BusinessLink Software Support

HSM Data Server's Guide

Version V2R1

This manual applies to Strategi version V2R1 and later and was last revised in September 2005.

ADVANCED BusinessLink Corp. may have patents and/or patent pending applications covering subject matter in this document. The furnishing of this document does not give any license to these patents.

Every effort has been made to ensure the accuracy of this manual. However, ADVANCED BusinessLink Corp. and Advanced BusinessLink (Australia) Pty. Ltd make no warranties with respect to this documentation, and shall not be liable for any errors, or for incidental or consequential damages in connection with the performance or use of this manual, or the examples pertaining to products and procedures as described herein. The information in this manual is subject to change without notice.

Other trademarks, trade names and brand and product names used in this manual are trademarks or registered trademarks of their respective holders.

Created in the United States of America.

A Note to Readers

The latest versions of this document and other Technical Support Bulletins can be downloaded from ADVANCED BusinessLink Corp.'s Support Website, http://support.businesslink.com.

You may print this in duplex format using Adobe's Acrobat Reader, which is available for download from http://www.adobe.com/products/acrobat/readstep.html.

With some installs of Adobe Acrobat, your printer may not resolve the characters correctly, and once printed, all characters will appear as rectangles or as symbols. If this happens, you will need to select "Print as image" from the Acrobat print dialogue. This will cause the print to occur correctly.

If you have any questions, comments or suggestions, please feel free to contact BusinessLink Software Support via email at support@businesslink.com.

Sincerely,

BusinessLink Software Support

Table of Contents

Introduction

This document describes the standard Strategi HSM Data Servers, which are designed to provide convenient access to databases from HSM resource files. The same access is also available from any HSM client program, but is generally less convenient than direct database programming there.

Strategi provides two implementations of HSM Data Server, MiniDBMS and JDBCDS. MiniDBMS is intended for single-user client systems, running under Pocket Strategi. JDBCDS is intended for multi-user access to data on any system that provides an underlying DBMS supporting JDBC SQL access to files, which are expected to contain data for all users. Server choice is dependent upon the intended usage of the application.

Configuring Strategi HSM Data Servers for Access

Prior to usage, the Strategi HSM Data Servers must be configured as follows:

Configuring MiniDBMS

In order to use MiniDBMS, an HSM server must be defined to use the Java class, com.businesslink.sgi.minidbms.Main.

Using a database file editor, modify hm.dbf to define an HSM server specifically for MiniDBMS and include the following information:

Column	Value
HMSVR	MINIDB (replace with your specific name)
HMAUTO	1
HMINST	1
HMDESC	MiniDBMS Server
HMCLAS	com.businesslink.sgi.minidbms.Main
HMCLSP	

The server name must conform to standard HSM server naming conventions, but is not required to have a certain name. For example, the server could be named MINIDB, LOCAL, or MYSERVER. Choose a good descriptive name to ease server management.

Upon restart of Pocket Strategi the newly defined MiniDBMS Server will be active and ready for database interaction.

Configuring JDBCDS

Current implementation of JDBCDS is available on OS/400 only. To configure a JDBCDS Server in host Strategi, perform the Create HSM Server (CRTHSMSVR) command, entering *STRATEGI for the Server Type, *JDBCDS for the Strategi Program, optionally include a class path, and enter an appropriate Job Description that contains the libraries necessary for JDBCDS to find the HDSDFN file.

Example:

```
*****************************************************************************
*                          Create HSM Server (CRTHSMSVR)                    *
*                                                                           *
* Server Name . . . . . .   APPDB           Name, *CONVERT, *SYSTEM         *
* Server Type . . . . . > *STRATEGI         *STRATEGI, *DTAQ, *SRVPGM...    *
* Strategi System Server:                                                   *
*   Strategi Program . . . . *JDBCDS        *CONVERT, *JDBCDS, *SGIDS…      *
*   Added Classpath (Optional) . .  *NONE                                   *
* Instances . . . . . .     1               Number                          *
* Autostart . . . . . .     *YES            *YES, *NO                       *
* Text Description . . .    'Acme Application Data Server'                  *
*                                                                           *
*****************************************************************************
```

Indexes and Filters

HSM Data Server includes the definition of multiple indexes over each file, using simple binary ascending or descending sequence within each field.

When the records are listed using an index, they are of course returned in index order. But any number of leading key fields can be specified as having to match the supplied key (e.g., simplifying retrieval of all lines for a particular customer.)

Filters can also be set up, causing listed records to be limited to those that match certain criteria without specifying a separate index. This is valuable because there can be many filters in force at any one time, and maintaining indexes for all the combinations would be prohibitive. Note that filters are treated as character fields. Filters test only for exact equivalence of whole fields, with a filter value of blank meaning all records should be returned. A filter defined as having more than one field tests each of the fields individually, ignoring trailing fields with a blank value (e.g., useful for drill-down selection of products by group and optionally subgroup). A given filter applies across all files for which it is defined (i.e., setting filter 1 to a particular value affects subsequent list accesses to all files within that Data Server that define a filter 1). A blank filter is interpreted as meaning accept any value; a filter can be cleared by setting it to blank.

Indexes and filters are referred to by number, starting at one. Numbers need not be consecutive, but if any indexes are defined then index one is required, and is treated as the primary key.

MiniDBMS

MiniDBMS is intended for single-user client systems, running under Pocket Strategi. It requires no underlying DBMS, storing its data in either simple text files (with .dta extensions) or dBase III compatible files (with .dbf extensions), with related field, key, and filter definitions in .dfn files.

On opening the file, it is read from beginning to end and in-memory indexes and filters are set up, and then maintained as records are changed, so filtered access to complex indexes is very fast. Files should be explicitly closed using the CLOSE opcode to ensure that the end of file marker is written to the file.

There is no provision for file or record locking, although the fact that files are opened for update may cause the underlying operating system to limit access to them by other programs.

For .dbf files, output formatting of numeric fields is triggered by having the field length in the .dfn file be longer than the .dbf field length. Input formatting is automatic, with numbers in character strings forced to conform to dBase numeric layout, truncating or ignoring excess or invalid characters as necessary to ensure valid dBase data. It is expected that application code (Javascript or a separate application-specific HSM server) will have checked keyed data and returned errors to the user before any numeric data is passed to MiniDBMS.

For .dta files, all data is treated as simple text, with numeric fields expected to be pre-formatted by the requestor (in fact MiniDBMS on .dta files does not distinguish between numeric and character fields).

Data is automatically forced to disk (by closing and reopening the underlying file handle) after 10 seconds of no activity, and at least every 60 seconds when there is continuing activity, as determined by HSM requests to each MiniDBMS server.

Any deleted records in the file are automatically reused for new records as they are added, but there is currently no file compression or reorganization.

All records on all files are accessible, as in a normal DBMS.

JDBCDS

JDBCDS is intended for multi-user access to data on any system that provides an underlying DBMS supporting JDBC SQL access to files, which are expected to contain data for all users. It has been initially deployed on OS/400 running under Strategi.

It includes automatic subsetting of data (under Strategi only), so each user sees only the data relevant to them. This is done by, in the definition of the HSM server that accepts the original request, specifying a Strategi User Attribute to pass with the request. The value of that attribute for the requesting user stays with the request if DHSM is used, and is also passed on with secondary HSM requests made by any HSM server. JDBCDS can then subset the file data by requiring specified fields to match parts of that user attribute data, or the Strategi user number itself.

HSM Data Server indexes need bear no relation to the underlying database, although it may be valuable to set up corresponding real indexes (including user subsetting fields) to optimize performance. Index zero is a special case, since record numbers cannot be consistently used to access a file, let alone a subset of a file, in JDBC. To support record number requests, there must be a numeric record number field in the underlying table, and it is then internally set and used as a normal key field, transparent to the application making HSM requests.

Field lengths as seen by HSM, along with key and filter definitions, are stored in an HDS definition along with the name of the underlying file to use and other information such as how to subset data. Note that the lengths as returned may differ from the underlying DB2 field lengths, and will be returned in HDSDFN field sequence regardless of DB2 field order.

Numeric field formatting is triggered by keywords in that definition file.

Commitment of data to disk is at the discretion of the underlying DBMS.

File Names

HSM requests refer to files by two-character names.

For MiniDBMS, these request file names relate directly to the actual file names, e.g. file XX will be XX.dfn and XX.dbf, or XX.dta, in the file system. Note that the file name is capitalized while the extension is lowercase, although either case can be used on requests.

For JDBCDS, the actual host database file name is determined by the definition file, and the definition file itself is currently implemented assuming IBM DB2/400 file and member naming. In particular, the definition file must:

- Be in the library list
- Be named HDSDFN
- Have members with names corresponding to the two-character request file names

MiniDBMS DTA Files

The .dta alternative for MiniDBMS files is laid out as a simple text file with fixed-length lines. Each line contains the field data for a single record, each field space-padded to its maximum length, without any tabs or other extraneous characters. Binary data that might confuse simple text processing is not supported.

Each line must end with a Carriage Return and Line Feed (0x0D 0x0A), preceded by a record status flag that must be one of:

H Header – a single free-format line at the start, useful for marking field positions
A Active record
D Deleted record

There can be an additional End of File character (0x1A) at the end, but otherwise the file length must be consistent with the line length inferred by the definition file.

Such files are therefore easily edited with most text editors, and very useful for testing, but note that if both a .dbf and a .dta exist, the .dbf will be used.

With a few exceptions (e.g., APYMODDBF, SETRECORD), access to the two types of file is transparently identical, and MiniDBMS will open and use any mixture of types, so individual files can be switched.

Definition File Common Groups

For each database file there is a definition file, with extension .dfn, that describes the data as seen by clients making requests of the HSM Data Server.

This file is laid out in typical .ini file fashion, as a simple text file in which groups defined in square brackets head up keyword=value pairs. There is a limit of 9 keys and 99 filters for JDBCDS and a limit of 9 keys and 9 filters for MiniDBMS.

Note that comments are inserted by beginning the line with a semicolon.

For example, the PH.dfn example file below defines product history by customer and date, is listed in code or description sequence with the newest entries first, and is able to limit the listing by two levels of product group.

A definition file might be as follows:

```
[FIELDS]
CUSTOMER=5
DATE=8
CODE=4
GROUP=2
SUBGROUP=2
DESCRIPTION=50
QUANTITY=9

* A comment line starts with an asterisk
[KEY 1]
CUSTOMER=A
DATE=D
CODE=A

[KEY 2]
CUSTOMER=A
DATE=D
DESCRIPTION=A

[FILTER 1]
GROUP
SUBGROUP
```

Group FIELDS

Required, this lists all fields and their character lengths as seen by the HSM requests and replies. MiniDBMS currently allows .dbf file fields with the same length as in the dBase file to be omitted, but this will change for better control and compatibility with JDBCDS.

Each entry in the group is the name of a field, with its character length after the equals sign.

Order of field definition is not important for MiniDBMS DBF files, but it determines the record layout for MiniDBMS DTA files, and is used in JDBCDS to interpret data on a SETRECORD.

In JDBCDS, fields should only be listed here if HSM clients directly access them. If any of the user data subsetting fields are listed, then the file must be defined as read-only (because applications must of course not be able to directly set those fields).

Group KEY n

Only required if the file has keys (as opposed to a file that is accessed by record number), a number of keys can be defined. If any key is defined, then key 1 must be, and is considered the primary key (used for example in GETRECORDS). Otherwise, key numbers can be skipped, allowing different files to use the same numbers for similar keys.

Each entry in the group is a field name, with A for Ascending or D for descending after the equals sign. MiniDBMS defaults to ascending if the equals sign is missing. The sequence of entries determines the sequence of fields in the key.

Group FILTER n

Only required if the file has filters; a number of filters can be defined. For a given filter number, the number of fields and their length must be the same on all files where that filter number is defined (within a set of files processed by a single HSM Data Server). If a given file defines any, they can be for any filter number.

Each entry in the group is simply the field name, with the sequence of entries determining the sequence of fields in the filter.

Definition File JDBCDS Groups

Two definition file groups are only relevant for, and present for, JDBCDS, and normally come ahead of the common groups. For example, the history file above might have the following:

```
[File]
HsmName=PH
SystemName=MYLIB/ABCPH
DigitGroupingSeparator=','
SimpleDigitGrouping=TRUE
BlankNumericWhenZero=TRUEX
ReadOnly=TRUE
*RelativeRecordNumber=<field>

[Client Subset]
CUSTOMER=Client Attribute,1,5
```

which would map requests for file PH to OS/400 file MYLIB/ABCPH, use the same numeric field editing as MiniDBMS, not use record number access (the RelativeRecordNumber keyword is left in for completeness, but commented out), and expect the customer code to be at the start of the Strategi User Attribute picked up when the original client request was made.

Group FILE

Required, this defines various file-level properties:

- HsmName = the two letter file name that will be used on HSM requests to refer to this file
- SystemName = the name of the underlying SQL table, as needed for JDBC. A new special value for [FILE] SystemName has been added, *HDSDFN. This indicates to use the database file in the same library as the definition file.
- DigitGroupingSeparator = Comma or other separator to use when editing numeric fields
- SimpleDigitGrouping = set to True to automatically edit numeric fields with commas when possible
- BlankNumericWhenZero = set to True to deliver blank instead of 0.0 for zero numeric fields
- ReadOnly = set to True to reject update requests, False to allow updates
- RelativeRecordNumber = Name of field to use for the record number if this file has no keys

Group CLIENT SUBSET

Required, this defines which fields, if any, are to be used to limit the requester's view of data in the file. The group consists of a list of field names, and which aspect of user identification they must match. Available identification information includes:

- Client Attribute (as picked up from a Strategi User Attribute by the initial HSM server definition)
- Client Id (e.g., the 9-digit Strategi user number)
- Client Type
- Client Location

- Authority Type
- Authority ID

All are accessed as per standard HSM definitions, and sub stringed by start position and length, so:

```
[Client Subset]
CUSTOMER=Client Attribute,1,5
```

requires the customer code to be in the first five positions of the user attribute.

If any of the subset field names also appear in the FIELDS group and therefore visible to HSM requests, then the FILE group must specify the file as read-only.

Group FILTER CONSTANTS

JDBCDS allows a special form of filter that is always on. This guarantees data display will be limited without relying on application HSM requests to correctly do so. The group consists of a list of fields and their required values, for example:

```
[Filter Constants]
ISPUBLIC=Y
```

Field Modifiers

Field modifiers can also be added to the field definitions. These are specified on the same line as comma separated kwd=value pairs.

For example:
[FIELDS]
someField=10, DbName=DBSOMFLD, FieldType=TextDigit, DigitGrouping=false
anotherField=14, DbName=DBANOTHR, FieldType=TextDigit, DigitGrouping=true

The allowable modifiers are:

- DbName - allows the underlying database field name to be different from the HSM field name. Note that the client subset group continues to use the underlying database field name, since the subset field is usually *not* listed in [FIELDS]; all other groups naturally refer to the HSM field name as listed in [FIELDS].
- FieldType - allows the HSM field type to overridden. Normally the field is numeric if the underlying database field is numeric, and otherwise text. This modifier introduces some specific variants.
  - TextDigit - A text field that contains only digits, and so is formatted as for a numeric field, with commas added for HSM output depending on size and whether UseDigitGrouping=true is set in the FILE group. An example would be the Open Count in Strategi's WRKSGIF, which was incorrectly defined as a text, not numeric field in the database.
  - CodeDigit - A text or numeric field whose contents are digits considered to be a code. The value is formatted without commas and is right aligned and zero filled. An example would be a Strategi user number, or Send File Reference Number, etc.
- DigitGrouping - Allows digit grouping to be enabled or suppressed on a field-by-field basis for Numeric and TextDigit fields (not Text or CodeDigit).

- BlankWhenZero - Allows blanking the value when it is zero to be enabled or suppressed on a field-by-field basis for Numeric, TextDigit and CodeDigit fields (not Text).

With HSM names separated from database names using DbName, we officially support defining a field several times, perhaps with different characteristics. Was someField lengthened? Add someField2 with the new length. Do you sometimes want someField with comma formatting, and sometimes right-aligned and zero padded? Define it twice with different HSM names and different characteristics on each. This is great for maintaining compatibility with existing resources

Definition File Creation

For MiniDBMS, the .dfn file is a simple text file, created using a simple text editor.

For JDBCDS, the definition file is a database file in the native DBMS. For OS/400 this means a Physical File that is not directly editable. Instead, a corresponding source file is maintained with normal OS/400 source maintenance tools and the final file created by Strategi command CRTHDSDFN. The source file contains simple lines laid out as for the text file version.

CRTHDSDFN parameters are:

FILE	Definition file (including library) to create or update. The name must be HDSDFN for JDBCDS.
MBR	Member in that file to create or replace
SRCFILE	Source file (including library) to create or update
SRCMBR	Source member to use, or *MBR if the same as target MBR above
REPLACE	*YES (default) to replace existing target, *NO to not replace
TEXT	Text for target member, *SRCMBR (default) for same as SRCMBR description

HSM Data Server Opcodes

All requests may return the standard HSM errors as well as the replies detailed below.

Many conditions, such as attempting to update non-existent keys, are deliberately not treated as errors (i.e., do not return a *ERROR opcode), simply returning the successful completion opcode with a data flag indicating that records were not found. This allows HSM resource files to treat all unexpected opcodes as fatal errors, and to test the not-found return value only when it matters (which is particularly useful when there are many return data fields and missing records can be ignored).

Request and reply layouts are listed with the field length followed by a description of the field. For arrays, letters are used for variable numbers, so Nx15 is an array of 15-character fields with the number of elements N defined previously, and NxL is an array where the length of entry also varies.

Current implementations of MiniDBMS allow key data supplied on requests to be shorter than the total key length, including allowing less than ten digits in a record number, but JDBCDS requires key data to be at least the total key length, and future versions of MiniDBMS will also do so. Key data is allowed to be greater than the key length, with extra data ignored, so a single set of HSM resource file statements can be used for multiple keys (e.g., listing a file in varying sequences.)

The *ERROR replies have the standard seven character code plus one space plus descriptive text layout, with detail potentially varying from release to release. JDBCDS error codes are prefixed with HDS, and MiniDBMS codes are prefixed with MDB. MiniDBMS requires that files be explicitly opened, and has a specific error code MDB0011, which will not change in future releases, that means the file being referred to was not open.

Note that comments in HSM begin with a semicolon. See the Strategi Administrator's Guide for additional details on the use of HSM.

File Opcodes

File opcodes do not apply to JDBCDS because JDBCDS accesses files on a request-by-request basis through JDBC SQL statements.

Opcode APYMODDBF *MiniDBMS*

Currently supported only by MiniDBMS, and only for the dBase file format, this takes a file of modifications and applies them to the main file. Records in the modification file are applied to the main file based on their primary key, adding or updating if the modification record is active, deleting if the modification record is deleted.

Only minimal layout compatibility checking is done, and there is no ability to map differing file layouts.

Request Opcode APYMODDBF

2	File name
1024	File path relative to the location of the file being updated

Reply Opcode APYMODDBF

No data

Opcode CLOSE *MiniDBMS*

Relevant for MiniDBMS only, this closes a file or files. It is not an error to close a file that is not open.

For JDBCDS, there is no concept of closing a file, since they are accessed on a request-by-request basis through JDBC SQL statements.

Note that this opcode writes the end of file marker to the file. If this opcode is not called on a file before uploading a file via Pocket Strategi, the upload will fail because of the lack of the end of file marker.

Request Opcode CLOSE

2 File name, or ** for all open files

Reply Opcode CLOSE

1 Indicator returns:
1 - at least one file was closed
0 - no files were closed

Example:

```
[server request]
server=MYSERVER
opcode=CLOSE
start_length_field=1,2,"**"
; HSM comment line (starts with semicolon)

[reply]
opcode=CLOSE
start_length_field=1,1,CLOSEFLAG
```

Opcode OPEN *MiniDBMS*

Relevant for MiniDBMS only, this defines what files to open. A file is considered to exist if both a definition and data file exist. For example, file XX exists if XX.dfn exists along with either XX.dbf or XX.dta (if both XX.dbf and XX.dta exist, the dbf will be used). If a .dta files is opened, and it is of zero length, then a header record is written to it with the field names (or as much of each as fits in the field width).

If there is something wrong (e.g., invalid layout) in any of the files opened, opening will continue, and all the errors will be reported together at the end (so specifying the file name as "**" will open all the files it can).

For JDBCDS, there is no concept of opening a file, since they are accessed on a request-by-request basis through JDBC SQL statements.

Request Opcode OPEN

2 File, or ** for all files in the directory
128 Absolute or relative (to Pocket Strategi root) path to file(s) to be opened

Future versions of MiniDBMS will accept only a single path in a separate initialization request, and all opens will use that path. This will mean that all files for a particular instance of MiniDBMS will be required to be in one directory, which is already normal practice.

Reply Opcode OPEN

No data

Example:

```
[server request]
server=MYSERVER
opcode=OPEN
start_length_field=1,2,"**"
start_length_field=*,128,"application/myapp"

[reply]
opcode=OPEN
```

Opcode SETMAXOPN *MiniDBMS*

Supported only by MiniDBMS, this allows the number of concurrently open operating system file handles to be explicitly set, in case the automatic settings are not adequate. The number is a total across all HSM Data Servers running with the same server class, and must be at least one for each Data Server that might be running concurrently (so for a single application server it must be at least two, since Pocket Strategi currently runs one such server internally).

On client systems with limitations on the number of open files, MiniDBMS automatically limits itself to ten less than the number of test opens that succeed, based on repetitively opening the first real file. If that number is at least 50, no limit is assumed, and every file apparently open corresponds to a system file handle. If it is less, then system file handles are dynamically closed and re-opened to allow a larger number of database files to appear to be open.

Request Opcode SETMAXOPN

4 Maximum number of operating system file handles to use, across all HSM Data Servers

Reply Opcode SETMAXOPN

1 Indicator returns:
1 - request was applied
0 - request was not applied
4 New maximum number of open handles

Filter Opcodes

Filter opcodes apply to MiniDBMS and/or JDBCDS as indicated.

Opcode SETFILTER *MiniDBMS*

This sets the specified filter value. This command will be deprecated in the near future when MiniDBMS support for opcode SETFILTER2 is complete. This opcode is now deprecated for JDBCDS in favor of opcode SETFILTER2.

MiniDBMS allows the filter setting to be for a particular file, so a given filter can be set differently for different files. This functionality will not be available in the opcode SETFILTER2, and should not be used in this command to ensure future compatibility.

Request Opcode SETFILTER

2	File name
1	Filter number
*	Filter value (see “Indexes and Filters” section for details)

Reply Opcode SETFILTER

No data

Opcode SETFILTERS *JDBCDS*

This sets the specified filter value. Currently only available for JDBCDS, but will be made available for MiniDBMS in the near future.

Request Opcode SETFILTERS

2	Filter number
*	Filter value (see “Indexes and Filters” section for details)

Reply Opcode SETFILTERS

No data

Opcode SETFILTER2 *JDBCDS*

This sets the specified filter value. Currently only available for JDBCDS, but will be made available for MiniDBMS in the near future. Replaces the now deprecated SETFILTER for use in JDBCDS. This is the preferred opcode to set filters.

Request Opcode SETFILTER2

2	Count of Files to be Named
N x 2	Files to Set Filters On
2	Filter Number
128	Filter Value (see “Indexes and Filters” section for details)

Reply Opcode SETFILTER2

No data

Example: using values of 01 for Count of Files to be Named and ** for Files to Set Filters On will set all filters for that file.
Note: Using a value of ** for Files to Set Filters On in a list of 2 or more files will result in an error.

Record Opcodes

Record opcodes apply to MiniDBMS and JDBCDS as indicated.

Opcode DLTRECORD JDBCDS and MiniDBMS

This deletes the record with the corresponding primary key. It is not an error to delete a record that does not exist.

Request Opcode DLTRECORD

2 File name
1 Key number
4 Length of key data supplied (K)
K Key data (record number if key number is zero)

Reply Opcode DLTRECORD

1 Indicator returns:
1 - a record was deleted
0 - no matching record was found

Opcode GETMTADTA JDBCDS and MiniDBMS

This returns the file metadata. The initial version only allows for file field definitions, but future versions will allow for key fields, filters, and other server settings.

Note that the reply opcode reflects the type of data returned, not the request opcode, since the opcode is intended to define the format of the data.

If there are more fields than the supplied array size allows for, or the file is not open, an error is returned.

Request Opcode GETMTADTA

10 Type of data to be returned - see below
* Detail related to type of data

Type FILEFIELDS returns field definitions, given request detail:
4 Size of arrays (A)
2 File name

Reply Opcode FILEFIELDS

4	Number of array elements used (i.e., the number of fields in file)
A x 15	Array of field names
A x 4	Array of field lengths, with leading zeros

These are as seen by HSM. For MiniDBMS DBF files, the lengths may differ from the dBase field lengths, although they will be returned in dBase field sequence, so if the HSM length is the same as the dBase length the returned values will be correct for SETRECORD. Future versions will have a separate type that returns the SETRECORD layout including expected number of decimals.

For JDBCDS files, the lengths may differ from the underlying DB2 field lengths, and will be returned in HDSDFN field sequence regardless of DB2 field order.

```
*******
* Example Use
* GETMTADTA opcode
*******
[DO]
assign_field_value=NbrOfArrElm,10

[SERVER REQUEST]
server=MYJDBCDS
opcode=GETMTADTA
start_length_field=1,10,"FILEFIELDS"
start_length_field=*,4,NbrOfArrElm
start_length_field=*,2,FileName

[REPLY]
opcode=FILEFIELDS
start_length_field=1,4,NbrElmUsd
start_length_field=*,10X15,FieldNames
start_length_field=*,10X4,FieldLengths

[REPLY]
opcode=*OTHER
start_length_field=1,9999,replyData
```

Opcode GETRECORD JDBCDS and MiniDBMS

This opcode returns nominated data fields from one record in a file, identified by its key.

Request Opcode GETRECORD

2	File
1	Key number
4	Number of fields requested (N)
4	Length of key data supplied (K)
K	Key data
N x 15	Names of fields to return

Reply Opcode GETRECORD

1	Indicator returns: 1 - record was found 0 - record was not found (in which case all returned values are blank)
L1	Value of field 1, length as per the definition file
…	…
LN	Value of field N, length as per the definition file

Opcode GETRECORDS JDBCDS and MiniDBMS

This opcode returns nominated data fields from a number of records in a file, identified by their keys. The start of the key can be provided separately, simplifying requests where part of the key is common to all records requested. Not all array elements need be used, again simplifying HSM resource files, which can be coded with fixed array sizes.

Request Opcode GETRECORDS

2	File
1	Key number
4	Number of fields requested (N)
4	Size of arrays (A)
4	Number of elements used in arrays
4	Common key prefix length (may be zero) (P)
4	Array key data length (K)
P	Common key prefix (if any)
A x K	Array of key data
N x 15	Names of fields to return

Reply Opcode GETRECORDS

1	Indicator returns: 1 - all records were found 0 - not all records were found (in which case some returned values are blank)
A x L1	Array of field 1 values, length as per the definition file
…	…
A x LN	Array of field N values, length as per the definition file

Example:

```
[server request]
server=MYSERVER
opcode=GETRECORDS
assign_field_value=ARRSZE,"15"
assign_field_value=ARRUSECM,"15"
assign_field_value=FIELDS,"1"
assign_field_value=FILE,"CM"
assign_field_value=KEY,"1"
assign_field_value=KEYLEN,"9"
assign_field_value=PFXLEN,"0"
;
start_length_field=1,2,FILE
```

```
start_length_field=*,1,KEY
start_length_field=*,4.0,FIELDS
start_length_field=*,4.0,ARRSZE
start_length_field=*,4.0,ARRUSECM
start_length_field=*,4.0,PFXLEN
start_length_field=*,4.0,KEYLEN
start_length_field=*,15x9,SMCUSNR
start_length_field=*,15,"CMCUSNM"

[reply]
opcode=GETRECORDS
start_length_field=1,1,ALLFOUND
start_length_field=*,15x30,CMCUSNM
```

Opcode LSTRECORDS JDBCDS and MiniDBMS

This gets nominated data fields from a number of records in a file, identified by a key and starting position.

The direction to go from the starting key can be

- F Forward, including any record that exactly matches the key
- B Backward, including any record that exactly matches the key
- S Start, as for Forward, but with blank treated as high values for descending key fields

Note that the positioning keys returned must be treated as opaque – the server will re-map spaces and other characters as necessary so that the values will survive translation to and from HTML and other formats, as well as perform any other reformatting.

Key fields to match can be set non-zero to return only records for which the specified number of key fields match the supplied key.

Request Opcode LSTRECORDS

2	File
1	Key number
1	Direction (F, B, or S, as above)
1	Number of key fields to match (usually 0)
1	Indicates that filters should be: 1 - obeyed 0 - ignored
4	Number of fields requested (N)
4	Size of arrays (A)
4	Length of key data (K)
K	Key to position to
N x 15	Names of fields to return

Reply Opcode LSTRECORDS

1	Indicator returns: 1 - filters were used (i.e., they were requested and were not all blank) 0 - filters were not used
4	Number of array elements used (i.e., records found) (may be zero)
K	Refresh positioning key (opaque key for first record returned, blank if

none)

K	Forward positioning key (opaque key for following record in key sequence, blank if none)
K	Backward positioning key (opaque key for preceding record in key sequence, blank if none)
A x L1	Array of field 1 values, length as per the definition file
…	…
A x LN	Array of field N values, length as per the definition file

Example:

```
[server request]
server=MYSERVER
opcode=LSTRECORDS
assign_field_value=ARRSZE,"15"
assign_field_value=FILE,"SM"
assign_field_value=FIELDS,"5"
assign_field_value=KEY,"1"
assign_field_value=KEYLEN,"26"
;
start_length_field=1,2,FILE
start_length_field=*,1,KEY
start_length_field=*,1,DIR
start_length_field=*,1,KEYFIELDSTOMATCH
start_length_field=*,1,FILTERS
start_length_field=*,4.0,FIELDS
start_length_field=*,4.0,ARRSZE
start_length_field=*,4.0,KEYLEN
start_length_field=*,26,POS
start_length_field=*,15,"SMCUSNR"
start_length_field=*,15,"SMDOCNR"
start_length_field=*,15,"SMDOCDT"
start_length_field=*,15,"SMDOCDS"
start_length_field=*,15,"SMSHPCS"

[reply]
opcode=LSTRECORDS
start_length_field=1,1,FILTERSUSED
start_length_field=*,4.0,ARRUSE
start_length_field=*,26,REF
start_length_field=*,26,NXT
start_length_field=*,26,PRV
start_length_field=*,15x9,SMCUSNR
start_length_field=*,15x9,SMDOCNR
start_length_field=*,15x8,SMDOCDT
start_length_field=*,15x30,SMDOCDS
start_length_field=*,15x6,SMSHPCS
```

Opcode SETRECORD JDBCDS and MiniDBMS

This requires data laid out as per the underlying file, and adds or updates the record based on the primary key of the supplied record. For MiniDBMS, only the data part of the record, not the active/deleted flag is supplied, but the data must be exactly correct as per the physical file, ignoring the definition file, and therefore varies depending on whether that file is a .dta or a .dbf. For JDBCDS, all the HSM fields defined for the file must be supplied, in the order and with the length defined for HSM.

Request Opcode SETRECORD

2	File name
10	Record number if file has no keys, ignored for files with a primary key
*	Record content

Reply Opcode SETRECORD

No data

Opcode UPDRECORDS JDBCDS and MiniDBMS

This updates nominated data fields in a number of records in a file, identified by their keys. The start of the key can be provided separately, simplifying requests where part of the key is common to all records involved. Not all array elements need be used, again simplifying HSM resource files, which can be coded with fixed array sizes.

It is not an error to attempt update of non-existent records.

Request Opcode UPDRECORDS

2	File
1	Key number
4	Number of fields updated (N)
4	Size of arrays (A)
4	Number of elements used in arrays
4	Common key prefix length (may be zero) (P)
4	Array key data length (K)
P	Common key prefix (if any)
A x K	Array of key data
N x 15	Names of fields to return
A x L1	Array of field 1 values, length as per the definition file
…	…
A x LN	Array of field N values, length as per the definition file

Reply Opcode UPDRECORDS

1	Indicator returns: 1 - all records were found 0 - not all records were found

Opcode ENSADDCAP *MiniDBMS*

This ensures that internal control arrays will have the capacity to accept the specified number of additional records, and it can also change the number of entries by which the arrays will be extended.

Applications can use this to control MiniDBMS in cases where the default values for free records or record extension are inefficient and/or log-intensive.

These settings have no effect on the dbf or dta file on disk, which on being re-opened will revert to a capacity equal to the number of records physically in the file, extending by 10% of that initial number or by 10 if there are less than 100 records initially.

If there are already enough free entries to accept the added capacity, there is no change to size.

If the new extension value is zero or not supplied, the existing setting is left unchanged. Note that the extension value must be of length 10 or the value will not be used. Leading space or zero padding will meet this requirement.

Request Opcode ENSADDCAP

2	File
10	Number of free record entries to be ensured
10	Number of entries for following extensions

Reply Opcode ENSADDCAP

No data is returned

Opcode ENSTOTCAP *MiniDBMS*

This ensures internal control arrays have a total capacity of the specified number of records, and it can also change the number of entries by which the arrays will be extended.

Applications can use this to control MiniDBMS in cases where the default values for free records or record extension are inefficient and/or log-intensive.

These settings have no effect on the dbf or dta file on disk, which on being re-opened will revert to a capacity equal to the number of records physically in the file, extending by 10% of that initial number or by 10 if there are less than 100 records initially.

If there are already enough entries, there is no change to size, and size is not reduced to match the current number of entries.

If the new extension value is zero or not supplied, the existing setting is left unchanged. Note that the extension value must be of length 10 or the value will not be used. Leading space or zero padding will meet this requirement.

Request Opcode ENSTOTCAP

2 File
10 Number of record entries to be ensured
10 Number of entries for following extensions

Reply Opcode ENSTOTCAP

No data is returned

Opcode GETCAPINF *MiniDBMS*

This returns the capacity information for the MiniDBMS file.

The first two values reflect the internal MiniDBMS array information. The last two entries reflect the physical dbf or dta file information.

Request Opcode GETCAPINF

2 File

Reply Opcode GETCAPINF

10 Number of entries allowed for in MiniDBMS control arrays
10 Number of entries by which control arrays will extend
10 Number of records in dbf or dta file, active and deleted
10 Number of active records in file

Appendix 1 – MiniDBMS Data File Size Calculation

Although the MiniDBMS has the ability to store and track files with millions of records, there is a limit imposed (as is true for all databases) on the number of records, but the real limiting factor is the amount of memory available on the computer.

MiniDBMS uses a signed 32-bit integer to keep track of the number of records, limiting it to 2G minus 1 (i.e., 2,147,483,647 records). This is half the dBase file limit of 4G minus 1, or 4,294,967,295.

But the practical limit is the memory required to hold the fast, in-memory indexes and filters that are built as a file is opened (the theoretical limit is 2Gb for each index).

The memory requirement is calculated for a file as follows:

numberOfRecords * (numberOfKeys*24 + keyLengthsEven + filterLengths)

Where:

- numberOfKeys is the number of keys defined, not the highest key number (e.g., if only KEY1 and KEY3 are defined, that counts as two not three)
- keyLengthsEven is the total bytes in all the keys, rounded up to an even number for each key (e.g., if KEY1 is 17 bytes long and KEY3 is 19 bytes, total will be 18 + 19 = 38)
- filterLengths is the total length of any filters defined on the file

Here are a few examples from the field:

	# of Records	File Size	Runtime Memory
High # of Records	114,401	7MB	5MB
Max Memory	54,919	8MB	12MB

Appendix 2 – Strategi Data Server

The Strategi Data Server is a specific instance of JDBCDS that provides READ-ONLY access to certain Strategi database files. Currently the files supported are SGICT (connections), SGITT (file transmissions) and SGIUS (users).

HDSDFN example for SGICT access:

```
[FILE]
HsmName=CT
SystemName=*HDSDFN/SGICT
ReadOnly=TRUE
DigitGroupingSeparator=','
SimpleDigitGrouping=TRUE
BlankNumericWhenZero=FALSE

[CLIENT SUBSET]

[FIELDS]
connectNumber=10, Type=CodeDigit
userNumber=9, Type=CodeDigit
connectDts=14
disconnectDts=14
statusCode=2, Type=CodeDigit
clientCode=2
clientVersion=100
clientIpAddress=12, Type=CodeDigit
linkSecurity=1
linkDetail=50
activityCount=14, Type=TextDigit
accessPoint=50

[KEY 1]
connectNumber=A

[KEY 2]
connectNumber=D

[KEY 3]
connectDts=A
connectNumber=A

[KEY 4]
disconnectDts=A
connectNumber=A

[FILTER CONSTANTS]

[FILTER 01]
userNumber

[FILTER 20]
clientCode
```

```
[FILTER 21]
accessPoint
```

HDSDFN example for SGITT access:

```
[FILE]
HsmName=TT
SystemName=*HDSDFN/SGITT
ReadOnly=TRUE
SimpleDigitGrouping=TRUE
DigitGroupingSeparator=','
BlankNumericWhenZero=FALSE

[CLIENT SUBSET]

[FIELDS]
sendNumber=10, Type=CodeDigit
sendUnitNumber=10, Type=CodeDigit, BlankWhenZero=true
userNumber=9, Type=CodeDigit
statusCode=3
priorityCode=1
openCount=6, Type=TextDigit
sentRcvdCode=1
textDescription=40
origin=40
fileName=64
fileTypeCode=1
mimeType=64
fileSize=14, Type=TextDigit
fileDts=14
createDts=14
queueArriveDts=14
sentEventDts1=14
sentEventDts2=14
deleteDts=14
transferGroup=15
trackingNumber=10, Type=CodeDigit, BlankWhenZero=true

[KEY 1]
sendNumber=A

[KEY 2]
sendNumber=D

[KEY 3]
statusCode=A
sendNumber=D

[KEY 4]
transferGroup=A
trackingNumber=D
sendNumber=D
```

[KEY 5]
createDts=D
sendNumber=D

[KEY 6]
sentEventDts1=D
sendNumber=D

[KEY 7]
sentEventDts2=D
sendNumber=D

[KEY 8]
deleteDts=D
sendNumber=D

[KEY 9]
sendUnitNumber=D
sendNumber=D

[FILTER CONSTANTS]

[FILTER 01]
userNumber

[FILTER 02]
transferGroup

[FILTER 20]
sentRcvdCode

[FILTER 21]
statusCode

[FILTER 22]
textDescription

[FILTER 23]
fileName

[FILTER 24]
priorityCode

[FILTER 25]
fileTypeCode

[FILTER 26]
mimeType

[FILTER 27]
sendUnitNumber

HDSDFN example for SGIUS access:

```
[FILE]
HsmName=US
SystemName=*HDSDFN/SGIUS
ReadOnly=TRUE
DigitGroupingSeparator=','
SimpleDigitGrouping=TRUE
BlankNumericWhenZero=FALSE

[CLIENT SUBSET]

[FIELDS]
userNumber=9, Type=CodeDigit
Status=1
userClass=3
userName=40
remoteNumber=5, Type=CodeDigit, BlankWhenZero=true
passChangeDts=14
passControl=1
createdBy=40
createdDts=14
expiryDts=14
expiryRule=1
expiryDays=3, Type=CodeDigit, BlankWhenZero=true
expiryAction=1
ipAddress=12
ipAddressSubnet=12
sslKeyCode=1
sslCertCode=1
allowEmulation=1
allowSendSpool=1
allowSendFile=1
allowHsm=1
allowGuiStyle=1
allowPocketSgi=1
allowRemote=1
allowHttpLogin=1
allowRecvFile=1
emuUserName=10
emuDeviceName=10
emuIdleTime=4, Type=CodeDigit
emuSessions=2, Type=CodeDigit
emuRetention=4, Type=CodeDigit
textDescription=40
lastLoginDts=14
lastEnabledDts=14
firstName=20
lastName=30
title=40
organization=40
emailAddress=64

[KEY 1]
userNumber=A
```

[KEY 2]
userNumber=D

[KEY 3]
userName=A

[KEY 4]
remoteNumber=A
userNumber=A

[KEY 5]
lastLoginDts=A
userNumber=A

[FILTER CONSTANTS]

[FILTER 01]

[FILTER 21]
status

[FILTER 22]
userClass

[FILTER 23]
emuUserName

[FILTER 24]
textDescription

[FILTER 25]
createdBy

[FILTER 26]
organization

[FILTER 27]
sslKeyCode

[FILTER 28]
sslCertCode

[FILTER 30]
allowEmulation

[FILTER 31]
allowSendSpool

[FILTER 32]
allowSendFile

[FILTER 33]
allowHsm

[FILTER 34]
allowGuiStyle

[FILTER 35]
allowPocketSgi

[FILTER 36]
allowRemote

[FILTER 37]
allowHttpLogin

[FILTER 38]
allowRecvFile

Appendix 4 – Data Server Log Files

When a data server is started, a corresponding log file is created in the Strategi IFS directory '/STRATEGI/tmp'. This log may be useful in helping to troubleshoot any problems with your server.

This log file has the following naming convention:

```
/STRATEGI/TMP/jdbcds-XXXXXXXXX-001.txt
```

Where

- "XXXXXXXXX" is the name of the data server
- "001" is the log file sequence (001 is usually the current log)

Index