

PxPlus

C-Library File IO Routines

Introduction	<i>3</i>
PVK_AllocEnv()	<i>5</i>
PVK_DeAllocEnv()	<i>5</i>
PVSetEnvMode()	<i>5</i>
PVGetEnvMode()	<i>7</i>
PVK_OpenExt()	<i>8</i>
PVK_close()	<i>10</i>
PVK_read()	<i>10</i>
PVK_seek()	<i>11</i>
PVK_write()	<i>11</i>
PVK_insert()	<i>12</i>
PVK_update()	<i>12</i>
PVK_remove()	<i>13</i>
PVK_getpos()	<i>13</i>
PVK_setpos()	<i>14</i>
PVK_geterrno()	<i>15</i>
PVK_strerror()	<i>16</i>
PVK_dict()	<i>17</i>
PVK_deffh()	<i>17</i>
PVK_register()	<i>18</i>
PVK_RegisterKey()	<i>18</i>
Example	<i>19</i>



PxPlus is a trademark of PVX Plus Technologies Ltd.

All other products referred to in this document are trademarks or registered trademarks of their respective trademark holders.

©2018 PVX Plus Technologies Ltd. — Printed in Canada

25 Centurian Drive, Suite 204, Markham, Ontario, Canada, L3R 5N8

All rights reserved. Reproduction in whole or in part without permission is prohibited.

The capabilities, system requirements and/or compatibility with third-party products described herein are subject to change without notice. Refer to the PVX Plus website www.pvxplus.com for current information.

File IO Routines

The PxPlus C-Library is an add-on interface that enables PxPlus Keyed, Indexed, and EFF files to be accessed by programs written in 'C' and other programming languages. It consists of the following file IO functions:

PVK_AllocEnv()	<i>Allocate Environment</i>
PVK_DeAllocEnv()	<i>De-allocate Environment</i>
PVSetEnvMode()	<i>Set Environment Variables</i>
PVGetEnvMode()	<i>Get Environment Variables</i>
PVK_open()	<i>File Open (Obsolete)</i>
PVK_openEx()	<i>Extended File Open (Obsolete)</i>
PVK_OpenExt()	<i>Extented File Open</i>
PVK_close()	<i>File Close</i>
PVK_read()	<i>Read a Record from a File</i>
PVK_seek()	<i>Position within Keyed/Indexed File</i>
PVK_write()	<i>Write/Rewrite a Record</i>
PVK_insert()	<i>Write a New Record</i>
PVK_update()	<i>Update an Existing Record</i>
PVK_remove()	<i>Remove a Record</i>
PVK_getpos()	<i>Get Address/Position within File</i>
PVK_setpos()	<i>Set Address/Position of File</i>
PVK_geterrno()	<i>Return Last Error Status</i>
PVK_strerror()	<i>Return Last Error Message</i>
PVK_dict()	<i>Read Dictionary</i>
PVK_deffh()	<i>Pointer to Internal Structure Block</i>
PVK_register()	<i>File Open (Obsolete)</i>
PVK_RegisterKey()	<i>Register Usage of Library</i>

In addition to the above functions, two 'C' header files are provided:

- PVKIO.H - contains file structures and function prototypes
- SYBEX.H - contains computer word size definitions and macros.

Environments Provided

These functions have been pre-compiled for the 32-bit and 64-bit Windows environment.

Registration

Use and distribution of this package is prohibited without first obtaining an authorized registration key. A warning message to this effect is presented whenever a file is opened unless the application first invokes the **PVK_RegisterKey()** function with a valid registration string and registration number.

Distribution of the PXPIO routines is restricted to only those companies that apply for and receive a registration string and number directly from PVX Plus Technologies Ltd.

PVK_AllocEnv()

Allocate Environment

Format **HPVKENV** PVK_AllocEnv();

Where:

HPVKENV Handle to the environment structure, 4-byte value (32-bit version of the library) or 8-byte value (64-bit version of the library).
Returns null on failure.

Description **PVK_AllocEnv()** is used to allocate the environment. The environment handle must be passed to the following functions in order to provide thread-safety of PXPIO operations; **PVK_RegisterKey()**, **PVK_OpenExt()**, **PVSetEnvMode()**, **PVGetEnvMode()** . The environment handle must be freed at the end of the session to avoid resource leaks.



Warning: Attempting to pass a bad or invalid environment handle to any PXPIO function can cause unpredictable results that may lead to abnormal termination.

PVK_DeAllocEnv()

Free Environment

Format void **PVK_DeAllocEnv**(**HPVKENV** *hEnv*);

Where:

hEnv Handle to environment structure.

Description **PVK_DeAllocEnv()** is used to de-allocate the environment.

PVSetEnvMode()

Set Environment Variables

Format **intptr_t** **APIDEF** **PVSetEnvMode**(**HPVKENV** *hEnv*, **int** *iFlag*, **intptr_t** *iValue*);

Where:

hEnv Handle to environment structure.

<i>iFlag</i>	Selector of the environment variable to be modified. Can be one of the following constants:
	PV_BURST_MODE 1
	PV_DIRTY_READ 2
	PV_LOCK_MODE 3
	PV_READ_ONLY 4
	PV_MAX_MB 5
<i>iValue</i>	Corresponding value for <i>iFlag</i> :
	PVK_BURST_ON 1
	PVK_BURST_OFF 0
	PVK_DIRTY_ON 1
	PVK_DIRTY_OFF 0
	PVK_DONT_CHECK_LOCK 1 (<i>Don't check, never lock read records</i>)
	PVK_CHECK_LOCK 2 (<i>Check for extracted records</i>)
	PVK_CHECK_LOCK_NOWAIT 4 (<i>Check for extracted records, exit if locked</i>)
	PVK_HDR_LOCK_NOWAIT 8 (<i>Don't wait for a locked header</i>)
	PVK_READONLY_ON 1
	PVK_READONLY_OFF 0
	PV_MAX_MB (<i>Maximum size of file segment in MB, integer value 0 to 2000</i>)

Description **PVK_SetEnvMode()** is used to set the value of environment variables. If successful, the function returns the previous value of the modified environment variable. If specified and *iValue* is not valid, **ERR_BAD_TYPE (-5)** is returned. Returns **PV_ERROR (-1)** on failure.

Additional Notes

PV_BURST_MODE:

- Normal processing of a file involves locking each area of the file as it is read. Activating burst mode greatly reduces the number of locks issued against a file. With Burst mode set, the PXPIO routines lock the file header for either 50 file operations or three-tenths of a second, whichever occurs first. This decreases the number of times the file must be locked, and the number of times that internal buffers may need to be reloaded.

PV_DIRTY_READ:

- Dirty Read mode of operation skips the normal file consistency checks. Dirty reads can speed file processing by reducing the number of locks issued against a file. However this may result in inconsistent data should the file be updated while being read by the PXPIO routines.

PV_LOCK_MODE:

- The Lock Mode is used to control whether to check for locked / extracted records when reading and writing. The default setting is to not check for locked records for backwards compatibility with older versions of the PXPIO routines.
- Note: This flag should normally be set to PVK_CHECK_LOCK when files are being updated concurrently by PxPlus and applications using the PXPIO routines. A setting of PVK_DONT_CHECK_LOCK will allow the PXPIO routines to read and write a record that is extracted in PxPlus. The remaining settings provide a quicker means of checking for a locked record or file header and will return immediately rather than retrying the lock.

PV_MAX_MB :

- The PV_MAX_MB setting is used to control the approximate size of a file in mega bytes before additional segments are created. This setting is functionally equivalent to the 'MB' (Mega-Bytes) system parameter in PxPlus. Values for PV_MAX_MB must be in the range of zero (0) to two thousand (2000). The default is two thousand (2000). Specifying a value of zero (0) resets this parameter to its default.

PVGetEnvMode()

Get Environment Variables

Format `intptr_t APIDEF PVGetEnvMode(HPVKENV hEnv, int iFlag);`

Where:

hEnv Handle to environment structure.

iFlag Selector of the environment variable to retrieve the value.

Description **PVK_GetEnvMode()** is used to get the value of the environment variable. If successful, the function returns the value of the environment variable specified by *iFlag*. Returns PV_ERROR (-1) on failure.

PVK_open()

File Open (Obsolete)

Description *Obsolete.* Supported for backwards compatibility only. Refer to the **PVK_OpenExt()** function.

PVK_openEx()*Extended File Open (Obsolete)*

Description *Obsolete.* Supported for backwards compatibility only. Refer to the [PVK_OpenExt\(\)](#) function.

PVK_OpenExt()*Extended File Open*

Format `int PVK_OpenExt(HPVKENV hEnv, char *path, char *pswd, int pswd_sz, INT16 opt, INT32 *open_err);`

Where:

hEnv Handle to environment structure created by [PVK_AllocEnv\(\)](#).

path Pointer to a null terminated string containing the pathname of the keyed/direct/indexed/view file to open.

pswd Pointer to a buffer that contains the optional password required to access a keyed or direct file.

pswd_sz Indicates the length of the pswd buffer.

opt Indicates whether a file should be opened in read-only mode (Windows or UNIX) or for exclusive use (Windows Only).

open_err Error code (see error code values below).

Description **PVK_OpenExt()** is used to open a PxPlus keyed/direct/indexed/EFF files or Views which requires a password or extended options. It will return the logical file handle for the file provided it can be opened. All subsequent file I/O calls to PXPIO functions must specify the returned handle.

Valid *opt* values include **WSF_INPUT** for read-only and **WSF_LOCK** for exclusive mode. A value of -1 is returned if the file cannot be opened.

Opt Table

```
#define FAM_READONLY 0x0000 /* File in read only mode */
#define FAM_READWRITE 0x0001 /* File in read write mode */
#define WSF_LOCK 0x0400 /* File was opened with exclusive use */
```

PXPIO Error Codes

```
#define ERR_OK 0 /* no error */
#define ERR_CANT_OPEN 1
#define ERR_BAD_FH 2
#define ERR_NOSUCH_KEY 3
#define ERR_EOF 4
#define ERR_BAD_TYPE 5
```

```
#define ERR_KEYNO 6
#define ERR_KEY_LENGTH 7
#define ERR_NO_MEMORY 8
#define ERR_KIO_OFS 9
#define ERR_KIO_FAILED 10
#define ERR_KIO_WRONG 11
#define ERR_KSZ_WRONG 12
#define ERR_RSZ_WRONG 13
#define ERR_SEEK_FAILED 14
#define ERR_READ_FAILED 15
#define ERR_READ_SHORT 16
#define ERR_BAD_FUNCTION 17
#define ERR_INDEXED_FILE 18
#define ERR_WRITE_FAILED 19
#define ERR_KIO_BADADR 20
#define ERR_KIO_DELCHN 21
#define ERR_KIO_NOEOF 22
#define ERR_BUSY 23 /* File or Data busy */
#define ERR_FILE_FULL 24
#define ERR_NOT_REGISTERED 25
#define ERR_DOM 26 /* Duplicate key not allowed - if
missing Rpt ERR_NO_SUCH_KEY */
#define ERR_KIO_RSIZ 27 /* Keyed file error (Record length
invalid) */
#define ERR_KIO_BADSEG 28 /* Invalid segment number */
#define ERR_IND_HEADER 29 /* Unable to access Indexed file
header */
#define ERR_KIO_DECOMPFAIL 30 /* Decompress of record failed */
#define ERR_PSWD_WRONG 31 /* Wrong password supplied */
#define ERR_BAD_OFFSET 32 /* Bad Read Offset */
#define ERR_NO_SUCH_FILE 33 /* File does not exist (or already
exists) */
#define ERR_RESTRICT_FAILED 34
#define ERR_ACCESS_VLTN 35 /* Access violation, attempt to write
to ReadOnly file */
#define ERR_TX_BEGIN 36 /* Begin transaction without finishing
previous */
#define ERR_TX_ROLLBACK 37 /* Rollback/Commit without proper
Begin transaction */
#define ERR_FILE_BUSY 38 /* File is busy */
#define ERR_MISSING_INFO 39 /* Not enough information passed in */
#define ERR_OBJ_VER_WRONG 40
#define ERR_BAD_BUFFER 41 /* Incorrect buffer returned from
page_get */
#define ERR_SYS_NOFH 42 /* os error: No more file handles
available (too many open files)*/
#define ERR_NET_FAILED 43 /* network error */
#define ERR_VERSION 44
#define ERR_SECURITY_FAILED 45 /* Logon failed */
#define ERR_PVK_NOTSUPPORTED 46 /* Feature is not supported */
```

PVK_close()

File Close

Format `int PVK_close(int fh);`

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

Description **PVK_close()** closes the file and releases all resources (memory) associated with the specified file handle.

PVK_read()

Read a Record from a File

Format `int PVK_read(int fh, char *dtabfr, int dtasz, int function);`

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

dtabfr Pointer to the data buffer to receive the record data.

dtasz Size (in bytes) of the data buffer.

function Type of read to be performed - values are:

PVKRD_CUR	Returns current
PVKRD_NEXT	Returns next
PVKRD_PRIOR	Returns prior
PVKRD_LOCK	OR'ed into function to lock the record
PVKRD_UNLOCK	OR'ed into function to unlock all records

Description **PVK_read()** is used to read a record from a PxPlus Keyed, Indexed, or EFF file. The return value will contain the length of the record in bytes or -1 if an error occurred. A return value of -2 indicates that the supplied buffer was not large enough to store the entire data record.

A record may be locked or extracted by specifying **PVKRD_LOCK** in conjunction with the appropriate function (e.g., **PVKRD_NEXT | PVKRD_LOCK**).



Note: For files with an external key (Direct files) the data returned will consist of the external key followed by the data.

For example, a Direct file with a 6 character key and an 80 character record size will return an 86 byte record - characters 1-6 will be the external key padded with nulls followed by the record data.

PVK_seek()

Position within a File

Format int **PVK_seek**(int *fh*, char **keybfr*, int *keysz*, int *keyno*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.
keybfr Pointer to a buffer containing the key.
keysz Size of the key in bytes.
keyno Key number to use (0=Current key, 1=Primary, 2=first alternate, etc.).

Description **PVK_seek()** is used to position the key pointer to a specified location within a file for subsequent processing. By default the Key IO routines read using the primary access key (KEY 1). An alternate key chain may be specified in the *keyno* parameter. If *keyno* is set to 0, the current key is used. If successful, a status of 0 is returned

PVK_write()

Write/Rewrite a Record

Format int **PVK_write**(int *fh*, char **dtabfr*, int *dtasz*, char **keybfr*, int *keysz*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.
dtabfr Pointer to the data buffer to receive the record data.
dtasz Size of the record in bytes.
keybfr Pointer to a buffer containing the external key, if applicable.
keysz Size of the external key in bytes.

Description **PVK_write()** is used to write or rewrite a record to a PxPlus keyed, indexed, or EFF file.

The **PVK_insert** and **PVK_update** functions may be used if an application needs to differentiate between creating new records versus updating existing records.

The data buffer must contain a properly formatted record with the length of the record specified. The value supplied in *dtasz* should contain the actual size of the record rather than the size of the data buffer. **PVK_write** will pad the data record with nulls as required for files with fixed length records.

The key buffer and length must contain the necessary key information for a file with an external key. If no external key is defined for the file then the *keysz* field must be set to zero.

The calling application is responsible for constructing a valid PxPlus data record using field separators as required.

If successful this function will return 0 otherwise it will return -1.

PVK_insert()

Write a New Record

Format `int PVK_insert(int fh, char *dtabfr, int dtasz, char *keybfr, int keysz);`

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

dtabfr Pointer to the data buffer to receive the record data.

dtasz Size of the record in bytes.

keybfr Pointer to a buffer containing the external key, if applicable.

keysz Size of the external key in bytes.

Description **PVK_insert()** is used to write a new record into a PxPlus keyed, indexed, or EFF file. It returns an error if a record with the same key value exists.

The data buffer must contain a properly formatted record with the length of the record specified. The value supplied in *dtasz* should contain the actual size of the record rather than the size of the data buffer. **PVK_insert()** will pad the data record with nulls as required for files with fixed length records.

The key buffer and length must contain the necessary key information for a file with an external key. If no external key is defined for the file then the *keysz* field must be set to zero.

The calling application is responsible for constructing a valid PxPlus data record using field separators as required.

If successful this function will return 0 otherwise it will return -1.

PVK_update()

Update an Existing Record

Format `int PVK_update(int fh, char *dtabfr, int dtasz, char *keybfr, int keysz);`

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

dtabfr Pointer to the data buffer to receive the record data.

dtasz Size of the record in bytes.

keybfr Pointer to a buffer containing the external key, if applicable.

keysz Size of the external key in bytes.

Description **PVK_update()** is used to update an existing record in a PxPlus keyed, indexed, or EFF file. The **PVK_update()** function will return an error if the record does not already exist.

The data buffer must contain a properly formatted record with the length of the record specified. The value supplied in *datasz* should contain the actual size of the record rather than the size of the data buffer. **PVK_update** will pad the data record with nulls as required for files with fixed length records.

The key buffer and length must contain the necessary key information for a file with an external key. If no external key is defined for the file then the *keysz* field must be set to zero.

The calling application is responsible for constructing a valid PxPlus data record using field separators as required.

If successful this function will return 0 otherwise it will return -1.

PVK_remove()

Remove a Record

Format int **PVK_remove**(int *fh*, char **keybfr*, int *keysz*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.
keybfr Pointer to a buffer containing the external key of the record to remove.
keysz Size of the primary key in bytes.

Description **PVK_remove()** is used to remove a record from a PxPlus keyed file. The length and value of the primary key for the record must be specified.

Records can only be removed from a file using the primary key. Alternate keys cannot be used.

This function cannot be used with indexed files.

If successful this function will return 0 otherwise it will return -1.

PVK_getpos()

Get Address/Position within File

Format long **PVK_getpos**(int *fh*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

Description **PVK_getpos()** returns the address of the record associated with the current key pointer for the specified file handle.
A return value of -1 is returned if the function is unsuccessful.

PVK_setpos() *Set Address/Position of File*

Format int **PVK_setpos**(int *fh*, long *addr*);
Where:
fh File handle returned from a prior call to **PVK_OpenExt()**.
addr Address/position of the record.

Description **PVK_setpos()** sets the current record address based on the specified address.
If successful, a status of 0 is returned.

PVK_get_max_mb() *Deprecated*

Description *Deprecated.* This has been replaced by the **PVGetEnvMode()** function with an *iFlag* setting of PV_MAX_MB.

PVK_set_max_mb() *Deprecated*

Description *Deprecated.* This has been replaced by the **PVSetEnvMode()** function with an *iFlag* setting of PV_MAX_MB.

PVK_CheckLock() *Deprecated*

Description *Deprecated.* This has been replaced by the **PVGetEnvMode()** and **PVSetEnvMode()** functions with an *iFlag* setting of PV_LOCK_MODE.

PVK_dirty()

Deprecated

Description *Deprecated.* This has been replaced by the **PVGetEnvMode()** and **PVSetEnvMode()** functions with an *iFlag* setting of PV_DIRTY_READ.

PVK_burst()

Deprecated

Description *Deprecated.* This has been replaced by the **PVGetEnvMode()** and **PVSetEnvMode()** functions with an *iFlag* setting of PV_BURST_MODE.

PVK_geterrno()

Return Last Error Status

Format int PVK_geterrno(void);

Description This function returns the last known error status. It will return one of the following values (see PVKIO.H):

```
#define ERR_CANT_OPEN      1
#define ERR_BAD_FH        2
#define ERR_NOSUCH_KEY    3
#define ERR_EOF            4
#define ERR_BAD_TYPE      5
#define ERR_KEYNO         6
#define ERR_KEY_LENGTH    7
#define ERR_NO_MEMORY     8
#define ERR_KIO_QFS       9
#define ERR_KIO_FAILED   10
#define ERR_KIO_WRONG    11
#define ERR_KSZ_WRONG    12
#define ERR_RSZ_WRONG    13
#define ERR_SEEK_FAILED  14
#define ERR_READ_FAILED  15
#define ERR_READ_SHORT   16
#define ERR_BAD_FUNCTION 17
#define ERR_INDEXED_FILE 18
#define ERR_WRITE_FAILED 19
#define ERR_KIO_BADADR   20
#define ERR_KIO_DELCHN   21
#define ERR_KIO_NOEOF    22
#define ERR_BUSY         23
#define ERR_FILE_FULL    24
#define ERR_NOT_REGISTERED 25
```

```
#define ERR_DOM 26
#define ERR_KIO_RSIZE 27
#define ERR_KIO_BADSEG 28
#define ERR_IND_HEADER 29
#define ERR_KIO_DECOMPFAIL 30
#define ERR_PSWD_WRONG 31
#define ERR_BAD_OFFSET 32
#define ERR_NO_SUCH_FILE 33
#define ERR_RESTRICT_FAILED 34
#define ERR_ACCESS_VLTN 35
#define ERR_TX_BEGIN 36
#define ERR_TX_ROLLBACK 37
#define ERR_FILE_BUSY 38
#define ERR_MISSING_INFO 39
#define ERR_OBJ_VER_WRONG 40
```

PVK_strerror()

Return Last Error Message

Format `char * PVK_strerror(void);`

Description This function returns the text of the current error status. Values are:

```
"Can't open data file"
"Bad file handle number"
"Invalid key specified"
"End of file reached"
"Bad file type -- Not a KEYED file"
"Key number invalid"
"Length invalid"
"No system memory available"
"File error : Offset error"
"File error : Read of key buffer failed"
"File error : Key header address invalid"
"File error : Key size invalid"
"File error : Record size invalid"
"File error : Seek failed"
"File error : Read failed"
"File error : Truncated read"
"Bad internal function code"
"File type is indexed"
"Write command failed"
"Keyed Io returned bad address"
"Deleted record chain corrupted"
"No EOF marker found in keyed file"
"File header or record busy -- retry later"
```

```
"File full" "Registration
Failure" "Duplicate key not
allowed"
"File error : Record length invalid"
"File error : Invalid Segment number"
"Unable to access Indexed file header"
"File error : Decompression failed"
"File error : Password Incorrect"
"Bad record offset"
"File does not exist"
"Unknown operator in restrict routine"
"Access Violation: File is in Read Only mode"
"Begin transaction without ending previous transaction"
"Rollback/Commit without Begin transaction"
"File header is busy -- retry later"
"Required information missing"
"Views object version wrong"
```

PVK_dict()

Read Dictionary

Format int **PVK_dict**(int *fh*, int *dctidx*, char **dctbfr*, int *dctbsz*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.
dctidx Data dictionary entry.
dctbfr Pointer to the data buffer to receive the data dictionary record.
dctbsz Size of the data buffer in bytes.

Description This function can be used to read the embedded data dictionary records held within a file. The format of the information contained within the data dictionary is subject to change and as such, is not documented in this reference manual.

PVK_deffh()

Pointer to Internal Structure Block

Format struct PVKINF * **PVK_deffh** (int *fh*);

Where:

fh File handle returned from a prior call to **PVK_OpenExt()**.

Description This function may be used to obtain a pointer to the internal structures maintained by PXPIO.



Note: The values contained within this structure should not be modified by an outside application. Any attempt to do so, may result in file corruption and/or cause the application to become unstable.

See PVKIO.H for more details on this structure.

If successful, this function will return valid pointer otherwise it will return *null*.



Warning: Passing a bad or invalid file handle can cause unpredictable results that may lead to abnormal termination.

PVK_register()

Deprecated

Description *Deprecated.* This has been replaced by the **PVK_RegisterKey()** function.

PVK_RegisterKey()

Register Usage of Library

Format int **PVK_RegisterKey**(HPVKENV *hEnv*, char **reg_str*, long *reg_num*);

Where:

hEnv Handle to environment structure created by **PVK_AllocEnv()**.

reg_str Registration string provided by PVX Plus Technologies Ltd.

reg_num Registration number provided by PVX Plus Technologies Ltd.

Description **PVK_RegisterKey()** must be called prior to opening the first file in order to provide the DLL with a valid registration string and key. Without this registration information, a warning message that requires user intervention will be displayed whenever a file is opened.

The PXPIO routines are not to be redistributed as part of any application without first having purchased and obtained a proper registration string and number from PVX Plus Technologies Ltd.

Example

```

/*      sample.c : Sample PXPIO console application*/
#include <stdio.h>
#include <windows.h>
#include "pvkio.h"
int main(int argc, char* argv[])
{
    HMODULE hPvkio;
    FARPROC PVK_OpenExt, PVK_close, PVK_read, PVK_write, PVK_seek;
    FARPROC PVK_AllocEnv, PVK_DeAllocEnv, PVK_RegisterKey;
    HPVKENV hEnv;

    int fh, keysz, dtasz, i, sts, fc;
    char bfr[256], keybfr[4+1], dtabfr[4+256+1], pswd[32];
    INT16 opt = 0;
    INT32 open_err = 0;

    memset(pswd, 0x00, sizeof(pswd));

    /* Load the DLL and locate necessary entrypoints */
    if ((hPvkio = LoadLibrary("pxpio.dll")) EQ NULL) return -1;
    if ((PVK_OpenExt = GetProcAddress(hPvkio, "PVK_OpenExt")) EQ NULL) return -2;
    if ((PVK_close = GetProcAddress(hPvkio, "PVK_close")) EQ NULL) return -2;
    if ((PVK_read = GetProcAddress(hPvkio, "PVK_read")) EQ NULL) return -2;
    if ((PVK_write = GetProcAddress(hPvkio, "PVK_write")) EQ NULL) return -2;
    if ((PVK_seek = GetProcAddress(hPvkio, "PVK_seek")) EQ NULL) return -2;
    if ((PVK_AllocEnv = GetProcAddress(hPvkio, "PVK_AllocEnv")) EQ NULL) return -2;
    if ((PVK_DeAllocEnv = GetProcAddress(hPvkio, "PVK_DeAllocEnv"))EQ NULL) return -2;
    if ((PVK_RegisterKey = GetProcAddress(hPvkio, "PVK_RegisterKey"))EQ NULL) return -2;

    /* Create a new Environment */
    hEnv = (HPVKENV) (*PVK_AllocEnv) ();
    if (hEnv EQ NULL) return -3;
    (*PVK_RegisterKey) (hEnv, "<Insert License Name and Number here>", 12345678L);
    fh=(int) ((*PVK_OpenExt) (hEnv, "testfile", pswd, (int) sizeof(pswd), opt, &open_err));
    if (fh EQ (int) -1) return -4;

    /* Insert/Update 10 records */
    for(i=1;i<=10;i++)
    {
        sprintf(keybfr, "%04d", i);
        sprintf(dtabfr, "This is record #d%c", i, 0x8a);
        keysz = strlen(keybfr);
        dtasz = strlen(dtabfr);

        sts = (int) ((*PVK_write) (fh, &dtabfr, dtasz, &keybfr, keysz));
        sprintf(bfr, "Writing: %s - %s - sts=%d\n", keybfr, dtabfr, sts);
        printf(bfr);
    }

    /* Seek to key 0005 and read until end of file */
    sts = (int) ((*PVK_seek) (fh, "0005", 4, 1));
    fc = PVKRD_CUR;
    for(;;)
    {
        sts = (int) ((*PVK_read) (fh, &dtabfr, sizeof(dtabfr), fc));
        if (sts EQ -1) break; /* EOF */
        dtabfr[sts] = 0;
        sprintf(bfr, "Read: %s - sts=%d\n", dtabfr, sts);
        printf(bfr);
        fc = PVKRD_NEXT;
    }

    (*PVK_close) (fh);
    (*PVK_DeAllocEnv) (hEnv)
    ; FreeLibrary(hPvkio);
    return 0;
}

```