An instance of the P2P Agent application configured to provide routing services including round-robin selection of Transport Agents, message-queuing and fail-over retransmission.

RSA

An internet encryption and authentication system that uses an algorithm developed in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. The RSA algorithm is the most commonly used encryption and authentication algorithm and is included as part of the Web browser from Netscape and Microsoft.

S

Serializer Service

The set of application tasks that execute within the context of a single thread of execution to serialize the access of shared application resources by other application threads. The Serializer Service is started automatically at application startup and is required by all P2P Agent Roles. Serialized resources include shared memory areas, directories, and the database.

Service

A discrete set of P2P Agent application tasks that provide a logical service to the Agent. The Services supported by P2P Agent are: Serializer, Outbound, Inbound, Control, PKI, Work-Order, User Interface, Out-Beacon, and In-Beacon. Sets of concurrently executing Services are combined to define P2P Agent Roles.

SHA-1

Specifies the Secure Hash Algorithm used to verify a file's integrity. The SHA-1 generates a condensed representation of a message called a message digest. The SHA-1 is used by both the transmitter and intended receiver of a message in computing and verifying a digital signature.

S/MIME

Secure MIME - S/MIME (Secure/Multipurpose Internet Mail Extensions) provides a consistent way to send and receive secure MIME data. Based on the popular Internet MIME standard, S/MIME provides the following cryptographic security services for electronic messaging applications: authentication, message integrity and non-repudiation of origin (using digital signatures) and privacy and data security (using encryption).

SMTP

Simple Mail Transport Protocol - An Internet standard for transporting email.

SSL

Short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL works by using a public key to encrypt data that's transferred over the SSL connection. Both Netscape Navigator and Internet Explorer support SSL, and many Web sites use the protocol to obtain confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection start with https: instead of http:.

T

TCP/IP

Transmission Control Protocol/Internet Protocol or the suite of standard protocols that enable computers to inter-communicate on the Internet.

Thread

A logical sequence or program instructions that are executed independently.

Transport Agent

An instance of the P2P Agent application configured to provide transport services including the compression, encryption and delivery of data, the verification of digital signatures and the construction and transmission of Delivery Notifications.

Triple Data Encryption Standard

Triple Data Encryption Standard (DES3) is a derivative of Data Encryption Standard (DES) that has served as the cornerstone of data encryption for almost 40 years. DES-3 is DES run three times with three different keys. It uses a 192-bit key and has an effective strength of 112-bits.

U

UCC

Uniform Code Council, Inc.

UDP

User Datagram Protocol. A simple, datagram-oriented, transport layer protocol, used by P2P Agent to facilitate dynamic pools of Transport Agents marshaled by a Router Agent. The Transport Agents use UDP as the underlying protocol to transmit small informative packets of data identifying their inbound protocol ports.

User-Interface Service

The set of application tasks that execute within the context of a single thread of execution to return HTML-formatted application-status information to a web-browser. The User-Interface Service is not required by any P2P Agent Role. However, any P2P Agent Agent can enable the User-Interface Service so that its current status can be remotely viewed via a Web-browser.

W

Work-Order

A set of one or more Console Commands sent to a P2P Agent Agent to accomplish one or more specific tasks. The typical use of a Work-Order is to initiate an outbound delivery of data (a send).

Work-Order Service

The set of application tasks that execute within the context of a single thread of execution to query the database or a directory for Work Orders.

X

X.509V3

X.509 Public Key Certificate and CRL Profile, Version 3, defined in CERT. The version of X.509 Public Key Certificate supported by P2P Agent.