ProvideX

VERSION 6

C-Library
File IO Routines

best software

Publication Release: V6

File IO Routines

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

PVK_open( )	*File Open*
PVK_openEx( )	*Extended File Open*
PVK_close( )	*File Close*
PVK_read( )	*Read a Record from a File*
PVK_seek( )	*Position within Keyed/Indexed File*
PVK_write( )	*Write/Rewrite a Record*
PVK_insert( )	*Write a New Record*
PVK_update( )	*Update an Existing Record*
PVK_remove( )	*Remove a Record*
PVK_getpos( )	*Get Address/Position within File*
PVK_setpos( )	*Set Address/Position of File*
PVK_get_max_mb( )	*Return Multi-Segmented Threshold*
PVK_set_max_mb( )	*Set Size for Multi-Segmented Files*
PVK_CheckLock( )	*Set Locking for Extracted Records*
PVK_dirty( )	*Set Dirty Read Mode Option*
PVK_burst( )	*Set Burst Mode Option*
PVK_geterrno( )	*Return Last Error Status*
PVK_strerr( )	*Return Last Error Message*
PVK_dict( )	*Read Dictionary*
PVK_deffh( )	*Pointer to Internal Structure Block*
PVK_register( )	*Register Usage of Library*

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 following environments:

- 32-bit Windows
- AIX
- RedHat
- SCO UNIX

Registration

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

Distribution of the PVKIO routines is restricted to only those companies that apply for and receive a registration string and number directly from Best Software Canada.

PVK_open( ) *File Open*

Format

int **PVK_open**(char **path*);

Where:

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

Description

PVK_open( ) is used to open a ProvideX keyed/direct/indexed file and to return the logical file handle for the file. All subsequent file IO calls to PVKIO functions must specify the returned handle.

There is a maximum of 100 files allowed to be open at any time.

A value of -1 is returned if the file cannot be opened.

PVK_openEx( ) *Extended File Open*

Format

int **PVK_openEx**(char **path*, char **pswd*, int *pswd_sz*, INT16 *opt*);

Where:

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

Description

PVK_openEx( ) is used to open a ProvideX Keyed or Direct file 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 IO calls to PVKIO functions must specify the returned handle.

Valid *opt* values include `WSF_INPUT` for Read-Only and `WSF_LOCK` for Exclusive mode.

There is a maximum of 100 files allowed to be open at any time.

A value of -1 is returned if the file cannot be opened.

PVK_close( ) File Close

Format

int **PVK_close**(int *fh*);

Where:

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

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_open( )**.

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 ProvideX Keyed or Indexed 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 Keyed/Indexed File*

Format

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

Where:

fh	File handle returned from a prior call to **PVK_open( )**.
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_open( )**.
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 ProvideX keyed or indexed 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 ProvideX 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_open( )**.
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 ProvideX keyed or indexed 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 ProvideX 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_open( )**.
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 ProvideX keyed or indexed 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 *dtasz* 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 ProvideX 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_open( )**.
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 ProvideX 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_open( )**.

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_open( )**.
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( ) *Return Multi-Segmented Threshold*

Format

int **PVK_get_max_mb**(void);

Description

PVK_get_max_mb( ) returns the size setting for multi-segmented files. This setting represents the approximate size of a file in mega-bytes before additional segments are created.

The return value will be in the range of 1 to two thousand (2000).

For more information, refer to the ProvideX '**MB**' (Mega-Bytes) system parameter .

PVK_set_max_mb( ) *Set Size for Multi-Segmented Files*

Format

int **PVK_set_max_mb**(int *max_mb*);

Where:

max_mb Approximate file segment size.

Description

PVK_set_max_mb( ) 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 ProvideX.

Values for *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 this default.

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

PVK_CheckLock( ) *Set Locking for Extracted Records*

Format

int **PVK_CheckLock**(int *flag*);

Where:

flag Determines whether to check for record locks:

PVK_DONT_CHECK_LOCK	Don't check and never lock read records.
PVK_CHECK_LOCK	Check for EXTRACTed records.
PVK_CHECK_LOCK_NOWAIT	Check for EXTRACTed records and exit immediately if locked.
PVK_HDR_LOCK_NOWAIT	Don't wait for a locked header.

Description

PVK_CheckLock( ) 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 PVKIO routines.

Note: This flag should normally be set to PVK_CHECK_LOCK when files are being updated concurrently by ProvideX and applications using the PVKIO routines. A setting of PVK_DONT_CHECK_LOCK will allow the PVKIO routines to read and write a record that is extracted in ProvideX. 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.

The function will return the previous state of the flag.

PVK_dirty( ) *Set Dirty Read Mode Option*

Format

void **PVK_dirty**(int *flag*);

Where:

flag Determines whether to use Dirty read mode:
TRUE - uses the dirty read mode to process files
FALSE - does not use the dirty read mode (default)

Description

PVK_dirty( ) is used to set the dirty read mode state. The default setting is to not use the dirty read mode for processing files.

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 PVKIO routines.

There is no return value for this function.

PVK_burst( ) *Set Burst Mode Option*

Format

void **PVK_burst**(int *flag*);

Where:

flag Determines whether to use burst mode:
TRUE - uses the burst mode to process files
FALSE - does not use the burst mode (default)

Description

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 PVKIO 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.

There is no return value for the **PVK_burst( )** function.

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_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
#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_strerr( ) *Return Last Error Message*

Format char * **PVK_strerr**(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_open( )**.
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_open( )**.

Description

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

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 unpredicable results that may lead to abnormal termination.

PVK_register( ) *Register Usage of Library*

Format

void **PVK_register** (char **reg_str*, long *reg_num*);

Where:

reg_str	Registration string provided by Best Software Canada.
reg_num	Registration number provided by Best Software Canada.

Description

This function 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 PVKIO routines are not to be redistributed as part of any application without first having purchased and obtained a proper registration string and number from Best Software Canada.

Example

```
/*    sample.c : Sample PVKIO console application    */

#include <stdio.h>
#include <windows.h>

#include "pvkio.h"

int main(int argc, char* argv[])
{
    HMODULE hPvkio;
    FARPROC PVK_open, PVK_close, PVK_read, PVK_write, PVK_seek, PVK_register;

    int fh, keysz, dtasz, i, sts, fc;
    char bfr[256], keybfr[4+1], dtabfr[4+256+1];

        /* Load the DLL and locate necessary entrypoints */
    if ((hPvkio = LoadLibrary("pvkio32.dll")) EQ NULL) return -1;

    if ((PVK_open     = GetProcAddress(hPvkio, "PVK_open"))     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_register = GetProcAddress(hPvkio, "PVK_register")) EQ NULL) return -2;

    (*PVK_register)("<Insert License Name and Number here>",12345678L);

    if ((fh = (int)((*PVK_open)("testfile"))) EQ (int)-1) return -3;

        /*    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);
    FreeLibrary(hPvkio);
    return 0;
}
```