Portable Systems Group

NT OS/2 File System Support Routines Specification

Author: *Gary D. Kimura*

Revision 1.0, August 10, 1990

1. Introduction

This specification describes a library of file system support routines for use by the different file systems within **NT OS/2**. They are executive level routines that are too file system specific to belong in the Ex or public Rtl component, and are also inappropriate for the I/O component.

The name of this component is **FsRtl**. Each set of routines within **FsRtl** is tailored for supporting a specific file system function. Most components within **FsRtl** define an abstract data type and routines for manipulating the data. In addition, the header file for **FsRtl** defines some global data types common to multiple file systems.

The global data types defined within **FsRtl** are:

- Logical Block Number (**LBN**). **LBN** is the moniker used to identify physical blocks on the disk. The numbering sequence is from zero to *N*-1 where *N* is the number of sectors on the disk.

- Virtual Block Number (**VBN**). The **VBN** identifies the sectors of a file relative to the start of the file. A value of 0 corresponds to the first sector of data for a file, a value of 1 corresponds to the second sector, and so forth.

The individual categories of support provided by **FsRtl** are:

- Byte Range File Locks (**FILE_LOCK**). This package implements a set of routines for handling byte range file locking. The routines provide a consistent method of all file system to maintain and implement byte range file locks.

- Name Support. This package provides string manipulation routines and macros that are tailored for file system usage.

- Mapped Control Block (**MCB**). This package provides for in-memory retrieval mapping support. Retrieval mapping is the correspondence between LBN's and VBN's for a given file.

- Volume Mapped Control Block (**VMCB**). The Pinball and Fat file systems treat the ancillary structures of the on-disk file system as one large file, called the Volume File. This allows the file systems to utilize memory management for maintaining a cache of these sectors stored in memory. This package provides necessary support for constructing and maintaining an artificial mapping between LBNs and VBNs in the volume file.

- Notify Change Directory Routines. \\ still needs to be added \\

To use the FsRtl component a file system must explicitly include the file <fsrtl.h> (i.e., the file is not automatically included with <ntos.h>).

The remainder of this document describes, in detail, each of the preceding components, and also miscellaneous macros.

2. Miscellaneous Support Macros

This module defines all of the general File System Rtl routines

2.1 FsRtlCompleteRequest

VOID
FsRtlCompleteRequest(
IN PIRP *Irp*,
IN NTSTATUS *Status*
)

Routine Description:

This routine is used to complete an IRP with the indicated status. It does the necessary raise and lower of IRQL.

Parameters:

Irp - Supplies a pointer to the Irp to complete

Status - Supplies the completion status for the Irp

Return Value:

None.

3. Byte Range File Lock Routines

The file lock package provides a set of routines that allow the caller to handle byte range file lock requests. A variable of type FILE_LOCK is needed for every file with byte range locking. The package provides routines to set and clear locks, and to test for read or write access to a file with byte range locks.

The main idea of the package is to have the file system initialize a FILE_LOCK variable for every data file as its opened, and then to simply call a file lock processing routine to handle all IRP's with a major function code of LOCK_CONTROL. The package is responsible for keeping track of locks and for completing the LOCK_CONTROL IRPS. When processing a read or write request the file system can then call two query routines to check for access.

Most of the code for processing IRPS and checking for access use paged pool and can encounter a page fault, therefore the check routines cannot be called at DPC level. To help servers that do call the file system to do read/write operations at DPC level there is a additional routine that simply checks for the existence of a lock on a file and can be run at DPC level.

Concurrent access to the FILE_LOCK variable must be control by the caller.

The functions provided in this package are as follows:

- o FsRtlInitializeFileLock - Initialize a new FILE_LOCK structure.
- o FsRtlUninitializeFileLock - Uninitialize an existing FILE_LOCK structure.
- o FsRtlProcessFileLock - Process an IRP whose major function code is LOCK_CONTROL.
- o FsRtlCheckLockForReadAccess - Check for read access to a range of bytes in a file.
- o FsRtlCheckLockForWriteAccess - Check for write access to a range of bytes in a file.
- o FsRtlAreThereCurrentFileLocks - Check if there are any locks currently assigned to a file.
- o FsRtlGetNextFileLock - This procedure enumerates the current locks of a file lock variable.

3.1 FsRtlInitializeFileLock

VOID
FsRtlInitializeFileLock(
IN PFILE_LOCK *OpaqueFileLock*,
IN POOL_TYPE *PoolType*,
IN PCOMPLETE_LOCK_IRP_ROUTINE *CompleteLockIrpRoutine* **OPTIONAL**
)

Routine Description:

This routine initializes a new FILE_LOCK structure. The caller must supply the memory for the structure. This call must precede all other calls that utilize the FILE_LOCK variable.

Parameters:

OpaqueFileLock - Supplies a pointer to the FILE_LOCK structure to initialize.

PoolType - Supplies the pool type to use when allocating additional internal storage. If nonpaged pool is selected then all of the routines in this package can be called a DPC level. However, using nonpaged pool for storing file lock information is not a wise use of nonpaged pool.

CompleteLockIrpRoutine - Optionally supplies an alternate routine to call for completing IRPs. FsRtlProcessFileLock by default will call IoCompleteRequest to finish up an IRP; however if the caller want to process the completion itself then it needs to specify a completion routine here. This routine will then be called in place of IoCompleteRequest.

Return Value:

None.

3.2 FsRtlUninitializeFileLock

VOID
FsRtlUninitializeFileLock(
IN PFILE_LOCK *OpaqueFileLock*
)

Routine Description:

This routine uninitializes a FILE_LOCK structure. After calling this routine the File lock must be reinitialized before being used again.

This routine will free all files locks and completes any outstanding lock requests as a result of cleaning itself up.

Parameters:

OpaqueFileLock - Supplies a pointer to the FILE_LOCK struture being decommissioned.

Return Value:

None.

3.3 FsRtlAreThereCurrentFileLocks

BOOLEAN
FsRtlAreThereCurrentFileLocks(
IN PFILE_LOCK *OpaqueFileLock*
)

Routine Description:

This routine tells its if caller there are any current file locks for the file. It does this test by simply checking the file lock variable and not by accessing other pieces of memory. Therefore if the FileLock variable is in nonpaged pool then this test can be done at DPC.

Parameters:

FileLock - Supplies the File lock being queried

Return Value:

BOOLEAN - TRUE if there are current locks on the file and FALSE otherwise

3.4 FsRtlProcessFileLock

NTSTATUS
FsRtlProcessFileLock(
IN PFILE_LOCK *OpaqueFileLock***,**
IN PIRP *Irp***,**
IN PVOID *Context* **OPTIONAL**
)

Routine Description:

This routine processes a file lock IRP it does either a lock request, or an unlock request. It also completes the IRP. Once called the user (i.e., File System) has relinquished control of the input IRP.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueFileLock - Supplies the File lock being modified/queried.

Irp - Supplies the Irp being processed.

Context - Optionally supplies a context to use when calling the user alternate IRP completion routine.

Return Value:

NTSTATUS - The return status for the operation.

3.5 FsRtlCheckLockForReadAccess

BOOLEAN
FsRtlCheckLockForReadAccess(
IN PFILE_LOCK *OpaqueFileLock*,
IN PIRP *Irp*
)

Routine Description:

This routine checks to see if the caller has read access to the range indicated in the IRP due to file locks. This call does not complete the Irp it only uses it to get the lock information and read information. The IRP must be for a read operation.

Parameters:

OpaqueFileLock - Supplies the File Lock to check.

Irp - Supplies the Irp being processed.

Return Value:

BOOLEAN - TRUE if the indicated user/request has read access to the entire specified byte range, and FALSE otherwise

3.6 FsRtlCheckLockForWriteAccess

BOOLEAN
FsRtlCheckLockForWriteAccess(
IN PFILE_LOCK *OpaqueFileLock*,
IN PIRP *Irp*
)

Routine Description:

This routine checks to see if the caller has write access to the indicated range due to file locks. This call does not complete the Irp it only uses it to get the lock information and write information. The IRP must be for a write operation.

Parameters:

OpaqueFileLock - Supplies the File Lock to check.

Irp - Supplies the Irp being processed.

Return Value:

BOOLEAN - TRUE if the indicated user/request has write access to the entire specified byte range, and FALSE otherwise

3.7 FsRtlGetNextFileLock

PFILE_LOCK_INFO
FsRtlGetNextFileLock(
IN PFILE_LOCK *OpaqueFileLock***,**
IN BOOLEAN *Restart*
)

Routine Description:

This routine enumerate the file lock current denoted by the input file lock variable. It returns a pointer to the file lock information stored for each lock. The caller is responsible for synchronizing call to this procedure and for not altering any of the data returned by this procedure.

The way a programing will use this procedure to enumerate all of the locks is as follows:

```
for (p = FsRtlGetNextFileLock( FileLock, TRUE );

     p != NULL;

     p = FsRtlGetNextFileLock( FileLock, FALSE )) {

    // Process the lock information referenced by p

  }
```

Parameters:

OpaqueFileLock - Supplies the File Lock to enumerate. The current enumeration state is stored in the file lock variable so if multiple threads are enumerating the lock at the same time the results will be unpredictable.

Restart - Indicates if the enumeration is to start at the beginning of the file lock list or if we are continuing from a previous call.

Return Value:

PFILE_LOCK_INFO - Either it returns a pointer to the next file lock record for the input file lock or it returns NULL if there are not more locks.

4. Name Support Routines

The name support package is for manipulating DBCS strings (later this will be extended to also handle UNICODE strings). The routines allow the caller to dissect and compare strings.

There are two exported typedef's defined by this package. The first is a structure called CODEPAGE. Every Dbcs routines takes as input a code page record. This record contains the double byte character and upcase information. If a code page not supplied the routines currently use the default US code page.

We need to work out the routines for a file system to contruct a code page.

The second typedef is an enumerated type called COMPARISON_RESULTS that is used when comparing two strings. It indicates if one string is less than, equal to, or greater than the other. The comparison routines also know how to handle wild cards.

The following routines are provided by this package:

o FsRtlDissectDbcs - This routine takes a path name string and breaks into two parts. The first name in the string and the remainder. It also checks that the first name is valid for an OS/2 file.

o FsRtlUpcaseDbcs - This routines takes a string and computes its upcased equivalent.

o FsRtlDbcsContainsWildCards - This routines tells the caller if a string contains any wildcard characters (i.e., * or ?).

o FsRtlCompareDbcs - This routine compares two strings.

o FsRtlIsDbcsInExpression - This routine is used to compare a string against a template (possibly containing wildcards) to sees if the string is in the language denoted by the template.

o FsRtlIsNameValid - This routine checks to see if a string contains valid characters.

o FsRtlIsPathValid - This routine checks to see if a string contains valid names separated by backslashes.

o FsRtlFirstDbcsCharacter - This routine is used to extract the first character from a DBCS string.

o FsRtlIslegalDbcsCharacter - This routine is used to decide if a DBCS character value is legal.

o FsRtlToUpperDbcsCharacter - This routine is used to upcase a single DBCS character.

4.1 FsRtlFirstDbcsCharacter

USHORT
FsRtlFirstDbcsCharacter(

IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *Name***,**
OUT PSTRING *RemainingName*
)

Routine Description:

This routine takes an input Dbcs string and returns as its function value the first character in the string and as an output parameter the remaining name after the first character. If the name is empty then a value of zero is returned. If the first character in the input name is invalid then an invalid character is returned and the remaining name is advanced by one byte through the input name.

Example of its results are:

Name	Function Result	RemainingName
empty	0	empty
A	A	empty
~A	~	A (~ denotes an illegal char)
AB	A	B

Note that given a Dbcs string denoted by a STRING variable Str the 1st, 2nd, and subsequent Dbcs characters can be extracted using the following programming construct

```
    while (Str.Length != 0) {

         Dbcs = FsRtlFirstDbcsCharacter( NULL, Str, &Str );

          //  Dbcs now contains the next character in the
string.

      }
```

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Name - Supplies the input string being examined

RemainingName - Receives the remaining part of the input string after the first character.

Return Value:

USHORT - Receives the first character found in the input string.

4.2 FsRtlDissectDbcs

BOOLEAN
FsRtlDissectDbcs(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *InputName*,
IN BOOLEAN *Is8dot3*,
OUT PSTRING *FirstPart*,
OUT PSTRING *RemainingPart*
)

Routine Description:

This routine takes an input Dbcs string and dissects it into two substrings. The first output string contains the name that appears at the beginning of the input string, the second output string contains the remainder of the input string.

In the input string backslashes are used to separate names. The input string must not start with a backslash. Both output strings will not begin with a backslash.

If the input string does not contain any names then both output strings are empty. If the input string contains only one name then the first output string contains the name and the second string is empty.

Note that both output strings use the same string buffer memory of the input string.

This routine returns a function result of TRUE if the input string is well formed (including empty) and contains only valid characters (including wildcards). It returns FALSE if the input string is illformed, contains invalid characters.

Example of its results are:

InputString	FirstPart	RemainingPart	Function Result
empty	empty	empty	TRUE
A	A	empty	TRUE
A\B\C\D\E	A	B\C\D\E	TRUE
*A?	*A?	empty	TRUE

`\A`	`empty`	`empty`	`FALSE`
`A[,]`	`empty`	`empty`	`FALSE`
`A\\B+;\C`	`A`	`\B+;\C`	`TRUE`

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when dissecting the input string; otherwise a generic code page is used.

InputName - Supplies the input string being dissected

*Is*8dot3 - Indicates if the first part of the input name must be 8.3 or can be long file name.

FirstPart - Receives the first name in the input string

RemainingPart - Receives the remaining part of the input string

Return Value:

BOOLEAN - TRUE if the input string is well formed and its first part does not contain any illegal characters, and FALSE otherwise.

4.3 FsRtlUpcaseDbcs

VOID
FsRtlUpcaseDbcs(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *InputName***,**
OUT PSTRING *OutputName*
)

Routine Description:

This routine copies and upcases a Dbcs input string into a caller supplied output string according to the following upcase mapping rules.

For character values between 0 and 127, upcase normally.

For character values between 128 and 255 and not DBCS, use the upcase table in the code page to upcase a single character.

For character values between 128 and 255 and DBCS, do not alter.

The first two points above are handled transparently via the Code Page

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when upcasing the input string; otherwise a generic code page is used.

InputName - Supplies the input string to upcase

OutputName - Receives the output string, the output buffer must already be supplied by the caller

Return Value:

None.

4.4 FsRtlDbcsContainsWildCards

```
BOOLEAN
FsRtlDbcsContainsWildCards(
    IN PCODEPAGE CodePage OPTIONAL,
    IN STRING Name
    )
```

Routine Description:

This routine checks if the input Dbcs name contains any wild card characters (i.e., * or ?).

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Name - Supplies the name to examine

Return Value:

BOOLEAN - TRUE if the input name contains any wildcard characters and FALSE otherwise.

4.5 FsRtlCompareDbcs

```
COMPARISON_RESULTS
FsRtlCompareDbcs(
    IN PCODEPAGE CodePage OPTIONAL,
    IN STRING Expression,
    IN STRING Name,
    IN COMPARISON_RESULTS WildIs,
    IN BOOLEAN CaseInsensitive
    )
```

Routine Description:

This routine compares a Dbcs expression with a Dbcs name lexigraphically for LessThan, EqualTo, or GreaterThan. If the expression does not contain any wildcards, this procedure does a complete comparison. If the expression does contain wild cards, then the comparison is only done up to the first wildcard character. The Name parameter must not contain wild cards. The wildcard character compares as less then all other characters. So the wildcard name "*.*" will always compare less than all other strings.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Expression - Supplies the first name (expression) to compare, optionally with wild cards. (Upcased already if CaseInsensitive is supplied as TRUE.)

Name - Supplies the second name to compare - no wild cards allowed.

WildIs - Determines what Result is returned if a wild card is encountered in the Expression String.

CaseInsensitive - TRUE if Name should be Upcased before comparing.

Return Value:

COMPARISON_RESULTS -

LessThan if Expression < Name

EqualTo if Expression == Name

GreaterThan if Expression > Name

4.6 FsRtlIsDbcsInExpression

BOOLEAN
FsRtlIsDbcsInExpression(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *Expression***,**
IN STRING *Name***,**
IN BOOLEAN *CaseInsensitive*
)

Routine Description:

This routine compares a Dbcs name and an expression and tells the caller if the name is equal to or not equal to the expression. The input name cannot contain wildcards, while the expression may contain wildcards.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Expression - Supplies the input expression to check against (Caller must already upcase if passing CaseInsensitive TRUE.)

Name - Supplies the input name to check for.

CaseInsensitive - TRUE if Name should be Upcased before comparing.

Return Value:

BOOLEAN - TRUE if Name is an element in the set of strings denoted by the input Expression and FALSE otherwise.

4.7 FsRtlIsNameValid

BOOLEAN
FsRtlIsNameValid(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *Name***,**
IN BOOLEAN *Is8dot3***,**
IN BOOLEAN *CanContainWildCards***,**
OUT PSTRING *LongestValidPrefix* **OPTIONAL**
)

Routine Description:

This routine scans the input Dbcs name and verifies that if only contains valid characters.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Name - Supplies the input name to check.

*Is*8dot3 - Specifies if the Name must be 8.3 or can be a long file name.

CanContainWildCards - Indicates if the name can contain wild cards (i.e., * and ?).

LongestValidPrefix - This optional output parameter receives a string denoting the largest valid prefix found for the input string. If the input string is completely valid then the longest valid prefix is equal to the input string.

Return Value:

BOOLEAN - TRUE if the input name is valid and FALSE otherwise.

4.8 FsRtlIsPathValid

BOOLEAN
FsRtlIsPathValid(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN STRING *Name*,
IN BOOLEAN *Is8dot3*,
IN BOOLEAN *MustHaveLeadingBackslash*,
IN BOOLEAN *CanContainWildCards*,
OUT PSTRING *LongestValidPrefix* **OPTIONAL**
)

Routine Description:

This routine scans the input Dbcs string and verifies that it is only composed of valid names separated by backslashes.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input string; otherwise a generic code page is used.

Name - Supplies the input name to check.

*Is*8dot3 - Specifies if the Name must be 8.3 or can be a long file name.

MustHaveLeadingBackslash - Specifies if the name must start with a leading backslash.

CanContainWildCards - Indicates if the name can contain wild cards (i.e., * and ?).

LongestValidPrefix - This optional output parameter receives a string denoting the largest valid prefix found for the input string. If the input string is completely valid then the longest valid prefix is equal to the input string.

Return Value:

BOOLEAN - TRUE if the input name is valid and FALSE otherwise.

4.9 FsRtlIsLegalDbcsCharacter

BOOLEAN
FsRtlIsLegalDbcsCharacter(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN USHORT *DbcsCharacter*
)

Routine Description:

This routine takes an input character (either double byte or single byte) and indicates to the caller if the character is legal. The input to this procedure should be the return value of having called FsRtlFirstDbcsCharacter.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input character; otherwise a generic code page is used.

DbcsCharacter - Supplies the input character being examined

Return Value:

BOOLEAN - TRUE if the input character is legal and FALSE otherwise

4.10 FsRtlToUpperDbcsCharacter

USHORT
FsRtlToUpperDbcsCharacter(
IN PCODEPAGE *CodePage* **OPTIONAL,**
IN USHORT *DbcsCharacter*
)

Routine Description:

This routine takes an input character (either double byte or single byte) and returns its upcased equivalent character. The input to this procedure should be the return value of having called FsRtlFirstDbcsCharacter.

Parameters:

CodePage - Is an optional input parameter. If supplied it specifies the code page to use when examining the input character; otherwise a generic code page is used.

DbcsCharacter - Supplies the input character being upcased

Return Value:

USHORT - Recieves the upcased character

5. Mapped Control Block Routines

The MCB routines provide support for maintaining an in-memory copy of the retrieval mapping information for a file. The general idea is to have the file system lookup the retrieval mapping for a VBN once from the disk, add the mapping to the MCB structure, and then utilize the MCB to retrieve the mapping for subsequent accesses to the file. A variable of type MCB is used to store the mapping information.

The routines provided here allow the user to incrementally store some or all of the retrieval mapping for a file and to do so in any order. That is, the mapping can be inserted to the MCB structure all at once starting from the beginning and working to the end of the file, or it can be randomly scattered throughout the file.

The package identifies each contiguous run of sectors mapping VBNs and LBNs indenpendent of the order they are added to the MCB structure. For example a user can define a mapping between VBN sector 0 and LBN sector 107, and between VBN sector 2 and LBN sector 109. The mapping now contains two runs each one sector in length. Now if the user adds an additional mapping between VBN sector 1 and LBN sector 106 the MCB structure will contain only one run 3 sectors in length.

Concurrent access to the MCB structure is control by this package.

The following routines are provided by this package:

o FsRtlInitializeMcb - Initialize a new MCB structure. There should be one MCB for every opened file. Each MCB structure must be initialized before it can be used by the system.

o FsRtlUninitializeMcb - Uninitialize an MCB structure. This call is used to cleanup any anciallary structures allocated and maintained by the MCB. After being uninitialized the MCB must again be initialized before it can be used by the system.

o FsRtlAddMcbEntry - This routine adds a new range of mappings between LBNs and VBNs to the MCB structure.

o FsRtlRemoveMcbEntry - This routines removes an existing range of mappings between LBNs and VBNs from the MCB structure.

o FsRtlLookupMcbEntry - This routine returns the LBN mapped to by a VBN, and indicates, in sectors, the length of the run.

o FsRtlLookupLastMcbEntry - This routine returns the mapping for the largest VBN stored in the structure.

o FsRtlNumberOfRunsInMcb - This routine tells the caller total number of discontiguous sectors runs stored in the MCB structure.

o FsRtlGetNextMcbEntry - This routine returns the the caller the starting VBN and LBN of a given run stored in the MCB structure.

5.1 FsRtlInitializeMcb

VOID
FsRtlInitializeMcb(
IN PMCB *OpaqueMcb*,
IN POOL_TYPE *PoolType*
)

Routine Description:

This routine initializes a new Mcb structure. The caller must supply the memory for the Mcb structure. This call must precede all other calls that set/query the Mcb structure.

If pool is not available this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueMcb - Supplies a pointer to the Mcb structure to initialize.

PoolType - Supplies the pool type to use when allocating additional internal Mcb memory.

Return Value:

None.

5.2 FsRtlUninitializeMcb

VOID
FsRtlUninitializeMcb(
IN PMCB *OpaqueMcb*
)

Routine Description:

This routine uninitializes an Mcb structure. After calling this routine the input Mcb structure must be re-initialized before being used again.

Parameters:

OpaqueMcb - Supplies a pointer to the Mcb structure to uninitialize.

Return Value:

None.

5.3 FsRtlAddMcbEntry

BOOLEAN
FsRtlAddMcbEntry(
IN PMCB *OpaqueMcb*,
IN VBN *Vbn*,
IN LBN *Lbn*,
IN ULONG *SectorCount*
)

Routine Description:

This routine is used to add a new mapping of VBNs to LBNs to an existing Mcb. The information added will map

Vbn to Lbn,

Vbn+1 to Lbn+1,...

Vbn+(SectorCount-1) to Lbn+(SectorCount-1).

The mapping for the VBNs must not already exist in the Mcb. If the mapping continues a previous run, then this routine will actually coalesce them into 1 run.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

An input Lbn value of zero is illegal (i.e., the Mcb structure will never map a Vbn to a zero Lbn value).

Parameters:

OpaqueMcb - Supplies the Mcb in which to add the new mapping.

Vbn - Supplies the starting Vbn of the new mapping run to add to the Mcb.

Lbn - Supplies the starting Lbn of the new mapping run to add to the Mcb.

SectorCount - Supplies the size of the new mapping run (in sectors).

Return Value:

BOOLEAN - TRUE if the mapping was added successfully (i.e., the new Vbns did not collide with existing Vbns), and FALSE otherwise. If FALSE is returned then the Mcb is not changed.

5.4 FsRtlRemoveMcbEntry

VOID
FsRtlRemoveMcbEntry(
IN PMCB *OpaqueMcb*,
IN VBN *Vbn*,
IN ULONG *SectorCount*
)

Routine Description:

This routine removes a mapping of VBNs to LBNs from an Mcb. The mappings removed are for

Vbn,

Vbn+1, to

Vbn+(SectorCount-1).

The operation works even if the mapping for a Vbn in the specified range does not already exist in the Mcb. If the specified range of Vbn includes the last mapped Vbn in the Mcb then the Mcb mapping shrinks accordingly.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueMcb - Supplies the Mcb from which to remove the mapping.

Vbn - Supplies the starting Vbn of the mappings to remove.

SectorCount - Supplies the size of the mappings to remove (in sectors).

Return Value:

None.

5.5 FsRtlLookupMcbEntry

BOOLEAN
FsRtlLookupMcbEntry(
IN PMCB *OpaqueMcb*,
IN VBN *Vbn*,
OUT PLBN *Lbn*,
OUT PULONG *SectorCount* **OPTIONAL**

)

Routine Description:

This routine retrieves the mapping of a Vbn to an Lbn from an Mcb. It indicates if the mapping exists and the size of the run.

Parameters:

OpaqueMcb - Supplies the Mcb being examined.

Vbn - Supplies the Vbn to lookup.

Lbn - Receives the Lbn corresponding to the Vbn. A value of zero is returned if the Vbn does not have a corresponding Lbn.

SectorCount - Receives the number of sectors that map from the Vbn to contiguous Lbn values beginning with the input Vbn.

Return Value:

BOOLEAN - TRUE if the Vbn is within the range of VBNs mapped by the MCB (even if it corresponds to a hole in the mapping), and FALSE if the Vbn is beyond the range of the MCB's mapping.

For example, if an MCB has a mapping for VBNs 5 and 7 but not for 6, then a lookup on Vbn 5 or 7 will yield a non zero Lbn and a sector count of 1. A lookup for Vbn 6 will return TRUE with an Lbn value of 0, and lookup for Vbn 8 or above will return FALSE.

5.6 FsRtlLookupLastMcbEntry

BOOLEAN
FsRtlLookupLastMcbEntry(
IN PMCB *OpaqueMcb*,
OUT PVBN *Vbn*,
OUT PLBN *Lbn*
)

Routine Description:

This routine retrieves the last Vbn to Lbn mapping stored in the Mcb. It returns the mapping for the last sector or the last run in the Mcb. The results of this function is useful when extending an existing file and needing to a hint on where to try and allocate sectors on the disk.

Parameters:

OpaqueMcb - Supplies the Mcb being examined.

Vbn - Receives the last Vbn value mapped.

Lbn - Receives the Lbn corresponding to the Vbn.

Return Value:

BOOLEAN - TRUE if there is a mapping within the Mcb and FALSE otherwise (i.e., the Mcb does not contain any mapping).

5.7 FsRtlNumberOfRunsInMcb

ULONG
FsRtlNumberOfRunsInMcb(
IN PMCB *OpaqueMcb*
)

Routine Description:

This routine returns to the its caller the number of distinct runs mapped by an Mcb. Holes (i.e., Vbns that map to Lbn=0) are counted as runs. For example, an Mcb containing a mapping for only Vbns 0 and 3 will have 3 runs, one for the first mapped sector, a second for the hole covering Vbns 1 and 2, and a third for Vbn 3.

Parameters:

OpaqueMcb - Supplies the Mcb being examined.

Return Value:

ULONG - Returns the number of distinct runs mapped by the input Mcb.

5.8 FsRtlGetNextMcbEntry

BOOLEAN
FsRtlGetNextMcbEntry(
IN PMCB *OpaqueMcb*,
IN ULONG *RunIndex*,
OUT PVBN *Vbn*,
OUT PLBN *Lbn*,
OUT PULONG *SectorCount*
)

Routine Description:

This routine returns to its caller the Vbn, Lbn, and SectorCount for distinct runs mapped by an Mcb. Holes are counted as runs. For example, to construct to print out all of the runs in a a file is:

```
  for (i = 0; FsRtlGetNextMcbEntry(Mcb,i,&Vbn,&Lbn,&Count);
i++){

        // print out vbn, lbn, and count

        }
```

Parameters:

OpaqueMcb - Supplies the Mcb being examined.

RunIndex - Supplies the index of the run (zero based) to return to the caller.

Vbn - Receives the starting Vbn of the returned run, or zero if the run does not exist.

Lbn - Recieves the starting Lbn of the returned run, or zero if the run does not exist.

SectorCount - Receives the number of sectors within the returned run, or zero if the run does not exist.

Return Value:

BOOLEAN - TRUE if the specified run (i.e., RunIndex) exists in the Mcb, and FALSE otherwise. If FALSE is returned then the Vbn, Lbn, and SectorCount parameters receive zero.

6. Volume Mapped Control Block Routines

The VMCB routines provide support for maintaining a mapping between LBNs and VBNs for a virtual volume file. The volume file is all of the sectors that make up the on-disk structures. A file system uses this package to map LBNs for on-disk structure to VBNs in a volume file. This when used in conjunction with Memory Management and the Cache Manager will treat the volume file as a simple mapped file. A variable of type VMCB is used to store the mapping information and one is needed for every mounted volume.

The main idea behind this package is to allow the user to dynamically read in new disk structure sectors (e.g., FNODEs). The user assigns the new sector a VBN in the Volume file and has memory management fault the page containing the sector into memory. To do this Memory management will call back into the file system to read the page from the volume file passing in the appropriate VBN. Now the file system takes the VBN and maps it back to its LBN and does the read.

The granularity of mapping is one a per page basis. That is if a mapping for LBN 8 is added to the VMCB structure and the page size is 8 sectors then the VMCB routines will actually assign a mapping for LBNS 8 through 15, and they will be assigned to a page aligned set of VBNS. This function is needed to allow us to work efficiently with memory management. This means that some sectors in some pages might actually contain regular file data and not volume information, and so when writing the page out we must only write the sectors that are really in use by the volume file. To help with this we provide a set of routines to keep track of dirty volume file sectors. That way, when the file system is called to write a page to the volume file, it will only write the sectors that are dirty.

Concurrent access the VMCB structure is control by this package.

The functions provided in this package are as follows:

o FsRtlInitializeVmcb - Initialize a new VMCB structure.

o FsRtlUninitializeVmcb - Uninitialize an existing VMCB structure.

o FsRtlSetMaximumLbnVmcb - Sets/Resets the maximum allowed LBN for the specified VMCB structure.

o FsRtlAddVmcbMapping - This routine takes an LBN and assigns to it a VBN. If the LBN already was assigned to an VBN it simply returns the old VBN and does not do a new assignemnt.

o FsRtlRemoveVmcbMapping - This routine takes an LBN and removes its mapping from the VMCB structure.

o FsRtlVmcbVbnToLbn - This routine takes a VBN and returns the LBN it maps to.

o FsRtlVmcbLbnToVbn - This routine takes an LBN and returns the VBN its maps to.

o FsRtlSetDirtyVmcb - This routine is used to mark sectors dirty in the volume file.

o FsRtlSetCleanVmcb - This routine is used to mark sectors clean in the volume file.

o FsRtlGetDirtySectorsVmcb - This routine is used to retrieve the dirty sectors for a page in the volume file.

6.1 FsRtlInitializeVmcb

VOID
FsRtlInitializeVmcb(
IN PVMCB *OpaqueVmcb*,
IN POOL_TYPE *PoolType*,
IN ULONG *MaximumLbn*
)

Routine Description:

This routine initializes a new Vmcb Structure. The caller must supply the memory for the structure. This must precede all other calls that set/query the volume file mapping.

If pool is not available this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueVmcb - Supplies a pointer to the volume file structure to initialize.

PoolType - Supplies the pool type to use when allocating additional internal structures.

MaximumLbn - Supplies the maximum Lbn value that is valid for this volume.

Return Value:

None

6.2 FsRtlUninitializeVmcb

VOID
FsRtlUninitializeVmcb(
IN PVMCB *OpaqueVmcb*
)

Routine Description:

This routine uninitializes an existing VMCB structure. After calling this routine the input VMCB structure must be re-initialized before being used again.

Parameters:

OpaqueVmcb - Supplies a pointer to the VMCB structure to uninitialize.

Return Value:

None.

6.3 FsRtlSetMaximumLbnVmcb

```
VOID
FsRtlSetMaximumLbnVmcb(
    IN PVMCB OpaqueVmcb,
    IN ULONG MaximumLbn
    )
```

Routine Description:

This routine sets/resets the maximum allowed LBN for the specified Vmcb structure. The Vmcb structure must already have been initialized by calling FsRtlInitializeVmcb.

Parameters:

OpaqueVmcb - Supplies a pointer to the volume file structure to initialize.

MaximumLbn - Supplies the maximum Lbn value that is valid for this volume.

Return Value:

None

6.4 FsRtlAddVmcbMapping

```
BOOLEAN
FsRtlAddVmcbMapping(
    IN PVMCB OpaqueVmcb,
    IN LBN Lbn,
    IN ULONG SectorCount,
    OUT PVBN Vbn
    )
```

Routine Description:

This routine adds a new LBN to VBN mapping to the VMCB structure. When a new LBN is added to the structure it does it only on page aligned boundaries.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueVmcb - Supplies the VMCB being updated.

Lbn - Supplies the starting LBN to add to VMCB.

SectorCount - Supplies the number of Sectors in the run

Vbn - Receives the assigned VBN

Return Value:

BOOLEAN - TRUE if this is a new mapping and FALSE if the mapping for the LBN already exists. If it already exists then the sector count for this new addition must already be in the VMCB structure

6.5 FsRtlRemoveVmcbMapping

VOID
FsRtlRemoveVmcbMapping(
IN PVMCB *OpaqueVmcb***,**
IN VBN *Vbn***,**
IN ULONG *SectorCount*
)

Routine Description:

This routine removes a Vmcb mapping.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueVmcb - Supplies the Vmcb being updated.

Vbn - Supplies the VBN to remove

SectorCount - Supplies the number of sectors to remove.

Return Value:

None.

6.6 FsRtlVmcbVbnToLbn

BOOLEAN
FsRtlVmcbVbnToLbn(
IN PVMCB *OpaqueVmcb*,
IN VBN *Vbn*,
IN PLBN *Lbn*,
OUT PULONG *SectorCount* **OPTIONAL**
)

Routine Description:

This routine translates a VBN to an LBN.

Parameters:

OpaqueVmcb - Supplies the VMCB structure being queried.

Vbn - Supplies the VBN to translate from.

Lbn - Receives the LBN mapped by the input Vbn. This value is only valid if the function result is TRUE.

SectorCount - Optionally receives the number of sectors corresponding to the run.

Return Value:

BOOLEAN - TRUE if he Vbn has a valid mapping and FALSE otherwise.

6.7 FsRtlVmcbLbnToVbn

BOOLEAN
FsRtlVmcbLbnToVbn(
IN PVMCB *OpaqueVmcb*,
IN LBN *Lbn*,
OUT PVBN *Vbn*,
OUT PULONG *SectorCount* **OPTIONAL**
)

Routine Description:

This routine translates an LBN to a VBN.

Parameters:

OpaqueVmcb - Supplies the VMCB structure being queried.

Lbn - Supplies the LBN to translate from.

Vbn - Recieves the VBN mapped by the input LBN. This value is only valid if the function result is TRUE.

SectorCount - Optionally receives the number of sectors corresponding to the run.

Return Value:

BOOLEAN - TRUE if the mapping is valid and FALSE otherwise.

6.8 FsRtlSetDirtyVmcb

VOID
FsRtlSetDirtyVmcb(
IN PVMCB *OpaqueVmcb***,**
IN ULONG *LbnPageNumber***,**
IN ULONG *Mask*
)

Routine Description:

This routine sets the sectors within a page as dirty based on the input mask.

If pool is not available to store the information this routine will raise a status value indicating insufficient resources.

Parameters:

OpaqueVmcb - Supplies the Vmcb being manipulated.

LbnPageNumber - Supplies the Page Number (LBN based) of the page being modified. For example, with a page size of 8 a page number of 0 corresponds to LBN values 0 through 7, a page number of 1 corresponds to 8 through 15, and so on.

Mask - Supplies the mask of dirty sectors to set for the Page (a 1 bit means to set it dirty). For example to set LBN 9 dirty on a system with a page size of 8 the LbnPageNumber will be 1, and the mask will be 0x00000002.

Return Value:

None.

6.9 FsRtlSetCleanVmcb

VOID
FsRtlSetCleanVmcb(
IN PVMCB *OpaqueVmcb***,**
IN ULONG *LbnPageNumber*
)

Routine Description:

This routine sets all of the sectors within a page as clean. All of the sectors in a page whether they are dirty or not are set clean by this procedure.

Parameters:

OpaqueVmcb - Supplies the Vmcb being manipulated.

LbnPageNumber - Supplies the Page Number (Lbn based) of page being modified. For example, with a page size of 8 a page number of 0 corresponds to LBN values 0 through 7, a page number of 1 corresponds to 8 through 15, and so on.

Return Value:

None.

6.10 FsRtlGetDirtySectorsVmcb

ULONG
FsRtlGetDirtySectorsVmcb(
IN PVMCB *OpaqueVmcb***,**
IN ULONG *LbnPageNumber*
)

Routine Description:

This routine returns to its caller a mask of dirty sectors within a page.

Parameters:

OpaqueVmcb - Supplies the Vmcb being manipulated

LbnPageNumber - Supplies the Page Number (Lbn based) of page being modified. For example, with a page size of 8 a page number of 0 corresponds to LBN values 0 through 7, a page number of 1 corresponds to 8 through 15, and so on.

Return Value:

ULONG - Receives a mask of dirty sectors within the specified page. (a 1 bit indicates that the sector is dirty).