



OE DataDistributor User Guide

V1.9
May 2012
DDUSG0512

Table of Contents

Chapter 1

Introduction	7
Purpose	7
Product	7
Audience	7
Revision Log	8

Chapter 2

DataDistributor Operation	9
DataDistributor on Windows®	9
Supported Instances	9
Starting DataDistributor	9
Stopping DataDistributor	9
DataDistributor on HP NonStop Guardian	10
Supported Instances	10
Starting DataDistributor	10
Stopping DataDistributor	11
DataDistributor Producer for IST/Switch (Solaris)	11
Supported Instances	11
Starting DataDistributor	12
Stopping DataDistributor	12
DataDistributor on Solaris (Non IST/Switch)	13
Supported Components	13
Starting DataDistributor	13
Stopping DataDistributor	14
DataDistributor on IBM® zOS	15
Supported Instances	15
Starting DataDistributor Consumer	15

Stopping DataDistributor Consumer	15
DataDistributor on AIX	16
Supported Components	16
Starting DataDistributor.	16
Stopping DataDistributor.	17

Chapter 3

Operational Behaviour18

General Processing Behaviour.	18
Common Processing.	18
DataDistributor Consumer.	18
DataDistributor Producer.	19
Transaction Data Input and Output.	19
Producer for IST/Switch	19
Producer for Connex on HP NonStop.	20
Producer for Delivery Queue.	21
Producer for Connex on IBM®	21
Producer for BASE24	22
Consumer for DataNavigator®.	23
Consumer for Settlement	28
Consumer for Received Queue.	30
Checkpoint Processing.	31
Checkpoint Structure.	32
Position Information.	35

Chapter 4

Troubleshooting37

Process Initialisation Failures.	37
Process Runtime Errors	38
Error Recovery Processing	39
Network Errors	39

Producer Data Source Errors	41
Consumer Target System Errors.....	44

Chapter 5

Trace Output	46
Transaction Trace Output	46

Chapter 6

HP NonStop Producer Statistics Reporting	47
Extraction Lag Reporting	47

Chapter 7

Consumer Load Statistics	49
---------------------------------------	-----------

Chapter 8

Record Filtering.....	51
Filtering Connex on HP NonStop Log Records	51

Chapter 9

Connection Validation.....	54
TCP/IP Address Validation	55
Consumer Identity Validation	56

Chapter 10

PA-DSS Compliance and Cryptography Services	58
Producer Platform Cryptography	58
HP/Connex	58
IBM/Connex	59
IST/Switch	59
Consumer Platform Cryptography	60
Encryption/Decryption Processing Rules	60
Encryption Header	61
ICSF Cryptography Library	63
OpenSSL Cryptography Library	64
Key Generator Utility	65
Overview	65
Operation	65
ICSF Configuration and Output	66
OpenSSL Configuration and Output	67
Password Encryption Utility	70
Overview	70
Operation	70

Chapter 11

FTP Named Pipe Support	72
Overview	72
DDPIPE Operation	73
z/OS Platform	73
Configuration	74
Pipe Reader	74
Pipe Writer	76
Interaction with FTP	78
Sending Data to a Named Pipe	78

Receiving Data from a Named Pipe	79
--	----

Chapter 12

File Processing Support.....	81
Overview	81
DDFILE Operation	82
z/OS Platform	82
Configuration	83
File Process	83

Chapter 13

Secure Socket Layer Communication Support	85
Overview	85
SSL Environment Configuration	86
Overview	86
Open SSL Configuration (IST SWITCH)	86
System Secure Socket Layer Configuration (Z/OS)	87
DataDistributor SSL Configuration	89
Overview	89
OpenSSL Component Configuration	90
Z/OS SSL Component Configuration	91
Index.....	92



Introduction

Purpose

This document provides a guide to the use of DataDistributor in the various operating environment in which it is supported.

This guide covers the following subjects:

- How to start and stop the DataDistributor processes.
- A brief description of normal DataDistributor operation.
- A description of the transaction data input and output sources that are used by the DataDistributor processes.
- How to find information in case of failure or errors which occur in the operation of DataDistributor process.

Product

- OE DataDistributor

Audience

This manual is intended for Fidelity National Information Services (FIS) and Customer staff responsible for the operation of OE DataDistributor.

Revision Log

Date	Effective w/version	Chapter	Change
May 2012	V1.9	3	Added a note on page 25 and page 29 .
		10	Added the following section: <ul style="list-style-type: none"> • Password Encryption Utility.
		13	Added the following chapter: <ul style="list-style-type: none"> • Secure Socket Layer Communication Support.
September, 2010	V1.6	3	Added the following information under Transaction Data Input and Output: <ul style="list-style-type: none"> • “Producer for BASE24” on page 22. Added the following information under Checkpoint Processing Checkpoint Structure: <ul style="list-style-type: none"> • “Producer for BASE24” on page 34. Modified the following heading under Checkpoint Processing Position Information: <ul style="list-style-type: none"> • “Producer for Connex on HP NonStop/BASE24” on page 35.
		4	Added the following information under Producer Data Source Errors: <ul style="list-style-type: none"> • “BASE24 Producer” on page 44.
		12	Added the new chapter: <ul style="list-style-type: none"> • “File Processing Support” on page 81.
May, 2010	V1.6	3	Updated the text information for: <ul style="list-style-type: none"> • “Consumer for Settlement” on page 28.
		10	Added the new chapter: <ul style="list-style-type: none"> • “PA-DSS Compliance and Cryptography Services” on page 58.
		11	Added the new chapter: <ul style="list-style-type: none"> • “FTP Named Pipe Support” on page 72.
October, 2009	V1.0		Initial release of this manual.



DataDistributor Operation

DataDistributor on Windows®

Supported Instances

Currently only OE DataDistributor Consumer is supported on the Windows® platform.

The DataDistributor Consumer processes are required to be configured as Windows Services. Refer to the *OE DataDistributor Installation and Configuration Guide* for use of the **cnscfg** command line tool, which is used to configure the Consumer Services.

Starting DataDistributor

Each DataDistributor Consumer Service is started using the standard tools supplied by the Windows Operating System for controlling services.

If the Consumer Services are configured as automatic - each configured service will start when the Windows Server is started.

If the Consumer Services are configured as manual (or as automatic, but the Windows Server hasn't been re-started since the services were configured), the Services are started selecting the required service from within the Windows Services Administrative Tool (Control Panel - Administrative Tools - Services) and using the Start Service option.

Stopping DataDistributor

Each DataDistributor Consumer Service is started using the standard tools supplied by the Windows Operating System for controlling services.

Each service can be stopped by selecting the required service from within the Windows Services Administrative Tool (Control Panel - Administrative Tools - Services) and using the Stop Service option.

DataDistributor on HP NonStop Guardian

Supported Instances

Both DataDistributor Producer (Connex and Base24) and Consumer instances are supported on HP NonStop Guardian.

DataDistributor Producer and Consumer processes may be started from TACL using appropriate commands (see next section) or configured to run as PATHWAY Servers. Refer to the *OE DataDistributor Installation and Configuration Guide* for information on configuring DataDistributor Producer or Consumer as a PATHWAY Server.

Starting DataDistributor

Starting from PATHWAY

Each DataDistributor Producer or Consumer process is started using standard PATHCOM commands. For example:

```
PATHCOM START SERVER PRD-SERVER-1
PATHCOM START SERVER CNS-SERVER-1
PATHCOM START SERVER *
```

Starting from TACL

DataDistributor Producer or Consumer processes are started from TACL using the RUN command, as follows:

```
RUN <location>.producer /nowait, name $<pid>, cpu 0/ $<pid> <config
file>
RUN <location>.consumer /nowait, name $<pid>, cpu 0/ $<pid> <config
file>
```

Where:

Command	Description
<subvol>	Is the location of the installed Producer or Consumer executable.
<pid>	Is the process id assigned to the process e.g. \$PRD01.
<config file>	Is the name of the configuration file for this process (full path).

Stopping DataDistributor

Stopping from PATHWAY

Each DataDistributor Producer or Consumer process is stopped using standard PATHCOM commands. For example:

```
PATHCOM FREEZE SERVER PRD-SERVER-1
PATHCOM FREEZE SERVER CNS-SERVER-1
PATHCOM STOP SERVER PRD-SERVER-1
PATHCOM STOP SERVER CNS-SERVER-1
PATHCOM THAW SERVER PRD-SERVER-1
PATHCOM THAW SERVER CNS-SERVER-1
```

Stopping from TACL

The Producer process can be stopped by issuing the STOP command to the appropriate Process ID.

```
STOP $PRD01
STOP $CNS01
```

DataDistributor Producer for IST/Switch (Solaris)

Supported Instances

The DataDistributor Producer for IST/Switch is a specific instance of the Solaris Producer built, packaged and configured for use with FIS IST/Switch products.

The Producer is intended to be configured as an IST/Switch controlled process running under the IST Task Manager. Consult the appropriate IST/Switch Documentation for information regarding how to accomplish this.

Starting DataDistributor

To start the DataDistributor Producer issues the **run** command using the IST/Switch **mbcmd** utility. For example:

```
mbcmd run <process name>
```

where:

Command	Description
<process name>	Is the name of the Producer as assigned during the Task Manager configuration.

For further details of the IST/Switch mbcmd utility see the appropriate *IST/Switch* documentation.

Stopping DataDistributor

To stop the DataDistributor Producer issues the **kill** command using the IST/Switch **mbcmd** utility. For example:

```
mbcmd kill <process name>
```

where:

Command	Description
<process name>	Is the name of the Producer as assigned during the Task Manager configuration.

For further details of the IST/Switch mbcmd utility see the appropriate *IST/Switch* documentation.

DataDistributor on Solaris (Non IST/Switch)

Supported Components

Both OE DataDistributor Producer and Consumer processes are supported on Solaris. This section deals with instances of OE DataDistributor that are not associated with an IST/Switch Installation.

Starting DataDistributor

DataDistributor processes are started using the **dd_start** shell utility.

The command syntax is:

```
dd_start [<component group> [<component>]]
```

Where:

Command	Description
<component group>	Is the optional group of components to start, that is, consumer or producer.
<component>	Is the specific instance of consumer or producer to start, for example, cns01, prd01, etc.

Thus:

Command	Description
dd_start	Starts all configured DataDistributor processes.
dd_start consumer	Starts all consumer processes.
dd_start producer	Starts all producer processes
dd_start consumer cns01	Starts consumer cns01.
dd_start producer prd01	Starts producer prd01.

NOTE:

- The dd_start utility relies upon a known directory structure in order to locate and run the start file created during the configuration procedures for each process. Thus the “component group” name specified (i.e. consumer/producer) must match the directory created to contain the configuration directories for each process within that component group. Similarly, the component name specified must match the directory name created to hold the start file for that component. Refer to the *OE DD Installation and Configuration Guide* for details.
- It is important to ensure that all previous instances of DataDistributor Processes have completely terminated prior to issue the dd_start command. Failure to do so will result in processes failing to start, usually due to TCP/IP Socket bind errors, as required TCP/IP ports are still in use by the previous instance.

Stopping DataDistributor

DataDistributor processes can be stopped using the **dd_stop** shell script, which has the following syntax:

```
dd_stop [<component group> [<component>]]
```

where:

Command	Description
<component group>	Is the optional group of components to stop, that is, consumer or producer.
<component>	Is the specific instance of consumer or producer to stop, for example, cns01, prd05, etc.

Thus:

Command	Description
dd_stop	Stops all consumer and producer processes.
dd_stop consumer	Stops all consumer processes.
dd_stop producer	Stops all producer processes.
dd_stop consumer cns01	Stops consumer cns01.
dd_stop producer prd01	Stops producer prd01.

NOTE:	dd_stop is an shell interface to the DataDistributor Command Interface utility, ddcmd . This utility uses the configuration supplied in the \$DD_ENV_CONFIG_HOME/trgtcfg.xml file to send the stop command to the required processes via TCP/IP.
--------------	---

DataDistributor on IBM® zOS

Supported Instances

Both OE DataDistributor Producer and Consumer are supported on the IBM® zOS platform.

DataDistributor Producer processes can only be run as Connex on IBM tasks and must be started and stopped using standard Connex on IBM commands via the Operator Console. See the relevant Connex on IBM documentation.

DataDistributor Consumer processes can be run either as batch processes by submitting JCL containing the relevant commands, or by configuring the Consumer processes as “Started Tasks”.

See the OE DataDistributor Installation and Configuration Guide for information on the creation of the appropriate JCL or configuration of the Started Task.

Starting DataDistributor Consumer

Starting Using JCL

If using JCL to start the Consumer as a batch process, submit the JCL file to the Job Entry System to start the Consumer process

Using a Started Task

If running the Consumer as a Started Task, submit the JCL Procedure for the task using the Operator Start command.

Stopping DataDistributor Consumer

In order to stop the Consumer in a controlled manner send a STOP command to the appropriate JCL Job or Started Task from an Operator Console

Stopping a Batch Process

Additionally, it is possible to bring a Consumer process running as a batch job to an abrupt halt by cancelling the JCL Job.

Although the use of the STOP command is preferred, cancelling the job should not result in any loss of transaction data due to the checkpoint mechanism employed by DataDistributor.

DataDistributor on AIX

Supported Components

Only OE DataDistributor Consumer processes are supported on AIX.

Starting DataDistributor

DataDistributor processes are started using the **dd_start** shell utility.

The command syntax is:

```
dd_start [<component group> [<component>]]
```

where:

Command	Description
<component group>	Is the optional group of components to start, that is, consumer or producer.
<component>	Is the specific instance of consumer or producer to start, for example, cns01, prd01, etc.

Thus:

Command	Description
dd_start	Starts all consumer configured DataDistributor processes.
dd_start consumer	Starts all consumer processes.
dd_start consumer cns01	Starts consumer cns01.

NOTE: The `dd_start` utility relies upon a known directory structure in order to locate and run the **start** file created during the configuration procedures for each process. Thus the “component group” name specified (that is, consumer/producer) must match the directory created to contain the configuration directories for each process within that component group. Similarly, the component name specified must match the directory name created to hold the start file for that component.

See the *OE DataDistributor Installation and Configuration Guide* for details.

Stopping DataDistributor

DataDistributor processes can be stopped using the **dd_stop** shell script, which has the following syntax:

```
dd_stop [<component group> [<component>]]
```

where:

Command	Description
<component group>	Is the optional group of components to stop, that is, consumer or producer.
<component>	Is the specific instance of consumer or producer to stop, for example, <code>cns01</code> , <code>prd05</code> , etc.

Thus:

Command	Description
<code>dd_stop</code>	Stops all consumer and producer processes.
<code>dd_stop consumer</code>	Stops all consumer processes.
<code>dd_stop consumer cns01</code>	Stops consumer <code>cns01</code> .

NOTE: `dd_stop` is a shell interface to the DataDistributor Command Interface utility, `ddcmdnd`. This utility uses the configuration supplied in the `$DD_ENV_CONFIG_HOME/trgtcfg.xml` file to send the stop command to the required processes via TCP/IP.



Operational Behaviour

This section provides a general description of the expected “normal” operational processing behaviour of OE DataDistributor Processes in the various supported environments.

General Processing Behaviour

Common Processing

On start-up all OE DataDistributor Processes perform the following tasks:

- Load and read their respective configuration files.
- Create and initialise the required runtime components:
 - Create and initialise the required Event Logger component.
 - Create and/or read the process Checkpoint file or database table entry.
 - Create and initialise the required TCP/IP Connectivity.

DataDistributor Consumer

On start-up the OE DataDistributor Consumer performs the following additional tasks:

- Create and initialise the appropriate “Target System” components.
 - **For DataNavigator®:**
 - Connect to the configured DataDistributor database.
 - Prepare ODBC or OCCI (Oracle) statements for database interaction.
 - Initialise the DataDistributor Control and Queue Tables if these do not already exist. This processing creates the configured Queue table entries and sets the required initial values in the Control table.
 - **For Settlement:**
 - Locate and open any required Settlement Data Sets.
 - **For Received Queue:**
 - Connect to the configured DataDistributor database.
- Attempt to connect to its configured Producer process.
- Negotiate with the Producer regarding Communications Session parameters.

- Initiate the transfer of transaction data.
- Process received data by forwarding to the appropriate Target Systems and acknowledging receipt to the Producer.

DataDistributor Producer

On start-up the OE DataDistributor Producer performs the following additional tasks:

- Create and initialise the appropriate “Transaction Data Source” components.
 - **For Connex on HP NonStop:**
 - Locate and open the appropriate Log Control file (LGWARM) and read the entry for the associated Logger Process.
 - **For IST/Switch:**
 - Initialise the IST Switch Database access environment for performing extraction of transactions from the IST/Switch SHCLOG tables.
 - **For Delivery Queue processing:**
 - Connect to the configured DataDistributor database.
 - Prepare OCCI (Oracle) statements for database interaction.
 - **For Connex on IBM®:**
 - Initialise the Connex runtime environment including reading the CR Extract file (this must be done prior to opening and processing the Producer configuration as the details of this file are contained within the CR Extract).
- Listen for connections from the associated Consumer process.
- Negotiate with the Consumer regarding Communications Session parameters.
- Extract and send transaction data from the configured Data Source on request.

Transaction Data Input and Output

This section describes the Input and Output sources of transaction data as processed by the various Producer and Consumer instances.

Producer for IST/Switch

The SHCLOG_CTRL table controls the extraction of transaction data.

This table contains a row for every transaction that is logged by IST/Switch – each row corresponds to a row in the main SHCLOG table. Additional transaction information may be held in subsidiary IST/Switch tables:

- SHCLOG_ADD
- SHCLOG_CHEQUE
- SHCLOG_DN (DataNavigator® Specific Data)
- SHCLOG_<client> (Client specific data).

Each record extracted from the database by the Producer will contain data from the SHCLOG base table plus the optional subsidiary tables, dependent on the nature of the transaction.

The SHCLOG_CTRL table is updated by the Producer on receipt of an acknowledgement from the Consumer to indicate which transactions have been transferred.

Producer for Connex on HP NonStop

The Connex on HP NonStop Log Control file (LGWARM), controls the extraction of transaction data.

This file contains an entry for each Logger configured for the Switch.

The Producer uses this entry to determine which files must be processed for its configured Logger process.

Files may be in one of the following three states:

- Currently Open (in use by the Logger).
- Closed (completed by the Logger following a Swap Logs command).
- Unloaded (unloaded - either by DataDistributor or by a separate UNLOAD process).

On receiving a request from the Consumer, the Producer will transfer any “unprocessed” Closed files first, followed by the Currently Open file. “Unprocessed” is determined on the basis of a **last close** timestamp held in the Producer’s checkpoint file (this timestamp indicates the close time of the last file processed by the Producer).

Under certain circumstances, the Producer will also process previously Unloaded files if the Producer detects that they have been Unloaded since the last close time held by the Producer. Typically this will only occur if there has been a severe outage of DataDistributor during which Connex on HP NonStop has swapped and unloaded log files.

The expected situation is that the Producer will always be processing the Currently Open file - that is, it will be transferring data to the Consumer as soon as possible after the logger writes it to the log file. It will also therefore swap logs in line with the Logger.

Producer for Delivery Queue

Data is extracted from the defined delivery queue table. The name of the table to be referenced is specified in the configuration. The structure of the table is as follows:

Column Name	Data Type	Null	PKey	Notes
ENTRY_NO	NUMBER(18)	N	Y	A sequential number that identifies the row.
ENTRY_TIMESTAMP	TIMESTAMP	N	N	The time at which the entry was inserted.
ENTRY_STATUS	CHAR(1)	N	N	Indicates status of record as confirmed as sent ('C') or unsent (' ' space).
ENTRY_TYPE	NUMBER(5)	N	N	For ease of identification of the data contained within the entry.
ENTRY_LENGTH	NUMBER(5)	N	N	The length of the data stored in ENTRY_DATA
ENTRY_DATA	BLOB	N	N	Contains a single record for delivery.

Each row in the table represents a data item to be delivered to the corresponding Consumer process. Rows are selected by ENTRY_NO, as indicated in the initial request received from the Consumer. Rows with an ENTRY_NO greater than the last value extracted are available to be retrieved.

On receipt of the acknowledgement from the Consumer, the Producer will update the status flag in the delivery queue table to 'C' to mark the row as confirmed as sent/received.

Producer for Connex on IBM®

The Connex on IBM® MQ Control Task (or tasks) provides the source of the transaction log records.

On receiving an initial request from the Consumer, the Producer will send an "Active" message to the MQ Control task via the Message Delivery System (MDS).

The MQ Control Task will then send Log Records to the Producer via the MDS which the Producer will store in a buffer for sending to the Consumer. After "n" records have been sent (where "n" is a value configurable via the configuration of the MQ Control Task) the MQ Control Task will send a "Commit" message to the Producer. The Producer will then send the records accumulated in the buffer to the Consumer.

Once the Consumer has processed the records, it will send an Acknowledgement back to the Producer, which will then send a “Commit Reply” message to the MQ Control task to indicate that all records up to and including the “Commit” have been processed. MQ Control will then queue more records to the Producer.

Producer for BASE24

The Producer for BASE24 is configured to extract logged record data from a designated set of BASE24 log files (PTFL or TLF).

TLF and PTLF files are generated on a daily basis and named according to the date to which they apply. The naming convention is as follows:

<2 character prefix>YYMMDD

e.g.

PT091227

PT091228

etc.

Both Base24 and Base24-pos system maintain two concurrent (i.e. available for logging purposes) daily files – one for the “current” business day and one for the “next” business day. Transactions are logged to the appropriate file according to the date at which the transaction was acquired in the location of the POS or ATM terminal. The purpose of the daily files is to allow terminals to switch from one business day to another at different times according to their location.

At Network Settlement time all logging to the “current” file is deemed complete. This is best illustrated by example:

Date	Current File	Next File
091226	PT091226	PT091227
091227	PT091227	PT091228
091228	PT091228	PT091229
etc.		

Thus at Network Settlement time the old Next File becomes the new Current File and a new Next File is created.

Since data may be written to both the Current and Next files concurrently, DataDistributor Producers are configured as a pair (with corresponding Consumer processes) such that data can be extracted from both files concurrently as it is written.

Each Producer is configured with an initial start date which indicates from which file extraction for that Producer is to begin. One Producer will be configured to extract from the “Current” File (as described above), the other from the “Next” file.

On receiving a request for transfer from its associated Consumer the Producer will begin transfer from the appropriate file. Position information will be exchanged to allow subsequent repositioning in the file following any restart. On reaching the end of the current file (and after a suitable delay) the Producer will check for the existence of its next file to process which will be 2 days forward from the date of its current file.

Consumer for DataNavigator®

Consumer for DataNavigator® feeds transaction data to DataNavigator via an intermediate database consisting of the following tables:

- DATADIST_QUEUE_*nn*(where *nn* = 01, 02, 03 etc.)
- DATADIST_CONTROL

There is a DATADIST_QUEUE_*nn* table created for each configured Consumer process. The name of the table corresponds to the Queue Number assigned to the Consumer in its configuration (see the OE DD Installation and Configuration Guide and OE DD Configuration Reference for details).

The Queue tables are initialised with a pre-set number of rows, corresponding to values set in the Consumer configuration.

The DATADIST_CONTROL table is shared amongst the configured Consumer processes and the processes responsible for reading the queue (the Queue Reader). There is a row in this table for each queue.

Each queue is implemented as a “ring-buffer” in which all rows in the database table are pre-allocated and keyed with a sequential number. The transaction data received by the Consumer process is then loaded into the queue using SQL UPDATE statements rather than INSERT statements.

The ring buffer concept requires that data is loaded into the queue starting at row number one and then added sequentially until the end of the table is reached, at which point the loading will start over at row number one. During normal operation, rows are not inserted into or deleted from the table, they are merely re-used once they have been read and processed by the Queue Reader.

The Consumer and Queue Reader processes will keep track of their positions in the queue through the use of read and write pointers located in a separate Control table.

The Consumer process will read the read pointer to determine the last row processed by the Queue Reader, to avoid over-writing unread rows in the table. It will load data into the queue table in sequential order, and update the write pointer to indicate last written row. On start-up, the Consumer can use the write pointer to determine where to start loading new transactions.

The Queue Reader process will read the write pointer and compare it to the read pointer to determine if there are rows to read. It will read and process the rows in the table sequentially, and update the read pointer to indicate the last processed row. On start-up, the Queue Reader can use the read and write pointers to determine if there are rows to process.

The queue table rows are pre-allocated by the Consumer process on initial start-up based on a value specified within the Control table. The mechanism also allows for automatic expansion of the queue during processing, thus avoiding queue full situations. The parameters for controlling the queue expansion are also specified in the control table.

The structure of the Database tables and their processing is described in more detail in the following sections:

Queue Table Structures

This section defines the structure of the DataDistributor Queue and Control tables.

In each case the columns are expressed in terms of their Oracle Data Types.

- DATADIST_CONTROL

Column Name	Data Type	Null	PKey	Notes
QUEUE_NO	CHAR(2)	N	Y	ID Number assigned to the queue table controlled by this row. It matches the <i>_nn</i> in the queue table name.
QUEUE_SIZE	NUMBER(10)	N	N	Number of rows that exist in the queue table.
QUEUE_SIZE_INC	NUMBER(10)	N	N	Number of rows to add to the queue table at a time during queue initialisation and queue expansion.
QUEUE_SIZE_MAX	NUMBER(10)	N	N	The maximum size to which the queue can expand. This will be set to QUEUE_SIZE to prevent queue expansion as this is not required for this interface.
WRITE_PTR	NUMBER(10)	N	N	The last row in the queue updated by the Consumer.
READ_PTR	NUMBER(10)	N	N	The last row in the table read by the Queue Reader.
EMPTY_SPACE	VARCHAR(3000)	N	N	Empty space used to force each row to be placed on a separate data page. Set to maximum length.

Each row in this table contains the control data needed to process a single queue table. The data in this table is read and updated by the Consumer process and by the process that reads the queue(s). The table will be initialized with appropriate starting values when a queue is created (See [Queue Initialisation](#) for more information).

The primary key for this table is QUEUE_NO. This table has no referential integrity with other tables.

The last column of this table, EMPTY_SPACE, is used to reduce locking on this table in the DB2 mainframe environment. Typically, page level locking is used in this environment. The column forces each row of the table to be placed on a separate page. When a row is initialized with the start values for a specific queue, the EMPTY_SPACE column should be initialized to its maximum length. Thereafter, there should be no need to access or update this column.

- DATADIST_QUEUE_nn

Column Name	Data Type	Null	PKey	Notes
ENTRY_NO	NUMBER(10)	N	Y	A sequential number that identifies the row.
ENTRY_LENGTH	NUMBER(5)	N	N	The length of the data stored in ENTRY_DATA
ENTRY_DATA	BLOB	N	N	Contains a single financial transaction.

Each table will hold the records for a single queue. Each row contains an ENTRY_DATA column, which will either be empty (following initialisation) or contain a single logged transaction record. The queue will be created containing an initial number of rows according to the initialisation parameters specified for the Consumer process associated with the queue (see [Queue Initialisation](#) for more information).

The data in this table is read and updated by the DataDistributor Consumer process and read by the process which reads the queues.

The primary key for this table is ENTRY_NO. This table has no referential integrity with other tables.

NOTE:

When creating the DataDistributor Queue tables, consideration must be given as to the required size of the entry data which needs to be accommodated. The size of the data records being handled will be dependent on the switch from which data is being received.

For example, consider whether data records from a HP Connex switch will exceed the default 8000 byte limit if Extended tokens are being logged.

Queue Initialisation

Queue table initialisation is performed by the DataDistributor Consumer process on initial start-up.

The DATADIST_CONTROL and DATADIST_QUEUE_*nn* tables must be created, as specified in section [Queue Table Structures](#), prior to starting the DataDistributor Consumer processes.

The DataDistributor Consumer process uses values specified in its configuration file to initialise the tables. The required values are:

- Queue Number (no default)
- Queue Size (default 1000)
- Queue Size Increment (default 50)
- Maximum Queue Size (defaults to Queue Size)
- Maximum Rows per Transaction (default 50).

On start-up the Consumer process will read the DATADIST_CONTROL table to determine whether a row exists for the specified Queue Number.

If a row exists, the Queue is considered initialised already, and processing will continue as described in section [Queue Processing](#) for more information.

If no row exists, the Queue will be considered to be un-initialised, and the Consumer process will initialise the queue tables as described in the following sub-sections.

- DATADIST_QUEUE_*nn*

The Consumer will insert the required number of rows, as specified by the configured Queue Size, into the DATADIST_QUEUE_*nn* table, where *nn* corresponds to the queue number, as follows:

ENTRY_NO	Sequential number starting from 1.
ENTRY_LENGTH	Set to 0.
ENTRY_DATA	BLOB value of zero length.

The rows will be inserted into the table in groups corresponding to the Maximum Rows per Transaction setting.

- DATADIST_CONTROL

Following the successful population of the DATADIST_QUEUE_*nn* table, the Consumer will insert a row into the DATADIST_CONTROL table as follows:

QUEUE_NO	As specified in Consumer configuration.
QUEUE_SIZE	As specified in Consumer configuration.
QUEUE_SIZE_INC	As specified in Consumer configuration.
QUEUE_SIZE_MAX	As specified in Consumer configuration.
WRITE_PTR	Equal to QUEUE_SIZE.
READ_PTR	Equal to QUEUE_SIZE.
EMPTY_SPACE	Spaces, padded to maximum length of column.

Queue Processing

This section describes the processing of the queues and control table that is to be performed by the DataDistributor Consumer and the Queue Reader processes.

Queue processing from the Consumer process perspective involves writing (via UPDATE) to the queue table and correctly maintaining the WRITE_PTR value within the control table.

Queue processing from the Queue Reader perspective involves reading from the queue table and correctly maintaining the READ_PTR value within the control table.

The WRITE_PTR will only be updated by the Consumer process.

During normal operation the READ_PTR will only be updated by the Queue Reader. The exception to this is during Queue Initialisation (see [Queue Initialisation](#)) when the Consumer process will set the initial value.

During processing, READ_PTR and WRITE_PTR values may exist in the database and in the memory of each program using the pointer. It is important for each process to know when to use its in-memory value to improve performance, and when to read/update the database table to ensure that correct values are used/updated.

Each process sets the pointer value to point at the last row processed. When a batch of rows is processed (written or read), the pointer is set to point at the last row processed when the batch is committed.

The following sub-sections describe the significant queue states that will be encountered during Consumer and Queue Reader processing.

- Initial Queue State

The initial queue state is $WRITE_PTR = READ_PTR = QUEUE_SIZE$.

This initial state is designed to be compatible with the Queue Rollover processing specified in the next section, such that writing and reading begin at row number 1, and defines a Queue Empty state.

- Queue Rollover State

Consumer

Prior to inserting a transaction, if $WRITE_PTR + 1$ is greater than $QUEUE_SIZE$, the Consumer process will rollover to the start of the queue and insert the transaction in row number 1.

Queue Reader

Prior to reading a transaction, if $READ_PTR + 1$ is greater than $QUEUE_SIZE$, the Queue Reader will rollover to the start of the queue and read the transaction in row number 1.

Queue Full State

Prior to inserting a transaction, if $WRITE_PTR + 1 = READ_PTR$, the Consumer process will not be able write the transaction to the queue.

NOTE:

The Consumer can therefore never cause $WRITE_PTR$ to become equal to $READ_PTR$ during normal processing.

Additionally, the Consumer cannot rollover to the start of the queue if $READ_PTR = 1$ (see Queue Rollover State).

These are considered **Queue Full** states.

- Queue Empty State

The Queue Reader process cannot proceed to read rows when $READ_PTR = WRITE_PTR$.

This is considered a **Queue Empty** state.

Consumer for Settlement

Consumer for Settlement feeds transaction data to FIS Settlement application (also known as CMS) through the creation and population of Data Sets that can then be imported into the Settlement Application.

The number of Data Sets that are created is dependent upon the nature of the Transaction Data Source from which the DataDistributor Producer is extracting the data.

- For Connex on HP NonStop, the data sets created correspond directly to the Connex on HP NonStop logs files. As such there will be many files created for import by Settlement on any one processing day, that is, as many as there are Connex on HP NonStop Log files created on that day.

- For IST/Switch, one data set will be created per IST/Switch processing day. The data set will be created at the start of the business day and closed once the Producer sends the EOD notification to the Consumer. The exception to this will be in situations where large volumes of transaction data require that multiple (>1) data sets are allocated per day as each allocated data set becomes full.

Data sets are allocated as FB (Fixed Block) with transaction records being padded with trailing spaces to ensure they are the correct (fixed) length.

NOTE:

When configuring the size of the datasets, consideration must be given as to the required size of the log record which needs to be accommodated. The size of the log records being handled will be dependent on the switch from which data is being received.

For example consider whether data records from a HP Connex switch will require increased record sizes if Extended tokens are being logged.

Data Sets are named according to the naming convention specified in the *OE DataDistributor Configuration Reference* - (Section 2 - [DataDistributor Producer for IST/Switch \(Solaris\)](#)).

The Consumer releases each completed data set to Settlement by placing an entry in a "Settlement Audit" file indicating the name of the data set to be imported. The location of the Audit file is specified in the Consumer configuration file.

Consumer for Received Queue

Data is received from the corresponding Producer for Delivery Queue process.

Data records received by the Consumer are inserted into the configured Received Queue table for subsequent processing by the appropriate application.

The table structure is as follows:

Column Name	Data Type	Null	PKey	Notes
ENTRY_NO	LARGEINT	N	Y	A sequential number that identifies the row. The value used will be that extracted from the DELIVERY_QUEUE for the item.
ROW_NO	INTEGER	N	Y	Zero if this is the only row for the ENTRY_NO. Value increases from one when the ENTRY_NO has more than one row.
ENTRY_STATUS	CHAR(1)	N	N	Will be ' ' when written by the Consumer. May be amended by subsequent processing as required.
ENTRY_TIMESTAMP	LARGEINT	N	N	A Julian timestamp that describes a GMT time. This value is converted from the ORACLE Timestamp recorded by the sending application when the item for delivery is inserted into the delivery queue on the sending platform.
ENTRY_TYPE	SMALLINT	N	N	For ease of identification of the data contained within the entry. This value is that extracted from the DELIVERY_QUEUE for the item.
ENTRY_LENGTH	SMALLINT	N	N	The total length of data for ALL rows with the same ENTRY_NO. This length will therefore match the length as specified for the item in the DELIVERY_QUEUE. Where an item spans multiple rows in this table, the length of each individual item can be determined from the VARCHAR data.
ENTRY_DATA	CHAR VARYING (4000)	Y	N	Contains a segment of a single record for delivery.

Checkpoint Processing

DataDistributor employs a Checkpoint mechanism that enables the state of data transfer to be recorded at periodic intervals, thus enabling transfer to continue from the correct position following scheduled or unexpected termination.

The Checkpoint information is recorded in either a process-specific unstructured file or within a process-specific row within a database table (currently Oracle database only).

Both Consumer and Producer processes record checkpoint information. The content of the checkpoint file or row is dependent upon the type of DataDistributor process and the nature of the target application or data source with which it is interacting.

However at the very least, the checkpoint data that is stored will contain sufficient 'positional' data to allow data transfer to resume at the correct point following an interruption. Typically this information consists of the name of a file from which data is being transferred and a position within that file. The interpretation of "file" and "position" however depends upon the nature of the data source from which data is being extracted. In some instances, positional information is not required at all, as other mechanisms are employed to ensure that data is correctly sent and received.

Details of the checkpoint structures data stored are given in section [Checkpoint Structure](#). Details of the position information used by each Producer are given in section [Position Information](#).

From a restart perspective it is the Consumers position information that is taken as being "correct". Consequently, this information is sent to the Producer in the initial request that the Consumer sends to the Producer, to enable the Producer to position itself correctly within the data source from which data is to be extracted.

The Consumer also includes its checkpoint information in the acknowledgement message that it sends periodically (after a configurable number of data transfer messages have been received) to the Producer. The Producer stores this checkpoint information within its own checkpoint data. This data is stored in this fashion in order to allow for potential loss of checkpoint information at the Consumer end. The Producer will send its version of the Consumer checkpoint data to the Consumer during a Session Negotiation dialogue following connection from the Consumer. The Consumer will only use this information if it detects that its own checkpoint data is absent.

Checkpoint Structure

The checkpoint data stored by each DataDistributor process type is described in the following sections. The data stored is dependant on the type of process.

Each Producer process will store the checkpoint data that it receives from its connected Consumer in an acknowledgment message, prior to storing any checkpoint data of its own.

Consumer for DataNavigator®

The following checkpoint data is stored:

Field	Size	Usage
Position	8	Position (as provided by Producer).
File Name	128	File name (as provided by Producer).
File Completed	4	File marked as complete.
Record Count	4	Record count for current file.

Consumer for Settlement

The following checkpoint data is stored:

Field	Size	Usage
Position	8	Position (as provided by Producer).
File Name	128	File name (as provided by Producer).
File Completed	4	File marked as complete.
Record Count	4	Record count for current file.
Position	8	Position in Settlement File.
File Name	128	Settlement File Name.
Last Date	8	Date of last Settlement file (as used in filename).

Consumer for Received Queue

The following checkpoint data is stored:

Field	Size	Usage
Position	8	Position (as provided by Producer).
File Name	128	File name (as provided by Producer).
File Completed	4	File marked as complete.
Record Count	4	Record count for current file.

Producer for Connex on HP NonStop

The following checkpoint data is stored:

Field	Size	Usage
Consumer data	variable	Storage of checkpoint data sent by Consumer.
Record Count	4	Record count for current file.
Current Log	4	File in progress is “current” log (True/False).
900 Count	2	Number of 900 records counted in file.
901 Count	2	Number of 910 records counted in file.
Previous Record 901	4	Last record processed was a 901 (True/False).
Previous 900 Close Time	6	Close Time recorded in last 900 (Connex on HP NonStop 48 bit timestamp).
Last Timestamp	6	Timestamp (48 bit) of last extracted record.
Amount Hash	8	Hash of 400 record amounts.
Timestamp Hash	4	Hash of record timestamps.
Last Close Timestamp	6	Close Timestamp (48 bit) of last file processed.

Producer for Delivery Queue

The following checkpoint data is stored:

Field	Size	Usage
Consumer data	variable	Storage of checkpoint data sent by Consumer.
Log Status	2	Last status indicator sent to Consumer.

Producer for IST Switch

Field	Size	Usage
Consumer data	variable	Storage of checkpoint data sent by Consumer.
Log Status	2	Last status indicator sent to Consumer.
Record Count	4	Record count for control period (if controls configured).
Amount Hash	8	Amount Hash for control period (if controls configured).
Open Timestamp	16	Timestamp for start of control period (if controls configured).
Current Business Date	4	Current IST Switch Business Date.

Producer for Connex on IBM®

The following checkpoint data is stored:

Field	Size	Usage
Consumer data	variable	Storage of checkpoint data sent by Consumer.
Log Status	2	Last status indicator sent to Consumer.

Producer for BASE24

The following checkpoint data is stored:

Field	Size	Usage
Consumer data	variable	Storage of checkpoint data sent by Consumer.
Record Count	4	Record count for current file.
Current Log	4	File in progress is “current” log (always TRUE).
Processing Date	7	Date of the file being currently processed.
Reset Flag	4	Indicates whether the Producer has been reset.
Reset Start Time	8	Tandem 64 bit timestamp representing the reset start time.
Reset File Name	34	The name of the file in which to restart.
Reset Position	8	Position in file to restart.
Reset End Time	8	Tandem 64 bit timestamp representing the reset end time.

Field	Size	Usage
Pre Reset Filename	34	Name of the file being processed prior to a reset.
Pre Reset Position	8	Position in file being processed prior to a reset.
Control Record Count	32	Number of records processed in current Control period.
Control Hash	64	Hash of timestamps from records in current Control Period.
Recovery Record	56	Current recovery record.

Position Information

Producer for Connex on HP NonStop/BASE24

Field	Size	Usage
Position	8	Position within file being transferred. Contents depend on whether file is Entry Sequenced (ES) or Relative (R) format. 1st 4 bytes contain record number. 2nd 4 bytes contain either block number (ES) or previous record length (R).
File Name	128	Name of file being transferred.

Producer for Delivery Queue

Field	Size	Usage
Position	8	Last datadist_delivery_queue.entry_number value for records in transfer buffer.
File Name	128	Fixed value equal to loggerId value in Producer configuration.

Producer for IST Switch

Field	Size	Usage
Position	8	Contains the current business date (1 st 4 bytes) and the current log status (2 nd 4 bytes).
File Name	128	File name is of the form SHCYYYYMMDD where YYYYYMMDD is the current business date.

Producer for Connex on IBM®

Field	Size	Usage
Position	8	Contains 0. No relevant position information for this Producer. Position is maintained by the MQ Control Task checkpoint mechanism.
File Name	128	Fixed value equal to loggerId value in Producer configuration.

4

Troubleshooting

This section describes how to find and determine the possible cause of processing failures encountered in the use of DataDistributor.

Processing failures generally fall into two categories.

- Failures/errors encountered during initialisation of the process, that is, the process fails to start successfully.
- Failures encountered during normal processing, that is, once the process is up and running.

These types of failure are described in more detail in the following sections.

Process Initialisation Failures

Process initialisation failures are generally as a result of errors encountered during loading of the process configuration file.

Processing initialisation failures are reported to one of two places, depending upon the point at which the failure occurs:

- To a file called INITFAIL, which will be found in a location that is dependent on the operating environment in which the process is running:
 - On Windows® the file will be found in the same folder that contain the process configuration file.
 - On the Connex on HP NonStop, the file will be found in the default sub-volume for the process.
 - On Solaris, the file will be found in the directory from which the process was started.
 - On the IBM® Mainframe, the file will be found in the location specified by the DD Name entry in the process start-up JCL.
- To the process event log, the location of which will be dependent on the nature of the EventLogger component specified in the process configuration file and the type of event being logged. See Chapter 2 in the *OE DataDistributor Configuration Reference* for a discussion of the available Event Logger configurations and event type filtering.

The INITFAIL file will be used if the error encountered occurs such that the configured event logger component has not yet been initialised.

In all other cases errors will be logged to the appropriate event log output location.

Generally process initialisation errors fall into one of the following categories:

- Failure to find and or load the specified process configuration file.
- Failure to find and or load the process Event Template file.
- Failures to parse a specified configuration file - this is usually caused by syntax errors introduced into the XML configuration file.
- Missing or invalid attributes specified within the configuration file for which no meaningful default value can be supplied.
- Security failures associated with connections to specified databases.

The OE DD Event Message Guide provides a description of all messages that can be logged by DataDistributor giving the likely cause and possible action to be taken.

Process Runtime Errors

This category of errors refers to those that occur once the DataDistributor system (Producer and Consumer) has been successfully configured and started such that connection has been made and transaction data has begun to flow.

Process Runtime errors generally fall into one of three sub-categories:

- Network TCP/IP IO Errors.
- Producer errors accessing or extracting data from the configured Data Source.
- Consumer errors accessing or distributing data to the configured Target System.

DataDistributor has been designed wherever possible to recover from such errors internally and without operator intervention, such that data transfer continues without significant interruption.

The DataDistributor checkpoint and other data transfer control mechanisms are designed to ensure that recovery occurs without loss of data (although under some circumstances, recovery may result in the re-transmission of data in order to guarantee this).

Only repeated or persistent errors are likely to cause a DataDistributor process to terminate abnormally. In these cases it is necessary to consult the OE DataDistributor Event Messages Guide in order to ascertain the likely cause and possible resolution of the problem. Once the problem has been cured, the relevant DataDistributor process can be restarted. As above, the DataDistributor checkpoint mechanisms should ensure that no data is lost.

The following sub-sections describe some of the runtime errors that may be encountered and the expected action that DataDistributor undertakes in order to effect recovery. Additionally, suggested action to be taken by the operator in the event that the problem is persistent is given.

Error Recovery Processing

On encountering an error during processing a DataDistributor Process will attempt recovery by terminating the current connection session with its corresponding process.

Following the initiation or detection of the loss of connection, each process will then return to a state that allows the transfer of data to resume from the last known position:

- The Producer process returns to a state of waiting for a connection from the Consumer.
- The Consumer returns to a state of attempting to connect to its Producer.
- The Consumer uses its checkpoint data to resume the transfer from the correct position.

Network Errors

The default DataDistributor behaviour upon encountering TCP/IP Network errors during processing is to attempt to reconnect indefinitely. Once connection has been made, data transfer will resume from the last known position as indicated by the Consumer checkpoint data.

The connection retry processing is controlled by the Consumer configuration attributes on the ProducerConnection component, namely:

Attribute	Description
netIoErrorMax	Specifies the number of consecutive IO errors that will be tolerated before process termination. Default no maximum (retry indefinitely).
netIoErrorRetryInterval	Specifies the number of seconds to delay before attempting to reconnect.

Initial Connection Failures

It is the Consumers responsibility to make the initial connection to its configured Producer.

The connection configuration is specified in terms of a TCP/IP Address and port number, which correspond to the address of the host upon which the Producer is running and the port upon which the process will accept connections.

Connection failures typically fall into one of two categories:

- **Connection Timed Out**

Typically this indicates a network routing error, such that a connection cannot be made from the host upon which the Consumer is running, to the host upon which the Producer is running.

- **DataDistributor will:**
 - Retry the connection attempt following a configured delay.
- **The Operator should check:**
 - That the TCP/IP Address specified in the Consumer configuration file is the correct address for the target Producer host.

- **Connection Refused**

Typically this indicates that the Producer is not listening on the port specified in the Consumer configuration.

- **DataDistributor will:**
 - Retry the connection attempt following a configured delay.
- **The Operator should check:**
 - That the required Producer process is running on the target host.
 - That the Producer and Consumer configuration matches in terms of the port specified for the connection.

Connection Failures During Processing

During data transfer, the loss of connection will be detected on either a TCP/IP socket **send** or **recv** operation.

Typically, the loss of connection will be detected immediately on a **send** operation.

For a **recv** operation, DataDistributor may only detect the loss of connection after a configurable timeout has occurred. The length of time for which a process will wait for a **recv** operation to complete is governed by the **readTimeout** value on the appropriate connection component in the process configuration file.

Following detection of the loss of connection:

- **DataDistributor will:**
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
- **The Operator should check:**
 - The Producer and Consumer event log output for indication of any other error that may have caused one or other of the processes to terminate the connection.

Connection Timeouts and Poll Intervals

DataDistributor processes employ timeout values for TCP/IP operations, which are designed to ensure that the processes detect abnormalities in the flow of data in a timely fashion.

The **readTimeout** values are applied to TCP/IP Socket **recv()** operations to ensure that expected messages are received in timely fashion.

Typically DataDistributor interfaces are operating at high-volumes, with data transfer messages (from the Producer) and acknowledgement messages (from the Consumer) being exchanged continuously.

On occasion, it may transpire that the volume of data being generated becomes low enough that the Producer has no data to extract. This is also likely for interfaces that are by their very nature of low volume.

In order to ensure that the Consumer is not continually asking the Producer to send data when none is available to be sent, the Consumer employs a configurable delay, which it introduces when the Producer indicates that it has no data to send.

The default values for the Consumer poll interval and the Producer readTimeout are set such that the poll interval is less than the readTimeout. This is to ensure that the Consumer sends a prompt for more data before the Producer times out waiting for that prompt.

On occasion, it may be necessary to fine-tune these settings to ensure that network trip time for the messages does not cause the Producer to repeatedly time out waiting for the next request from the Consumer.

Producer Data Source Errors

The type of errors encountered will be dependant upon the nature of the configured Data Source.

Connex on HP NonStop Producer

Problems encountered are likely to relate to accessing the Connex on HP NonStop Log Control file (LGWARM) or a Connex on HP NonStop Log File.

On encountering an error reading the LGWARM file:

- **DataDistributor will:**
 - Report the error to the event log.
 - Terminate pending resolution of the problem, since the Producer cannot continue without being able determine which Connex on HP NonStop log file to process.

- **The Operator should check:**
 - The location of the LGWARM file is correctly specified.
 - The identity of the Logger process for which the Producer is configured is correctly specified. The identity is case-sensitive.

On encountering an error reading a log file:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.

IST Producer

Problems encountered are likely to relate to accessing the IST Switch Database using the IST Database API.

On encountering a problem accessing the IST Switch Database:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - Check the IST DB API debug trace (if configured) for details pertaining to the problem.
 - User and Password details specified in the configuration are correct.
 - Security permissions for the DB are correct.
 - Verify that the correct business date is being extracted.
 - Verify that all expected EOD records have been generated and that the Producer extraction is not inadvertently “stuck” on a particular business date.

Connex on IBM® Producer

Problems encountered are likely to relate to the interaction with the MQ Control Task via the Message Delivery System (MDS).

On encountering a problem either sending or extracting messages from the MDS Queue:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - Check the configuration for the relevant MDS queues (MQ Control Task and Producer Task) are correct.

Delivery Queue Producer

Problems encountered are likely to relate to accessing the configured Delivery Queue table.

On encountering a problem accessing the Delivery Queue table:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - That the table specified in the configuration exists.
 - User and Password details specified in the configuration are correct.
 - Security permissions for the specified table are correct.

BASE24 Producer

Problems encountered are likely to relate to accessing the BASE24 Log File.

On encountering an error reading a log file:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [“Error Recovery Processing” on page 39](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.

Consumer Target System Errors

The type of errors encountered will be dependant upon the nature of the configured Target System.

Settlement Consumer

Problems encountered are likely to relate to accessing the Settlement Import files.

On encountering a problem accessing the Settlement Import files:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - That the specified Settlement import file is not full.
 - Security permissions for the creation of the file.

DataNavigator® Consumer

Problems encountered are likely to relate to accessing the DataDistributor Queue database.

On encountering a problem accessing the DataDistributor Queue Database tables:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - That the required queue (DATADIST_QUEUE_nn) table specified in the configuration exists.
 - That the DATADIST_CONTROL table exists and contains an entry for the required queue.
 - User and Password details specified in the configuration are correct.
 - Security permissions for the specified table are correct.
 - Check whether the queue is full as described in section [Queue Processing](#).

Received Queue Consumer

Problems encountered are likely to relate to accessing the configured Received Queue database table.

On encountering a problem accessing the DataDistributor Received Queue tables:

- **DataDistributor will:**
 - Report the error to the event log.
 - Follow the recovery procedure outlined in section [Error Recovery Processing](#).
 - If the problem is persistent, the Producer may terminate, pending investigation of the problem.
- **The Operator should check:**
 - The event log for details of the reported problem.
 - That the define =T_RECEIVED_QUEUE exists or is specified on the PATHWAY Server definition, and that this references the correct table.

Security permissions for the specified table are correct.



Trace Output

Transaction Trace Output

An optional (via Configuration) Transaction Trace mechanism has been provided, to allow the output of formatted information regarding records processed via DataDistributor.

This trace output is configured by specifying a log record processing rule within the process configuration file. See Chapter 2 in the *OE DataDistributor Configuration Reference Guide* (FormatTxnRecord) for a description.

This rule can be configured for both the Producer itself, in order to provide formatted output of the transactions that have been extracted from the datasource, and for the connected Consumer in order to provide formatted output of the transactions which have been received, prior to their being forwarded to the required target systems.

This feature has been provided largely as a tool to aid testing - to provide a mechanism through which the data extraction and transfer can be verified. It is not expected that this mechanism should be used within a Production environment, since it generates a large amount of output and could potentially lead to degradation in performance.

6

HP NonStop Producer Statistics Reporting

Extraction Lag Reporting

The Connex on HP NonStop and BASE24 Producers provides a mechanism for reporting on the delay between the logging of records to a log file by a Logger process and their extraction from that file by the Producer.

The assumption is that in general the Producer will be reading records from the log file almost as soon as those records are written to the file by the logger. Reporting of the lag between writing and extraction is reported in one of two ways.

Periodically (as specified by the `txnRateReportInterval` setting on the `LogFileSet` component), the Producer will log statistics relating to the rate of extraction from the current file. This report will include the current rate of transfer (txns per second as measured over the reporting interval) and the extraction lag in centiseconds.

Additionally, it is also possible to set a reporting threshold for the lag between logging and extraction, such that if the lag exceeds the specified threshold, a message will be logged indicating this fact.

The lag reporting is controlled through the Producer configuration (`LogFileSet` component), which allows the setting of the following values:

- `lagReportThreshold` (centiseconds)
- `lagReportIncrement` (centiseconds)
- `lagReportInterval` (seconds).

Periodically the Producer will calculate the difference between the current timestamp and the log timestamp of an extracted record.

If this difference is greater than the `lagReportThreshold` value, an event message will be logged by the Producer indicating this fact.

Subsequently, the Producer will report as follows:

- If the lag increases or decreases by the `lagReportIncrement` value, an event message will be displayed.

- If the lag remains "constant" (that is does not increase or decrease by the increment value), after lagReportInterval seconds have elapsed.

The Producer will also log an event once the lag falls back within the threshold value.

So for example, if the threshold value is 5, and the increment is 1. A message will be logged once the lag exceeds 5 seconds. Subsequent messages will be logged if the lag gets better or worse by at least a second. Once the lag falls below 5 seconds, a message will be logged.



Consumer Load Statistics

The Consumer can be configured to collect statistics relating to how long it is taking to process records received from the Producer.

This is achieved by setting a non-zero value for the `loadReportThreshold` attribute for the Consumer component (see Chapter 2 in the *Configuration Reference Manual*).

If a non-zero value is set, the Consumer will record timestamps for various stages (milestones) involved in the processing of a set of records received from the Producer. If the overall time taken to process the records exceeds the `loadReportThreshold` value, the Consumer will log event messages which display the details of the milestones and additional statistics relating to the processing of the records, as follows:

The reported Milestones are as follows:

Milestone	Description
0	Wait for a message
1	Message received
2	Start processing buffer
3	Updated all targets
4	Start EOF Processing (if appropriate)
5	EOF Processing Complete (if appropriate)
6	Start Checkpoint update
7	Checkpoint update Complete
8	Message processed (may include acknowledgement sent)

Additionally, statistics relating to the load into each Target System will be reported displaying the following values:

- Total record load time
- Shortest record load time
- Longest record load time
- Average record load time.



Record Filtering

The DataDistributor configuration mechanism allows the introduction of certain additional processing, over and beyond the transfer of data between end-point.

This additional processing includes the ability to filter records extracted by a Producer such that certain records are not sent.

This level of processing can be configured for both Producer and Consumer, but typically will be performed by the Producer, so that unwanted records are not needlessly sent to a Consumer.

The Filtering Mechanism consists of two main components:

- An XML definition of the record to be examined
- Rules which make filtering choices based on the value of fields in the record

The mechanism is best explained by an example, which follows in the next sub-section. Full details of the configuration components referenced in the example are given in the *OE Configuration Reference*.

Filtering Connex on HP NonStop Log Records

The following example XML shows filtering of Connex on HP NonStop Log records by a Producer, based on the Message Code value contained in the standard Connex on HP NonStop Log Header. The example shows the configuration required to allow all 400, 466 and 600 Connex Messages to be sent to the Consumer, whilst all other messages are discarded (that is, not sent).

The XML begins by defining the structure of the Connex on HP NonStop Log Header as a DDL Message, which consists of DDL Field definitions of the fields in the header. A DDL Sequence is a named collection of fields.

Each field is of a certain type (numeric, character etc.) and size. This enables the processing to determine how to interpret the extracted messages and therefore extract the appropriate field values which are going to be used for filtering.

There then follows a GetField rule definition. In this simple example, there is a single GetField rule, called getMsgCode. More complex filtering requirements will make use of additional GetField rules. The rule defines the name of the field to be examined, and the

name of the DDL Input Message within which that field is to be found. In this example, the field is the **internalHeader.msgCode** field, in the **connexLogHeader** message definition. A dot notation is used to define sub-fields within a sequence.

For each extracted record, the `getMsgCode` rule, will determine the Message Code value. The `ConditionalRule` entries, embedded within the body of the `getMsgCode` rule, provide potential values for Message Code upon which the processing is to filter. In this example, we have chosen to filter upon values 400, 466 & 600. When a record with a matching value is detected, the corresponding rule, as defined within the body of the `ConditionalRule`, is invoked. In each case, the rule invoked is `forwardRecord`. This corresponds to the `ForwardRecord` processing rule, which is simply a mechanism for marking the current record as eligible for inclusion in the buffer of data to be sent to the Consumer. If the `msgCode` value for the record does not match with any of the `ConditionalRule` values, the default `discardRecord` rule is invoked. This corresponds to a `DiscardRecord` processing rule, which is simply a mechanism for marking the current record as ineligible for inclusion in the buffer of data to be sent.

Finally, the “allRecords” `ProcessingRuleList`, is the entry point for the set of rules contained in the file. This rule is referenced in an individual Producer process configuration file (via the `firstRule` attribute on the `LogFileSet` component). The `ProcessingRuleList` performs each `<rule>` specified in sequence.

```

<rules>
<DDLMessage name="connexLogHeader">
  <DDLSequence name="logHeader">
    <DDLConnexionHPNonStopTimestamp name="loggerTimestamp" length="6" />
    <DDLConnexionHPNonStopTimestamp name="timestamp" length="6" />
    <DDLField name="node" length="8" />
    <DDLIntegerField name="mftVersion" length="4" />
  </DDLSequence>
  <DDLSequence name="internalHeader">
    <DDLIntegerField name="msgCode" length="2" />
    <DDLIntegerField name="step" length="2" />
    <DDLField name="source" length="6" />
    <DDLField name="destination" length="6" />
    <DDLField name="junk" length="8" />
    <DDLIntegerField name="acknowledge" length="2" />
    <DDLIntegerField name="msgLength" length="2" />
  </DDLSequence>
</DDLMessage>

<GetField name="getMsgCode" field="internalHeader.msgCode"
input="connexLogHeader" byteReverse="0">
<ConditionalRule match="400">forwardRecord</ConditionalRule>
<ConditionalRule match="600">forwardRecord</ConditionalRule>
<ConditionalRule match="466">forwardRecord</ConditionalRule>
discardRecord
</GetField>

<ProcessGeneric name="processGeneric" />

<DiscardRecord name="discardRecord" />
<ForwardRecord name="forwardRecord" />

<ProcessingRuleList name="allRecords">
<rule>processGeneric</rule>
<rule>getMsgCode</rule>
</ProcessingRuleList>
</rules>

```

9

Connection Validation

DataDistributor Producer provides basic authentication mechanisms for validating connections received from remote parties are from known sources (that is, validly configured DataDistributor Consumers).

Authentication operates at two levels:

- TCP/IP Address validation
- Consumer processes identification validation.

These are described in the following sub-sections.

NOTE:

It is to be expected that for the most part, standard network security mechanisms will be in place to prevent unwanted connections from unknown end-points being made. This extra level of authentication is provided to assist in certain circumstances where network security mechanisms will not be sufficient to provide the necessary protection, such as:

- Circumstances where multiple Consumer test systems are configured to connect to the same Producer sub-system. TCP/IP address and/or Consumer identity validation can be utilized to prevent multiple Consumer system being allowed to connect simultaneously, causing spurious and unwanted issues.
- Circumstances where network probes are being used to validate the existence of the Producer process by making periodic connections to the Producers main listening port. TCP/IP address validation can be utilized to allow probe connections to be made without causing connectivity issues for the main Consumer process.

TCP/IP Address Validation

DataDistributor Producer processes can be configured to validate that connections they receive are from known TCP/IP addresses.

Lists of acceptable TCP/IP addresses can be maintained in an XML configuration file which is referenced in the main Producer configuration file via the `validConnectionDetails` attribute as follows:

```
<ConsumerConnection  name="consumerConnection"
                      byteReverse="0"
                      port="9191"
                      windowSize="5"
                      bufferSize="64000"
                      tcpIpProcessName="$zb01b"
                      tcpIpBufferSize="32000"
                      compressionMode="1"
                      validConnectionDetails="$DISK1.ADVGDDCF.VCONXML"
                      validateWhenConnected="1" >
  <validIdentifiers name="identifiers">
    <identifier>cns01</identifier>
  </validIdentifiers>
</ConsumerConnection>
```

The contents of the XML file should be of the form displayed below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- Valid Connections -->
<connections>
  <connectionList name="primaryConnections">
    <!-- list of addresses which are valid for connecting for the
    -->
    <!-- primary purpose of the server e.g. Consumer to Producer
    -->
    <address>172.20.76.92</address>
  </connectionList>
  <connectionList name="probeConnections">
    <!-- list of addresses which are valid for connecting
    -->
    <!-- as network probes -->
    <address>172.30.49.105</address>
  </connectionList>
</connections>
```

Two lists of valid addresses can be defined.

The first specifies a list of TCP/IP addresses from which “primary” connections can be received. Primary connections are those from actual Consumer processes to which data is to be sent.

The second specifies a list of TCP/IP addresses from which “probe” connections can be received. Probe connections are those which are made simply for the purpose of testing for the existence of the Producer (that is, is the Producer running and listening on the required port).

Connections received from valid Primary addresses will be accepted and maintained for the purposes of sending data.

Connections received from valid Probe addresses will be accepted and immediately disconnected. This will occur “silently” and such connection activity will not register in the event log.

Connections from any other addresses will not be accepted (will be accepted and dropped) with an event log message being generated which will warn of the unexpected connection.

Additionally, the Producer will log a warning message when a connection is received from a valid Primary TCP/IP address which differs from the last known TCP/IP address to be accepted by the Producer.

If no validConnectionDetails are configured, TCP/IP address validation will not be performed.

Consumer Identity Validation

DataDistributor Producer processes can be configured to validate that connections they receive are from known Consumer processes.

Lists of acceptable Consumer identities can be maintained within in the main Producer configuration file via the validIdentifiers sub-element as follows:

```
<ConsumerConnection  name="consumerConnection"
                        byteReverse="0"
                        port="9191"
                        windowSize="5"
                        bufferSize="64000"
                        tcpIpProcessName="$zb01b"
                        tcpIpBufferSize="32000"
                        compressionMode="1"
                        validConnectionDetails="$DISK1.ADVGDDCF.VCONXML"
                        validateWhenConnected="1" >
  <validIdentifiers name="identifiers">
    <identifier>cns01</identifier>
    <identifier>cns02</identifier>
  </validIdentifiers>
</ConsumerConnection>
```


Each identifier element specifies the value of a valid Consumer identifier. This value is supplied in each message sent from a Consumer process to the Producer and will be validated when the first message is received from the Consumer following connection being successfully established.

The identifier value must match that configured for the Consumer via the identity attribute on the Consumer component, for example:

```
<Consumer  name="consumer"
            identity="cns01"
            node="\PXD"
            logger="LG03"
            fileIoErrorMax="3"
            netIoErrorMax="3"
            netIoErrorRetryInterval="30"
            requestDelayInterval="30"
            txnStalenessReportLimit="10"
            asciiEbcdicConversionMode="1">
  <TargetSystem>
  </TargetSystem>
</Consumer>
```

The values must match exactly (case-sensitivity applies) in order for the connection to be accepted.

If a message is received from the Consumer which contains a value which is not in the list of valid identifiers, the connection will be dropped by the Producer and an event message logged warning of the fact.

If no validIdentifiers are configured, identity validation will not be performed.



PA-DSS Compliance and Cryptography Services

As part of PA-DSS Compliance changes, OE DataDistributor has added support for Encryption and Decryption of sensitive data via Cryptographic Service functions.

The Cryptographic Services provided are dependent on the platform and nature of the DataDistributor Producer and Consumer processes which are deployed.

The PA-DSS requirements require that where DataDistributor processes are responsible for committing sensitive data to disk, that data must not be held in the clear. It is therefore the responsibility of the DataDistributor Consumer process to encrypt log records where necessary, either in full, or in part, prior to writing them to disk.

The requirement to encrypt data on the Consumer side is dependent on the way sensitive data is held on the Switch platform from which the Producer extracts and transfers the data. In summary, the Consumer is required to encrypt data received from HP/Connex and IST/Switch, but not IBM/Connex.

The following sections discuss these items in more detail.

Producer Platform Cryptography

HP/Connex

Sensitive data within the HP/Connex Log files is stored in encrypted form whereby the entire log record is encrypted. Encryption is performed using HP/Connex PA DSS Cryptography functions and the associated Key Management System (KMS). Detailed discussion is beyond the scope of this document.

The HP/Connex Producer is required to decrypt the log records that it reads from the log files. This is performed using HP/Connex PA DSS decryption routines.

The Producer must be configured to perform this decryption by specifying that the logs from which it is extracting records are encrypted. This is achieved by adding the encryptedData attribute to the TandemLogFile configuration component as follows:

```
<TandemLogFile  name="logFile"
                recordSize="4072 "
                endOfFileWaitInterval="50 "
                encryptedData="1 " />
```

See the *OE DataDistributor Configuration Reference* for further details.

The Consumer that receives the data from the HP/Connex Producer is therefore required to encrypt that data prior to committing those records to disk.

IBM/Connex

Sensitive data within the IBM/Connex Log files is stored by encryption of specific fields within the data using the Integrated Cryptographic Service Facility (ICSF).

The IBM/Connex Producer is not required to decrypt the log records that it reads from the log files. Instead the log records are transferred to the Consumer as-is and written to disk by the Consumer in the same fashion. It is the responsibility of the receiving application (e.g. DataNavigator) to decrypt the encrypted fields as required.

IST/Switch

Sensitive data within the SHCLOG database is stored by encryption or tokenization of specific fields within the data.

The Database API which the Producer uses to perform extraction of data from the SHCLOG database presents the records to the Producer in unencrypted (or de-tokenized) form.

The Consumer that receives the data from the IST/Switch Producer is therefore required to encrypt that data prior to committing those records to disk.

Consumer Platform Cryptography

Cryptographic Services are provided on the Consumer platform via the use of a configurable cryptographic library component and a configurable processing rule that enables the encryption (or decryption) of records as required.

The specific cryptographic library implementation is dependent on the nature of the platform on which the Consumer is operating.

On the z/OS platform the cryptography is provided using the z/OS Integrated Cryptographic Service Facility.

On UNIX platforms and Windows (Win32) the cryptography is implemented using the OpenSSL Cryptographic library.

Regardless of the platform, a Key Generator Utility is provided to enable generation, storage and optional export of data encryption keys.

Each record is encrypted either wholly or in part; an encryption header is provided at the start of each record which details the nature of the encryption performed.

These are discussed in the following subsections.

Encryption/Decryption Processing Rules

Encryption (and Decryption) of records is provided as a configurable processing rule which must be invoked either as an individual rule or as part of a Processing Rule List.

The encryption or decryption will be performed using services provided by the configured cryptographic library component (see following sections).

The rules provided are as follows:

```
<DecryptRecord name="decryptRecord" />
<EncryptRecord name="encryptRecord" />
```

These must be specified within either the main process configuration file or within a rules file, for example:

```
<EncryptRecord name="encryptRecord" />
<ConnexTxnDistributor name="txnDistributor"
    firstRule="encryptRecord"/>

or

<ConnexTxnDistributor name="txnDistributor"
    rulesFile="c:\ddtest\crypto\rules.xml"
    firstRule="allRecords"/>
```

where the contents of the rules file are similar to the following:

```
<EncryptRecord name="encryptRecord" />
<ProcessingRuleList name="allRecords" >
    <rule>formatRecord</rule>
    <rule>encryptRecord</rule>
</ProcessingRuleList>
```

Encryption Header

There are two versions of the encryption header currently supported for inter-operability between encrypting and decrypting platforms.

DataDistributor supports both Version 1 and Version 2 levels of encryption and decryption. The version to be used is determined by setting of the version attribute on the appropriate Crypto Library component (default is version 1).

Version 1 Header

The Cross-Platform PA-DSS Header is shown in the following table:

Field#	Field Name	Offset	Length	Attr	Description
1	Eye Catcher	0	6	Char	Value ">DSS*<"
2	Version	6	2	Binary	Value 1 - This version of the header.
3	Encryption Key Check Value	8	4	Char	Character Representation of Hexadecimal check digits of the encryption key.
4	Encryption Method	12	2	Binary	1 indicates 3DES CBC.

Field#	Field Name	Offset	Length	Attr	Description
5	Length	14	2	Binary	Length of the decompressed and decrypted message.
6	Encryption Compression Method	16	2	Binary	Value 0 Uncompressed Value 1 Compressed.
7	Compressed Length	18	2	Binary	Length of encrypted/ compressed record.
8	Data	20	4070	Char	Encrypted and potentially compressed data.

Version 2 Header

The Version 2 header provides enhancements which allow:

- Identification of the encrypting platform
- Partial encryption of records from a specific offset and for a specific length

The Cross-Platform PA-DSS Header is shown in the following table:

Field#	Field Name	Offset	Length	Attr	Description
1	Eye Catcher	0	6	Char	Value ">DSS*<"
2	Version	6	2	Binary	Value 2 – This version of the header.
3	Encryption Key Check Value	8	4	Char	Character Representation of Hexadecimal check digits of the encryption key.
4	Encryption Method	12	2	Binary	1 indicates 3DES CBC.
5	Length	14	2	Binary	Length of the decompressed and decrypted message.
6	Encryption Compression Method	16	2	Binary	Value 0 - Uncompressed Value 1 - Compressed.
7	Compressed Length	18	2	Binary	Length of encrypted/ compressed record

Field#	Field Name	Offset	Length	Attr	Description
8	Source Platform	20	2	Binary	0 - DD FTP Pipe Process 1 - CMS 2 - HP/Connex 3 - IBM/Connex 4 - DD Consumer 5 - DataNavigator.
9	Encrypted Field Offset	22	2	Binary	Starting offset of the encrypted data field.
10	Encrypted Field Length	24	2	Binary	Length of the data to be encrypted. This must be a multiple of 8 bytes.
11	Data	26		Char	Encrypted and potentially compressed data.

ICSF Cryptography Library

Configuration of the ICSF services for use on the z/OS platform is achieved by adding the ZosICSFCryptoLibrary configuration component to the appropriate configuration file - either the main configuration file for the process or within a separate rules file, which can be shared amongst processes.

Full details of the configuration options are provided in the *OE DataDistributor Configuration Reference*.

The following example shows the configuration for a Consumer process (sourcePlatform is 4) which will enable encryption (or decryption) of the first 56 bytes of each record processed.

The key to be used is determined by reading the configured key extract file (see section [“ICSF Configuration and Output” on page 66](#)).

```
<ZosICSFCryptoLibrary name="cryptoLibrary"
    sourcePlatform="4"
    version="2"
    encryptionOffset="0"
    encryptionLength="56"
    keyExtract="// 'D96658.CNSKEY' " />
```

OpenSSL Cryptography Library

Configuration of the OpenSSL services for use on the UNIX and Windows platforms is achieved by adding the OpenSSLCryptoLibrary configuration component to the appropriate configuration file - either the main configuration file for the process or within a separate rules file, which can be shared amongst processes.

Full details of the configuration options are provided in the OE DataDistributor Configuration Reference.

The following example shows the configuration for a Consumer process (sourcePlatform is 4) which will enable encryption (or decryption) of the first 56 bytes of each record processed.

The key to be used is determined by reading the configured key repository. The key within the repository is itself encrypted with a Master Encryption Key which is generated using a secure algorithm based on a string which is provided in the configured seed file (the value can be any string value but must match that provided when the key was generated - see section [“OpenSSL Configuration and Output” on page 67](#)).

```
<OpenSSLCryptoLibrary name="cryptoLibrary"
    sourcePlatform="4"
    version="2"
    encryptionOffset="0"
    encryptionLength="56"
    seedFile="c:\ddtest\crypto\seed.txt"
    keyRepository="c:\ddtest\crypto\keys" />
```


Key Generator Utility

Overview

The DataDistributor Key Generator Utility is provided to enable generation of keys for use with the appropriate Cryptographic services used to perform encryption and decryption of records.

Operation

z/OS Platform

The Key Generator Utility is invoked by submitting a JCL file similar to that shown below:

```
//DDKYG01 JOB (W,K21415S,G2SE),
//          'C EXEC',
//          CLASS=T,
//          MSGCLASS=A,
//          NOTIFY=&SYSUID
/*XEQ DDSNBDEV
/*JOBPARM S=A005
/*ROUTE PRINT LOCAL
/* Specify Key Generator program and start up parameters
/* including the location of the configuration file
//EXEC      EXEC PGM=DDKEYGEN, PARM='POSIX(ON),          /KYG01
//          /*'DSEFTS.DD.CFG(KYG01CFG)'''
/* Specify the location of the Key Generator load library
//STEPLIB DD DSN=DSEFTS.ID.OE.V015BR01.LOAD,DISP=SHR
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//MSGFILE  DD SYSOUT=*
//ABNLIGNR DD DUMMY
/* Specify DD names for files to be opened SHARED by Utility
//INITFAIL DD DSN=DSEFTS.DD.INITFAIL,DISP=SHR
//KYG01LOG DD DSN=DSEFTS.DD.KYG01LOG,DISP=SHR
```

The JCL runs the DDKEYGEN program passing an identifier (KYG01) and the location of an XML configuration file.

UNIX Platform

The utility is run from the command line as follows:

```
ddkeygen kyg01 /app/ddtest/crypto/kyg01cfg
```

The utility is passed an identifier (kyg01) and the location of an XML configuration file.

Windows Platform

The utility is run from the command line as follows:

```
ddkeygn.exe kyg01 c:\ddtest\crypto\kyg01cfg
```

The utility is passed an identifier (kyg01) and the location of an XML configuration file.

ICSF Configuration and Output

Configuration of the key generation utility is via XML, an example of which is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="//DD.CFG(EVNTSXML)"
    logFileName="DD:CNSKGLOG"
    wrapLog="1"
    purgeOnOpen="1"
    traceLevel="6"
    firstRule="logNormal"/>
  <ZosICSFCryptoLibrary name="cryptoLibrary"
    keyLabelStem="DD.TEST.KEY"
    keyRepository="//'DSEFTS.DD.KEY'" />
  <!-- actions 1 = generate -->
  <KeyGenerator name="keyGenerator"
    identity="kyg01"
    action="1" />
</components>
```

Aside from the usual DataDistributor Event Log configuration details, the configuration specifies the type of Crypto Library for which keys are to be generated and configures the action for the tool; in the example shown the action is 1 for generate which is the only supported action for ICSF Keys.

The ZosICSFCryptoLibrary component specifies the key label stem for the required key label and the location of the key file into which the key is to be placed. Key labels are formed by concatenating the key label stem with a 4 character hexadecimal check digit value obtained by performing encryption of data using the generated key.

The key will be stored in a the configured key extract file in the following form:

DKEY	<4 character checkdigit><key label>
------	-------------------------------------

For example (based in the above configuration):

DKEY	ABCDDSEFTS.DD.KEYABCD
------	-----------------------

where ABCD represents the 4 character check digit value.

OpenSSL Configuration and Output

Configuration of the key generation utility is via XML, an example of which is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="c:\ddtest\evntxml "
    purgeOnOpen="1"
    traceLevel="7"
    firstRule="logAlert"/>
  <OpenSSLCryptoLibrary name="cryptoLibrary"
    seedFile="c:\ddtest\crypto\seed.txt"
    keyRepository="c:\ddtest\crypto\keys" />
  <!-- actions 1 = generate; 2 = export -->
  <KeyGenerator name="keyGenerator"
    identity="ddkg01"
    action="2"
    transportKeyFile="c:\ddtest\crypto\trankey"
    exportKeyFile="c:\ddtest\crypto\keyexport"/>
</components>
```

Aside from the usual DataDistributor Event Log configuration details, the configuration specifies the type of Crypto Library for which keys are to be generated and configures the action for the tool; in the example shown the action is 2 for export which will export a previously generated key (action = 1).

The utility will generate a key and store it in the specified key repository; the data key is itself encrypted using a Master Encryption Key. The MEK is generated using a secure algorithm using a seed value (a character string) specified in the configured seed file. The seed file (or seed value) must be provided when configuring a Consumer (or other process) which uses the key held in the key repository to enable it to be decrypted.

The key repository stores the key in a record of the following format:

Field#	Field Name	Offset	Length	Attr	Description
1	Check Digits	0	4	Char	Check Digits for the key.
2	Key Length	4	2	Binary	Length of the key value.
3	Encryption Key	6	32	Char	Hexadecimal Character Representation the key value.
4	MEK Check Digits	36	4	Char	Master Encryption Key Check Digits.
5	Timestamp	40	16	Char	Length of the decompressed and decrypted message.

A previously generated key can also be processed for export by running the tool with the action attribute set to 2.

Keys are exported by using a three-part transport key mechanism. Three separate transport keys are generated and stored in separate key files. These transport key parts are then combined to create a final transport key which is used to encrypt the data key which is to be exported. The intention is that the three key parts are distributed separately and combined to form the final transport key.

Each key (transport key part and exported key) is stored in file of the following form:

Field#	Field Name	Offset	Length	Attr	Description
1	TK Check Digits	0	4	Char	Check Digits for the combined transport key.
2	Space	4	1	Char	Space filler.

Field#	Field Name	Offset	Length	Attr	Description
3	Encryption Key	5	32	Char	Hexadecimal Character Representation the key value.
4	Space	37	1	Char	Space filler.
5	Check Digits	38	4	Char	Check Digits for the key.

The following example shows the contents of the three transport key part files and the exported key file itself:

```
F482 515432101601D60DA7F849D086AD5B37 3299
F482 CB3262BFA10ECE92B3801CD5A7750D8A D4DC
F482 4F04C1EA3D8C4C7573F2A1A415E33225 6680
F482 6D4B4AED53E1BEF2C423BEF1F38C8133 2A49
```

The key file names are determined from the Key Generator configuration, which specifies the name of the export file and the transport key files. The specified transport key file name is generated with a two character suffix (P1, P2, P3) to distinguish the different key parts, for example:

```
c:\ddtest\crypto\trankeyP1
c:\ddtest\crypto\trankeyP2
c:\ddtest\crypto\trankeyP3
```

Password Encryption Utility

Overview

The DataDistributor Password Encryption Utility is provided to enable a simple mechanism for encrypting passwords for use with DataDistributor. For example, to encrypt the password specified for use with the Consumer for DataNavigator.

The utility consists of a command line tool to enable entry of a password which is then encrypted and store in a file for future reference. Encryption is supported using OpenSSL cryptographic libraries and is available on Unix and Windows platforms.

Operation

UNIX Platform

The utility is run from the command line as follows:

```
ddpasswd /app/ddtest/crypto/pwe01cfg
```

The utility is passed the location of an XML configuration file.

Windows Platform

The utility is run from the command line as follows:

```
ddpasswd.exe c:\ddtest\crypto\pwe01cfg
```

The utility is passed the location of an XML configuration file.

OpenSSL Configuration and Output

Configuration of the key generation utility is via XML, an example of which is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="c:\ddtest\evntsxml"
    purgeOnOpen="1"
    traceLevel="7"
    firstRule="logAlert"/>
  <OpenSSLCryptoLibrary name="cryptoLibrary" />
  <PasswordEncryption name="passwordEncryption"
    identity="ddpwe01"
    passwordFile="c:\ddtest\crypto\dbpasswd"/>
</components>
```

Aside from the usual DataDistributor Event Log configuration details, the configuration specifies the type of Crypto Library to be used to encrypt the password and the location of the repository for the result.

The utility will prompt for entry of the password twice. If the entered passwords match the password will be encrypted and stored in the repository specified in the utility configuration.

The password file can then be referenced in the appropriate Consumer database configuration file, for example:

```
<DbConfig name="dbConfig"
  dataSourceName="oedd"
  userId="oedd_user"
  passwordFile="c:\ddtest\crypto\dbpasswd"/>
```



FTP Named Pipe Support

Overview

OE DataDistributor provides support for the processing of data through UNIX Named Pipes (or FIFO) in cooperation with FTP file transfer.

At present this facility is only available on the z/OS Platform.

Named Pipes provide a mechanism for communication between processes - data written to the pipe by one process can be read from the pipe by the other.

z/OS provides a mechanism for writing and reading data from Named Pipes via FTP; data which is sent to a server via FTP can be written to a Named Pipe instead of a regular file or data set. Similarly, data can be read from a Named Pipe for transfer via FTP.

The DataDistributor Pipe Process Utility (DDPIPE) provides mechanisms for processing data transferred by a Named Pipe to and from FTP.

- A PipeReader component for reading data from a Named Pipe, performing processing on the records received, and then writing that data to a specified file or dataset.
- A PipeWriter component for reading data from a specified file or dataset, performing processing on the records, and then writing the data to a Named Pipe.

The processing to be performed on records read from and written to the Named Pipe is configurable via DataDistributor Processing rules, including the encryption and decryption of data.

DDPIPE Operation

z/OS Platform

The DDPipe Utility is invoked by submitting a JCL file similar to that shown below:

```
//DDNMP01 JOB (W,K21415S,G2SE) ,
//          'C EXEC' ,
//          CLASS=T,
//          MSGCLASS=A,
//          NOTIFY=&SYSUID
/*XEQ DDSNBDEV
/*JOBPARM S=A005
/*ROUTE PRINT LOCAL
/* Specify ddpipes program and start up parameters
/* including the location of the configuration file
//EXEC      EXEC PGM=DDPIPE, PARM='POSIX(ON)',      /NMP01
//          /*'DSEFTS.DD.CFG(NMP01CFG)'''
/* Specify the location of the ddpipes load library
//STEPLIB DD DSN=DSEFTS.ID.OE.V015BR01.LOAD,DISP=SHR
//SYSOUT    DD SYSOUT=*
//CEEDUMP   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//MSGFILE   DD SYSOUT=*
//ABNLIGNR  DD DUMMY
/* Specify DD names for files to be opened SHARED by Utility
//INITFAIL  DD DSN=DSEFTS.DD.INITFAIL,DISP=SHR
//NMP01LOG  DD DSN=DSEFTS.DD.NMP01LOG,DISP=SHR
```

The JCL runs the DDPipe program passing an identifier (NMP01) and the location of an XML configuration file.

In order to initiate the process in a coordinated fashion with the required FTP session, the recommended approach is to submit JCL similar to the above using the z/OS FTP JES interface (see section [“Interaction with FTP” on page 78](#)).

Configuration

Pipe Reader

Configuration of the DDPIPE Pipe Reader facility is via XML.

The XML configuration specifies the specifics of the Named Pipe from which data is to be read, and the specifics of the data set to which records from the pipe are to be written; optional processing of records prior to being written to the data set is specified by configurable rules:

The following example shows the configuration of a Pipe Reader to read fixed size records from a Named Pipe and write those records to a Fixed Block format dataset, having first performed partial encryption of the records using the EncryptRecord rule and the ICSF Cryptolibrary:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="//DD.CFG(EVNTSXML)"
    logFileName="DD:NMP01LOG"
    wrapLog="1"
    purgeOnOpen="1"
    traceLevel="6"
    firstRule="logNormal"/>
  <ZosICSFCryptoLibrary name="cryptoLibrary"
    sourcePlatform="0"
    version="2"
    encryptionOffset="0"
    encryptionLength="56"
    keyExtract="//'D96658.CNSKEY'" />
  <EncryptRecord name="encryptRecord" />
  <DataSet name="outputFile"
    dataSetName="//'D96658.CNSAPM3C'"
    recfm="FB"
    lrecl="526"
    blockSize="6312"
    primaryExtents="10"
    secondaryExtents="10"
    units="CYL" />
  <NamedPipe name="namedPipe"
    format="fixed"
    defaultPipeName="/tmp/nmp01r"
    maxRecordLength="500" />
  <PipeReader name="pipeProcess"
    firstRule="encryptRecord"
    identity="ddpr01"/>
</components>
```

The NamedPipe configuration specifies the name of the pipe from which data is to be read – the pipe will be created if it does not exist. Also specified is the format of the records to be read (fixed) and the size of each record (500 bytes).

The DataSet configuration specifies the name of the output dataset to which the records read from the pipe are to be written once encrypted. The data set will be allocated using the sizing characteristics specified in the configuration.

NOTE:

The record length of each record has to be increased to accommodate the encryption header added by the encryption routine.

Full details of the configuration parameters for these components are available in the *OE DataDistributor Configuration Reference*.

Pipe Writer

Configuration of the DDPIPE Pipe Writer facility is via XML.

The XML configuration specifies the specifics of the Data Set from which data is to be read, and the specifics of the Named Pipe to which records from the data set are to be written; optional processing of records prior to being written to the named pipe is specified by configurable rules:

The following example shows the configuration of a Pipe Writer to read fixed size records from a Dataset and write those records to a Named Pipe, having first performed partial decryption of the records using the DecryptRecord rule and the ICSF Cryptolibrary:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="//DD.CFG(EVNTSXML)"
    logFileName="DD:NMP01LOG"
    wrapLog="1"
    purgeOnOpen="1"
    traceLevel="6"
    firstRule="logNormal"/>
  <ZosICSFCryptoLibrary name="cryptoLibrary"
    sourcePlatform="0"
    version="2"
    encryptionOffset="0"
    encryptionLength="56"
    keyExtract="//'D96658.CNSKEY'" />
  <DecryptRecord name="decryptRecord" />
  <DataSet name="inputFile"
    dataSetName="//'D96658.CNSAPM1E'"
    recfm="FB"
    lrecl="526"
    blockSize="6312"
    space="cyl,(10,10)" />
  <NamedPipe name="namedPipe"
    format="fixed"
    defaultPipeName="/tmp/nmp01w"
    maxRecordLength="500" />
  <PipeWriter name="pipeProcess"
    firstRule="decryptRecord"
    identity="ddpr01"/>
</components>
```

The DataSet configuration specifies the name of the input dataset from which the records. The data set will be allocated using the sizing characteristics specified in the configuration.

The NamedPipe configuration specifies the name of the pipe to which data is to be written – the pipe will be created if it does not exist. Also specified is the format of the records to be written (fixed) and the size of each record (500 bytes).

NOTE:

In the example shown, the fixed length records (500 bytes) in the data-set are preceded by the Standard Encryption Header (26 bytes) which will be removed before records are written to the pipe.

Full details of the configuration parameters for these components are available in the *OE DataDistributor Configuration Reference*.

Variable Length Record Transfer

Records read from variable format files will be written to the pipe preceded by a Record Descriptor Word (RDW) which indicates the length of the variable length record that follows.

Interaction with FTP

The following sections discuss the commands which need to be supplied to FTP in order to interact with the DDPIPE process, acting either as a Pipe Reader or a Pipe Writer.

Since both ends of the Pipe must be open in order for data to flow, the recommended approach is to use initiate the Pipe Process via the z/OS FTP interface to the JES command sub-system. Then FTP commands can be submitted to send data to or retrieve data from the Named Pipe as the pipe process will be running.

For further information on the FTP commands specified in the following sub-sections, consult the z/OS IP User's Guide and Commands document (Sections 4 and 5).

Sending Data to a Named Pipe

Login to the FTP server which hosts the Named Pipe.

Start the Pipe Reader process to read the records which are to be sent via FTP; the site commands sets characteristics for the FTP session on the remote host:

site	filetype=JES	(submit files to JES)
put	ddnmpjcl	(send JCL file to JES)

Set the session characteristics for submitting to the Named Pipe:

binary	(binary transfer of data)
site unixfiletype=FIFO	(unix files are pipes)
site filetype=SEQ	(reset filetype)
site fifoioptime=3600	(IO timer)

If transferring variable length records, specify that record descriptor words (RDW) are to be transferred to allow the Pipe Reader to parse the records written to the Pipe.

```
locsite RDW
```

Send the data using the FTP PUT command - the dataset is local, the Named Pipe is remote:

```
put MY.DATASET /tmp/nmp01
```

Receiving Data from a Named Pipe

Login to the FTP server which hosts the Named Pipe.

Start the Pipe Writer process to read the records which are to be sent via FTP:

```
site filetype=JES      (submit files to JES)
put ddnmpjcl           (send JCL file to JES)
```

Set the session characteristics for submitting to the Named Pipe:

```
binary                (binary transfer of data)
site unixfiletype=FIFO (unix files are pipes)
site filetype=SEQ      (reset filetype)
site fifovertime=3600  (IO timer)
```

Set space characteristics for the local dataset using the locsite command (this example is for FB records of length 500 bytes):

```
locsite recfm=FB
locsite lrecl=500
locsite blockside=6000
```



Send the data using the FTP GET command - the Named Pipe is remote, the dataset is local:

```
get /tmp/nmp01w MY.DATASET
```


12

File Processing Support

Overview

OE DataDistributor provides support for the processing of records stored in Files or Datasets.

At present this facility is only available on the z/OS Platform.

The DataDistributor File Process Utility (DDFILE) provides mechanisms for processing read from a file using a File Processing component for reading data from an input Dataset, performing processing on the records retrieved and then writing those processed records to a specified output dataset.

The processing to be performed on records read from and written to the Data Sets is configurable via DataDistributor Processing rules, including the encryption and decryption of data.

DDFILE Operation

z/OS Platform

The DDFILE Utility is invoked by submitting a JCL file similar to that shown below:

```
//DDFUT01 JOB (W,K21415S,G2SE) ,
//          'C EXEC' ,
//          CLASS=T,
//          MSGCLASS=A,
//          NOTIFY=&SYSUID
/*XEQ DDSNBDEV
/*JOBPARM S=A005
/*ROUTE PRINT LOCAL
/* Specify ddpipe program and start up parameters
/* including the location of the configuration file
//EXEC      EXEC PGM=DDFILE,PARM='POSIX(ON)',      /FUT01
//          /*'DSEFTS.DD.CFG(FUT01CFG)'''
/* Specify the location of the ddfile load library
//STEPLIB DD DSN=DSEFTS.ID.OE.V015BR01.LOAD,DISP=SHR
//SYSOUT    DD SYSOUT=*
//CEEDUMP   DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSUDUMP  DD SYSOUT=*
//MSGFILE   DD SYSOUT=*
//ABNLIGNR  DD DUMMY
/* Specify DD names for files to be opened SHARED by Utility
//INITFAIL  DD DSN=DSEFTS.DD.INITFAIL,DISP=SHR
//FUT01LOG  DD DSN=DSEFTS.DD.NMP01LOG,DISP=SHR
```

The JCL runs the DDFILE program passing an identifier (FUT01) and the location of an XML configuration file.

Configuration

File Process

Configuration of the DDFILE File Process facility is via XML.

The XML configuration specifies the specifics of the input DataSet from which data is to be read, and the specifics of the output Data Set to which records from the input are to be written; optional processing of records prior to being written to the data set is specified by configurable rules.

The following example shows the configuration of a File Process to read records from a Fixed Block DataSet and write those records to a Fixed Block format dataset, having first performed partial encryption of the records using the EncryptRecord rule and the ICSF Cryptolibrary:

```
<?xml version="1.0" encoding="UTF-8"?>
<components>
  <AlertLogger name="logAlert" />
  <FileLogger name="logNormal" />
  <EventLogger name="eventLogger"
    templateFile="//DD.CFG(EVNTSXML)"
    logFileName="DD:FUT01LOG"
    wrapLog="1"
    purgeOnOpen="1"
    traceLevel="6"
    firstRule="logNormal"/>
  <ZosICSFCryptoLibrary name="cryptoLibrary"
    sourcePlatform="0"
    version="2"
    encryptionOffset="0"
    encryptionLength="56"
    keyExtract="//'D96658.CNSKEY'" />
  <EncryptRecord name="encryptRecord" />
  <DataSet name="inputFile"
    dataSetName="//'D96658.CNSAPM3C'"
    recfm="FB"
    lrecl="526"
    blockSize="6312"
    primaryExtents="10"
    secondaryExtents="10"
    units="CYL" />
  <DataSet name="outputFile"
    dataSetName="//'D96658.CNSAPM3E'"
    recfm="FB"
    lrecl="526"
    blockSize="6312"
    primaryExtents="10"
    secondaryExtents="10"
    units="CYL" />
  <FileProcess name="fileProcess"
    firstRule="encryptRecord"
    identity="ddpr01"/>
</components>
```

The FileProcess configuration specifies the name of the DataSet from which data is to be read.

The output DataSet configuration specifies the name of the output dataset to which the records read from the input are to be written once encrypted. The data set will be allocated using the sizing characteristics specified in the configuration. Full details of the configuration parameters for these components are available in the OE DataDistributor Configuration Reference.

13

Secure Socket Layer Communication Support

Overview

DataDistributor provides support for using Secure Socket Layer (SSL) communications protocols over TCP/IP. This ensures secure end-to-end transmission of data from Producer platform to Consumer platform.

Both Producer and Consumer must be configured to use SSL for communications to be enabled.

SSL support is currently available on the following platforms:

- IST Switch Producer (Unix)
- IBM z/OS

For the **IST Switch Producer**, SSL is implemented using the OpenSSL SSL library. A distribution of the required OpenSSL code is provided with the IST Switch release.

On the **IBM z/OS platform**, SSL is implemented based on the System Secure Socket Layer programming interface, which is part of the Cryptographic Services facilities provided on that platform.

Regardless of the platform, OpenSSL support requires configuration both of DataDistributor itself but also of the OpenSSL environment in relation to the establishment of the required keys and certificates.

Detailed information pertaining to the configuration of keys and certificates is provided in the documentation for each SSL implementation. An overview is provided here as a starting point.

SSL Environment Configuration

Overview

Regardless of the SSL implementation, SSL configuration largely consists of the following main tasks:

- Generation of private keys for encryption
- Generation (and storage) of certificates for authentication

The exact requirements for key and certificate generation and storage depend on whether the connection is acting as a Server or a Client in the interface. In DataDistributor terms a Producer is a Server process and the Consumer is a client.

In general terms the requirements for configuration are:

- A private key must be generated for a Server process (since the Server is responsible for generating public keys based on the private key for sharing with the Client).
- A Server will also be required to provide a certificate which the client will use to verify authenticity.
- A Client may also be required to provide a certificate if the Server is performing Client authentication (this is configurable).
- Both Server and Client can store certificates (a truststore) for known and trusted connections.
- For the purposes of DataDistributor Producer to Consumer communications, self-signed (as opposed to signed by a Certificate Authority) SSL Certificates are adequate for fulfilling the protocol and authentication requirements.

Open SSL Configuration (IST SWITCH)

OpenSSL provides a command line tool (openssl) for the generation of private keys and certificate files. The OpenSSL documentation provides details on its use.

IST Switch generates the relevant OpenSSL keys and self-signed certificate files (for both Server and Client) during the installation process. Consult the IST Switch documentation for details. DataDistributor Producer for IST Switch should be configured to utilize the files generated during installation (See subsequent sections).

If required however a Self Signed Certificate can be created independently of the switch for test purposes as follows:

```
openssl req -new -x509 -nodes -out server.crt -keyout server.key
```

This creates a certificate file server.crt in PEM format and a certificate key file in server.key. You will be prompted to provide details for the certificate.

To create a combined certificate and key file, use the following:

```
openssl req -new -x509 -nodes -out server.crt -keyout server.crt
```

Both certificate and key are written to the file server.crt

To export a certificate, for example for importing into a z/OS key database (see [System Secure Socket Layer Configuration \(Z/OS\)](#)), the following command should be used:

```
openssl x509 -in server.crt -outform DER -out export.der
```

System Secure Socket Layer Configuration (Z/OS)

System SSL on z/OS is configured using the gskkyman tool which can be accessed from OMVS via TSO. Use of the tool is described in the relevant IBM documentation Example:

<http://publib.boulder.ibm.com/infocenter/zos/v1r11/index.jsp>

(See Cryptographic Services on [page 58](#) for more details on System Secure Socket Layer Configuration.)

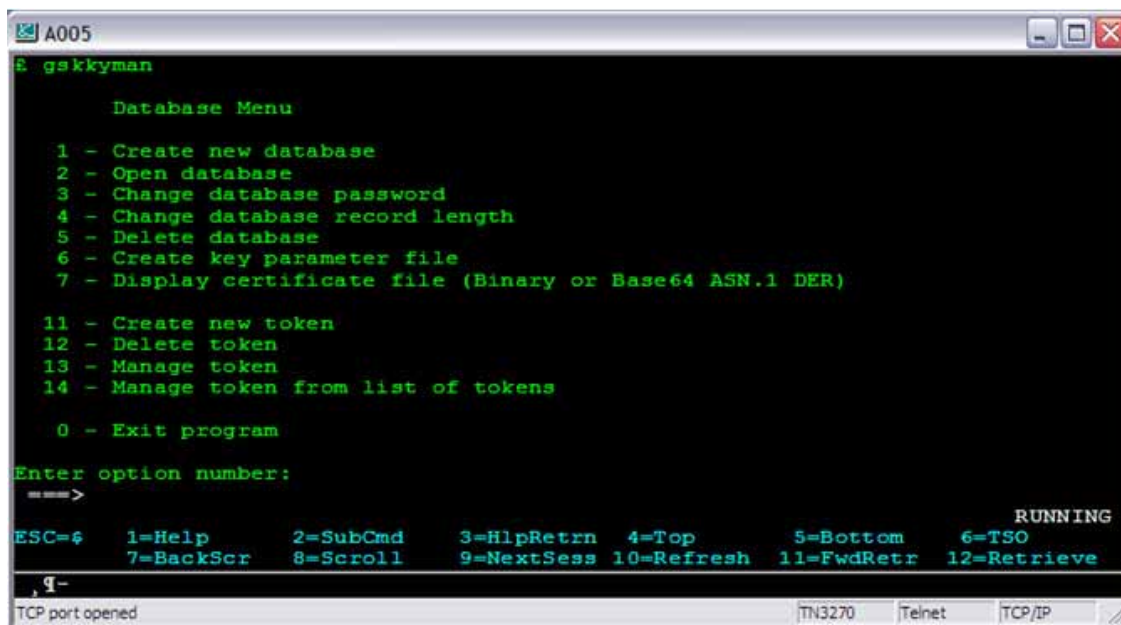
At a minimum there are three basic configuration steps required:

- Create a key database
- Store the database password
- Import the server certificate for verification purposes

These steps are described below:

Create a Key Database

1. Enter gskkyman from OMVS via TSO, the following menu is displayed.



```

A005
R gskkyman

      Database Menu

      1 - Create new database
      2 - Open database
      3 - Change database password
      4 - Change database record length
      5 - Delete database
      6 - Create key parameter file
      7 - Display certificate file (Binary or Base64 ASN.1 DER)

      11 - Create new token
      12 - Delete token
      13 - Manage token
      14 - Manage token from list of tokens

      0 - Exit program

Enter option number:
>

ESC=¢  1=Help    2=SubCmd   3=HlpRetrn  4=Top      5=Bottom   6=TSO
        7=BackScr 8=Scroll   9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

,q-
TCP port opened      TN3270  Telnet  TCP/IP
  
```

2. Select option 1 - Create new database.
3. You will be prompted to enter the key database name, this is a file within the Unix System Services Hierarchical File System (HFS), example:
`/datadist/datadist.kdb`
4. You will be prompted to enter a password for the key database.
5. The Key database will then be created.

Store Key Database Password

1. From the main menu, choose option 2 to open the key database you created above. You will be prompted to enter the database name and the password.
2. Choose option 10 - Store Database Password.
3. The database password will be store in a stash file with the same name as the key database but with a .sth suffix, example:
`/datadist/datadist.sth`

Import Server Certificate File

This is required if the certificate that the Server (i.e. Producer) is using is not signed by a Certificate Authority for which certificates are already stored in the database - this includes Self-Signed certificates.

1. From the main menu, choose option 2 to open the key database you created above. You will be prompted to enter the database name and the password.
2. Choose option 7 - Import a certificate.
3. You will be prompted to enter the name of the certificate file example:
`/datadist/export.der`
4. You will be prompted to provide a label for the certificate within the database example: DataDistributor

DataDistributor SSL Configuration

Overview

SSL is implemented as an additional configuration option within DataDistributor which builds on the existing TCP/IP configuration.

Regardless of the SSL implementation, there are two steps required to configure SSL for a Producer or Consumer:

- Indicate the use of SSL by adding an attribute to the existing TCP/IP Component (ConsumerConnection or ProducerConnection)
- Add an SSL Component to the configuration dependent on the SSL implementation

OpenSSL Component Configuration

This section describes the configuration required to add SSL support using OpenSSL software in this instance to a Producer process (that is, Producer for IST Switch).

The following shows the configuration which is required:

```
<OpenSSLConnection name="sslConnection"
    keyFile="server.key"
    certificateFile="server.crt" />
<ConsumerConnection
    name="consumerConnection"
    byteReverse="0"
    port="9101"
    windowSize="1"
    useSSL="1"
    bufferSize="32000"
    readTimeout="30"
    acceptTimeout="30"
    tcpIpBufferSize="32000"
    compressionMode="0">
</ConsumerConnection>
```

The ConsumerConnection component contains an additional attribute useSSL which indicates that SSL is required.

Additionally a new component OpenSSLConnection is specified this contains configuration attributes which specify the key and certificate files which are to be used.

See the OE DataDistributor Configuration Reference Guide for more details of the configuration attributes.

Z/OS SSL Component Configuration

This section describes the configuration required to add SSL support using z/OS System SSL software - in this instance to a Consumer process.

The following shows the configuration which is required:

```
<ZosSSLConnection name="sslConnection"
keyDatabase="/datadist/datadist.kdb"
keyStash="/datadist/datadist.sth" />
<ProducerConnection name="producerConnection"
    byteReverse="0"
    ipNode="mkeux080"
    port="9101"
    useSSL="1"
    readTimeout="30"
    bufferSize="64000"
    tcpIpBufferSize="32000" />
```

The ProducerConnection component contains an additional attribute useSSL which indicates that SSL is required.

Additionally a new component ZosSSLConnection is specified. This contains configuration attributes which specify the key database and password stash files which are to be used.

See the OE DataDistributor Configuration Reference Guide for more details of the configuration attributes.

Index

C

checkpoint processing	31
checkpoint structure	32
position information	35
checkpoint processing-checkpoint structure	
consumer for DataNavigator®	32
consumer for received queue	33
consumer for settlement	32
producer for BASE24	34
producer for Connex on HP NonStop	33
producer for Connex on IBM®	34
producer for delivery queue	33
producer for IST switch	34
checkpoint processing-position information	
producer for Connex on HP NonStop/ BASE24	35
producer for Connex on IBM®	36
producer for delivery queue	35
producer for IST switch	35
configuration	83
file process	83
consumer identity validation	56
consumer load statistics	49
consumer platform cryptography	60
encryption header	61
encryption/decryption processing rule	60
ICSF cryptography library	63
OpenSSL cryptography library	64
consumer platform cryptography-encryption header	
version 1 header	61
version 2 header	62

D

DataDistributor on AIX	16
starting DataDistributor	16
stopping DataDistributor	17
supported components	16
DataDistributor on HP NonStop guardian	10
starting DataDistributor	10
stopping DataDistributor	11
supported instances	10
DataDistributor on HP NonStop guardian- starting DataDistributor	
starting from PATHWAY	10
starting from TACL	10
DataDistributor on HP NonStop guardian- stopping DataDistributor	
stopping from PATHWAY	11
stopping from TACL	11
DataDistributor on IBM® zOS	15
starting DataDistributor consumer	15
stopping DataDistributor consumer	15
supported instances	15
DataDistributor on IBM® zOS-starting DataDistributor consumer	
starting using JCL	15
using a started task	15
DataDistributor on IBM® zOS-stopping DataDistributor consumer	
stopping a batch process	15
DataDistributor on Solaris (non IST/switch)	13
starting DataDistributor	13
stopping DataDistributor	14
supported components	13
DataDistributor on Windows®	9
starting DataDistributor	9
stopping DataDistributor	9
supported instances	9

DataDistributor producer for IST/switch (Solaris)	11
starting DataDistributor	12
stopping DataDistributor	12
supported instances	11
DDFILE operation	82
z/OS platform	82
DDPIPE configuration	74
pipe reader	74
pipe writer	76
variable length record transfer	78
DDPIPE operation	73
z/OS platform	73

E

extraction lag reporting	47
------------------------------------	----

F

filtering Connex on HP NonStop log records 51	
FTP named pipe support overview	72

G

general processing behaviour	18
common processing	18
DataDistributor consumer	18
DataDistributor producer	19

I

interaction with FTP	78
receiving data from a named pipe . .	79
sending data to a named pipe	78

K

key generator utility	65
ICSF configuration and output	66
OpenSSL configuration and output .	67
operation	65
overview	65
key generator utility-operation	
UNIX platform	66
Windows platform	66
z/OS platform	65

O

overview of file processing support	81
---	----

P

password encryption utility	70
operation	70
openSSL configuration and output .	71
UNIX platform	70
Windows platform	70
overview	70
process initialisation failures	37
process runtime errors	38
consumer target system errors	44
error recovery processing	39
network errors	39
producer data source errors	41
process runtime errors-consumer target system errors	
DataNavigator® consumer	45
received queue consumer	45
settlement consumer	44
process runtime errors-network errors	
connection failures during processing 40	
connection timeouts and poll intervals .	41
initial connection failures	39

process runtime errors-producer data source errors	
BASE24 producer	44
Connex on HP NonStop producer	41
Connex on IBM® producer	43
delivery queue producer	43
IST producer	42
producer platform cryptography	58
HP/Connex	58
IBM/Connex	59
IST/Switch	59

S

secure socket layer communication support	
DataDistributor SSL configuration	89
openssl component configuration	90
overview	89
Z/OS SSL component configuration	91
overview	85
SSL environment configuration	86
open SSL configuration (IST SWITCH)	86
overview	86
system secure socket layer	

configuration (Z/OS)	87
create a key database	88
store key database password	88
SSL environment configuration	
system secure socket layer configuration (Z/OS)	
import server certificate file	89

T

TCP/IP address validation	55
transaction data input and output	19
consumer for DataNavigator®	23
consumer for received queue	30
consumer for settlement	28
producer for BASE24	22
producer for Connex on HP NonStop	20
producer for Connex on IBM®	21
producer for delivery queue	21
producer for IST/switch	19
transaction data input and output-consumer for DataNavigator	
queue table structures	24
ueue initialisation	26
transaction data input and output-consumer for DataNavigator®	
queue processing	27
transaction trace output	46

Statement of Confidentiality

The information contained in or supplied with this document is submitted solely for the purpose of evaluating the products and services of FIS, and/or its affiliates and subsidiaries. The information contained in or supplied with this document, in its entirety, is the confidential and proprietary information of FIS, and it may not be copied by or disclosed to any person or entity (other than to the intended recipient), without the prior written consent of FIS. With or without FIS' prior written consent, FIS accepts no liability whatsoever for any consequences arising from the reproduction of the information contained in or supplied with this document, or from its disclosure to any person or entity, including to the intended recipient. FIS additionally accepts no liability for the use of the information contained in, or supplied with this document, by the intended recipient, or by any other person or entity, with or without FIS' express prior consent. The intended recipient shall not use any part of the information contained in, or supplied with this document, in any way to the competitive disadvantage of FIS, and will take all steps designed to assure its compliance with this provision.

This proposal is neither an offer nor intended by FIS, upon acceptance by the intended recipient, or otherwise, to create a binding agreement with FIS. Such an agreement shall be reflected only by a definitive contract, executed by both parties.

Trademarks

All other trademarks are the property of their owners.

Company, product, and service names used by FIS within, or supplied with this document may be trademarks or service marks of other persons or entities.

Copyright

Copyright© 2012 FIS.
All Rights Reserved.