VsamEx[treme]™

The Original "un-database"

(With SHA+ Encryption)

Reference Manual

Feburary 2011

Windows & Linux Edition

Software Source

PO Box 23306
San Jose, CA 95153
United States of America

Email
software_src@earthlink.net

Web
www.1-software-source.com

Limited Warranty

Software Source provides the **VsamEx[treme]™** software and accompanying materials with the following limited warranty:

When used in accordance with instructions, Software Source warrants this product against any defects due to faulty materials or workmanship, for a period of sixty days from the purchase date. If Software Source receives notification within this warranty period of defective materials or workmanship, and determines that such notification is correct, then Software Source will replace the defective product distribution and/or documentation at no charge. This warranty does not cover damage due to accident, abuse, misuse, or improper installation of the product. Software Source authorizes no other warranty, written or oral, and there are no implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. In no event will Software Source be liable for any damages, including any general, special, incidental, indirect, exemplary, or consequential damages arising out of the use, misuse, or inability to use the **VsamEx[treme]™** product. The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective product distribution and/or documentation and shall not include or extend to any claim for or right to recover any other damages, including, but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if Software Source has been specifically advised of the possibility of such damages; in no event, however, will Software Source's liability exceed the actual amount paid for the product.

NOTICE:

ANY USE OF THIS PRODUCT IMPLIES AGREEMENT WITH THE TERMS OF THIS WARRANTY! IF YOU DO NOT AGREE TO THESE TERMS PRIOR TO USING THIS PRODUCT, PLEASE RETURN IT TO RECEIVE A FULL REFUND OF ITS PURCHASE PRICE.

Our Special Thanks for contributions to this project go to:

- ***Jay J. Falconer*** *at Bitwise Software International, Inc., http://www.shoppingQ.com, for his help in testing VsamEx for Linux, his many suggestions and his creation of a VsamEx C++ class wrapper that ships with the product.*

- ***Mike Whittingham*** *at Chaos Software, Inc., http://www.chaossoftware.com, for his long time support, kind critical comments and meaningful suggestions.*

Visit their websites to see real world applications of VsamEx[treme] ™ and its predecessor VB/ISAM™!

Tony Altwies
Software Source

TABLE OF CONTENTS

Copyright Notice and Trademarks

Software License Keys

VsamEx[treme]™ **Libraries, License Keys**, **software, and documentation** are the sole property of **Software Source**. Said property is licensed not sold. **Software Source** retains sole title and rights to all said property except rights specifically granted to others by **Software Source**. You may purchase and register rights to use a **License Key** directly from **Software Source**. **Software Source** grants licenses so purchased for the exclusive use with **VsamEx** for the purpose of software development, provided all of the following conditions are met:

1. Only one person at a time on one machine may use a particular **License Key** for the purpose of developing applications linked to **VsamEx[treme]**™.
2. You must use a **License Key** provided to you by **Software Source**.
3. You must register the transfer, if any, of the **License Key** with **Software Source**.
4. You may not enable unregistered use of **VsamEx[treme]**™ in a development environment.
5. You may not use a **License Key** already registered to someone else.
6. **Software Source** must have a record of originally issuing the **License Key**.

As long as all of the above conditions are met, you are expressly granted the rights to copy, transfer, and distribute your applications linked with the **VsamEx[treme]**™ **Library or DLL,** as part of the software you develop using **VsamEx[treme]**™, without Royalty**.**

FUNCTIONAL DESCRIPTIONS

VsamAddField

short VsamAddField (

LONG	Datasetnumber,	LPSTR	lpFieldName,
LPSTR	lpFieldType,	WORD	wDimension,
BOOL	IsIndex,	WORD	wWidth,
LPSTR	lpJust,	LPWORD	lpFieldNumber)

Description: Add a new Data class field description to a dataset.

Records in a dataset may be thought of as being made up of attribute values called fields. When a Dataset is created, it initially contains one Data class field definition; the "Primary" key field definition (see VsamCreate). All records in a dataset are physically made up of Data class fields. Fields are sparse and records do not need to contain all fields. As a minimum, a record must contain a Primary key field. This function allows users to define additional fields. Fields may be added to the dataset at any time. Records that do not contain a field will not return values for that field when requested (see VsamFetchField).

Arguments: *Initial field attributes*

DatasetNumber&: The reference number returned by VsamOpen.

lpFieldName$: The Field Name as a string. You may optionally specify the field number as "%n" where n is the string value of the field number.

lpFieldType$: The Field Type as a string ($,%,&,#,Cf.w).
Where:

"$"	=	String	(Variable length)
"%"	=	Integer	(2 byte numeric)
"&"	=	Long	(4 byte numeric)
"!"	=	Single	(4 byte Floating)
"#"	=	Double	(8 byte Floating)
"@"	=	Currency[1]	(8 Byte numeric)
"Cf.w"	=	Compound[2]	(Concatenated strings)

1. Windows Only.
2. Automatically sets this field as an index field.

See the Supported Field Types section

Examples:
"C5.3:6.10" field 5 for 3 bytes plus field 6 for 10 bytes.
"C7:12.20" all of field 7 plus field 12 for 20 bytes.

***wDimension%:** For numeric fields, this indicates the number of array elements in this field. The maximum number is 65500/ value size. So for LONG's, the max is 65500/4 = 1625 elements. (*Remember: maximum record size is 65500).*
(Currently only 1 is allowed)

isIndex%: Boolean - TRUE if this field is an index field. A Compound definition will set this attribute to TRUE, regardless of what is set in this parameter.

wWidth%: Field width (user attribute default = 255).

lpJust$: Field justification.
Where: "L" = Left; "R" = Right; "C" = Center
lpJust and wWidth do not affect the raw record data. They are only intended as reference for building print records.

lpFieldNumber%: The field number assigned is returned in this parameter. Once The system reserves this field number, it will not be assigned to any other field until this field has been "Purged".

**** This parameter is intended for future use and only 1 dimension is support for now.***

Remember, VsamEx maintains a separate, independent table entry for each index field. Index pointers, including the Primary index pointer, may be moved <u>independent</u> of one another and do not <u>interfere</u> with each other. However, any add or change to a field definition will cause all index pointers to be reset and <u>undefined</u> until re-established by a VsamGet, BOF or EOF operation! This applies to Dictionary index pointers as well.

Function return codes:
VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_NO_ROOM – *No room left for field definitions*
All field definitions and their attributes must fit inside 65530 bytes.
VIS_BAD_PARAMETER_VALUE – The f*ield type is not legal*
VIS_ALREADY_EXISTS – *The field definition already exists*

VsamBOF

short VsamBOF (LONG Datasetnumber, LPSTR Index)

Description: Sets the file pointer in a specified index to BOF: Before the First entry in the index, if any.

Arguments:

DatasetNumber&: The reference number returned by VsamOpen.

Index$: To select the primary index, use "%0" or "Primary". To select a secondary index, use the Field name or Field number in quotes preceded by the "%" symbol. (see the VsamAddField function description).

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamCancel

short VsamCancel (LONG DatasetNumber)

Description:
This Function will cancel any currently active **VsamMovePtr** operation on the dataset specified. Then use **VsamGet**/XCURRENT to determine the location of the pointer.

Arguments:

DatasetNumber&: The reference number of the dataset returned by **VsamOpen**.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*

VsamClose

short VsamClose (LONG DatasetNumber)

Description: Closes an open **VSAM** dataset.

*» Caution: if you make any changes to a dataset (with **VsamPut**, **VsamDelete**, or **VsamWriteDict in non-shared modes**), you must call **VsamClose** before ending your program; if you don't, the last change you made to the dataset may not be saved -- even though the system later closes the files. For performance reasons, **VsamEx** doesn't flush its private memory buffer to disk, while in **non-shared** modes, until you close the dataset. (See the **VsamFlush** function description.) Because closing datasets is so important, we recommend that you place your VsamClose calls in the code section where they'll always be executed no matter how your program ends.*

The other alternative is to use a shared mode, in which case buffers are always flushed!

Arguments:

***DatasetNumber&*:** The reference number returned by **VsamOpen**.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_HANDLE – *The dataset handle does not belong to VsamEx.*
VIS_DOS_ERROR – *An operating system error occurred*
VIS_DISK_ERROR – *A problem with one of the disk files was encountered*

VsamCsvDefMap

short VsamCsvDefMap (LONG DatasetNumber, LPSTR Fname)

Description: Initalize Csv Loader by defining a CSV field to Vsam Field mapping.

Arguments:

DatasetNumber&: The reference number of the dataset returned by **VsamOpen**.

Fname$: The name of the ".cmf" file that contains the import mappings. The contents are as follows:

CSV HEADER	Comma separated field names,
PRIMARY KEY MAPPING	"Primary" Must be the name of the Rec key.
VSAM FIELD NAME and MAPPING	All subsequent Vsam fields that get CSV Mappings.

.
.
Example*: Line 1 is always assumed to be the CSV Header Specification.*
"Last Name","First Name","Country","City","Street Name","House Number","Resident"

Primary,40=Last Name,"_",First Name","_",Country,"_",Resident:%02d"
Fields may be concatenated with literal delimiters or formatted text
Field Width for display may be defined i.e. "40".

Country,20=Country
City,10=City
Street Name,35=Street Name
House Number,5=House Number:%04d
Resident,5=Resident:%02d

Here we have defined Vsam records consisting of the Key (Primary) formed from each CSV field specified along with a formatting string. Each field in Vsam is named along with its display width and the corresponding CSV fields that it is produced from.

Function return codes:

VIS_OK - *Function call completed successfully*

VsamCsvWriteRec

short VsamCsvWriteRec (LONG DatasetNumber,
LPSTR CsvInputData
WORD Mode)

Description: Write a Vsam Record to an open Dataset initialized from the VsamCsvDefMap function. VsamCreate may be used to create the dataset.

Arguments:

DatasetNumber&: The reference number of the dataset returned by **VsamOpen**.

CsvInputData$: A single data line from a CSV style file where fields are separated by commas (,).

1. The data lines may be quoted using Double quote (“). Beginning and ending quotes will be removed.
2. If a field does not begin with a Double Quote (“) the field will end when the first comma (,) is encountered.
3. If a field begins with a Double Quote (“), commas (,) may be present inside the field.
4. A Double Quote in the field may be indicated by using two Double Quotes in a row (“”).
5. In a Quoted field, after resolving all Double Quote pairs (“”), the field ends at the next Double Quote (“). All data until the next comma (,) is ignored .

Mode%: How records will be written with duplicate keys.

1. Mode = 0 return UPDATE_VILOATION if record is already present with the specified Key, i.e. ADD_ONLY.
2. Mode = 1 indicates that if a duplicate key exists, the key for the current record will be appended with a numeric sequence number i.e., in the form “-%04d”. The process will increment the sequence number and continue until an UPDATE_VILOATION is not encountered.
3. Mode = 2 indicates that the record data with that key will be overwritten, i.e. ADD_OR_REPLACE.

Function return codes:

VIS_OK - *Function call completed successfully*

VsamCreate

short VsamCreate(LPSTR DatasetName,
LONG GroupSize,
LPSTR Encrypt)

Description: Creates a new **VsamEx** dataset, establishing its default “Primary” field definition. The dataset remains unopened.

Arguments:

DatasetName$: The name you want for the new dataset, optionally including a full pathname. **VsamEx** will create two files: *DatasetName$***.VOD**, and *DatasetName$***.VOM**. If you include an extension, the first two chars will be used to create dataset extensions; i.e. “mydata.nxt” would produce “.nx**d**”, “.nx**m**”, etc., instead of the default “.vod”, and “.vom”, etc.

GroupSizet$: This parameter will establish the initial size of each dataset group. It should always be defined in increments of 1024. We typically use 1024, 2048, 4096, 8192, 16384 and **32768** (maximum size). Experimental results show that 2048 is the best choice in most environments (and default value) for maximum performance.

NOTE:
The Group defines the smallest unit of data moved between the storage media and low level VsamEx routines. This is true even in networked systems. This number is important in that it also determines the maximum size that a dataset can grow to. To calculate the maximum extent your dataset can grow to, simply multiply the maximum number of groups (65,530) by the size of each group.

For example: A Group size of 8192 will yield a maximum dataset size of 536,821,760. Once established, Group Size cannot change except as a consequence of rebuilding the dataset using the **VsamRebuild** function. Of course, a more primitive and somewhat slower way would be to create a new dataset and write a little program to read records from the old dataset and put them back into the newer dataset. This method will still not create a dataset as compact, or as fast as is done in **VsamRebuild**.

Encrypt$: This key will enable Dataset Encryption. It may be any arbitrarily long string of characters (usually printable) that does not include character 0). If the Encryption key is NULL, then no encryption is performed. Once a dataset has been created with an Encryption Key, **VsamOpen** will only open the dataset properly if the exact same key is passed to it.

While the actual encryption key is not stored anywhere, the results of using an invalid key are not always predictable! When the create function was called, various critical dataset information was encrypted using the key passed at that time. When the VsamOpen function is called, the same critical information required to access the dataset must be decrypted before anything meaningful can be done. Therefore, the same Encryption key must be used in both cases.

Encryption is done using a modified ultra high speed Secure Hashing Algorithm (SHA). The cost in performance can vary depending on the record size and group size. We have estimated it to be approximately 4% loss in performance when using encryption. ***If you loose your key, the probability is very high that your data will not be recoverable!!***

** Even we at Software Source, with our ultra fast quark computers and special gluon grease, have found it necessary to travel back in time in order to recover lost encryption data! So far, the only individual we know of that can afford these services works at Microsoft!*

VsamCreate

Encryption/Decryption is performed in such a way that VsamEx data moves from host to client and back, fully encrypted even over LAN and WAN.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_PARAMETER_VALUE - *Invalid GroupSize or no DataSet Name*
VIS_ALREADY_EXISTS *Try another DatasetName$.*
VIS_DOS_ERROR - *Could not create at least one of the files.*
VIS_OUT_OF_MEMORY – *Failed to create the "Primary" field*
VIS_DISK_ERROR – *Operating system error or hardware failure*
VIS_OUT_OF_FILE_HANDLES – *Windows OS Error*

Example:

```
  int   rc, grp_size;
  grp_size = 8192;

  rc = VsamCreate("MyDataset", grp_size, "MyEncryptionKey");
  show_status("Create", rc);
```

VsamDelete

short VsamDelete(LONG DatasetNumber, LPSTR PrimaryKey)

Description: VsamDelete finds a record by primary key lookup and deletes it. Secondary Indexes are automatically updated as required. This does *not* reposition any index pointers.

Arguments:

DatasetNumber&: The reference number returned by **VsamOpen.**

PrimaryKey$: The key to be used as a record locator in the primary index.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_NOT_FOUND - *The primary-index lookup was unsuccessful*
VIS_ACCESS_DENIED - *The dataset is not open for READ_WRITE access*
VIS_INVALID_KEY - *The lookup key value you supplied was either null (0 bytes long); was longer than 252; or contained a binary 0 [NULL] or 1 [Ctrl-A].* ***Nothing was done!***
VIS_DISK_ERROR – *The operating system reported a file error*
VIS_DATA_VALIDITY_CHECK – *The dataset may be corrupted*

VsamDeleteDict

short VsamDeleteDict (LONG DatasetNumber, LPSTR DictKey)

Description: VsamDeleteDict will delete a dictionary record with the specified key.

DatasetNumber&: is the same value returned from **VsamOpen.**

DictKey$: key name of the Dictionary element being deleted.

VIS_OK - *Function call completed successfully*
VIS_BAD_HANDLE – *The dataset handle does not belong to VsamEx.*

VsamDeleteField

short VsamDeleteField (LONG DatasetNumber, LPSTR FldName)

Description: This function will set the *Deleted* mode flag for a particular record field. This field will no longer exist logically. While subsequent operations will treat any requests for data from this field as being non-existent, the Dictionary defining element will remain in the Dataset dictionary until a **VsamRebuild** operation has been completed. Once the rebuild is complete, we will have removed this field from all records in the dataset. The physical dataset field number for this field will now be available for re-use! Meanwhile, as records are read and re-written, this field will be removed individually from records

DatasetNumber&: is the same value returned from **VsamOpen.**

FldName$: is the name of the record field being deleted.

NOTE: This sets the "Fdel" attribute to "T". You can restore this field to normal status by using the **VsamSetFieldAttribute** function and specifying "F" as the value. Remember, any records written when this attribute is "T" will have this field removed from their records prior to storing them in the dataset. Resetting the value to ""F" will not recover any of the removed fields in records written with "T" set.

VIS_OK - *Function call completed successfully.*
VIS_BAD_HANDLE – *The dataset handle does not belong to VsamEx.*

VsamEncrypt

```
short VsamEncrypt (LPSTR  lpSourceData,
                   DWORD   lData,
                   LPSTR   lpEkey,
                   LONG    CryptOpt,
                   LPSTR   lpResult,
                   DWORD   lResult)
```

Description: Encrypts a Source data string into AsciiHex or Decripts an AsciiHex string into a Data string. The Data string may be any binary data block of length lData including binary zero.

Arguments:

lpSourceData$: A pointer to the source input data buffer. For ***CryptOpt*** = 0 (Decrypt), this buffer will contain AsciiHex and therefore must be exactly 2 times the length of the lResult buffer. **(lSourceData == 2 * lResult)**

lSourceData&: The length of the data buffer in bytes.
For CryptOpt = 1: lResult == 2 * lSourceData
For CryptOpt = 0: lSourceDatat == 2 * lResult

lpEkey$: A string pointer to the Encryption key (zero terminated). The same key must be use to both Encrypt and Decrypt the data.

CryptOpt&: ***1*** **= Encrypt, 0 = Decrypt**.

lpResult$: A pointer to the buffer to receive the output. For ***CryptOpt*** = 1 (Encrypt), this buffer will receive AsciiHex Text and therefore must be exactly 2 times the length of lpSourceData. **(lResult == 2 * lSoutceData)**

lResult&: The length of the Result buffer in bytes.
For CryptOpt = 1: lResult == 2 * lSourceData
For CryptOpt = 0: lSourceDatat == 2 * lResult

Function return codes:
VIS_OK - *Function call completed successfully*
VIS_BAD_PARAMETER_VALUE - *Buffer sizes do not correspond or eKey is NULL*.

VsamEnumAttribValues

```
short VsamEnumAttribValues (LONG DatasetNumber,
                            WORD     Option,
                            LPSTR    lpAttribName,
                            LPSTR    lpBuf,
                            LPWORD   lpBufSize,
                            LPWORD   lpwNumFields);
```

Description: Returns a list of attribute values, separated by commas, in a buffer.

Arguments:

DatasetNumber&: The reference number returned by **VsamOpen.**

Option%: 0 = ACTIVE fields; 1 = DELETED fields; 2 = ALL fields.

lpAttribName$: A string pointer to the Attribute name (zero terminated). See **VsamSetFieldAttribute** for a list of Predefined default attribute names and descriptions. User attribute values may be enumerated as well.

lpBuf$: A pointer to the buffer to receive the attribute values separated by a comma. Fields without a value for that attribute will return a null value. Field Number ("Fnum" attribute) values will be returned in the form of "%n" where n is the actual field number in ascii.

lpBufSize%: Size, in bytes, of the buffer pointed to by lpBuf$. If the buffer is too small, it will contain the size, in bytes, needed.

lpNFields%: Pointer to a variable to return number of fields processed. This also represents the number of fields returned - separated with the comma (",").

Function return codes:
VIS_OK - *Function call completed successfully*
VIS_NO_ROOM – *Not enough buffer space to complete the operation*

VsamEnumFieldAttrib

```
short VsamEnumFieldAttrib (LONG DatasetNumber,
                           LPSTR     lpFldName,
                           LPSTR     lpBuf,
                           LPWORD  lpBufSize,
                           LPWORD lpwNumAttribs);
```

Description: Returns a list of Attribute Names for a given field byFldName/FldNum, separated by commas, in a buffer.

Arguments:

DatasetNumber&: The reference number returned by **VsamOpen.**

lpFldName$:	A pointer to the Field Name/Number (“ fieldname or %n”).
lpBuf$:	A pointer to the buffer to receive the field attribute names separated with a comma.
lpBufSize%:	Size, in bytes, of the buffer pointed to by lpBuf$. If the buffer is too small, it will contain the size, in bytes, needed.
lpNFields%:	Pointer to a variable to return number of field attribute names found.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_NO_ROOM – *Not enough buffer space to complete the operation*
VIS_BAD_PARAMETER – *Field does not exist*

VsamEOF

short VsamEOF (LONG DatasetNumber, LPSTR Index)

Description: Sets the file pointer in a specified index to its EOF position: After the last entry in the index.

Arguments:

DatasetNumber&**:** The reference number returned by **VsamOpen**.

Index$: To select the primary index, use "%0" or "Primary". To select a secondary index, use the Field name or Field number in quotes preceded by the "%" symbol. (see the VsamAddField function description).

Remember, VsamEx maintains separate table entries for each index. Pointers into the dataset may move independent of one another and do not interfere with each other.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamFetchField

short VsamFetchField (LONG DatasetNumber, LPGSTR Record, LPSTR FieldName, WORD Element, LPSTR FieldType, Void *FieldData, WORD *LenData)

Description: This Function will fetch field data from the record data (Record GSTR) read by **VsamGet.** The data will be stored in the memory pointed to by FieldData as the type specified by FieldType.

Arguments:

DatasetNumber&: Is the same as was returned from **VsamOpen**.

Record: Is the GSTR handle returned by **VsamGet** function.

**** This is a structured pointer to an allocated memory buffer and it belongs to your application. You must be sure to free it when you are through with it. (Use VsamFreeRec)***

***FieldName$*:** Is the record field name (optionally "%n" to use the field Number). You cannot use this function to retrieve the Primary (%0) field. Remember, Primarys are returned as a separate parameter.

****Element%*:** Is the array element to return {0 – (n-1)}. -1 returns all values into an array. (Only 1 element is supported at this time)

***FieldType$*:** Is a string value which defines the basic field type. I.E. "$" defines a string, "%" defines an integer, etc.

***FieldData$*:** Is a pointer to a buffer where the field data will be returned. If the type is a string or an array of values, the actual number of bytes returned will be stored in LenData. Its basic field type must be what is defined in the FieldType$ parameter. Furthermore, it must correspond to the field specified in Record$ that is defined above.

VsamFetchField

LenData%: On input, this is the size of the FieldData$ buffer. On output, this is the actual length in bytes of the data returned if the field is a string or an array. *If the data will not fit in the buffer, an error is returned and the value stored here is the size in bytes of the buffer required, not including the terminating zero*.

* Designed for future enhancement - the ***Element%*** parameter will allow the user to specify an element in an array of fields of the type. Optionally, -1 will return the entire array of values or elements.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_NO_ROOM – *The output buffer is too small the value returned in LenData% indicates how large the buffer must be, not including the terminating zero.*

VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

Example:

```
ldat = sizeof(dat);
rc = VsamFetchField(hwmcb, gstrDat, “NameField”,  1, "$", dat, &ldat);
```

VsamFlush

short VsamFlush (LONG DatasetNumber)

Description: "Flushes" the dataset's memory buffers to disk so that the most recent updates you made using **VsamPut**, **VsamDelete**, etc. are written to the disk file - even if your program crashes before it executes **VsamClose** on this dataset. **VsamEx** includes this function for the cases in which your application can tolerate reduced performance as a trade-off for saving your data. See **VsamClose**.

Arguments:

***DatasetNumber%*:** The reference number returned by **VsamOpen**.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_DISK_ERROR – *The Operating system detected a failure during the write operation.*

VsamFreeRec

short VsamFreeRec (LPGSTR lpRecord)

Description: Free the Memory buffer allocated by the **VsamGet** function for a data record. (The memory space pointed to by a GSTR)

Arguments:

lpRecord&**:** The reference number returned by **VsamOpen**.

Function return codes:

VIS_OK - *Function call completed successfully*

VsamGet

short VsamGet (LONG DatasetNumber,
LPSTR Index,
WORD Options,
LPSTR Selector,
LPSTR RIndexEntry,
LPSTR RPrimaryKey,
LPGSTR *RRecord)

Description: Reads a record through an index access, either by key lookup or sequentially, repositioning the selected index pointer as described in the **FUNDAMENTALS -- USING INDEXES** section in the VsamEx User's Guide.

This is a powerful function that actually performs up to three operations in sequence, under the control of a three-part Options parameter.

The first, and fundamental, part of the *Options%* parameter specifies the access mode -- how you'll move the index pointer. The four alternative keywords are: **XLOOKUP, XNEXT, XPREVIOUS, XCURRENT.**

*» There's a major distinction between the Lookup access mode and the other three access modes (***XNEXT, XPREVIOUS, XCURRENT***). You can actually think of* **VsamGet** *as* **two** *functions that work very differently. We originally designed* **VsamEx** *with a "VsamFind" function for the lookup mode and a VsamStep function for the other modes, but we decided to combine them into* **VsamGet**.

Lookup access mode :

In lookup access mode, **VsamGet** repositions the selected index pointer according to your *Selector$* argument. In other words, **VsamGet**/lookup searches (seeks) that index for a match of your *Selector$* argument, and leaves the index pointer at a new location; the previous position of the pointer doesn't matter, and is forgotten. You can think of the index as a Rolodex file, the pointer as your finger, and the *selector$* argument as the card-header-name you're trying to find. The lookup ends with your finger either on a card headed by that name (**VIS_OK**) or with your finger between cards, where a card with the lookup name *would be* if it were there (**VIS_NOT_FOUND**).

To say this more formally: The lookup access mode uses the *Selector$* argument as a lookup key. If the lookup is successful, it moves the index pointer to the matching index entry and (optionally) retrieves the corresponding data record. If there is more than one matching index entry (possible only in the case of secondary indexes, not the primary index), it will select the earliest one (i.e., the one whose corresponding primary key is first).

Alternatively, if it doesn't find any match of the *Selector$* in the index, it returns **VIS_NOT_FOUND** and leaves the index pointer positioned at the insertion point ("phantom entry") where that *Selector$* entry would be if present.

next, previous, and current access modes:

By contrast, the other three access modes -- next, previous, and current -- move the selected index pointer one step forward or backward (or for current, not at all), from wherever the pointer had been positioned. You can think of a next access, for example, as the process of simply moving your finger forward in the Rolodex by one card.

If the next/previous/current access is successful, meaning you didn't step "off the edge" to **EOF** or **BOF**, and after the "step move" there is an index entry underneath the pointer, **VsamGet** can then perform the operational service of comparing that encountered index entry against your *Selector$* argument. If you ask for a comparison, the function return code tells you the result: VIS_OK means that your specified comparison test was "true". The second part of the *Options%* parameter specifies whether or not to perform a comparison, and if so, which kind of comparison to use. The eight alternative keywords are: XEQ (equal to), XNOT (not equal to), XBEGINS (begins with), XLT (less than), XGT (greater than), XLE (less than or equal to), XGE (greater than or equal to), XANY (accept anything). Note that in the next, previous, and current access modes, the Selector$ argument has an entirely different purpose than in the lookup mode.

If the specified comparison fails, the function immediately returns **VIS_NOT_FOUND**. This makes it convenient for the program to loop on **VsamGet**/Next **While** (or **Until**) it returns **VIS_OK** (or **VIS_NOT_FOUND**) so you can easily process all duplicate keys (i.e., "Smith"), or move beyond or before a group of entries.

» *Note: The comparison tests do* **NOT** *cause any further pointer movement; their sole purpose is to make it easy for you to know when to stop stepping through an index. See the coding examples at the end of this function description.*

VsamGet

Record retrieval:

For all access modes, there is a final choice: Whereas **VsamGet** normally, as a default, retrieves both the primary key and the data portion of the record, it will optionally retrieve the primary key only, which is faster if that is all you need. Your application may, for example, want to browse through one or more of the secondary indexes to accumulate a list of primary keys for later use in actual record retrieval. Option keyword: XNO_DATA. (The other choice is XGET_DATA, but that is the default, so you do not need to specify it.)

Arguments:

DatasetNumber&: The reference number returned by **VsamOpen**.

Index$: To select the primary index, use "%0" OR "Primary". Optionally, To select a secondary index, use the field name or number ("%n") for the corresponding Index field (see **VsamAddField**). The field must have the IsIndex field attribute set to true. This attribute is set for the "Primary" field as a default during **VsamCreate**.

Options%: Is the sum of the three numbers that specify your choices for each of the three phases of operation. The three operation phases are index access, comparison (after next/previous/current access only, not lookup), and record retrieval. The **.h/.BAS** files includes symbolic equivalents for the option numbers as Global Constants; since these values are mutually exclusive in their bit positions, you can "or" these symbolic components with a "+" in VB, or "|" in C++ as follows:
..., AccessOption + CompareOption + RetrievalOption,...

For example, you could specify the *Options%* argument as:
(XNEXT | XBEGINS | XNO_DATA) *(see tables below)*

All three phases of operation have default values shown below. ***You do not have to specify anything except non-default choices***. For example, to specify a Next access without any comparison tests, and with record retrieval into your **Options%** variable, just write XNEXT.

If you want the defaults for all three phases, you may use the argument value 0 -- but we recommend using **XLOOKUP** for clarity. The **VsamGet** function will return **VIS_BAD_PARAMETER_VALUE** if you specify invalid combinations.

VsamGet

The following table lists both the numeric and symbol equivalent values for the components of the *Options%* argument.

Phase	**Def ault**	**Value**	**Option Symbol**	**Description**
access	>	1%	XLOOKUP	RIndex$=Select$ (default)
		2%	XNEXT	Get next RIndex$
		4%	XPREVIOUS	Get previous RIndex$
		8%	XCURRENT	Get current RIndex$
compare	>	16%	XANY	OK for all RIndex$ and (default) Select$, no compares
		32%	XEQ	OK if RIndex$=Select$
		64%	XBEGINS	OK if Select$=left part of RIndex$
		128%	XNOT	OK if RIndex$!=Select$
		256%	XLT	OK if RIndex$< Select$
		512%	XGT	OK if RIndex$> Select$
		1024%	XLE	OK if RIndex$<= Select$
		2048%	XGE	OK if RIndex$>= Select$
Get	>	4096%	XGET_DATA	Get both key and data (default)
		8192%	XNO_DATA	Get only the key

When **VsamGet** is called, it returns the key and data, the record returned is a raw record (***GSTR)*** and is not decoded. Fields may then be extracted from it with **VsamFetchField**. See the example shown in the **VsamFetchField** Function Description.

Since, if any phase of *Option%* is left blank, the default value is used, it is not necessary to ever use XLOOKUP, XANY, or XGET_DATA. However, for application maintenance purposes, explicit arguments are easier to understand than invisible ones.

***Selector$*:** A zero terminated input argument used either as a lookup key into the specified index (in Lookup access mode) or for comparison against the retrieved index key (optionally, in Next, Previous, or Current access modes). If you don't need it for either purpose -- that is, you're doing a Next, Previous, or Current access without any comparison testing -- you must still supply a place holder, such as the null string ("").

VsamGet

****NOTE:*** ***When searching using a compound index, use the 0x02 character as a separator between compounded fields.***

Example:

Strcpy(Selector, "John"); // First Name
Strcat(Selector, "\x02"); // Compound Separator
Strcat(Selector, "A"); // First letter of last name

***RIndexEntry$*: ** Is a pointer to a buffer of at least 255 characters that will receive the zero terminated index key-entry found, if any. (Even if you don't need the key-entry, you must still provide the pointer to a buffer "throwaway" string variable to store it in.)

VsamGet will load a value into the string named in *RIndexEntry$* under the following circumstances:

in lookup mode if VIS_OK;

in Next, Previous, or Current mode if the initial index access resulted in an index key entry under the index pointer, without regard to the results of comparison testing (if any).

***RPrimaryKey$*:** Is a pointer to a buffer of at least 255 characters that will receive the returned corresponding primary key – zero terminated. VsamGet will load a value into this variable under the following circumstances:

in lookup mode if VIS_OK

in Next, Previous, or Current mode if the initial index access resulted in an index key entry under the index pointer, and any specified comparison test was also successful.

RRecord: If the function succeeds and data is requested (not XNO_DATA), VsamGet will store a **GSTR** handle that in itself points to the allocated buffer containing the raw data record.

Note: *Primary Keys are not part of the data record – they are always returned in an independent parameter (RPrimaryKey$).*

VsamGet

» Warning: ***VsamEx*** *can't tell if you've specified insufficient buffer space for return strings, so be careful; an error here could cause unpredictable behavior (like various parts of your body falling off, or in some cases extraneous growth in undesirable places)!*

Function return codes:
VIS_OK – *The function call was successful!*

VIS_NOT_FOUND - ***Either*** *the initial access operation was unsuccessful -- in which case the RIndexEntry$ and RPrimaryKey$ variables will not have been loaded with new values -- or the specific type of access failed because:*

XLOOKUP:	The *Selector$* value wasn't found in this index.
XNEXT:	You tried to read beyond the last entry in this index.
XPREVIOUS:	You tried to read before the first entry in this index.
XCURRENT:	The current pointer position n this index was either at (index) BOF, at (index) EOF, or between entries.

Or you specified a Next/Previous/Current step-access with a comparison test, and the step succeeded but the comparison test failed. In the latter case, the *RIndexEntry$* argument will have been loaded, but the *RPrimaryKey$* and *RRecord* arguments will be unchanged.

VIS_BAD_DATASET_NUMBER – *The dataset identifier is not valid!*
VIS_INVALID_KEY - *The Selector$ used as a lookup key was null; or was longer than 252 chars; or contained a binary 0x01byte.* ***Nothing was done!***

VIS_BAD_PARAMETER_VALUE - *Options% or Index field was invalid!*
VIS_OUT_OF_MEMORY – *The return buffer is too small!*
VIS_DISK_ERROR – *The operating system has a problem with the dataset!*
VIS_DATA_VALIDITY_CHECK - *The dataset is corrupt and needs repair!*
VIS_SEQUENCE_ERROR – *An index is out of sort, the dataset needs repair!*

NOTE: ***The GSTR returned is now the <u>property of the calling program and must be freed at some point by your application</u>. The only exception is if you continue to call VsamGet using the same GSTR pointer - VsamEx will internally free an old GSTR pointer to data before allocating a new GSTR to replace the old one. If you do not want to loose reference to the specific Raw record GSTR, you must move it (the GSTR pointer) to another variable and store ZERO(NULL) at the Variables (RRECORD) position so that VsamGet will allocate a new buffer.***

VsamGet

Coding Illustrations:

(1)

```
  int   rc;
  char key[256], rshkey[256], rkey[256];
  LPGSTR lpgstrDat = NULL;

  strcpy(key, “xyz”);
  rc = VsamGet(DsHandle,”Primary”, XLOOKUP, key, rskey, rkey, &lpgstrDat);
  show_record();
```

(2)

```
 while(rc == VIS_OK)
    {
       rc = VsamGet(DsHandle, ,”Primary”, XNEXT, key, rskey, rkey, &lpgstrDat);
       show_record();
    }
  if (*lpgstrDat)
    {
       FreeGstr(*lpgstrDat);
      *lpgstrDat = 0;
    }
  show_status("Read", rkey, rc);
```

VsamGetWithLock

short VsamGetWithLock (LONG DatasetNumber,
LPSTR Index,
WORD Options,
LPSTR Selector,
LPSTR RIndexEntry,
LPSTR RPrimaryKey,
LPGSTR *RRecord)

Description: Reads a record through an index access, either by key lookup or sequentially, repositioning the selected index pointer as described in the **FUNDAMENTALS -- USING INDEXES** section in the VsamEx User's Guide.

NOTE: This function operates exactly like **VsamGet** with the exception that it attempt to lock the primary key of the record to be read. If the lock of the primary key fails the function will not return the record and the return code will be that returned by an unsuccessful **VsamLock**, usually **VIS_ACCESS_DENIED**.

This is faster than asking for the lock and then reading the record as two separate operations.

VsamGetFieldAttribute

```
short VsamGetFieldAttribute (LONG DatasetNumber,
                 LPSTR lpFld,
                 LPSTR lpAttName,
                 LPSTR lpBuf,
                 LPWORD lpBufSize)
```

Description: Retrieves the value of a specific Field attribute.

Arguments:

DatasetNumber&**:** The reference number returned by **VsamOpen**.

lpFld$**:** The name of the dataset field.

lpAttName$**:** The name of the field attribute.

Predefined Attributes:

Field Name	***"Fnam"***	***String Name of the field***	
Field Number	***"Fnum"***	***n -***	***(Read Only!)***
Field Class	***"Fcls"***	***[D] - Data***	***(Read Only!)***
Field Type	***"Ftyp"***	***[%,&,!,#,@,$,C]***	***(Read Only!)***
Field Is an Index	***"Find"***	***[T, F, P, D] – True, False, Partial or Disable***	
	See: VsamSetFieldAttribute for detailed description		
Field width	***"Fwid"***	***field width in characters***	
Field Just	***"Fjst"***	***[L,C,R] – Left, Center, Right***	
Field Is Deleted	***"Fdel"***	***[T, F] - True, False***	

lpBuf$**:** Pointer to the buffer to return the attribute value in.

lpBufSize%**:** Length of the buffer. On Output, it is the size required.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*

VsamInfo

short VsamInfo (LONG DatasetNumber, LPVSTATS VstatsStructure)

Description: Retrieves dataset parameters and statistics (see below).

Arguments:

***DatasetNumber&*:** The reference number returned by **VsamOpen**.

***VstatsStruct*:** The name of a variable into which VsamInfo will place the retrieved information. The **Type** definition for VSTATS is included in the Vsam.h file. The structure is as follows:

```
typedef struct tagVSTATS
  {
    LONG   nrecords;          // total number of records in the dataset including xref’s
                              // and dictionary records
    LONG   grp_size;          // Group size for this dataset
    LONG   gps_used;          // Groups used in this dataset
    LONG   gps_unused;        // Groups in this dataset that are unused
                              // at the end of the dataset
    WORD   max_key_len;       // set to 252 in VsamEx datasets
    WORD   num_fields;        // number of fields defined in this database
    LONG   nPrimRecords;      // total number of Primary records in the dataset
    LONG   reserved[3];
  } VSTATS;

typedef VSTATS FAR * LPVSTATS;
```

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*

VsamKill

short VsamKill (LPSTR DatasetName)

Description: This function will delete all files associated with a Dataset. The Dataset must not be open in any other application or thread or the function will fail. If the function is successful, the ***files will be removed without moving them to the recycle bin***. Only disk recovery software (available only from third party vendors) will be able to recover them and then only if done in a timely manner. Timely manner in this case means before the operating systems has a chance to re-use any blocks in the old files. ***We strongly recommend that all the files comprising the dataset (.vod & corresponding .vom) be backed up to external media and archived before using this function to delete the dataset.***

Arguments:

***DatasetName*&:** The Dataset Name used in **VsamCreate**.

Function Return Codes:

VIS_OK **-** *You have successfully locked this string on this dataset*
VIS_ACCESS_DENIED **-** *Another process locked this string on this dataset*

VsamLock

short VsamLock (LONG DatasetNumber, LPSTR LockString)

Description: Sets a semaphore lock associated with a dataset. Other processes trying to lock that same semaphore will receive VIS_ACCESS_DENIED.

Arguments:

***DatasetNumber&*:** The reference number returned by **VsamOpen**.

***LockString$*:** Any variable length zero terminated string <= 255 characters.

Function Return Codes:

VIS_OK **-** *You have successfully locked this string on this dataset*
VIS_ACCESS_DENIED **-** *Another process locked this string on this dataset*
VIS_ALREADY_EXISTS **-** *Your process locked this string on this dataset*
VIS_NO_ROOM **-** *The process already has 32 strings locked on this dataset*
VIS_BAD_DATASET_NUMBER – *Invalid dataset number*
VIS_BAD_PARAMETER_VALUE – *Key length >255 characters.*

Examples:
Vsamlock(Dsn1&, "Jones");
Vsamlock(Dsn1&, "Accounting Division");
Vsamlock(Dsn1&, PrimaryKeyName$); // *used for record locking*

VsamMakeMap

short VsamMakeMap(LPSTR DatasetName, LPDWORD lpGcount, LPSTR lpEncryptKey)

Description: This function will recreate the map for an existing dataset. This may be necessary if a network or disk failure corrupts the map file (".vom"). This function will insure that all of the Dataset Groups are sequenced in sorted order.

VsamMakeMap *is always run as part of the* ***VsamRebuild*** *process. It is not necessary to run this function as a separate operation. if you suspect a dataset has been corrupted. Only* ***VsamVal*** *can determine if a dataset is corrupted or not!*

Arguments:

DatasetName$:	is the Name of the dataset.
lpGcount&:	is a pointer to a DWORD that will be updated with the number of Groups processed. This value is shared and may be viewed by a different thread or part of your program running off of a timer or a message pump.
lpEncryptKey$:	is an Encryption key that will be used to access the existing dataset.

Function return codes - Special Errors:

0	VsamMakeMap Completed OK
1 & 2	The Dataset has been truncated improperly and cannot be read!
3	The Temporary Map file cannot be written.
92	The Real Map file cannot be written.
93	The TDMILL structured write failed to update the new map. *(This is a location in the map where critical operating information is recorded!)*
94, 95, 96, 99	Cannot open either the Temporary or Real map file.
97	The Specified Dataset's data file cannot be opened.
98	Cannot allocate system memory for the operation.

VsamMovePtr

short VsamMovePtr (LONG DatasetNumber, LPSTR IndexField, LONG RelCount, WORD RelOption)

Description: Moves the index pointer of either a primary or secondary index, relative to it's current position. This function is much faster than executing VsamGet/NO_DATA.

Arguments:
DatasetNumber&: is the same as returned from **VsamOpen**.

***IndexField$*:**	is the index (Name or Number) on which to perform the move ptr.
***RelCount&*:**	is a plus(+) or minus(-) (long) number of records to skip.
***RelOption%*:**	is 1 if you want to DISABLE the internal RQM calls, otherwise it should be 0.

Note: Only one **VsamMovePtr** can be in operation at a time and will not respond again (except with VIS_BUSY), until it has completed or been terminated by **VsamCancel**. Since, in larger data sets, this operation has the potential of taking a very long time, **VsamMovePtr** has been made an asynchronous operation. This allows other parts of your application to run just fine. Although, if any other attempt is made to access a **VsamEx** function which will interfere with the relative move, that function will return VIS_BUSY until the **VsamMovePtr** operation is complete or terminated. This includes any other calls to **VsamMovePtr**.

Function Return Codes:

VIS_OK **–** *Function Call was successful and completed*
VIS_BAD_HANDLE – *Invalid dataset number*
VIS_BAD_PARAMETER_VALUE – *Field specified is not an index field*
VIS_BUSY – *Another MovePtr is under way, try again later*
VIS_NOT_FOUND - *end of list before count exhausted*
VIS_INTERRUPTED - *will return only if* ***VsamCancel*** *was called*

VsamOpen

short VsamOpen (LPSTR DatasetName, WORD AccessMode, LONG *rDatasetNumber, LPSTR LicenseKey, LPSTR Encrypt)

Description: Opens a named **VsamEx** dataset and returns its reference number (handle) for later access. You may have several datasets open concurrently, each with a different reference number. **VsamOpen** initializes all index pointers to **BOF**.

This function is used to open all **VsamEx** datasets.

Arguments:

***DatasetName$*:** The name of the dataset, optionally including a full path name. If an extension is specified, the first two characters will be used to construct the extensions, ending in "D", "M", and "L" respectively for the different parts of the VsamEx dataset.

***AccessMode%*:** This argument specifies the access privileges you want in a multiuser or multi-process environment; see the following section for important information about enforcement of these privileges.

***AccessMode%* = 0:** In this read-only mode, **VsamOpen** will fail if any other process has the dataset open for exclusive read/write access mode 1, and will return VIS_ACCESS_DENIED. You may not make any changes to a dataset open in read-only mode; **VsamPut**, **VsamDelete**, **VsamFlush**, and **VsamWriteNote** will all return VIS_ACCESS_DENIED. The **VSAM.H** file defines the global constant **READ_ONLY** = 0 to allow symbolic specification.

***AccessMode%* = 1:** In this exclusive read/write mode, **VsamOpen** will fail, and return VIS_ACCESS_DENIED, if any process has the dataset open in any access mode. The **VSAM.H** file defines the global constant **READ_WRITE** = 1 to allow symbolic specification.

VsamOpen

AccessMode% = 2: In this read only shared mode, **VsamOpen** cannot open a dataset and will return VIS_ACCESS_DENIED, if any process has the dataset open in mode 1 (READ_WRITE). The **VSAM.H** file defines the global constant **READ_ONLY_SHARED** = 2 to allow symbolic specification.

AccessMode% = 3: In this read/write shared mode, **VsamOpen** will return VIS_ACCESS_DENIED if any process has the dataset in mode 1 or mode 0. The **VSAM.H** file defines the global constant **READ_WRITE_SHARED** = 3 to allow symbolic specification.

***rDdatasetNumber&*:** The variable in which you want to receive the reference number (handle) to the dataset, returned if the call was successful (VIS_OK); if the call was unsuccessful, this value is undefined. You must save this number for later use in all references to this open **VsamEx** dataset.

» *Note: **VsamEx** dataset reference numbers are managed independently of the file numbers used for ordinary file access.*

***LicenseKey$*:** A string license key that is issued by Software Source. If this license key does not checkout, the dataset will not open.

***LicenseKey$* = (bad *Key*):** Return code = VIS_INVALID_PASSWORD

***LicenseKey$* = (valid *Key*):** Return code = VIS_OK

Encrypt$*:* This key must match the key specified in the VsamCreate function that created the dataset. Use NULL string for datasets without Encryption.
If you loose your key, the probability is very high that your data will not be recoverable.

Function return codes:
VIS_OK - *Function complete successfully*
VIS_ACCESS_DENIED - *There's a multi-user access conflict*
VIS_BAD_PARAMETER_VALUE - *Invalid CacheSize% or AccessMode%*
VIS_BAD_FILE - *One of the dataset files isn't a recognizable **VsamEx** file*
VIS_OLD_FILE – *The dataset is a VB/ISAM file that was opened in Read Only mode*
VIS_DOS_ERROR –A *failure when trying to open one of the dataset files*
VIS_DISK_ERROR – *Actual disk failure or bad values retrieved from file*
VIS_BUSY - *Loop and try again on this dataset if the net is heavily loaded*
VIS_BAD_PASSWORD – *A valid Encryption key is required*

VsamPut

```
short VsamPut (  LONG     DatasetNumber,
                 LPSTR    PrimaryKey,
                 LPGSTR   Record,
                 WORD     UpdateMode)
```

Description: Adds or replaces a master database record by primary key.

Arguments:

DatasetNumber&**:** The reference number returned by **VsamOpen**.

PrimaryKey$**:** The primary key that you supply.

Record**:** The address of the GSTR pointer returned from a previous VsamGet**.** **VsamPut** will write out the buffer pointed to by this variable as the data portion of the record,. It will update all secondary indexes to correspond to the components of its index fields.

UpdateMode%**:** A flag you can use to protect your data as follows:

***UpdateMode%* = 0:** Allows **VsamPut** to either add or replace a record, depending on whether the primary key you supply is new or already exists in the dataset. A new key will create ("add") a new record; by contrast, a non-new key will result in your data replacing the data in the existing record. The **VSAM.H** file defines the global constant **ADD_OR_REPLACE** = 0 so you can specify this symbolically.

***UpdateMode%* = 1:** Add only (disallow replace); that is, the argument key that you supply must be new. If the primary key is already in the dataset, **VsamPut** won't replace that record, but will instead return the **VIS_UPDATE_VIOLATION** code. The **VSAM.H** file defines the global constant **ADD_ONLY** = 1 so you can specify this symbolically.

***UpdateMode%* = 2:** Replace only (disallow add); that is, the argument key you supply must already be in the dataset. If it isn't, **VsamPut** won't add the record, but will instead return with a **VIS_UPDATE_VIOLATION** code. The **VSAM.h** file defines the global constant **REPLACE_ONLY** = 2 so you can specify this symbolically.

When **VsamPut** is called, it always stores data in the “Native” mode, i.e. as internally structured raw record data. **VsamPut** writes the key and data. A single field could be replaced by using **VsamStoreField** within the **Record** data before calling **VsamPut**. See the example shown in the “**VsamFetchField**” Function Description.

No other Values for *UpdateMode%* are valid.

Function return codes:

VIS_OK – *Function Complete OK*
VIS_UPDATE_VIOLATION - *See UpdateMode%, above*
VIS_ACCESS_DENIED - *Dataset is not open in READ_WRITE access mode*
VIS_INVALID_KEY - *The primary key you supplied was null; or was longer than 255; or contained a binary 0 [NULL] or binary 1 [Ctrl-A] byte. Nothing was done*

VIS_INVALID_SECONDARY_KEY - *One of the secondary-index-key fields in the record you want to write was longer 252, or contained a binary 0 [NULL] or binary 1 [Ctrl-A] byte. Nothing was done*

VIS_BAD_PARAMETER_VALUE - *The value you supplied for UpdateMode% was not valid. Nothing was done*

VIS_BAD_HANDLE
VIS_DISK_FULL - *There were not enough Dataset groups left to allow the next increment of space to expand the data file*

VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

Coding Illustration:

```
    sprintf(key, "%010ld", i);
    rc = VsamPut(DatasetNo, key, gstrDat, ADD_OR_REPLACE);
```

VsamPutWithUnlock

short VsamPutWithUnlock (LONG DatasetNumber,
LPSTR PrimaryKey,
LPGSTR Record,
WORD UpdateMode)

Description: Adds or replaces a master database record by primary key.

NOTE: This function will, after writing the record, unlock the primary key of the record being written. This function will not return an error if the record being written had not been locked. This is a companion function to **VsamGetWithLock**.

VsamOptimisticUpdate

```
short VsamOptimisticUpdate (LONG     DatasetNumber,
                            LPSTR    PrimaryKey,
                            LPGSTR Record,
                            LPSTR   ChkKey,
                            Void      *FieldData)
```

Description: Adds or replaces a master database record by primary key only if the contents of the field named by the **ChkKey** parameter in the existing record in the database is equal to the value in the **FieldData** parameter. If an existing record with the same **PrimaryKey** does not exist the record will be written.

Arguments:

***DatasetNumber&*:**	The reference number returned by **VsamOpen**.
***PrimaryKey$*:**	The primary key that you supply.
***Record*:**	The address of the GSTR pointer returned from a previous VsamGet**.** **VsamPut** will write out the buffer pointed to by this variable as the data portion of the record,. It will update all secondary indexes to correspond to the components of its index fields.
***ChkKey$*:**	The key field to compare before writing the record.
***FieldData*:**	The data value to compare against the field named by ChkKey$.

NOTE: This functions purpose is to make updates to a database over a network more efficient by providing and alternative to the need to lock and re-read records before updating them in most cases.

VsamReadDict

```
short VsamReadDict (LONG DatasetNumber,
                    LPSTR     DictKey,
                    LPSTR     DictData,
                    LPWORD    lDictData,
                    WORD      Options)
```

Description: This function will read the Dictionary data specified by DictKey$, into the buffer DictData$. The data is unstructured data and may contain any binary values.

Arguments:
DatasetNumber& is the same value returned from **VsamOpen**.

DictKey$	is a pointer to the key name of the Dictionary element. This must be large enough to hold the longest key in the dataset (255 characters max). It will contain the Key of the next record read with XNEXT or XPREVIOUS. If DictKey$ is "" (NULL) It will set to BOF in the Dictionary.
DictData$	is a pointer to a buffer large enough to hold the data portion of the record. The maximum Record size is approximately 65,500 bytes.
lDictData%	is the buffer length on input, and amount of data actually returned.
Option%	Specifics possible read actions:

XLOOKUP	Reads an exact record by key in the dictionary.
XNEXT	Reads the Next record in the dictionary.
XPREVIOUS	Reads the Previous record in the dictionary.

Function return codes:
VIS_OK – *Function call was successful*
VIS_NOT_FOUND - *The record was not found in the dictionary*
VIS_BAD_DATASET_NUMBER – *The dataset identifier is not valid*
VIS_BAD_PARAMETER_VALUE - *Options% was invalid*
VIS_NO_ROOM – B*uffer is too small, **lDictData%** = size needed including terminating 0.*
VIS_DISK_ERROR – *The operating system has a problem with the dataset*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_SEQUENCE_ERROR – *An index item is out of sequence; your dataset is corrupted*

VsamRebuild

short VsamRebuild (LPSTR DatasetName, LPSTR Ebuf, SHORT Options, LPDWORD Phase, LPDWORD RCount, LPSTR LicenseKey, LPSTR EncryptKey)

```
short VsamRebuild (LPSTR DatasetName,
                   LPSTR Ebuf,
                   SHORT Options,
                   LPDWORD Phase,
                   LPDWORD RCount,
                   LPSTR LicenseKey,
                   LPSTR EncryptKey)
```

Description: This function will read an existing dataset, extract the Data Definition, all Primary data records and build a new dataset with indexes. **See: VsamMakeMap.**

Arguments:

DatasetName&	The dataset name i.e**. "sourcename.xxx;destname.vod"**. If a single name is specified, without the ";" it represents both source and destination.
Ebuf$	A pointer to a log buffer. It should be at least 4096 bytes in length.
Options%	-1 = Cancel Rebuild process; 0 = rebuild without backup; 1 = rebuild and backup old data, i.e. ".ovd" & ".ovm" files are created.
Phase&	A pointer to a DWORD that is updated with a progress phase number.
Rcount&	A pointer to a DWORD that receives updates representing records processed. *On input, if it is a multiple of 1024, it will change Group Size.*
LicenseKey$	License key used to Open the existing dataset.
EncryptKey$	Encryption key used to access the existing encrypted data.

Function return codes:
VIS_OK – *Function call was successful*
VIS_BUSY – *The dataset rebuild function is already running – cancel it first!*
VIS_INTERRUPTED - The user canceled the search or there were too many errors.
VIS_NOT_FOUND – *A logical group could not be found in the dataset!*
VIS_DISK_ERROR – *The operating system has a problem with the dataset*
VIS_SEQUENCE_ERROR – *An index item is out of sequence; your dataset is corrupted*

VsamReturnCode

LPSTR VsamReturnCode (WORD code)

Description: This function will convert a VsamEx return code to printable text.

Arguments:

Code%:	is the VsamEx function return code.

Function return:

Returns a pointer to the ASCII text representing the translation of the Function return code.

VsamSearch

Short VsamSearch (

```
                    LPSTR DatasetName,
                    LPSTR OutputFileName,
                    LPSTR SearchCriteria,
                    SHORT Opts,
                    LPLONG Hits,
                    LPLONG Count,
                    LPSTR EncryptKey)
```

Description: Performs a high speed search of the primary record section; primary key and all fields, including fields not indexed. The search will rapidly locate all Primary Records (Hits), whose primary key and/or data fields contain the criteria specified. Terms separated by the AND operator (first character of the Search Criteria) will be combined using Boolean 'and'. Terms separated by the ‘OR’ operator (second character of the Search Criteria) will be combined using Boolean 'or'. Parsing is from left to right and the first 'or' group which satisfies the test will designate that record a 'Hit'. Leading and trailing spaces are removed. Embedded spaces are removed, all but one, from each search term. A term may be negated using the ‘NOT’ symbol ‘~’ before the search logic (see below).

Arguments:

DatasetName$: The dataset file name. This function will operate asynchronously and will always try to open the dataset in READ/SHARED mode.

OutputFile$: The search results will normally be returned in this file, delimited by CrLf. In the case of a DynaSet (see Opts% below) it is the new Dataset Name.

SearchCriteria$: The ***first character*** of the search criteria is the character to be used as the ‘AND’ operator. The ***second character*** of the search list is the ‘OR’ operator (see below).

The remainder of the search list is a list of terms used to filter the records. They are delimited by the selected 'AND' and 'OR' operators. A search term is any arbitrary text (including spaces), except the 'AND' or 'OR' operators, as specified by the first two characters. A search term group is one or more search terms separated only by the 'AND' operator. To be a 'Hit' candidate, all of the search terms in a group must be found, as designated by the term, in the combined primary record/key. If there are multiple search term groups, they are separated by the 'OR' operator.

If there is more than one search term group separated by the "OR" operator, the comparisons in an individual record continues **only** until the first "OR" term **match** occurs. Parentheses are treated as ordinary string characters, and do not affect evaluation order. Optionally, a numeric field identifier ("%3" etc.) may be used or the Field Name, followed optionally by the NOT sign "~", and a search logic designator to restrict testing to a specific field as described below:

***Opts%*:** 0= return only keys as hits.
1=Search and return keys and formatted data as hits.
2 = Create a VsamEx DynaSet – *This is a duplicate Dataset with the same Field Definitions except that the Dataset Indexes have not been created. Each record is a record from the original dataset that was a "Hit" with the given Search Criteria.*
-1=Cancel search.

NOTE: Once the function has been called, it will not return until complete or canceled. This must be done by calling the function again (-1 in this parm) from a different thread or message pump.

***Hits&*:** The number of primary records found matching the search criteria. This parameter is passed by reference and its contents will be periodically updated. This value may be used as a progress indicator.

***Count&*:** The number of groups processed.
Hits and Count are periodically populated with snap-shot values this value may be used along with Hits to generate a progress indicator.

Search Logic	*Description*
=	‘Equal’ Search; requires the term to be an exact match. i.e. **5=Jones** would mean field 5 must be ***exactly equal*** to ‘Jones’
>	‘Greater Than’ Search; requires the data to sort alphanumerically greater than term. i.e. **Field5>Jones** would mean field 5 must be alphanumerically ***greater than*** ‘Jones’
<	‘Less Than’ Search; requires the data to sort alphanumerically less than term. i.e. **5<Jones** would mean field 5 must be alphanumerically ***less than*** ‘Jones’
:	‘Begins with’ Search; requires the field to begin with the term. i.e. **Primary:Jone** or **0:Jone** would mean field 0 only had to begin with ‘Jone’; ‘Jones’ would be a Hit., only in field 0 (the Primary Key).
[	‘Contains’ Search; requires the field only to contain the term. i.e. **12[Jon** would mean ‘Jones’ is a Hit, ‘Jone’ is a Hit, and likewise ‘ Wilma Jonell Smith’ is a Hit, but only if they are found in field 12.
~	The not sign may be used to negate the meaning of a particular match. For example, “**~:**” selects a record as a hit if the data does not begin exactly with the term. i.e. “**Name~:John”** indicates that records beginning with “John” in the Name field are Not Hits.

NOTES:

No logic *(Default) - All terms that are absent of search logic are applied such that if the term is contained anywhere in the record or key, that record is a Hit.*

*A **missing field** is considered NOT to be a MATCH in a term that calls for it. Likewise, if the term contains the not (~) a **missing field** is considered a MATCH.*

EXAMPLE 1: ‘&,Jones ,Smith & Wesson’

Records containing either the term 'Jones' , or both terms 'Smith' and 'Wesson' in either the primary key or data part of the record would be a Hit.

EXAMPLE 2: ‘&,Jones &Smith&Wesson'

Records containing the term 'Jones ' (space included), containing 'Smith' and additionally 'Wesson' in either the primary key or any part if the data portion of the record would be a Hit.

EXAMPLE 3: ‘+^5~:Jone^Smith+Wesson’

Records not beginning with ‘Jone’ in field 5, or both the terms ‘Smith’ and ‘Wesson’ found anywhere in the record would make it a Hit.

NOTE: Search operations are canceled by setting Opts% = -1.

Function return codes:

VIS_OK
VIS_ACCESS_DENIED (There's a multi-user access conflict.)
VIS_BAD_PARAMETER_VALUE
VIS_BAD_FILE (One of the dataset files isn't in VB/ISAM format.)
VIS_OUT_OF_MEMORY
VIS_DOS_ERROR (File open failure, probably because it couldn't find it.)
VIS_DISK_ERROR
VIS_OUT_OF_FILE_HANDLES
VIS_INTERRUPTED (Either the user canceled the search or the results overflowed.)

VsamSetDictBof

short VsamSetDictBof (LONG DatasetNumber)

Description: Sets the file pointer in the Dictionary to BOF: Before the First entry, if any.

Arguments:

DatasetNumber&: The reference number returned by VsamOpen.

Remember, VsamEx maintains a separate, independent table entry for each index. Pointers into the dataset may move independent of one another and do not interfere with each other.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamSetDictEof

short VsamSetDictEof (LONG DatasetNumber)

Description: Sets the pointer in the Dictionary to EOF: After the Last entry, if any.

Arguments:

DatasetNumber&: The reference number returned by VsamOpen.

Remember, VsamEx maintains a separate, independent table entry for each index. Pointers into the dataset may move independent of one another and do not interfere with each other.

Function return codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamSetFieldAttribute

short FAR PASCAL VsamSetFieldAttribute(

```
LONG     DatasetNumber,
LPSTR    lpField,
LPSTR    lpAttributeName,
LPSTR    lpDataValue)
```

Description: This function will Set the value of a field attribute. This function will replace the value of the specified field attribute if it previously existed. If the attribute value is initially nonexistent, it will be created as a User defined attribute. ***If the lpData points to a NULL value, The corresponding field attribute will be removed if it is not a Predefined attribute***.

Argument:

DatasetNumber&: is the same value returned from **VsamOpen**.

lpField$: is a pointer to the FieldName/(optionally the Field Number as "%n").

lpAttributeName&: a Pointer to the Attribute name. User defined or Predefined.

Predefined Attributes that may be modified:

Field Name	"Fnam"	String Name of the field
Field Is an Index	"Find"	[T, F, P, D] – True, False, Partial or Disable
Field width	"Fwid"	field width in characters.
Field Just	"Fjst"	[L,C,R] – Left, Center, Right.
Field Is Deleted	"Fdel"	[T, F] - True, False.

Certain Predefined attribute values are restricted as above. If you wish to change the "Find" (Indexed) attribute's setting for an existing string field, then ONLY certain changes are allowed depending on the current state of the index in the file. They are as follows:

Current State	to	New State
T	->	**D**
F	->	**P**
P	->	**D**
D	->	**P**

Where:

T = True - all records in the file are currently indexed for this field (fully indexed)
F = False - no records in the file are indexed for this field
P = Partial - some of the records are indexed for this field. Only records added or changed after setting the field's Find attribute to "P" will be indexed. Run **VsamRebuild** to index all remaining records.
D = Disabled - this field's previously active index (full or partial) has been disabled for all records.

Therefore, you cannot perform T -> F or F -> T direct transformations on existing string fields. The intermediate "P" and "D" codes must be used until a VsamRebuild can be run on the file.

When a VsamRebuild is run on the file, the index field will be "cleaned up" for all of the file's records, and the following index transformations will have been completed internally:

Old State	to	New State
P	->	**T**
D	->	**F**

Where:

P = Was Partially indexed - only some records were indexed on this field.
T = All records are now fully indexed for this field.
D = Was Disabled index - (index existed but was not active)
F = All Disabled indexes have been removed from the file

lpDataValue$ is a pointer to a buffer containing the zero terminated Value. For User defined attributes (not Predefined), any string value may be set for this attribute. ***Keep in mind that all data definitions are stored in the dataset as a special record whose maximum size is 65k bytes.***

Function return codes:
VIS_OK – *Function call was successful*
VIS_BAD_DATASET_NUMBER – *The dataset identifier is not valid*
VIS_DISK_ERROR – *The operating system has a problem with the dataset*
VIS_ACCESS_DENIED – *The field is protected – Read Only!*

VsamSetFieldToNull

short VsamSetFieldToNull (LONG DatasetNumber, GSTR *Record, LPSTR FieldName, WORD element)

Description: This function will remove the field from this record. Any subsequent call to retrieve data from this field in this record will result in the error VIS_NOT_FOUND.

DatasetNumber&	same as returned from **VsamOpen**.
Record:	A GSTR pointer to the record returned by **VsamGet**.
FieldName$:	is the name of the field to be removed. Optionally, you may use "%n" where n is the string value of the field number.
Element%:	Unused. Set to 0.

Function Return Codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamStoreField

short VsamStoreField (LONG DatasetNumber,
GSTR *Record,
LPSTR FieldName,
WORD element,
LPSTR FieldType,
VOID *FldData,
WORD lFldData)

Description: This function will store data from FldData into the *raw* record pointed to by Record (from a previous **VsamGet** call). (see **VsamFetchField**.)

DatasetNumber& same as returned from **VsamOpen**.

Record:	A GSTR pointer to the record returned by **VsamGet**. You can create a New record by setting this value to NULL (0). If a new record is created, the new GSTR will be stored at the address pointed to by Record.
FieldName$:	is the name of the field into which the FldData data will be placed. Optionally, you may use "%n" where n is the string value of the field number.
FieldType$:	is a string value which defines the field type (E.G. "$", or "%").
FldData:	is the variable containing the data to be stored into Record. Its inherent type must be what is defined in the FieldType$ parameter. Furthermore, it must correspond to the field specified in Record defined above.

Function Return Codes:

VIS_OK - *Function call completed successfully*
VIS_BAD_DATASET_NUMBER – *Invalid open dataset handle*
VIS_BAD_HANDLE – *Potential bad dataset*
VIS_BAD_PARAMETER_VALUE – *Field is not an index*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_BUSY – *The map is temporarily locked, please retry the operation*

VsamUnlock

short VsamUnlock (LONG DatasetNumber, LPSTR LockString)

Description: Removes the specified semaphore lock, if it exists.

Arguments:

DatasetNumber&**:** The reference number returned by **VsamOpen**.

LockString$**:** Any variable length zero terminated string <= 255 characters.

Function Return Codes:

VIS_OK **-** *You have successfully unlocked this string on this dataset*
VIS_NOT_FOUND **-** *You tried to unlock a LockString you do not have locked*
VIS_BAD_HANDLE – *Invalid dataset handle*
VIS_BAD_PARAMETER_VALUE – *LockString$ exceeds 255 bytes*
VIS_OUT_OF_MEMORY
VIS_DOS_ERROR

Examples:
VsamUnlock(Dsn1&, "Jones");
VsamUnlock(Dsn1&, "Accounting Division");
VsamUnlock(Dsn1&, PrimaryKeyName$); // *used for record locking*

VsamVal

```
short VsamVal (LPSTR        Filespec,
               LPSTR        Ebuf,
               SHORT        Foptions,
               LPDWORD      lpFCount,
               LPSTR        EncryptKey)
```

Description: The low level dataset is checked for errors by reading every record. This function will report any errors found that would prevent data to be read from the dataset. There are two modes of operation, Physical and Logical. Physical Validate reads physical records a Group at a time and checks for record key sequencing only within the Group. This is the fastest validation. Logical validate cycles through the dataset in key sequence and will check that all records are sequenced properly over the entire dataset. This can only be done if the validate routine can gain exclusive shared read access to the dataset.

Since VsamEx[treme] only checks data Groups for errors when they are accessed, it is possible that a Group has undergone physical damage through a hardware or network failure. This error will not be detected until some time later - the next time that group is accessed! This function should be run prior to creating archive copies of the dataset to insure that it is error free.

While VsamEx[treme] will operate and continue to fetch records from undamaged groups, it will report a problem with a group when it cannot retrieve a record because damage has been detected. If a group is damaged, it may be repaired by running VsamRebuild. This will not recover the lost data. It will only salvage what it can, if anything, from damaged group(s) and restore the dataset to a functional error free condition.

VsamVal

Arguments:

***FileSpec*$:**	The Dataset name.
***Ebuf*$:**	Pointer to a Log buffer – reserve at least 4096 bytes.
***Foptions%*:**	-1 = cancel; 0 = logical validate – *runs in exclusive "Read Only" mode!* 1 = Physical validate – *runs in" Read Only Shared" mode!*
***lpFcount&*:**	Pointer to a DWORD that will be updated with the number of groups.
***EncryptKey$*:**	An Encryption key that allows access to this dataset if it is encrypted.

Function Return Codes:
VIS_OK **–** *Dataset Validated properly!* <u>*Example Ebuf$ Contents:*</u>

```
Primary Records Expected and Found = 1507163
                   Groups  Expected = 65529
                   Groups  Found    = 65529
```

VIS_NOT_FOUND	– *Dataset was not found*
VIS_DISK_ERROR	– *Cannot read from the dataset*
VIS_INTERUPTED	– *Validation was interrupted by the user*
VIS_DOS_ERROR	– *Disk Error in one of the dataset files*

VsamWriteDict

short VsamWriteDict (LONG DatasetNumber,
LPSTR DictKey,
LPSTR DictData)

Description: This function will write/replace the Dictionary data element pecified by the DictKey$ string, with data from the DictData$ string. The data is considered to be a 0 terminated string. The Maximum data string size is approximately 65,530 bytes. There is no limit to the number of dictionary elements that may be written except those which limit VsamEx datasets in general. The Dictionary type records may be used to fill up the entire dataset. Dictionary records occupy the same amount of space in a data set as would a primary record with a single string as an element.

DatasetNumber& is the same value returned from **VsamOpen**.

DictKey$	is a string pointer to the key for the Dictionary element being written. If the Key specified already exists, the data portion of the record will be replaced by the data contained in DictData$, i.e. (overwritten).
DictData$	is a pointer to the Data buffer of the Dictionary element to be written.

Function return codes:

VIS_OK – *Function call was successful*
VIS_NOT_FOUND - *The record was not found in the dictionary*
VIS_BAD_DATASET_NUMBER – *The dataset identifier is not valid*
VIS_BAD_PARAMETER_VALUE - *Options% or Index field was invalid*
VIS_OUT_OF_MEMORY – *The return buffer is too small*
VIS_DISK_ERROR – *The operating system has a problem with the dataset*
VIS_DATA_VALIDITY_CHECK - *Your dataset may be corrupted*
VIS_SEQUENCE_ERROR – *An index item is out of sequence; your dataset is corrupted*

BATCH API

VsamBatchCancel

short VsamBatchCancel (LONG DatasetNumber)

Description: This function will discard all pending, non committed batch transactions in an open batch and close the batch.

Function return codes:

VIS_OK – *Function call was successful*

VsamBatchCreate

short VsamBatchCreate (LONG DatasetNumber, LPSTR BatchName, DWORD Options)

Description: This function will create and open a Batch receptacle to hold selected transactions. These transactions may be committed or canceled at any time prior to a close. The default option is AUTO_COMMIT and the batch will commit any transactions in the buffer once it becomes full. Only the final batch will need to be committed.

When a batch has been created, all calls to:

VsamPut
VsamPutWithUnlock
VsamDelete
VsamOptimistcUpdate
VsamWriteDict
VsamDeleteDict
VsamWriteList
VsamDeleteList

will be put in a batch to be processed all return codes will be VIS_OK.

DatasetNumber& is the same value returned from **VsamOpen**.

BatchName$ the ASCII name of the batch. No other user may create a batch with the same name, until the batch is committed and closed.

Options$ is one or more of the following:
BATCH_NOT_AUTO_COMMIT – returns "VIS_BATCH_FULL" if the batch cannot accept a requested transaction because the buffer is full.
or
BATCH_QUIT_ON_ERROR – return with a batch error immediately without waiting for a commit.

Function return codes:

VIS_OK – *Function call was successful*
VIS_BATCH_FULL – *There is no room in the buffer to save the requested transaction.*
VIS_BAD_DATASET_NUMBER – *The dataset identifier is not valid*

VsamBatchCommit

short VsamBatchCommit (LONG DatasetNumber)

Description: This function will cause a set of stored batch operations to be executed at the remote, or local site. Any errors encountered will be itemized and may be retrieved using VsamBatchErrors. This includes successful operations that return VIS_OK. In the case of a batch opened with the "BATCH_QUIT_ON_ERROR" option, only one error will be in the buffer (besides the OK's up to that point).

DatasetNumber& is the same value returned from **VsamOpen**.

Function return codes:

VIS_OK – *Function call was successful*

VsamBatchErrors

short VsamBatchErrors (LONG DatasetNumber,
LPSTR ErrorBuf,
WORD LenErrorBuf)

Description: This function returns a list of Batch errors as a string using the following format description:

"%d,%d,%s,%d,%s,%s|" where each error descriptor is separated by "|" and the error fields are separated by comma ",".

Field 1	error sequence number
Field 2	VsamEx Error code
Field 3	Ascii string - translation of error code
Field 4	Batch Command Code
Field 5	Ascii string - API Function Name of the transaction
Field 6	Ascii string - Primary Key of the record

Example:
1,1,VIS_NOT_FOUND,8,VsamDelete,00010|

DatasetNumber& is the same value returned from **VsamOpen**.

ErrorBuf$ is a pointer to your buffer where the error(s) will be recorded.

LenErrorBuf % size of the ErrorBuf.

Function return codes:

VIS_OK – *Function call was successful*

VsamBatchStatus

short VsamBatchErrors (LONG DatasetNumber, LPSTR StatusBuf, WORD LenStatusBuf)

Description: This function returns a summary status in ASCII:
Example:

"Batch Mode Active
Batch Name = Test Batch
Items in batch = 100
Batch Space used = 1234
Offset of first Item = 0
Batch Space Available = 61000
Batch Options = 3"
Or
"Batch Mode Inactive"

DatasetNumber& is the same value returned from **VsamOpen**.

StatusBuf$ is a string pointer to the buffer where status will be written.

LenStatusBuf % size of the StatusBuf.

Function return codes:

VIS_OK – *Function call was successful*

APPENDIX

APPENDIX - A - SUPPORTED FIELD TYPES

Field Type Descriptions:

Type name	Description	Type decl char	Value ranges
Integer	2-byte integer	%	-32,768 to 32,767
Long	4-byte integer	&	-2,147,483,648 to 2,147,483,647
Single	4-byte floating-point number	!	-3.402823E38 to -1.401298E-45 1.401298E-45 to 3.402823E38
Double	8-byte floating-point number	#	-1.79769313486232D308 to -4.94065645841247D-324 (minus) 4.94065645841247D-324 to 1.79769313486232D308 (plus)
Currency[1]	8-byte number with fixed decimal point	@	-922337203685477.5808 to 922337203685477.5807
String[2]	Variable length character string	$	0 to 64K characters
Compound[3]	concatenated string	Cf1.w2: f2.w2:….	1 to 252 characters for a virtual index. Fields (fn) may be any valid String field in the dataset, not necessarily other index fields.

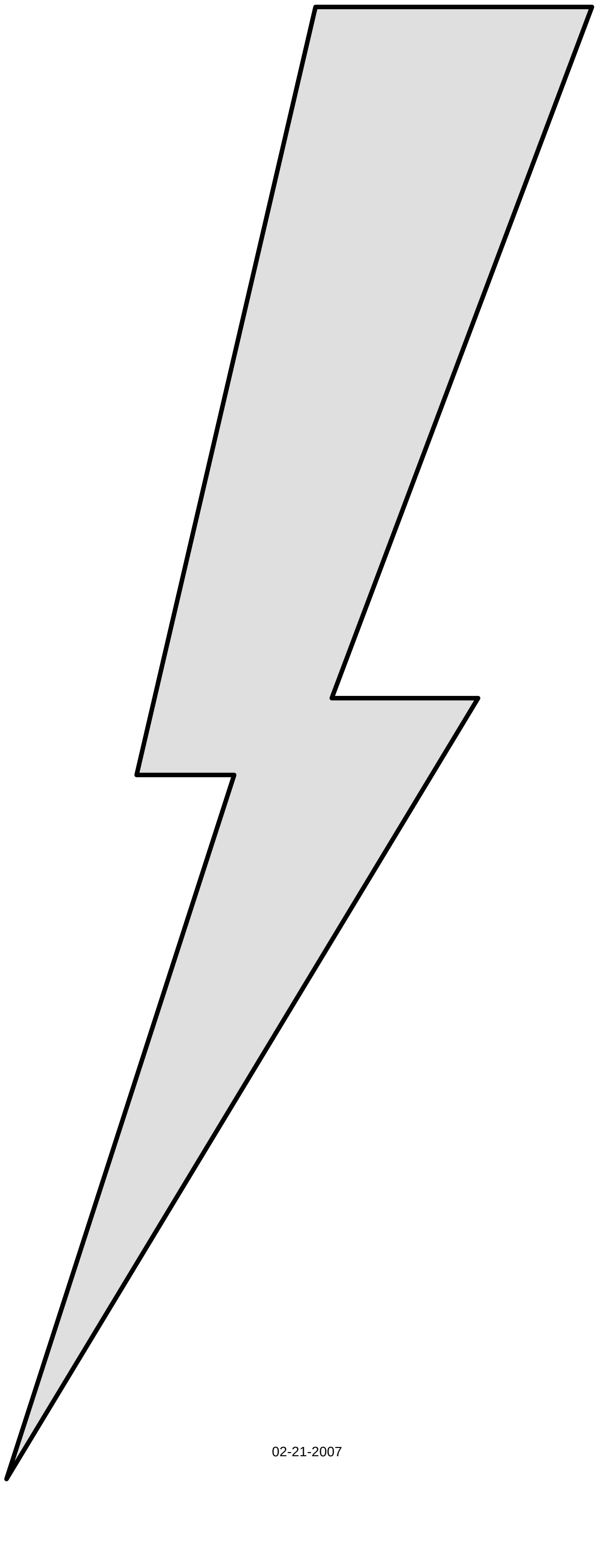
1 Not supported in Linux.
2 Only String fields may be index fields.
3 This type is automatically defined as an index field.

APPENDIX - B - NORMAL ERROR CODES

CODE	- NORMAL error messages	Description	Action
0	VIS_OK	All is OK	Normal return
1	VIS_NOT_FOUND	Object not found	programmers option
2	VIS_UPDATE_VIOLATION	update violation	programmers option
3	VIS_ACCESS_DENIED	Multi-user access	Programmers option
4	VIS_BAD_DATASET_NUMBER	Invalid dataset ref	Re-open Dataset
5	VIS_BAD_FORMAT	Invalid format string	See Function Spec
6	VIS_INVALID_KEY	Illegal char in key	See Function Spec
7	VIS_BAD_PARAMETER_VALUE	Invalid value	See Function Spec
8	VIS_BAD_FILE	File is not VsamEx	See VsamOpen Spec
9	VIS_ALREADY_EXISTS	Dataset Name exists	See VsamCreate Spec
10	VIS_NO_ROOM	No Note space left	See VsamWriteNote
11	VIS_DISK_FULL	No Disk space left	Add Disk space
12	VIS_OUT_OF_MEMORY	No Memory	Reduce Mem Use
13	VIS_DOS_ERROR	DOS failure	See Function Spec
14	VIS_DISK_ERROR	Disk read/write error	Fix Disk, then Rebuild
15	VIS_DATA_VALIDITY_CHECK	Corrupted dataset	Run Rebuild
16	VIS_INVALID_SECONDARY_KEY	Secondary index	See VsamPut Spec
17	VIS_SEQUENCE_ERROR	Records are out of sort	Rebuild dataset
18	VIS_OUT_OF_FILE_HANDLES	No system file handles	Close some files
19	VIS_BUSY	An operation was busy	Retry - notify the user.
20	VIS_INVALID_LICENSE_KEY	Invalid License Key	See VsamOpen
21	VIS_FUNCTION_UNAVAILABLE	Function not implemented	
22	VIS_INTERRUPTED	Asynchronous operation interrupted the Program	See Function Spec
23	VIS_BAD_PASSWORD	Invalid Encryption Key	Use a Valid Key
24	VIS_INVALID_LICENSE	Invalid Vservice Key	Use a Valid Key
25	VIS_OLD_FILE	VB/ISAM File Type	to convert, open in READ/WRITE mode
26	VIS_DATASET_FULL	The dataset is full for the number of Groups specified	Rebuild the dataset to compress the Groups. Increase Num. Groups.
27	VIS_BATCH_FULL	Out of room in batch buffer	Commit Batch
28	VIS_BATCH_ERROR	Error doing batch update	Check Errors
30	VIS_NETWORK_NOT_READY	Network error	
31	VIS_HOST_NOT_AVAILABLE	Network error	
32	VIS_SOCKET_ERROR	Network connection error	
33	VIS_CONNECT_ERROR	Network connection error	
34	VIS_SOCKET_TIMEOUT	Communication timeout	
35	VIS_CONNECTION_REFUSED	Vservice is not running	
36	VIS_MAX_CONNECTIONS_EXCEDED	Connections exceeded 2000	

APPENDIX - C - EXTENDED ERROR CODES

Code	EXTENDED error description	Recommended Action
101-103	Bad Group header	restore dataset
105	A single partial record is larger than 1/2 group.	“
222	Internal failure caused group split to create new group larger than maximum size of a group in this dataset.	"
300	Errors in the 300s can occur if some of the updates of a record have been completed but the system does not allow VsamEx to extend the file to finish updating the record.	“
321,333	Data record sub-parts are out of sequence.	"
345	Could not locate Primary record through an existing Secondary.	"
346	Secondary (Xref) couldn't be deleted - so we didn't delete primary.	"
347	Secondarys were deleted but could not delete the primary.	"
348	Secondary (Xref) update failed after successful primary update.	"
444	No address provided by Caller for return data from get.	Reboot
501	Record structure does is invalid.	Run Rebuild
502&3	Decode Buffer too small - Bad record structure.	"
503	Same as 502.	“
504	A field in the Dataset record is invalid.	“
555	Last partial piece of a record cannot be located - Record was truncated.	"
666	Dataset Record is too large.	“
765	Raw data read returned no data (NULL).	"
885	Dataset activation failed, usually because network rights are not set properly, or the Map file (.vom) is corrupted, or there is a problem creating or accessing the lock file.	Call net support Run Rebuild
886-7	Internal control block inconsistencies detected usually a memory problem cause by a GPF or semiconductor failure.	Reboot
888	Part of the VsamEx dataset became unavailable to the software	"
1001	A disk seek failure occurred in the data file - Fix the hardware.	Run Rebuild or restore dataset
1002	A Data read/write failure occurred in the data file.	“
1003	The Group size found was too big.	Run Rebuild or restore dataset
1004	A Disk seek failure occurred in the finder file - Fix the hardware.	Restore the dataset
1005	A finder read/write failure error occurred .	"
1014	An error was detected during a group split process which generated a bad header.	Run Rebuild or restore the dataset
1234	Internal Consistence error in assembling the data record usually a memory problem cause by a GPF or HW failure.	Reboot
1401-1450	Specific Disk Errors indicating code location where failure occurred.	Call for Support!
2001	Group Validity check - Record size was either too small or too large for the Group.	Run Rebuild or Restore dataset
2002	Group Validity check - A record had no key	"
2003	Group Validity check. Record size field exceeds Group limit	"
2005	Group Validity check - The total of all record size fields does not match the size of the data in the group.	"
7001-4	Unexpected attempt to truncate the Map – data was preserved	Call for Support!



02-21-2007