(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3756949号
(P3756949)
(45)発行日　平成18年3月22日(2006.3.22)　(24)登録日　平成18年1月6日(2006.1.6)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 501A
G06F 12/00 511
G06F 12/00 514A

請求項の数 29　(全 174 頁)

(21)出願番号　特願平7-529810
(86)(22)出願日　平成7年5月15日(1995.5.15)
(65)公表番号　特表平10-500508
(43)公表日　平成10年1月13日(1998.1.13)
(86)国際出願番号　PCT/US1995/006009
(87)国際公開番号　WO1995/031787
(87)国際公開日　平成7年11月23日(1995.11.23)
審査請求日　平成14年5月15日(2002.5.15)
(31)優先権主張番号　08/245,141
(32)優先日　平成6年5月13日(1994.5.13)
(33)優先権主張国　米国(US)

(73)特許権者
アップル　コンピュータ，　インコーポレイテッド
アメリカ合衆国　カリフォルニア州　９５０１４　クッパチーノ，　ワン　インフィナィト　ループ（番地なし）
(74)代理人
弁理士　大塚　康徳
(74)代理人
弁理士　松本　研一

最終頁に続く

(54)【発明の名称】ファイルシステムに記憶された情報に関する要求を処理する方法及び装置

(57)【特許請求の範囲】
【請求項１】
少なくとも１つのファイルシステムフォーマットの１つに従って構成された記憶媒体へのアクセスを要求するために適用される少なくとも１つのコーラを含むコンピュータにおいて、前記記憶媒体へのアクセス要求を処理するシステムであって、
コーラからの要求を受取り、その要求をメッセージで出力するインターフェース手段と、
前記ファイルシステムフォーマットに対応し、前記記憶媒体にアクセスするための要求を処理するフォーマットエージェント手段と、
前記フォーマットエージェント手段を特定する少なくとも１つの第１の識別子と、ディスパッチポートに割り当てられた複数のオブジェクトを特定する第２の識別子と、前記第１と第２の識別子の間をマッピングするための情報をマッピングするマッピング情報を記憶する記憶手段と、
前記インターフェース手段からの前記メッセージを受取るための前記ディスパッチポートを含み、前記マッピング情報に応答して前記フォーマットエージェント手段に前記メッセージを送出するディスパッチ手段とを有し、
前記マッピング情報は複数の制御ブロックを有し、前記複数の制御ブロックは、少なくとも１つのエージェント制御ブロックと、前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのボリューム制御ブロックとを備えることを特徴とするシステム。
【請求項２】

前記オブジェクトは、前記インターフェース手段からのメッセージを受取るためのファイルマネージャ・オブジェクトを備え、前記ディスパッチ手段は、前記メッセージに含まれる情報に基づいて、前記ファイルマネージャオブジェクトからの前記メッセージを前記フォーマットエージェント手段に送出する手段を備えることを特徴とする請求項１に記載のシステム。

【請求項３】

前記複数の制御ブロックは更に、少なくとも１つのボリューム制御ブロックと少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのパス制御ブロックを更に備えることを特徴とする請求項１に記載のシステム。

【請求項４】

前記複数の制御ブロックは更に、少なくとも１つのボリューム制御ブロックと少なくとも１つのエージェント制御ブロックに接続された少なくとも１つの反復制御ブロックを備えることを特徴とする請求項１に記載のシステム。

【請求項５】

前記ディスパッチ手段は、前記メッセージの受信と送信を同時に行うための複数のスレッドを実行する手段を備えることを特徴とする請求項１に記載のシステム。

【請求項６】

前記ディスパッチ手段は更に前記システムのためのメンテナンス動作を実行するスレッドを備えることを特徴とする請求項１に記載のシステム。

【請求項７】

ファイルマネージャを具備するコンピュータであって、

前記ファイルマネージャは、

少なくとも１つのファイルシステムフォーマットに従って構成されている記憶媒体へのアクセスを要求する少なくとも１つのコーラと、

コーラからの要求を受信し、その要求をメッセージで出力する、前記ファイルマネージャの複数のインターフェースモジュールで実行されるインターフェース手段と、

前記ファイルシステムフォーマットに対応し、前記記憶媒体にアクセスする要求を処理する、前記ファイルマネージャの少なくとも１つのフォーマットエージェントモジュールで実行されるフォーマットエージェント手段と、

前記少なくとも１つのフォーマットエージェントモジュールを特定するための少なくとも１つの第１の識別子と、ディスパッチポートに割り当てられた複数のオブジェクトを特定する第２の識別子と、前記第１と第２の識別子をマッピングするためのマッピング情報を記憶する、前記ファイルマネージャの記憶モジュールで実行される記憶手段と、

前記インターフェース手段から前記メッセージを受信するための前記ディスパッチポートを含み、前記マッピング情報に応答して前記少なくとも１つのフォーマットエージェントモジュールに前記メッセージを送信する、前記ファイルマネージャのディスパッチモジュールで実行されるディスパッチ手段とを有し、

前記マッピング情報は複数の制御ブロックを有し、少なくとも１つのエージェント制御ブロックと、前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのボリューム制御ブロックと、前記少なくとも１つのボリューム制御ブロックと少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのパス制御ブロックと、前記少なくとも１つのボリューム制御ブロックと少なくとも１つのエージェント制御ブロックに接続された反復制御ブロックとを備えることを特徴とするコンピュータ。

【請求項８】

前記オブジェクトは前記インターフェース手段からメッセージを受信するファイルマネージャオブジェクトを有し、前記ディスパッチ手段は、前記メッセージに含まれる情報に基づいて、前記ファイルマネージャオブジェクトからの前記メッセージを前記フォーマットエージェント手段に送出することを特徴とする請求項７に記載のコンピュータ。

【請求項９】

前記ディスパッチ手段は、前記メッセージの受信と送信を同時に行うための複数のスレッ

ドを実行する手段を備えることを特徴とする請求項８に記載のコンピュータ。
【請求項１０】
前記ディスパッチ手段は更に前記システムのためのメンテナンス動作を実行するスレッドを備えることを特徴とする請求項８に記載のコンピュータ。
【請求項１１】
オペレーティング・システムと、少なくとも１つの記憶媒体と、前記少なくとも１つの記憶媒体へのアクセスを要求するコーラと前記オペレーティング・システムとの間をインターフェースするインターフェース手段と、前記オペレーティング・システムと前記記憶媒体との間をインターフェースし、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットに対応している複数のエージェントモジュールと、前記少なくとも１つの記憶媒体へのアクセス要求を受取って処理するディスパッチャとを具備するコンピュータにおいて、
前記ディスパッチャーは、
少なくとも１つのエージェントモジュールを特定する少なくとも１つの第１の識別子と、要求が送られるべき複数のオブジェクトを特定する第２の識別子と、前記第１と第２の識別子の間をマッピングするためのマッピング情報を記憶する、前記ディスパッチャーの記憶モジュールで実行される記憶手段と、
前記インターフェース手段からの要求を受信し、前記マッピング情報に応答して前記少なくとも１つのエージェントモジュールに前記要求を送出する、前記ディスパッチャーの要求処理モジュールで実行される要求処理手段と、
要求の受信と送出を同時に行う複数のスレッドを実行する手段と、
メンテナンス動作を実行する手段と、
を備えることを特徴とするコンピュータ。
【請求項１２】
オペレーティング・システムと、少なくとも１つの記憶媒体と、前記少なくとも１つの記憶媒体へのアクセスを要求するコーラと前記オペレーティング・システムとの間をインターフェースするインターフェース手段と、前記オペレーティング・システムと前記記憶媒体との間をインターフェースし、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットに対応している複数のエージェントモジュールと、前記少なくとも１つの記憶媒体へのアクセス要求を受取って処理するディスパッチャとを具備するコンピュータにおいて、
前記ディスパッチャーは、
少なくとも１つのエージェントモジュールを特定する少なくとも１つの第１の識別子と、要求が送られるべき複数のオブジェクトを特定する第２の識別子と、前記第１と第２の識別子の間をマッピングするためのマッピング情報を記憶する、前記ディスパッチャーの記憶モジュールで実行される記憶手段と、
前記インターフェース手段からの要求を受信し、前記マッピング情報に応答して前記少なくとも１つのエージェントモジュールに前記要求を送出する、前記ディスパッチャーの要求処理モジュールで実行される要求処理手段とを備え、
前記マッピング情報は複数の制御ブロックを備え、前記制御ブロックは、少なくとも１つのエージェント制御ブロックと、前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのボリューム制御ブロックと、前記少なくとも１つのエージェント制御ブロックと前記少なくとも１つのボリューム制御ブロックに接続された少なくとも１つのパス制御ブロックと、前記少なくとも１つのボリューム制御ブロックと前記少なくとも１つのエージェント制御ブロックとに接続された少なくとも１つの反復制御ブロックとを備えることを特徴とするコンピュータ。
【請求項１３】
ポートと、前記ポートに割り当てられ、前記要求に対応するメッセージを受取るように適用される複数のオブジェクトを更に備え、
前記オブジェクトは前記エージェント制御ブロック、前記ボリューム制御ブロック、前記

パス制御ブロック及び前記反復制御ブロックに関連しており、そして前記ポートを通して前記オブジェクトから前記メッセージを取り出し、前記要求を処理する手段を備えることを特徴とする請求項１２に記載のコンピュータ。
【請求項１４】
前記要求処理手段は、前記メッセージが受信されるオブジェクトを特定するための識別子に基づいて、前記オブジェクトからの要求を前記少なくとも１つのエージェントモジュールに送出する手段を備えることを特徴とする請求項１３に記載のコンピュータ。
【請求項１５】
前記オブジェクトは、前記インターフェース手段からの要求を受取るためのファイルマネージャオブジェクトを備え、前記要求処理手段は、前記要求に含まれる情報に基づいて、前記ファイルマネージャオブジェクトからの要求を前記少なくとも１つのエージェントモジュールに送出する手段を備えることを特徴とする請求項１３に記載のコンピュータ。
【請求項１６】
オペレーティング・システムと、少なくとも１つの記憶媒体と、前記少なくとも１つの記憶媒体へのアクセスを要求するコーラと前記オペレーティング・システムとの間をインターフェースするインターフェース手段と、前記オペレーティング・システムと前記記憶媒体との間をインターフェースし、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットに対応している複数のエージェントモジュールとを具備するコンピュータにおいて、前記少なくとも１つの記憶媒体へのアクセス要求を扱って処理する方法であって、
前記複数のエージェントモジュールを特定するための少なくとも１つの第１の識別子と、要求が送出される複数のオブジェクトを特定するための第２の識別子と、前記第１と第２の識別子をマッピングするマッピング情報とをメモリに記憶する工程と、
要求処理モジュールで、前記インターフェース手段からの要求を受信する工程と、
前記メモリの前記マッピング情報に応答して前記少なくとも１つのエージェントモジュールに前記要求を送出する工程と、
要求の受信と送出を同時に行う複数のスレッドを実行する工程と、
メンテナンス動作を実行する工程と、
を備えることを特徴とする方法。
【請求項１７】
オペレーティング・システムと、少なくとも１つの記憶媒体と、前記少なくとも１つの記憶媒体へのアクセスを要求するコーラと前記オペレーティング・システムとの間をインターフェースするインターフェース手段と、前記オペレーティング・システムと前記記憶媒体との間をインターフェースし、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットに対応している複数のエージェントモジュールとを具備するコンピュータにおいて、前記少なくとも１つの記憶媒体へのアクセス要求を扱って処理する方法であって、
前記複数のエージェントモジュールを特定するための少なくとも１つの第１の識別子と、要求が送出される複数のオブジェクトを特定するための第２の識別子と、前記第１と第２の識別子をマッピングするマッピング情報とをメモリに記憶する工程と、
要求処理モジュールで、前記インターフェース手段からの要求を受信する工程と、
前記メモリの前記マッピング情報に応答して前記少なくとも１つのエージェントモジュールに前記要求を送出する工程とを有し、
前記マッピング情報を記憶する工程は、
少なくとも１つのエージェント制御ブロックを記憶する工程と、
前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのボリューム制御ブロックを記憶する工程と、
前記少なくとも１つのエージェント制御ブロックと前記少なくとも１つのボリューム制御ブロックとに接続された少なくとも１つのパス制御ブロックを記憶する工程と、
前記少なくとも１つのボリューム制御ブロックと前記少なくとも１つのエージェント制御

ブロックとに接続された少なくとも１つの反復制御ブロックを記憶する工程とを備えることを特徴とする方法。
【請求項１８】
前記要求処理モジュールは、
ポートと、前記エージェント制御ブロック、前記ボリューム制御ブロック、前記パス制御ブロック及び前記反復制御ブロックに関連する前記複数のオブジェクトを備え、
前記複数のオブジェクトは前記ポートに割り当てられ、前記要求に対応するメッセージを受取るように適用され、
前記方法は更に、前記ポートを通して前記オブジェクトからのメッセージを取り出して前記要求を処理する工程を備えることを特徴とする請求項１６に記載の方法。
【請求項１９】
前記要求を処理する工程は、前記メッセージが受信されるオブジェクトを特定するための識別子に基づいて、前記オブジェクトからの要求を前記少なくとも１つのエージェントモジュールに送出する送出工程を備えることを特徴とする請求項１８に記載の方法。
【請求項２０】
前記送出工程は、
前記要求がボリュームに関連したオブジェクトに送られる場合は、ボリュームオブジェクトから取り出された情報に基づいてボリューム制御ブロックを取り出し、
前記ボリューム制御ブロックから取り出された情報に基づいてエージェント制御ブロックを取り出し、
前記エージェント制御ブロックから取り出された情報に基づいて前記エージェントモジュールの１つに前記要求を送出し、
前記エージェントモジュールで受信された応答に従って前記インターフェース手段に応答することを特徴とする請求項１９に記載の方法。
【請求項２１】
前記送出工程は、
前記要求がパスに関連したパスオブジェクトに送られる場合は、前記パスオブジェクトから取り出した情報に基づいてボリューム制御ブロックを取り出し、
前記パス制御ブロックから取り出した情報に基づいてエージェント制御ブロックを取り出し、
前記エージェント制御ブロックから取り出した情報に基づいて前記エージェントモジュールの１つに前記要求を送出し、そして
前記エージェントモジュールから受信した応答に従って前記インターフェース手段に応答することを特徴とする請求項１９に記載の方法。
【請求項２２】
前記送出工程は、
前記要求が反復オブジェクトに関連するオブジェクトに送られる場合は、前記反復オブジェクトから受取った情報に基づいてボリューム制御ブロックを取り出し、
前記反復制御ブロックから受取った情報に基づいてエージェント制御ブロックを取り出し、
前記エージェント制御ブロックから受取った情報に基づいて前記エージェントモジュールの１つに前記要求を送出し、
前記反復モジュールが別のボリュームに移動される必要がある場合、前記反復制御ブロックを新たなボリューム制御ブロックに移動し、前記エージェント制御ブロックから受取った情報に基づいてエージェントモジュールの１つに要求を送出し、
前記エージェントモジュールから受信した応答に従って前記インターフェース手段に応答することを特徴とする請求項１９に記載の方法。
【請求項２３】
前記オブジェクトは、前記インターフェース手段からの要求を受取るファイルマネージャオブジェクトを備え、

前記方法は更に、前記要求に含まれる情報に基づいて、前記ファイルマネージャオブジェクトから前記少なくとも１つのエージェントモジュールに前記要求を送出する工程を備えることを特徴とする請求項１８に記載の方法。
【請求項２４】
前記要求に含まれる情報に基づく送出工程は、
前記要求がマルチボリューム要求であれば、前記要求により影響を受けるボリュームを決定し、
ファイルマネージャオブジェクトに送られたメッセージから取り出された情報に基づいて、前記ボリュームのボリューム制御ブロックを取り出し、
前記ボリューム制御ブロックから取り出した情報に基づいてエージェント制御ブロックを取り出し、
前記エージェント制御ブロックに関連したエージェントモジュールが同じであれば、前記エージェント制御ブロックから取り出された情報に基づいて前記エージェントモジュールの１つに前記要求を送出し、
前記エージェントモジュールが同じでない時は、前記ボリュームのそれぞれに対する要求を連続して作成して送出し、
前記エージェント制御ブロックから取り出した情報に基づいて、前記エージェントモジュールの１つに前記要求を送出し、
前記エージェントモジュールから受取った応答に従って前記インターフェース手段に応答することを特徴とする請求項２３に記載の方法。
【請求項２５】
前記要求に含まれる情報に基づく送出工程は、
前記要求がボリュームの実装要求の場合は、所定のメッセージを作成し、
前記エージェントモジュールの１つが当該ボリュームを実装できるまで、連続して前記エージェントモジュールに前記所定のメッセージを送出し、
前記エージェント制御ブロックから取り出した情報に基づいて、前記エージェントモジュールの１つに前記要求を送出し、
前記エージェントモジュールの１つから受信した応答に従って前記インターフェース手段に応答することを特徴とする請求項２３に記載の方法。
【請求項２６】
前記要求に含まれる情報に基づく送出工程は、
前記要求がパス名決定要求であれば、前記メッセージの一部からボリュームを決定し、
所定のボリュームに関連するボリューム制御ブロックから取り出した情報からエージェント制御ブロックを取り出し、
前記エージェント制御ブロックから取り出した情報に基づいて、前記エージェントモジュールの１つに前記要求を送出し、
前記エージェントモジュールの１つから受信した応答に従って前記インターフェース手段に応答することを特徴とする請求項２３に記載の方法。
【請求項２７】
少なくとも１つのファイルシステムフォーマットの１つに従って構成された記憶媒体へのアクセスを要求するために適用される少なくとも１つのコーラを含むコンピュータにおいて、前記記憶媒体へのアクセス要求を処理する方法であって、
少なくとも１つのフォーマットエージェント手段を特定する少なくとも１つの第１の識別子と、ディスパッチポートに割り当てられた複数のオブジェクトを特定する第２の識別子と、前記第１と第２の識別子の間をマッピングするためのマッピング情報を記憶する工程と、
コーラからの要求をインターフェース手段で受信する工程と、
前記ディスパッチポートを含むディスパッチ手段において、前記インターフェース手段からの要求を受信する工程と、
前記記憶手段のマッピング情報に応答して、前記フォーマットエージェント手段の１つに

前記要求を送出する工程と、
ディスパッチ手段の前記インターフェース手段からの要求を受信する工程と、
前記ファイルシステムフォーマットに対応する前記フォーマットエージェント手段の１つを介して前記記憶媒体をアクセスする要求を処理する工程とを備え、
前記マッピング情報を記憶する工程は、
少なくとも１つのエージェント制御ブロックを記憶し、前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのボリューム制御ブロックを記憶し、前記少なくとも１つのボリューム制御ブロックと前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つのパス制御ブロックを記憶し、前記少なくとも１つのボリューム制御ブロックと前記少なくとも１つのエージェント制御ブロックに接続された少なくとも１つの反復制御ブロックを記憶することを特徴とする方法。
【請求項２８】
要求の受信と送出を同時に行う複数のスレッドを実行する工程を更に有することを特徴とする請求項２７に記載の方法。
【請求項２９】
前記システムのメンテナンス動作を実行する工程を更に備えることを特徴とする請求項２７に記載の方法。
【発明の詳細な説明】
本願明細書は著作権の保護に関する事項を含んでおり、この著作権所有者は出願書類の開示或はある部分を、特許庁に記録されているままに再生されることに異議を唱えなるものではないが、一方、どのような物にも著作権が及ぶと考えている。
発明の背景
本願発明は、オペレーティング・システム（ＯＳ）の一部としてファイルシステムに記憶された情報に関する処理要求を行う方法及び装置に関するもので、特に、マルチファイルシステムに対する要求を処理する方法及び装置に関するものである。
コンピュータのＯＳは、幾つかの異なる動作モジュールを含み、その様なモジュールの１つにファイルマネージャがある。このファイルマネージャはハードディスクや着脱可能なディスクのようなデータ記憶媒体上に位置しているデータの編成、読出し及び書き込み等を管理している。各記憶媒体は、記憶媒体のデータをアクセスするための、それ自身に関連したフォーマットを有しており、その様なフォーマットの一例としてHFS, MS-DOS FAT, ProDOS, High Sierra, ISO9660, UNIX等がある。フォーマットは物理的なディスクフォーマット、ネットワークサーバーアクセス・プロトコル等を包含している。最近までは、コンピュータはたった１つのフォーマットの下で構成された記憶媒体にアクセスすることができた。しかし、コンピュータが、マルチフォーマットを用いてデータにアクセスできるようにする傾向に有り、マルチフォーマットを用いて記憶媒体にアクセスできる少なくとも２つのシステムが現在利用可能である。
欧州特許出願０４１５３４６（Willman）は、媒体を識別できるファイルシステムを自動的かつ動的に実装できるマイクロソフトＯＳ／２のＯＳで使用できるシステムを開示している。このWillmanのコンピュータシステムでは、１つ或はそれ以上のデータ記憶装置と複数のファイルシステム・ドライバが、ファイルシステムがリンクされて構成されているデフォルト・ファイルシステムを含んで設けられている。各ファイルシステム・ドライバは、特別なフォーマットのファイルシステムをアクセスできる。コンピュータシステムは連続して全ての周辺機器をモニタし、周辺の記憶装置における媒体のいかなる変化をも検知している。データ記憶装置において媒体が変更されると、或いはコンピュータシステムがデータ記憶装置に最初にアクセスされると、ファイルシステムドライバのリストで特定された最初のファイルシステムドライバがロードされ、ボリューム識別子が媒体から読み出される。この時、ボリューム識別子の位置は、ロードされたファイルシステムドライバによって特定される。そして媒体から読み出されたボリューム識別子はファイルシステムドライバに関連する識別子と比較されて、識別子が一致すればそのファイルシステムドライバがマウントされる。一致しなければ、リンクされたファイルシステムドライバのリス

トで次のファイルシステムドライバがロードされる。リンクされたファイルシステムドライバのリストの各ファイルシステムドライバがテストされるまで、或いは一致が見つかるまで、この過程が繰り返される。一致が見つからなかった場合は、デフォルトファイルシステムがマウントされる。
このシステムは、柔軟性に欠けている。特に、この識別子はシステムに用いることができる全てのボリュームフォーマット用に共通して知られた位置になければならない。その代わりに、ボリュームフォーマットにおける柔軟性増加を提供するため、各ファイルシステムドライバにそのボリュームを識別する独自の方法をとらせるのが望ましい。
マルチファイルシステムにアクセスする他のシステムにマイクロソフト（TM）ウィンドウズ（TM）ＮＴオペレーティング・システムがある。このシステムでは、各ファイルシステムドライバが保持する複数のオブジェクトにメッセージが送られ、そのオブジェクトはその要求の主題に対応している。従って、もし特定の種類のメッセージの処理に変更が必要な場合、その変更は各ファイルシステムドライバごとに行われなければならない。
その他のファイルマネージャーとして、例えばアップルコンピュータ株式会社によって製造されたマッキントッシュ（TM）ブランドのコンピュータ上で用いられるシステム７ファイルマネージャが知られている。このシステム７では要求を受け取ると、そのファイルマネージャは、その要求がアップルマッキントッシュ（TM）が用いるＨＦＳ（階層ファイルシステム）ボリュームへアクセスするための要求とみなす。もしその記憶媒体がＨＦＳでなければエラーが発生し、ファイルマネージャは要求されたアクセスの実行を試みるコードにその要求を出力する。
様々なプロセッサ上で実行することができ、多くのオペレーティング・システムで使用可能な、並行かつ非同期デバイスマネージャーを利用できる、ポータブルバージョンのファイルマネージャを提供することが望ましい。さらに、従来のファイルマネージャーのアプリケーションプログラマ・インターフェース（ＡＰＩ）に下位の互換性（downward compatibility）を提供するファイルマネージャーを提供することが望ましい。説明として、アプリケーションプログラマ・インターフェース（ＡＰＩ）は、第三者がそのオペレーティング・システムと通信ができる仕様（specification）を提供するものとして定義される。例えば、本発明の背景において、ＡＰＩは第三者がファイルマネージャーと通信できる仕様を提供するものと定義される。
従来のオペレーティング・システムは、特定のタスクを行う別々のモジュールを含まない、一体型コード構造（monolithic code structure）を用いて作られた。問題が明るみに出たのは、変更するごとに全システムを再構築しなければならず、システムの改良が困難になったからである。さらに、専用（特権：privileged）モードで実行される全てのコードも問題となる。これは、システムの構成要素のいずれかがエラーの原因となる構成要素と衝突する恐れがあったためである。
上記に対する解決策は、システムからできるだけ多くの機能性を、より特権性（privileged）の低いモードで実行される別々のサービス（例えば、ファイルマネージャー、入力・出力マネージャー、スクリーンマネージャ等）に分けて、コードの中心部により最も特権（privileged）化された機能を実行させることであった。その中心部はサービスの全くの最小限の機能を提供していたので、これを修正しなければならないことは稀であった。他のサービスの各々は独立して構築されたので、これらの修正はより容易であった。こうしてシステムコードのほとんどが縮小した特権（privilege）の下で実行されたので、システムはより安定した。また、システムがより容易に修正されたため、にわかに新しいものの導入が容易となり、多くの新しい性能が加えられた。このため、現代のオペレーティング・システムは、代表的にはこの中心部構造に従って設計されている。
一体型（monolithic）オペレーティング・システム用の従来のＡＰＩは、全体的な状況（global state）に大きく左右される。そこで、このような環境において、全ファイルマネージャーデータが個々に保護されたアドレススペースにあり、プログラマーが確実なシステムに向けて将来移行できることを想定したファイルマネージャーのため、新たなＡＰＩを開発することが望ましい。

発明の要約

本発明は、セントラルディスパッチャ、要求源として機能するマルチインターフェース、マルチローダブル・フォーマット・エージェント・モジュール（MULTIPLE LOADABLE FORMAT AGENT MODULE）またはファイルシステムドライバ、ストア（記憶装置（メモリ））、共用ライブラリを用いるファイルシステムに記憶された情報に関する要求を処理するシステムに関する。

本発明に係るシステムは、ファイルシステムの実行及び与えられたパブリックインターフェースから共に独立するように、ファイルシステムに記憶された情報に関する要求を表すことができる一組の共通メッセージ構造を用いる。これらのメッセージ構造は、その機構の構成要素間における通信の共通原語として用いられる。

複数のインターフェースモジュール（数はいくつでも構わない）は、システムの他の構成要素に、異なるパブリックインターフェースを提供する。従って特定の環境用に書かれたソフトウェアが、その環境において標準であるインターフェースを用いることができる。コーリングプログラムはインターフェースモジュールに様々な記憶媒体へのアクセス要求（リクエスト）を送る。これらのインターフェースモジュールは、そういった要求を処理させるためディスパッチャへ送る。

ディパッチャはこの要求を受けると、ストアからリトリーブした（取り出した）要求についての情報を基に、複数のフォーマットエージェントの１つにその要求を転送する。各フォーマットエージェントは、特定のファイルシステムフォーマットを用いて記憶媒体にアクセスするよう適応している。フォーマットエージェントは、記憶媒体と通信しながらディスパッチャから要求を受け取り、その要求を処理する。その要求の処理が終了すると、フォーマットエージェントは、元のコーラ(Caller：要求発生元)に返送されるべき何らかの情報をディスパッチャに応答する。その要求を実行するのに関わるフォーマットエージェント全てが終了しディスパッチャに応答したら、ディスパッチャは要求を伝えるのに用いたインターフェースモジュールに応答し、元のコーラに返送されるべき何らかの情報を送る。インターフェースモジュールは、その情報をインターフェースに適当な形式に翻訳し、そのコーラに返送する。

フォーマットエージェントの内部機構の一部は、異なるモジュール間でも非常によく似ているため、本発明に係るシステムは、フォーマットエージェント全てが用いることができる共通した共用ライブラリを提供する。このライブラリは、実行モジュール間で共用できる共通機構の実行を含むので、これらのモジュールの展開が容易となる。

本発明の一実施例によれば、少なくとも１つのコーラが記憶媒体へのアクセスが要求できるよう適応しているコンピュータにおいて、前記記憶媒体は少なくとも１つのファイルシステムフォーマットの一つに従って構成されており、システムは前記記憶媒体へのアクセス要求を処理するために提供される。前記システムは、コーラから要求を受け取り、受け取った要求の主題の少なくとも一部に基いて、前記要求を送信するための適当な宛先(destination)を複数の宛先の中から決定し、前記要求を前記適当な宛先に送信するインターフェース手段、前記記憶媒体にアクセスする要求を処理するための、前記ファイルシステムフォーマットに対応するフォーマットエージェント手段、前記フォーマットエージェント手段を識別する少なくとも１つの第１の識別子と、前記複数の宛先を識別する第２の識別子と、第２の識別子と第１の識別子間でマッピングを行うマッピング手段とを記憶する記憶手段、そして前記要求を前記インターフェース手段から受け取り、前記マッピング手段に応じて前記要求をフォーマットエージェント手段に送出するディスパッチ手段とを備える。

本発明の他の実施例によれば、記憶媒体へのアクセスを要求する少なくとも１つのコーラを持つファイルマネージャーと、少なくとも１つのファイルシステムフォーマットの１つに従って構成される前記記憶媒体と、コーラから要求を受け取り、受け取った要求の主題の少なくとも一部分に基いて、前記要求を送信するための適当な宛先を複数の宛先から決定し、前記要求を前記適当な宛先に送信する複数のインターフェースモジュールと、前記記憶媒体にアクセスする要求を処理するための、前記ファイルシステムフォーマットに対

応する少なくとも一つのフォーマットエージェントモジュール、前記少なくとも１つのフォーマットエージェントモジュールを識別する少なくとも１つの第１の識別子と、前記複数の宛先を識別する第２の識別子と、第２の識別子と第１の識別子間でマッピングを行うマッピング手段とを記憶するストア、そして前記インターフェースモジュールから前記要求を受け取り、前記マッピング手段に応じて前記要求を前記少なくとも１つのフォーマットエージェントモジュールに転送するディスパッチモジュールとが提供される。

本発明の他の実施例によれば、コンピュータにおいて、少なくとも１つの記憶媒体へのアクセス要求を処理するディスパッチャを備え、前記コンピュータは、オペレーティングシステムと、前記要求を送信したコーラと前記オペレーティング・システム間をインターフェースするインターフェース手段と、そして前記オペレーティング・システムと前記記憶媒体間をインターフェースする複数のエージェントモジュールとを備え、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットの１つに対応している。前記ディスパッチャは、前記少なくとも１つのエージェントモジュールを識別する少なくとも１つの第１の識別子と、要求を送信することができる複数のオブジェクトを識別する第２の識別子と、第２の識別子と第１の識別子間でマッピングを行うマッピング手段とを記憶するストアと、そして前記インターフェースモジュールから前記要求を受け取り、前記マッピング手段に応じて前記要求を前記少なくとも１つのエージェントモジュールに転送する要求処理モジュールとを備える。

本発明の他の実施例によれば、コンピュータにおいて、少なくとも１つの記憶媒体へのアクセス要求を処理する方法を提供するものであり、前記コンピュータはオペレーティングシステムと、前記要求を送信したコーラと前記オペレーティング・システム間をインターフェースするインターフェース手段と、前記オペレーティング・システムと前記記憶媒体間をインターフェースする複数のエージェントモジュールを備え、各エージェントモジュールは少なくとも１つのファイルシステムフォーマットの１つに対応している。前記方法は、前記複数のエージェントモジュールを識別する少なくとも１つの第１の識別子と、要求を送信することができる複数のオブジェクトを識別する第２の識別子と、第２の識別子と第１の識別子間でマッピングを行うためのマッピング情報とをストア（メモリ）に記憶する記憶行程と、そして前記インターフェースモジュールから前記要求を受け取る受信行程と、前記マッピング情報に応じて前記要求を前記少なくとも１つのエージェントモジュールに転送する転送行程とを含む。

本発明の他の実施例によれば、記憶媒体へのアクセスを要求する少なくとも一人のコーラを持つコンピュータにおいて、各記憶媒体は少なくとも１つのファイルシステムフォーマットの１つに従って構成されており、前記記憶媒体へのアクセス要求を処理する方法が提供される。前記方法は、少なくとも１つのフォーマットエージェント手段を識別する少なくとも１つの第１の識別子と、複数の宛先を識別する第２の識別子と、第２の識別子と第１の識別子間でマッピングを行うためのマッピング情報とを記憶手段に記憶する記憶行程と、インターフェース手段でコーラからの要求を受け取る受信行程と、要求の主題の少なくとも一部分に基づいて、前記要求を送信するための適当な宛先を複数の宛先から決定する決定行程と、前記要求を前記適当な宛先に送信する送信行程と、前記要求をインターフェース手段からディスパッチ手段で受け取る行程と、前記要求を、記憶手段のマッピング情報に応じて適当なフォーマットエージェント手段の１つに転送する行程と、前記ファイルシステムフォーマットに対応する前記フォーマットエージェント手段の１つを介して、前記記憶媒体へのアクセス要求を処理する行程とを含む。

さらに本発明の他の目的と特徴と利点は、以下の実施例の詳細な説明とそれに関連付けられた添付図面から当業者には明らかになるであろう。

【図面の簡単な説明】

本願の発明を、装置の好適な実施例と添付図面を参照してさらに詳細に説明する。

図１は本発明が実行できる典型的なコンピューターシステムを示すブロック図である。

図２は本発明によるファイルシステムに記憶される情報に関する要求を処理するファイルマネージャの構造全体を示すブロック図である。

図３ａは本発明の一実施例に係るストア（メモリ）の構成を示すブロック図である。
図３ｂと図３ｃはそれぞれ本発明の一実施例に係る制御ブロックヘッダとエージェント制御ブロックを表す図である。
図４は本発明の一実施例に係る共用ライブラリを表すブロック図である。
図５ａは本発明の一実施例に係るディスパッチャを示すブロック図である。
図５ｂは本発明の一実施例に係る要求タスク状況構造（request task state structrue）を表す図である。
図６は本発明の一実施例に基づいて、インターフェースがコーラ(caller)からの要求を処理する過程を示すフローチャートである。
図７ａ乃至図７ｆは本発明の一実施例に基づいて、ストアを形成する制御ブロックの存在を維持する、フォーマットエージェントによる処理を示すフローチャートである。
図８ａ乃至図８ｍは本発明の一実施例に係るディスパッチャ処理を示すフローチャートである。
詳細の説明
本説明において、ユーザとはある特定のソフトウエアの一部を使用する者、そして、コーラ(Caller：要求発生元)とは、本発明に於けるファイルマネージャを呼び出すソフトウエアの一部である。
本発明の目的は、セントラルディスパッチャと、コーラとディスパッチャ間のインターフェースとして機能するマルチプルインターフェースと、ファイルシステムフォーマットに従って大系づけられた記憶媒体にアクセスするためのマルチロード可能なフォーマットエージェントと、フォーマットエージェントとファイルシステムに関連する情報を記憶するためのストア(store：記憶装置)と、共通な共用ライブラリとを使用するファイルシステム内に保存された情報に関するリクエスト（要求）を取り扱うためのファイルマネージャである。このファイルマネージャは、ファイルシステムの内部実行(internal implementation)に何ら影響を与えることなくマルチプルパブリック・インターフェースを供給できるように、ファイルシステムへの外部インターフェースを内部実行から分離するための手段を提供する。また、外部インターフェースに明らかな影響を与えることなく複合実行（マルチ処理）（multiple implementation）がサポートされるように、個別のファイルシステムの実行をパブリックインターフェースから分離する手段を提供する。更に、個別のファイルシステムを実行するために必要な労力を低減する手段を提供する。
一実施例によると、ファイルシステム内のすべてのデータは、属性(attribute)またはフォーク(forks)の２種類の構成の内、どちらか一方の構成で保存される。
一実施例によると、属性とは、ボリューム、ディレクトリ、ファイル、フォーク、またはファイルクラス(file class)と言った特定のシステムユニットに関連する記述的なデータである。このような記述的情報のすべては、実際にそれがどのように保存されているかに関わらず、属性としてアクセスされる。従って、名前、特権(privilege)、ファイルのタイプおよびクリエータ(creator)、フォーク長のバイト数などは、すべて属性としてアクセスすることができる。ある属性の解釈が定義されてもよく、また個別のファイルシステムは、どの属性がどの種類のオブジェクトに関連づけることができるかについて制限を加えてもよい。
属性は、例えば、作者、文書のタイプ、ユニットのサイズ、ユニットに関連するキーワード等、ユニットにアクセスする他のソフトウエアにとって都合が良いと思われる、ユニットについての情報を提供する。属性は、ユニットをオープンしたりアクセスしたりすることなく、ソフトウエアに提供される。一実施例によると、属性は、必要に応じてプログラマーが定義することができる。特に、属性は、その属性を創造した（または、属性が添付されたものを所有する）プログラム以外のソフトウエアによってアクセスされる必要性のある、小さなデータアイテムを保存するのに使用される。属性は、そのファイルシステムにおけるいかなる固有の構成(persistent structure)、例えば、ボリューム、ファイルクラス、ディレクトリ、ファイル、そして、フォーク等、にも関連づけることができる。そして、それぞれの構造に関連する非常に多くの属性が存在する可能性があると仮定される

。
特定の属性は、識別子によって識別される。一実施例では、これはOSTypeとして知られる４文字コードでもよい。また、１組の識別子を使用しても良い。第１の識別子は、属性を所有するサービスを識別する。一実施例によると、例えば、「fmgr」はファイルマネージャ属性に使用され、「fndr」はファインダ属性に使用され、「hfs」はＨＦＳボリューム特定の属性に、アプリケーション記号はアプリケーションが生成した属性として使用される。第２の識別子は、同サービスが所有する異なる属性を見分けるために使用される。属性の値は、固定最大サイズ（ＡＰＩは勧めるが、小さい値を必要としない）を持たない、解釈されない１以上のバイト数（ゼロ長の属性をサポートする必要は無い）の列である。属性値は、関連するタイプを持たない。属性の名前を知る者はだれでも、値のフォーマットを知っていると仮定される。与えられた属性が２以上のフォーマットで値を持つことができる場合、開発者は、これを取り扱うための独自のプロトコルの開発を自由に行うことができる。
一実施例によると、属性は、システムによって保持されているファイルマネージャ内のすべての情報へのアクセスを提供するために使用される。オペレレーティングシステムが、そのファイルシステムにおける固有な構成について知っている全て、フォークデータ以外は、属性を介してアクセスすることができる。これは、他の手段を介してもアクセスすることができるファイルの名前や、フォークの長さなどの情報を含む。他の実施例では、サービス名「fork」を定義し、このフォーク名を属性名の第２のOSTypeとして利用することにより、フォークデータにもアクセスできるようにする。
フォーク(forks)とは、順序よく並べられたバイトの列で、入力および出力が実行される基本的なデータ単位である。フォークは、そのフォークを創造したソフトウエアに主にアクセスされる、大きなデータアイテムを保存するために用いられる。フォークはファイルにのみ関連づけられ、それぞれのファイルについて相対的に少ない数のフォークが存在すると仮定される。一実施例によると、フォークのデータは０バイトから２の63乗のバイト長であり、間を空けてメモリ内に割当てられている。
一実施例によると、ファイル内のフォークは、１つのOSTypeによって識別することができる。フォーク内のデータは、間を空けてアロケートしてもよい。フォークのフォーマットを定義するそれぞれのフォークに関連するOSTypeがある（標準ＨＦＳボリュームでは、これはリソースフォークとして「rsrc」にデフォルトし、データフォークとしてファイルのタイプにデフォルトする）。同時アクセスに対するすべての特権の制限は、フォークレベルで制御される。すべてのフォークは、ＩＤとそのデータを持つ。
一実施例によると、アプリケーションファイルは１つのフォーク、すなわち、ダイアログボックスや、メニューなどの構成を含むリソースフォークを有しても良い。データファイルは、テキストデータを含むデータフォークのみを有することができる。本発明の一実施例によると、２つ以上のフォークを定義することができる。他のフォークは、そのファイルを作成したソフトウエアが使用できる情報を含んでも良い。例えば、言語処理プログラムにより、タイプされたとおりに文書を第２言語に翻訳し、文書の２つのバージョンを２つの異なるフォークに保存することができる。フォーク自身の存在をユーザは知らないかもしれないが、言語処理プログラムは、文書のどのバージョンをリトリーブするためにどのフォークにアクセスするかを知っている。マルチプルフォークの他の使用については、本発明の開示を理解することにより、通常の技術を有する者により変更可能である。
一実施例では、フォーク内のデータは、オープンパスを介してアクセスされる。パスはある特定のフォークのために作り出されたランタイム・コンストラクト(runtime constructs)であり、特定のアクセス方法を使用するように定義される。与えられたフォークについて１以上のパスを作ることができ、異なるパスは異なるアクセス方法を使用することができる。しかし、先に作り出されたパスは、後で作り出されたパスを利用してできることについて制限を加えることができる。
一実施例では、ファイルとは、ファイルシステム内でコピーし、移動し、削除し、微細な改名をすることができるフォークと属性の集合である。全てのファイルは名前（文字列）

と、ペアレントディレクトリとしての役割を持つ１つのディレクトリと、そのデータが保存されている特定のボリュームを持つ。ファイルは、ボリューム内で唯一である値によって特定することができる。
一実施例によると、ファイルは任意の数のフォークを持ち、ファイル内のそれぞれのフォークは４バイトコードで見分けられる。すべてのファイルは、名前、タイプコード、クリエータコード、および一連の特権を持つ。
一実施例によると、ファイルは、ユニットとして容易に取り扱うことのできる情報の最大集合体を表す。ファイルの目的は、ユーザーの経験が情報の微細な集まりの概念を構築できる基礎を提供することである。プログラム内のファイルマネージャＡＰＩを介してより細かい情報にアクセスし、扱うことができるので、その目的は、ユーザーやコーラがファイルを総括的に取り扱うことのできるユニットとして見ることである。
一実施例によると、ファイルクラスは、同タイプやクリエータまたは両方を共用するボリューム内のファイルの論理グループである。あるファイルシステムの動作は、ファイルクラスに関連している。
一実施例によると、ディレクトリとは、ファイルとディレクトリの集合である。すべてのディレクトリは、名前（文字列）と、そのペアレントディレクトリとしての役割を持つ１つのディレクトリと、そのデータが保存されている特定のボリュームとを持つ。ディレクトリは他のディレクトリを含むことができるので、ボリューム上のファイルを含むディレクトリの階層が存在することがある。ディレクトリは、ボリューム中で唯一である値によって識別することもできる。
一実施例によると、ディレクトリの目的は、ファイルを探し出して使用するためのコンテクストを提供することである。ファイルは、関連ファイルをグループにまとめ、ファイル名を解釈するための多少のコンテクストを提供するために用いることのできる、ディレクトリのコンテクストで表現することができる。
一実施例によると、ファイルマネージャによって管理され、特殊な特徴を持った一つのスペシャルディレクトリがある。ファイルシステムのどれかのエンティティから始めて（どのファイル、ディレクトリ、またはボリュームでも良い）、そのエンティティのペアレントディレクトリに移り、更にそのペアレントディレクトリのペアレントディレクトリに移るというようにすると、最終的にメタルートディレクトリ(meta-root directory)に達する。このメタルートディレクトリはペアレントディレクトリを持たないが、非直接的には自分以外のファイルシステム内のもののペアレントである。メタルートはボリューム上には存在しないが、ファイルマネージャによって作り出され、ファイルマネージャ自身によって維持される。一実施例によると、ファイルマネージャがボリュームをマウントする度に（以下参照）、そのボリュームのルートディレクトリがメタルートをそのペアレントとして持つようにマウントする。メタルートは実際にはボリューム上にストアされないので、ファイルマネージャは、コーラがメタルート内またはそれを用いてできることについて厳しく制限している。制限された（拡張できない）一連の属性のみをメタルート上で扱うことができ、ファイルとディレクトリはメタルート上で作成することはできない。
ボリュームも、ファイルとディレクトリの集合である。すべてのボリュームは名前（文字列）を持っており、いくつかの形式の記憶媒体（ローカルなものまたは遠隔にあるもの）に関連している。ボリュームはファイルマネージャが使用できるユニット（単位）として機能する。一実施例によると、ボリュームは、ユニットとして利用可能にしたり利用不可能にされたファイルおよびディレクトリの最小集合体である。１つの脱着可能なディスクや、ディスクドライブのパーティション、またはディスクのセットなどのユニットは、ボリュームとして定義することができる。なお、ボリュームの内容をコンピュータが利用することができるが、その時点のユーザがデータに実際にアクセスすることを許可されない場合もある。
一実施例によると、すべてのボリュームは、他のディレクトリやファイルを含むことができ、ルートディレクトリとして知られる、少なくとも１つのディレクトリを持つ。そのボリューム中のすべてのエンティティについて、もしペアレントディレクトリと、そのペア

レントディレクトリのペアレントディレクトリと言ったようにディレクトリを見つけることができるとすれば、そのボリュームのルートディレクトリに最終的に必ず到達することができる。現時点では、ボリュームがファイルマネージャによってマウントされたときはいつでも、そのボリュームのルートのペアレントディレクトリをメタルート(meta-root0として設定する。

一実施例によると、ファイルマネージャのすべてのリクエストは、そのシステムの何人かのユーザまたはユーザのグループに代わって行われる。ファイルマネージャに関する限り、このようなユーザまたはユーザのグループは、それに固有な値（アイデンティティとして知られている）によって識別される。

一実施例によると、アクセス方法とは、フォーク中のデータがパスを介してアクセスすることのできるモデルのことである。一実施例によると、２つのアクセス方法が定義される。１つはフォーク上のさまざまな量のデータにランダムアクセスすることができるストリームであり、もう１つは、フォーク内に構成された記録の挿入、削除、および検索をすることができるBTreeである。

図１は、本発明に係るソフトウエアを実行することができる模範的なコンピュータを示すブロック図である。コンピュータ１０は、処理部（ＣＰＵ）１２、外部記憶装置１４、ディスク記憶装置１６、入出力（Ｉ／Ｏ）制御部１８、少なくとも一つの入力装置２０、少なくとも一つの出力装置２２、ランダムアクセスメモリ（ＲＡＭ）２４、および上記それぞれの構成要素に接続する共通システムバス２６を持つ。コンピュータ１０は、バス２６およびＩ／Ｏ制御部１８を介して入力および出力装置２０および２２に接続する。更に、コンピュータ１０は、ネットワーク２７と通信することも可能である。

コンピュータ１０は処理部１２によって実行される多数のソフトウエアの部分を含む。これらのソフトウエアの部分の一つが、オペレーションシステム２８である。一実施例において、オペレーションシステム２８は、マルチプルアドレス空間を維持することができるマイクロカーネル(microkernel)オペレーションシステムである。ファイルマネージャ３０は、オペレーションシステム中に内在する。模範的な実施例では、本発明のコンピュータ１０は、カリフォルニア州のCupertinoにあるアップルコンピュータ社によって作られたアップルマッキントッシュ（TM）コンピュータシステムであり、ファイルマネージャ３０を含むマイクロカーネルオペレーションシステムを内包するマイクロプロセッサおよびメモリを持つ。コンピュータの構成部分は、本発明の開示を理解することにより、通常の技術者の技術範囲で変更することが可能である。本発明はマッキントッシュ環境で記述されたものであるが、本発明の精神の範囲内および通常の技術を持った技術者がDOS, Unix、または他のコンピュータ環境で実施できるものも本発明に含まれる。

一実施例によると、本発明はマイクロカーネル構造をしたオペレーションシステムの一部であり、カーネルは以下のサービスを提供するものとする。カーネルは、すべてのファイルシステムの共用データ構成をファイルシステムのコードが存在するすべてのアドレス空間へマッピングすることができる、アドレス空間管理に向けられている。これは、すべてのコードおよびデータを単一のアドレスルームへマッピングすることによって提供することができる。また、カーネルは、タスク創作およびタスク終了の告知、セマフォ同期機構およびメッセージの送り先となることのできる複数のオブジェクトを供給するメッセージシステムを提供すると良い。このメッセージシステムにより、クリエータ定義値をそれぞれのオブジェクトに関連させたり、複数のオブジェクトを同じポートにマッピングしたり、メッセージをポートから受け取ったり、適切なタイプのメッセージがそのポートに送られたときにコールされる機能を持ったり、メッセージの受信者がそのメッセージが元々どのオブジェクトに向けて送信されたものかを決定したり、そのオブジェクトのクリエータの定義値を導き出したりすることができる。そのようなマイクロカーネル機構の一例は、アップルマッキントッシュ（TM）コンピュータで使用されるNuKERNEL（TM）によって提供されている。このNuKERNEL（TM）システムは、１９９３年９月３０日出願の係属中の米国特許出願番号第０８／１２８，７０６号記載の「コンピュータに於ける仮想メモリの分散バッキングストア制御のためのシステム（System For Decentralizing Backing Store Co

ntrol Of Virtual Memory In A Computer）」、１９９４年３月３０日出願の米国特許出願番号第０８／２２０，０４３号記載の「オブジェクト指向のメッセージ渡しシステムおよび方法（Object Oriented Message Passing System And Method）」、および本出願の日前後にＴｈｏｍａｓ　Ｅ．　ＳａｕｌｐａｕｇｈおよびＳｔｅｖｅｎ　Ｊ．　Ｓｚｙｍａｎｓｋｉの名で出願された「オブジェクト指向のメッセージフィルタリングのシステムおよび方法（System and Method Of Object Oriented Message Filtering）」に記述されている。これらの３つの特許出願は、参照により本願に包含されるものとする。

メッセージシステムによると、サービスはサーバによって提供され、サービスを使用したいソフトウエアは、サーバのクライアントと呼ばれる。メッセージは、クライアントによって目的地（宛先）へ送信され、サーバによって受信される。メッセージは、クライアントに代わってサーバが機能を実行するようにし向ける。メッセージオブジェクトは、メッセージシステムクライアントへのさまざまなリソースを示す、アブストラクトエンティティである。これらのオブジェクトは、例えば、サーバによって管理されるボリュームや、フォークや、ファイルを表しても良い。クライアントは、オブジェクトにメッセージを送り、オブジェクトはObjectID（オブジェクトＩＤ）によって識別される。

メッセージポートは、サービスを表すアブストラクト・エンティティである。これらのポートは、例えば、装置のドライバ、ファイルマネージャ、またはウィンドウマネージャを示しても良い。サーバは、ポートからメッセージを受け取り、そのポートはポートＩＤ（PortID）によって識別される。オブジェクトはポートに割り振られている。オブジェクトに送られたメッセージは、オブジェクトのポートから受信される。ポートおよびオブジェクトは、サーバの代わりにメッセージシステムによって作成される。オブジェクトを作成するために、オブジェクトに向けて送られたメッセージがリトリーブされる（取出される）ポートを指定しなければならない。本発明の一実施例において、オブジェクトおよびポートはファイルマネージャのさまざまな構成部分間の通信を確立するために使用される。本発明の一実施例は、カーネル同期サービスを利用して、重要な(critical)セクションや、論理メモリをアロケートするカーネルメモリ管理サービスや、複数の実行スレッドを提供するカーネルタスクサービスを保護するために使われる、セマフォに対するサポートを提供する。

本発明の一実施例に係るファイルマネージャ３０の機構の概略ブロック図を図２に示す。一実施例によると、ファイルマネージャは、インターフェースモジュール３１０と、ディスパッチャ（またはディスパッチモジュール）３００と、ストア（記憶装置）３４０と、共用ライブラリ３３０と、フォーマットエージェントモジュール３２０とを含む、それぞれ個別にロードすることができるモジュールの集合として見ることができる。

インターフェースモジュール＃１、＃２など（３１０）は、システムの他のモジュールへの異なるパブリックインターフェースを示す。そのような２つのインターフェースモジュールを図２に示すが、このようなモジュールをいくつ提供しても良い。特定の環境のために書かれたソフトウエアは、その環境下において標準のインターフェースを使用することができる。これらのインターフェースモジュール３１０は、それらのインターフェースを介して得られたリクエストをメッセージ構成に翻訳し、それらのメッセージをディスパッチャ３００へ送信することにより、呼び出し(calling)ソフトウエアとオペレーションシステム、特にファイルマネージャとの間のインターフェースを行う。

ストア３４０は、それぞれのフォーマットエージェントを識別するための識別子と、ディスパッチャポートに割り当てられたオブジェクトを識別するための識別子と、フォーマットエージェント識別子とオブジェクト識別子間のマッピングをするためのマッピング情報とを保存する記憶構成を含む。一実施例によると、ストア３４０は、ディスパッチャ３００の概念的な部分である。ディスパッチャ３００は、インターフェースモジュール３１０からメッセージを受信し、ストア３４０に含まれるマッピング情報を用いて、どのフォーマットエージェントがリクエストを満足することができるかを決定する。その後、メッセージを適切なフォーマットエージェント＃１、＃２など（３２０）へと転送する。３つのフォーマットエージェント＃１、＃２、＃３が図示されているが、含まれるフォーマット

エージェントの数はいくつでも良い。一実施例によると、それぞれのフォーマットエージェントは、特定のファイルシステムフォーマットに関連する記憶媒体にアクセスすることができる。

適切なフォーマットエージェント３２０は、ディスパッチャ３００からメッセージを受信し、リクエストを処理することによって、オペレーションシステム、特にファイルマネージャと、アクセス中の記憶媒体間のインターフェースを行う。リクエストが完了すると、フォーマットエージェント３２０は、元のコーラへ戻るのに必要な情報をディスパッチャ３００へ応答する。リクエストを実行するのに関わったフォーマットエージェント３２０のすべてが終了し、ディスパッチャ３００への返答を終えると、ディスパッチャ３００はリクエストを初期化するのに用いたインターフェースモジュール３１０に返答し、コーラへ戻る必要のある情報を送る。インターフェースモジュール３１０は、その情報をインターフェースに適した形態に翻訳し、それをコーラに戻す。

フォーマットエージェント３２０の内部メカニズムの部分は、異なるモジュール間で似ているため、本発明に係るファイルマネージャは、すべてのフォーマットエージェント３２０が使用できる共用ライブラリ３３０を含んでもよい。このライブラリは、フォーマットエージェント３２０間で共用することができる共通のメカニズムの実施を含むため、これらのモジュールの開発が容易になる。共用ライブラリ３３０は、エージェントがストア３４０を形成する記憶構成を構築できるようにするために、フォーマットエージェント３２０が必要とするソフトウエアルーチンをも含む。

一実施例によると、一インターフェースモジュールは、本発明に係るファイルマネージャとファイルマネージャＡＰＩの過去の実施にインターフェースし、他のインターフェースモジュールは現在のオペレーションシステムと本発明に係るファイルマネージャ間のインターフェースを供給する。他のインターフェースモジュールは、本発明の開示を理解することにより通常の技術者の技術範囲内で提供することが可能である。

一実施例によると、図３に示すようにストア３４０は複数の制御ブロックを持つ。制御ブロックは、エージェント制御ブロック（エージェントＣ．Ｂ．）３４１０と、ボリューム制御ブロック（ボリュームＣ．Ｂ．）３４２０と、ファイル制御ブロック（ファイルＣ．Ｂ．）３４３０と、フォーク制御ブロック（フォークＣ．Ｂ．）３４４０と、パス制御ブロック（パスＣ．Ｂ．）３４５０と、イタレータ制御ブロック（イタレータＣ．Ｂ．）３４６０とを含む。

本発明の一実施例に係るファイルマネージャは、それが管理するファイルシステムエンティティ（例えば、ボリュームフォーマット、ボリューム、ファイル、フォーク、パス、イタレータ(iterators)）のランタイム状態を維持するために、これらのファイルシステムのエンティティーに関する頻繁にアクセスされる情報のコピーをメモリ内に維持するために、メモリに保存された制御ブロック構成を使用する。加えて、これらの制御ブロックのそれぞれは、リクエスト処理タスク（ＲＰＴｓ）の活動を調整するために使用されるセマフォを含む。

一実施例によると、これらの制御ブロックは、以下のように保存される。エージェント制御ブロック３４１０のリストがあり、リスト上のエージェント制御ブロックのそれぞれは、フォーマットエージェント３２０に対応する。それぞれのエージェント制御ブロック３４１０は、そのフォーマットを有するボリュームのためのボリューム制御ブロック３４２０のリストをもつことができる。ボリューム制御ブロック３４２０のそれぞれは、オープンのフォークを持つファイルのためのファイル制御ブロック３４３０のリストと、そのボリュームの何かを現在示しているイタレータのためのイタレータ制御ブロック３４６０とを有することもできる。それぞれのファイル制御ブロック３４３０は、そのファイルのためにオープンされたフォークのためのフォーク制御ブロック３４４０のリストを持つことができる。フォーク制御ブロック３４４０のそれぞれは、そのファイルのためにオープンされたそれぞれのアクセスパスのためのパス制御ブロック３４５０を持つことができる。

それぞれの制御ブロックは、タイプに関わらず、いくつかの共通フィールドを持つヘッダ３４００を有する。それらのフィールドは、制御ブロックタイプ、バージョン、ペアレン

ト制御ブロックポインタ、リストリンク、セマフォ、エージェント制御ブロックポインタ、およびエージェント特定データポインタである。この構成を図３ｂに示す。制御ブロックタイプフィールドは、制御ブロック（ボリューム、ファイルなど）のタイプである。バージョンフィールドは、使用された制御ブロック構成のバージョンである。ペアレント制御ブロックポインタフィールドは、これのペアレントである制御ブロックへのポインタである。例えば、ファイル制御ブロック３４３０にとって、このフィールドはそのフィールドをリスト中に有するボリューム制御ブロックへのポインタである。リストリンクフィールドは、同じペアレントを持つ前後の制御ブロックへのポインタである。セマフォフィールドは、制御ブロックへのアクセスを調整するために使用されるセマフォである。エージェント制御ブロックポインタフィールドは、その構成を所有するエージェント制御ブロックポインタへのポインタである（つまり、もしペアレントポインタをたどり続ければ、最終的にはこのポインタにたどり着く）。エージェント特定データポインタフィールドは、制御ブロックへのエージェント特定拡張へのポインタである。このフィールドは、ファイルマネージャの異なる実行が制御ブロックに含まれる、あるパブリックフィールドを定義するので、またすべてのボリュームフォーマットがフォーマットに特有のデータおよび状態の変遷を追うために追加のプライベートフィールドを必要とすると思われるので、含まれている。制御ブロックサービスモジュール３３５０（図４）は、フォーマットエージェント３２０がこれらの追加フィールドを保持するために、制御ブロックを追加のルーム（メモリ空間）にアロケートすることができるようにする。そして、エージェント特定データポインタは、この追加されたメモリ空間へのポインタである。制御ブロック内に含まれるフィールドの定義は、本発明の開示を理解することにより、当業者により変更可能である。

エージェント制御ブロック３４１０の一実施例は、図３ｃに示す構成をし、以下のフィールドを含む。即ち、制御ブロックヘッダ、ボリュームリストヘッダ、エージェントオブジェクトＩＤ、エージェントサイン（エージェント署名）、タイムアウト（時間切れ）ベースである。制御ブロックヘッダは、図３ｂに示す上記のフィールドを含む。ボリュームリストヘッダは、このボリュームフォーマットについてのボリュームのリストへのポインタである。エージェントObjectIDは、このボリュームフォーマットのためにフォーマットエージェントへメッセージが送られる先のObjectIDである。エージェントのサイン（署名）は、ボリュームフォーマットの個別の識別子である。タイムアウトベースは、このボリュームフォーマットのためにリクエストがかける時間を見積もるための基準として使用される時間の長さである。従って、使用可能なタイムアウトの長さである。

図４は、本発明の一実施例に係る共用ライブラリ３３０のブロック図である。共用ライブラリ３３０は、５つの論理コードモジュールを含む。それらは、レンジロックサービスモジュール３３１０と、BTreeサービスモジュール３３２０と、キャッシュサービスモジュール２２５と、ブロック入出力（Ｉ／Ｏ）サービスモジュール３３４０と、制御ブロックサービスモジュール３３５０である。

レンジロックサービスモジュール３３１０は、レンジロック機能を提供するために使用されるすべてのフォーマットエージェントのための標準メカニズムを提供するものである。エージェントがこのメカニズムを使用するか、個別に供するかを決定するのは、まだ、それぞれのフォーマットエージェントによる。この決定は、以下により詳しく説明する。BTreeサービスモジュール３３２０は、フォーマットエージェントが使用することのできる共通BTreeコードを含む。一実施例に係る機構により、同じコードを内部およびパブリックBTreeインターフェースの両方をサポートするために使用することができる。キャッシュサービスモジュール３３３０は、ファイルマネージャ・キャッシュの管理を行うものである。それは、キャッシュ記憶をアロケートして保持し、どのようにさまざまなキャッシュブロックが使用されているかについてのデータベースを維持している。一実施例によると、この構成により、大量のキャッシュを有してもキャッシュヒットを速く行うことができるようにし、さまざまなサイズのバッファを扱い、リード・拡張処理をサポートする。ブロックＩ／Ｏサービスモジュール３３４０は、ドライバの入出力を行うために、ディス

クベースのフォーマットエージェントに共通インターフェースを提供する。
制御ブロックサービスモジュール３３５０は、本発明に係るファイルマネージャによって内部的に使用されるさまざまな種類の制御ブロックをアロケートし、アクセスし、取り扱い、リリースするコードを含む。制御ブロックサービスモジュール３３５０は、すべての重要(critical)セクションが保護されるように、実行される複数のスレッドに於けるリクエストの同時処理を可能にするサービスを提供している。特に、同じデータを使用する実行の複数のスレッドが存在する場合、お互いにそのデータを変更しないような方法が必要とされる。
一実施例によると、これは、制御ブロックを獲得し、それをリリースすることによって達成される。例えば、ボリュームから情報を読み出す場合（ただし変更を加えない）、ボリュームの制御ブロックは、同時にほかの人も読み出せるように共用アクセスのために獲得され、アクセスが終了するとリリースされる。または、アクセスパスについての情報を更新するために、１つのスレッドのみがある時に排他的にアクセスできるように、パスの制御ブロックがその排他的アクセスだけのために獲得される。第１のスレッドが排他的アクセスのための制御ブロックを獲得している間に、実行される第２のスレッドがアクセスしようとした場合、その第２のスレッドは、第１のスレッドが、その制御ブロックをリリースするまで強制的に待たされることになるこのようにして、種々のスレッドが互いに強調して実行されることになる。一実施例によると、これらの構成のために実際のデータは、カーネルバンド内にアロケートされ、グローバルに参照されるが、これらを取り扱うコードはそれらがすべてのフォーマットエージェント３２０にアクセスできるように共用ライブラリに含まれている。
図５ａは、本発明の一実施例に係るディスパッチャ３００のブロック図である。ディスパッチャ３００は、２つのタスクモジュールを含む。ディスパッチメインテナンスモジュール３０４０と、要求処理モジュール３０３０である。またディスパッチャ３００は、ディスパッチストア３０５０と、ディスパッチメッセージポート３０１０とを含む。ボリューム制御ブロックオブジェクト３４２５と、パスオブジェクト３４４５と、イタレータオブジェクト３４６５は、上記のディスパッチメッセージポート３０１０に割り振られている。ディスパッチメッセージポート３０１０は、ボリュームまたはパス、またはイタレータを指示していない要求を受信するファイルマネージャオブジェクト(FMO)３０２０も含む。
一実施例によると、ディスパッチ・メインテナンスモジュール３０４０は、ファイルマネージャの初期化、シャットダウン、カーネルメッセージの取り扱いや要求のキャンセルや要求処理タスク数の保全およびキャッシュのフラッシュなどの定期的な保全作業などを含む保全動作の１つの実行するスレッドからなる。他のメインテナンスの手順は、他の実施例において、本発明の開示を理解することにより当業者によって実施可能であろう。
一実施例によると、要求処理モジュール３０３０は、ディスパッチャに送られたファイルマネージャ要求を実際に処理するために使われる要求処理タスク（ＲＰＴ）として知られる、複数のスレッドの実行からなる。どの時点においても、これらのタスクのそれぞれは、要求の処理か、または、ファイルマネージャポートに入力される新たなメッセージを待つことである。新しい要求メッセージが入力される度に、メッセージシステムは、ウェイティングしている次のＲＰＴにそれを渡す。
一実施例によると、ＲＰＴの経過を追うために、ディスパッチャ３００は、各実行中のＲＰＴにつき１つずつの、RequestTaskState構造のセットを含むディスパッチストア３０５０を維持している。一実施例において、これらの構造は二重にリンクするリスト中で維持される。この構造は、タスク識別子、現在処理中（もし処理中であれば）の要求のメッセージ識別子、そして要求が転送されるフォーマットエージェント（もしあれば）の経過を追う。
一実施例において、この構成は図５ｂに示すフォーマットを持ち、リストリンクと、タスクＩＤと、フラグと、ターミネーションアクションと、現在のメッセージと、キャンセルされたメッセージと、エージェント制御ブロックポインタとを含む。リストリンクフィー

ルドは、RequestTaskState構造の前後へのポインターである。フラグフィールドは、ＲＰＴが現在要求を処理しているか、また、その要求がフォーマットエージェントへ送られたかどうかを示すために利用される。ターミネーションアクションフィールドは、一実施例によれば、ＲＰＴが予期せず終了した場合に呼び出されるメインテナンスタスクのために割り込みをするソフトウエアである。現在のメッセージフィールドは、ＲＰＴが現在処理しているメッセージのメッセージＩＤである。もしＲＰＴが現在要求を処理しているのでなければ、現在のメッセージフィールドは、ある知られた値にリセットされる。キャンセルされたメッセージフィールドは、キャンセルされたものであれば、そのメッセージのメッセージＩＤである。エージェント制御ブロックポインタは、要求が転送された先のフォーマットエージェントのための制御ブロックへのポインタである。

一実施例によれば、ディスパッチャ３００は、以下の付加的なグローバルを含む。すなわち、RequestTaskQueueはRequestTaskState構造のリストへのヘッダであり、AgentControlBlockListはエージェント制御ブロックのリストへのヘッダであり、FileManagerPortはファイルマネージャポートＦＭＯ３０２０であり、FileManagerObjIDは、ファイルマネージャ要求のための通常のObjectIDであり、RequestTaskCountは実行中のＲＰＴの数であり、RequestTaskDesireは与えられた時間に実行させるＲＰＴの望ましい数であり、MasterSemaphoreは、エージェント制御ブロックのリストへのアクセスを調整して使用するために用いられるセマフォである。

図６は、本発明の一実施例に於ける、インターフェースモジュールがコーラからの要求をプロセスする工程を示すフローチャートである。要求は、指定された入力および出力パラメータと共にコールの形態でコーラから受信される。ステップ６１０において、インターフェースモジュールは、コールのいずれかの入力パラメータを解釈する。出力パラメータに書き込みするために入力パラメータが必要かどうが判断される（ステップ６２０）。もしインターフェースモジュールがそのコールに対する出力パラメータに書き込むための十分なデータを持っている場合、ステップ６２０の判断はｎｏとなり、出力パラメータはステップ６２５で書き込みがされる。

出力パラメータに書き込みするのにメッセージが必要とされた場合、インターフェースモジュールは出力パラメータを得るために、送信する必要のある第１メッセージを決定する（ステップ６３０）。メッセージは、ディスパッチャ３００へ送信される（ステップ６４０）。ディスパッチャ３００から応答を受信すると、インターフェースモジュールによって解釈される（ステップ６５０）。応答されたデータは、ステップ６６０で一時バッファに格納される。これにより、インターフェースモジュールは、コールが出力パラメータとして返される値を手に入れるために多くのメッセージを必要とする状況および、コールが多くのメッセージからの応答データによって決定される値を有する１つの出力パラメータを持つと言った状況を扱うことができる。ステップ６７０では、コールが完了したかどうか、つまり、すべての必要なメッセージが出力パラメータ値を決定するために送信されたかどうかを判断し、コールを遂行するために必要なすべてのアクションを実行する。もし完了していなければ、インターフェースモジュールは送信すべき次のメッセージを決定し（ステップ６７５）、ステップ６４０へ戻る。そうであれば、出力パラメータが一時バッファから書き込まれ（ステップ６８０）、インターフェースモジュールはコーラに戻る（ステップ６９０）。

一実施例において、この処理はステートマシーンとして実施される。そのようなステートマシーンにとって、インターフェースモジュールへのコールの目的は、情報を集めたり、ファイルシステム上で動作を行ったり、または両方を行うことであるとみなすことができる。従って、ステートは、集められた情報の特定のセットや、実行する必要のない動作（その動作がすでに実施されたかまたはコールへ提供された入力パラメータに基づいて実行する必要が無い場合のどちらか）の特定のセットを表す。同様に、トランジションは、更なる情報を集め、更なる動作を行い、または両方を行うためにディスパッチャに送るべきメッセージを表す。インターレースモジュールは、入力パラメータによって示されたすべての情報が収集され、入力パラメータによって示されたすべてのアクションが完了するま

でステートマシーンの実行を続ける。
一実施例によると、要求は同時に処理される。しかし、それぞれのボリュームは、単に一つのものであり、制御ブロックの１階層しか存在しない。従って、ボリュームおよび制御ブロックは、アクセスをＲＰＴ間で対等でなければならない共用リソースである。一実施例では、これは、それぞれの制御ブロックにセマフォを含め、制御ブロックと、制御ブロックが表すファイルシステム内のデータへのアクセスを対等にするためにこれらのセマフォを使用することにより実現することができる。
一実施例によると、この対等にするためのポリシーは、
１）　エージェント制御ブロックを追加または削除するために、ディスパッチャのためのマスターセマフォおよび追加または削除される制御ブロックのためのセマフォが、排他的アクセスのために獲得されなければならない。
２）　エージェント制御ブロックのセットをトラバースするために、ディスパッチャのためのマスターセマフォおよび現在のエージェントのための制御ブロックのためのセマフォが、共用アクセスのために獲得されなければならない。
３）　ボリュームが現在マウントされているかどうかを変更するために、ボリュームの制御ブロックのためのセマフォおよびボリュームのエージェント制御ブロックのためのセマフォが、排他的アクセスのために獲得されなければならない。
４）　ボリューム制御ブロックのセットをトラバースするために、ディスパッチャのためのマスターセマフォおよび現在のボリュームに関するエージェント制御ブロックセマフォが、共用アクセスのために獲得されなければならない。
５）　ファイル、フォーク或はパス制御ブロックを追加または削除する特定のボリュームのための制御ブロック階層の構造を変更するために、ボリューム制御ブロックのセマフォおよび追加または削除される制御ブロックのためのセマフォが、排他的アクセスのために獲得されなければならない。
６）　ある特定のボリューム（そのボリュームの下でのファイル、フォークおよびパス制御ブロックすべて）のための制御ブロック階層の構成をトラバースするかまたは、そのボリュームのためのメタデータ（ディスク上の共用構造、例えば、ディスクアロケーションマップ）を変更するために、ボリューム制御ブロックのセマフォが、排他的アクセスのために獲得されなければならない。
７）　ファイル、フォークまたはパスについての情報を読み出すために、適切な制御ブロックのセマフォが、共用アクセスのために獲得されなければならない。
８）　ファイル、フォークまたはパスについての情報を更新するために、適切な制御ブロックのセマフォが、排他的アクセスのために獲得されなければならない。
９）　複数のセマフォはマスタ、パス、フォーク、ファイル、ボリューム、ボリュームフォーマットの順番に従って獲得されなければならず、この逆の順番で解放されなければならない。
フォーマットエージェント処理
図７ａ乃至７ｆは、フォーマットエージェントがストア３４０を形成する制御ブロックを作成する、本発明の一実施例に係る処理を示すフローチャートである。図７乃至７ｆでは制御ブロック構成は図３に示す構成を参照している。エージェントがどのようにしてそのフォーマットで構成された記憶媒体にアクセスするか、などの残りの機能は、エージェント毎に異なったものとなる。本発明の開示を理解することにより当業者はそうした機能を実現することが可能であろう。
図７ａは、マウント要求に応じてボリューム制御ブロックがストア３４０で作成される、本発明の一実施例に係る処理を示している。最初に、ボリューム制御ブロックがプライベートフィールドのルーム（メモリ空間）で作成される（ステップ７１００）。ボリューム制御ブロックのプライベートおよびパブリックフィールドは、ステップ７１１０で書き込みがされる。一実施例では、プライベートフィールドはフォーマットエージェントにのみ知られており、一方パブリックフィールドはファイルマネージャにより要求されるフィールドである。排他的アクセスのためにエージェント制御ブロックセマフォが獲得され（ス

テップ７１２０）、ボリューム制御ブロックはエージェント制御ブロックのボリュームリストにリンクされる（ステップ７１３０）。そして、エージェントセマフォが解放され（ステップ７１４０）、フォーマットエージェントはディスパッチャへ戻る。

図７ｂ，７ｃは、オープン要求に応じてパス制御ブロックがストア３４０に作成される、本発明の一実施例に係る処理を示している。最初に、パス制御ブロックがプライベートフィールドに対するルームに割り当てられる（ステップ７２０９）。パス制御ブロックのプライベートおよびパブリックフィールドは書込みがされる（ステップ７２０５）。

ステップ７２１０では現存のフォークに対するフォーク制御ブロックが存在するかどうかが判定される。フォーク制御ブロックが既に存在している場合は、排他的アクセスのためにファイルが存在しているボリュームに対するボリューム制御ブロックセマフォが獲得される（ステップ７２１１）。パス制御ブロックはフォーク制御ブロックのパスリストにリンクされ（ステップ７２１２）、ボリューム制御ブロックセマフォは解放される（ステップ７２１３）。そして処理は終了する。

ステップ７２１０でフォーク制御ブロックが存在していない場合は、フォーク制御ブロックがプライベートフィールドに対するルームに割り当てられる（ステップ７２１５）。フォーク制御ブロックのプライベートおよびパブリックフィールドは書き込みがされ（ステップ７２２０）、パス制御ブロックはフォーク制御ブロックのパスリストにリンクされる（ステップ７２２５）。

図７ｃを参照すると、処理されているファイルに対するファイル制御ブロックが存在するかどうかが判定される（ステップ７２３０）。ファイル制御ブロックが存在する場合は、排他的アクセスのためにボリューム制御ブロックセマフォが獲得される（ステップ７２３１）。フォーク制御ブロックはファイル制御ブロックのパスリストにリンクされ（ステップ７２３２）、ボリューム制御ブロックセマフォが解放される（ステップ７２３３）。そして処理は終了する。ステップ７２３０で現在のファイルに対するファイル制御ブロックが存在しない場合は、ファイル制御ブロックがプライベートフィールドに対するルームに割り当てられる（ステップ７２３５）。ファイル制御ブロックのプライベートおよびパブリックフィールドは書き込みがされ（ステップ７２４０）、フォーク制御ブロックはファイル制御ブロックのフォークリストにリンクされる（ステップ７２４５）。排他的アクセスのためにボリューム制御ブロックセマフォが獲得される（ステップ７２５０）。ファイル制御ブロックはボリューム制御ブロックのファイルリストにリンクされ（ステップ７２５５）、ボリューム制御ブロックセマフォは解放される（ステップ７２６０）。そして処理は終了する。

図７ｄおよび７ｅは、クローズ要求に応じてパス制御ブロックがストア３４０から排除される、本発明の一実施例に係る処理を示している。排他的アクセスのためにボリューム制御ブロックセマフォが獲得される（ステップ７３００）。ステップ７３０５で、パス制御ブロック内のデータはフラッシュされる（つまり、パス制御ブロック内のデータは記憶媒体に書込まれる）。パス制御ブロックはフォーク制御ブロックのパスリストからリンク解除（リンクされなくなる）される（ステップ７３１０）。ステップ７２３０では、このフォークに対するパス制御ブロックがそれ以上あるかどうかが判定される。他にパス制御ブロックが存在する場合は、ボリューム制御ブロックセマフォは解放され（ステップ７３３２）、それら他のパス制御ブロックがオープンの状態のままであるようにし、そして処理は終了する。

ステップ７３３０で他にパス制御ブロックが存在しない場合は、階層のバックアップをする必要がある（図３参照）。従って、データはフォーク制御ブロックにフラッシュされ（ステップ７３３５）、フォーク制御ブロックはファイル制御ブロックのファイルリストからリンク解除される（ステップ７３３０）。

ステップ７３４５（図７ｅ）では処理されているファイルに対するさらに他のフォーク制御ブロックが存在するかどうかが判定される。他のフォーク制御ブロックが存在する場合は、ボリューム制御ブロックセマフォが解放され（ステップ７３４７）、他のホーク制御ブロックがオープンの状態のままであるようにし、そして処理は終了する。ステップ７３

４５で他にフォーク制御ブロックが存在しない場合は、階層のバックアップをする必要がある（図３参照）。従って、データはファイル制御ブロックにフラッシュされ（ステップ７３５０）、ファイル制御ブロックはボリューム制御ブロックのファイルリストからリンク解除され（ステップ７３３５）、ボリューム制御ブロックセマフォが解放される（ステップ７３６０）。そして処理は終了する。

図７ｆは、本発明の一実施例に係る、アンマウント要求に応じてボリューム制御ブロックがストア３４０から削除される処理を示している。最初に、共用アクセスを行うためにボリューム制御ブロックセマフォが獲得される（ステップ７４００）。さらに、そのボリュームに対する新たな要求がブロックされる。一実施例では、これはメッセージングシステムにより提供されるロックオブジェクトフィーチャを使用して行われる。ステップ７４１０では、このボリュームに対するファイル制御ブロックが存在するかどうかが判定される。ファイル制御ブロックが存在する場合は、ボリューム制御ブロックセマフォが解放され（ステップ７４１５）、処理は失敗となる。ファイル制御ブロックが存在しない場合は、データがボリューム制御ブロックからフラッシュされる（ステップ７４２０）。ステップ７４３０では、このボリュームに対するイタレータ（反復）制御ブロックが存在するかどうかが判定される。存在する場合は、各イタレータはエンティティを何も参照しないようにリセットされる（ステップ７４１５）。一実施例では、これはアンマウントされているボリュームを制御するフォーマットエージェントへメッセージを送ることにより達成される。このボリュームに対するイタレータ制御ブロックが存在しない場合、そしてステップ７４３５の後は、ボリューム制御ブロックセマフォが解放され、排他的に再獲得される（ステップ７４４０）。排他的アクセスのためにエージェント制御ブロックセマフォが獲得される（７４５０）。ボリューム制御ブロックはエージェント制御ブロックのボリュームリストからリンク解除される（ステップ７４６０）。エージェント制御ブロックセマフォは解放され（ステップ７４７０）、処理は終了する。

フォーマットエージェント３２０の目的はディスパッチャ３００からの要求を受信し、それらを処理し、そして要求の結果を応答する。一実施例では、上述のように、フォーマットエージェント３２０は共用ライブラリ３３０を利用して、いくつかのフォーマットで共通の動作を行う。いくつかの要求は制御ブロックのパブリックまたはプライベートフィールドのいずれかに格納されている情報を使用して実行可能であるが、一方、媒体の読み出し／書き込み、あるいはボリュームのデータを得るのに用いたデバイスへのアクセスを必要とするような要求もある。

一実施例では、ボリュームがブロック用のデバイスに格納されている場合は、フォーマットエージェント３２０は共用ライブラリ３３０のブロック入出力サービスモジュール３３４０を使用して媒体のデータを読み出したり書き込んだりすることができる。フォーマットエージェント３２０が、あるデータが繰り返し使用可能であると予測したり、またファイルマネージャにそのデータに対しバッファを割り当てさせようとする場合は、共用ライブラリ３３０のキャッシュサービスモジュール３３３０を使用することができる。エージェント３２０が、共用ライブラリ３３０のBTreeサービスモジュール３３２０によって扱うことのできるフォーマットを持つBTreeへのアクセスを必要とする場合は、そのモジュールを使用することができる。

一実施例では、共用ライブラリ３３０のこれら３つのモジュールは互いに依存している。BTreeサービスモジュール３３２０は必要とするデータに対するバッファを得るためにキャッシュサービスモジュール３３３０をコールすることができる。キャッシュサービスモジュール３３３０は実際にキャッシュにおいてデータを読み書きするためにブロックI/O（入出力）サービスモジュール３３４０をコールすることができる。

しかし、これらのモジュールを連結しないことが望ましい。従って、他の実施例では、これらのモジュールはフォーマットエージェント３２０をコールバックしてこれらの必要を満たしている。フォーマットエージェント３２０は、フォーマットエージェント３２０が共用ライブラリ３３０上で行うコールに応じて共用ライブラリのモジュールが使用することができる手続きインタフェースを有している。これらのコールのためのインタフェース

は、BTreeサービスモジュール３３２０がキャッシュサービスモジュール３３３０に対して行う必要のあるコールとキャッシュサービスモジュール３３３０がブロックI/Oサービスモジュール３３４０に対して行う必要のあるコールに対し同一の機能性を提供する。
この実施例によると、フォーマットエージェント３２０が共用ライブラリ３３０のBTreeサービスモジュール３３２０をコールすると、BTreeサービスモジュール３３２０は、そのキャッシュ動作のためにフォーマットエージェント３２０をコールする。するとフォーマットエージェント３２０はキャッシュサービスモジュール３３３０をコールしてこれらのコールを実行することを選択するか、または他の何らかのメカニズム（例えばプライベートキャッシュまたはキャッシュ使用しない直接の読み書き）を使用することができる。同様に、フォーマットエージェント３２０が共用ライブラリ３４０のキャッシュサービスモジュール３３３０をコールすると、そのキャッシュはそのブロックのI/O動作のためにフォーマットエージェント３２０をコールする。再び、フォーマットエージェント３２０はブロックI/O動作をコールしてこれらのコールを実行することを選択するか、または他の何らかのメカニズム（例えばネットワークドライバへのコールまたはラムディスクバッファの読み出し）を使用することができる。
ディスパッチャによる要求処理
ディスパッチャ３００の主な機能は、要求を正しいフォーマットエージェント３２０へ送ることである。どのようにこれを行うかは要求のタイプに依存しているが、大抵のケースを網羅する一般的な規則は以下の如くである。
１）要求が特定のボリューム、ディレクトリ、ファイルまたはフォークに向けられている場合、そのボリューム、ディレクトリ、ファイルまたはフォークに対するスペシフィケーションの一部はエンティティが存在するボリュームに対するボリュームObjectIDである。メッセージはそのObjectIDへ送られる。そのOjbectIDには適切なボリューム制御ブロックを指すポインタが関連付けられている。ディスパッチャはそのポインタを獲得し、その制御ブロックから、エージェント制御ブロックポインタを使用して、フォーマットエージェントに対するObjectIDを有するエージェント制御ブロックへ到達する。そしてメッセージを、そのフォーマットエージェントへ送る。こうした種類の要求にはFileCreate（ファイル生成）、Delete（削除）、PathOpen（パスオープン）、VolumeFlush（ボリュームフラッシュ）が含まれる。
２）要求が特定のアクセスパスまたはイタレータへ向けられている場合は、メッセージはそのアクセスパスまたはイタレータに対するObjectIDへ送られる。そのOjbectIDには適切なパス制御ブロックまたはイタレータ制御ブロックを指すポインタが関連付けられている。ディスパッチャはそのポインタを獲得し、その制御ブロックから、エージェント制御ブロックポインタを使用して、そのフォーマットエージェントに対するObjectIDを有するエージェント制御ブロックへ到達する。そしてメッセージをフォーマットエージェントへ送る。こうした種類の要求にはRead（読出し）、Write（書込み）、Access Pathsに対するClose（クローズ９、Iteratorsに対するIterateが含まれる。
３）要求が１つ以上のボリュームを含む（そして、それゆえに１つ以上のボリュームフォーマットと１つ以上のフォーマットエージェントを含む可能性がある）場合、メッセージはファイルマネージャのオブジェクトのObjectIDへと向けられる。ディスパッチャは要求を分割して各々１つのフォーマットエージェントしか含まないパーツにして、それらのメッセージをそれぞれ適切なフォーマットエージェントへと送る。例えば、FileCopy（ファイルコピー）要求は存在するファイルと作成されるべき新しいファイルのスペシフィケーションを含んでいる。これらのスペシフィケーションは含まれているボリュームに対するVolumeObjectIDを含んでいる。そこから、フォーマットエージェントに対するObjectIDが得られる（上記ケース１参照）。ディスパッチャは存在するファイルを有しているフォーマットエージェントへRead（読出し）要求を送り、新しいファイルを有しているフォーマットエージェントへWrite（書込み）要求を送る。こうした種類の要求にはFileCopyRename（ファイルコピー改名付）やFileMoveRename（ファイルムーブ改名付）が含まれる。
４）要求が固有のファイルシステム構造を含んでおらず、しかしその代わり内部ファイル

マネージャデータを含んでいる場合は、その要求はファイルマネージャのオブジェクトのobjectIDへ送られ、ディスパッチャはいずれのフォーマットエージェントへも送ることなくその要求を分解することができる。この種類の要求にはSetCurrentDirectory（現ディレクトリの設定）やGetFileManagerVersion（ファイルマネージャのバージョン情報取得）が含まれる。

ほとんどの種類の要求は先の４つのカテゴリへに入る。しかしながら、特殊なケースであって特殊な規則で扱われる２つの要求MountとPathNameResolveがある。こうした種類の全ての要求はファイルマネージャオブジェクトのObjectIDへ送られる。

５a）マウンティングの前にはボリュームのフォーマットは不明であり、従ってまだマウント要求を特定のフォーマットエージェントへ向けることができない。その代わりに、ディスパッチャはそのボリュームをマウント可能かどうか尋ねるメッセージを各フォーマットエージェントへ送る。ディスパッチャは「イエス」と応えるフォーマットエージェントを選択してマウントメッセージを送る。

５b）ファイルマネージャを実現する多くには、間接的であるためエンティティが存在するボリュームのVolumeObjectIDを含まない可能性のあるボリューム、ディレクトリまたはファイルを特定する方法を有している。一実施例では、これには部分的または関連するパスネーム・スペシフィケーション（ベースエンティティを特定し、ファイルシステムを巡回して目標へ到達するような１スペシフィケーション）を提供することが含まれている。これを分解するために、ディスパッチャは部分パスネームを分析（parse）して、この情報を使用してエンティティに対する適切なスペシフィケーションを決定しなければならない。ディスパッチャはフォーマットエージェントへ要求を送ってパスに含まれているエンティティの親ディレクトリを決定して、それらがボリューム上のルートディレクトリであるかどうかを判定する必要がある。

図８ａ乃至８ｍは、本発明の一実施例に係るディスパッチャ処理を示すフローチャートである。

図８ａはディスパッチャ３００が新たなエージェントをインストールする処理を示している。エージェント制御ブロックがプライベートフィールドのためのルームで作成される（ステップ８０００）。エージェント制御ブロックのプライベートおよびパブリックフィールドに書き込みがなされる（ステップ８０１０）。排他的アクセスのためにマスターセマフォが獲得される（ステップ８０２０）。エージェント制御ブロックはディスパッチャのエージェントのリストにリンクされる（ステップ８０３０）。そしてマスターセマフォは解放される（ステップ８０４０）。

図８ｂは、本発明の一実施例に係る、ディスパッチャ３００による要求処理手順を示している。ルーチンの開始時点で、ディスパッチメッセージポート３０１０内の次の要求がディスパッチャ３００によりリトリーブされる（ステップ８１０１）。要求タスクステート・ストラクチャ３０５０が処理されている要求に対して書き込まれ、そのストラクチャのフラグフィールドが、ストラクチャがビジーであり要求が現在処理されていることを示すようにセットされる。ステップ８１０４では、このタイプの要求がボリュームオブジェクトへ送られるものであるかどうかが判定される。このテスト（および後続のテスト８１０６、８１０８、８１５０、８１５１、８１５２および８１５３）は要求が正しいタイプのオブジェクトへ送られたかどうか決定するエラーチェッキングキャパビリティ（不図示）を含んでいる。そうでない場合は、適切なエラーメッセージが作成される。要求がボリュームオブジェクトへ送られるタイプではない場合は、要求がパスオブジェクトへ送られるタイプであるかどうかが判定される（ステップ８１０６）。そうでない場合は、要求がイタレータオブジェクトへ送られるタイプであるかどうかが判定される（ステップ８１０８）。そうでない場合は、要求がマルチプルボリューム要求のタイプであるかどうかが判定される（図８ｃ、ステップ８１５０）。そうでない場合は、要求がディスパッチャ単独で扱われるタイプであるかどうかが判定される（ステップ８１５１）。そうでない場合は、要求がマウント要求であるかどうかが判定される（ステップ８１５２）。そうでない場合は、要求がパスネーム分解(resolve)要求であるかどうかが判定される（ステップ８１５

３）。そうでない場合は、ディスパッチャは「不明要求」エラーで応答する（ステップ８１５４）。ディスパッチャは図８ｂのＣへループして、要求タスクステートフラグを、要求が処理されていないことを示すアンビジーにセットし（ステップ８１００）、ディスパッチポートから次の要求を得る（ステップ８１０１）。

ステップ８１５０、８１５１、８１５２および８１５３における判定はメッセージ自体の内容に基づくものである。これらのメッセージはファイルマネージャオブジェクトFMO３０２０でディスパッチャ３００に受信される。

ステップ８１０４で要求がボリュームオブジェクトへ送られるタイプである場合は、ルーチンは図８ｄのステップ８１１０へ進む。ステップ８１１０では、ボリュームオブジェクトからのボリューム制御ブロックポインタがリトリーブされる。エージェント制御ブロックポインタはボリューム制御ブロックからリトリーブされる（ステップ８１１１）。ステップ８１１２では、要求に対して共通の前処理が行われる。この前処理には、メッセージが処理される前に実行可能であり、かつボリュームフォーマットの個々の実現にかかわらず同じ方法で行われる、つまりファイルシステムフォーマットから独立している、何らかの動作が含まれている。一実施例では、これにはメッセージ内の少数のフィールドの有効性の検証、メッセージ構造のトレーラ（追跡）パートのフィールドの書き込み、およびコーラまたはインタフェースモジュールに提供された何らかの大容量バッファの現在のアドレス空間へのマッピングが含まれている。

ステップ８１１３では、エージェント制御ブロック内のObjectIDを使用して要求が適切なフォーマットエージェントへ送られる。これには要求タスクステート構造３０５０において送られたフラグを、要求が送られたことを示すようにセットすることが含まれる。ステップ８１１４では、その要求に対して共通の後処理動作が行われる。この後処理には、要求が処理された後で実行可能であり、かつボリュームフォーマットの個々の実現にかかわらず同じ方法で行われる、つまりファイルシステムフォーマットから独立している、何らかの動作が含まれている。これには前処理ステップ８１１２で行われた、いずれかのマッピングのクリーンアップ、連続的にメッセージを処理した結果として生じたいずれかのイベントをポストすること、およびファイルマネージャの性能に関する統計要素（スタティスティクス）を収集することが含まれる。

ステップ８１１５では、その要求がうまく処理されたかかどうかが判定される。そうである場合は、ディスパッチャはその要求に対し適切なイベントを作成する。これらを検出し扱うイベントマネージャは、またイベントは上記の参照した出願中である「オペレーティングシステムのイベント情報の分配方法および装置」に記載されている。その要求がうまく処理されない場合は、ディスパッチャはフォーマットエージェントから受信した応答を使用して発信元のインタフェースモジュールへ応答を送る。そしてルーチンは図８ｂのＣへループする。

ステップ８１０６で、その要求がパスオブジェクトへ送られるタイプである場合は、ルーチンは図８ｅのステップ８１２０へ進む。ステップ８１２０では、ボリューム制御ブロックポインタがパスオブジェクトからリトリーブされる。エージェント制御ブロックポインタがパス制御ブロックからリトリーブされる（ステップ８１２２）。ステップ８１２２～８１２７は上述の図８ｄのステップ８１１２～８１１７と同じであるので、ここで説明は繰り返さないこととする。

ステップ８１０８で要求がイタレータオブジェクトへ送られるタイプである場合は、ルーチンは図８ｆのステップ８１３０へ進む。ボリューム制御ブロックポインタがイタレータオブジェクトからリトリーブされ（ステップ８１３０）、エージェント制御ブロックポインタがイタレータ制御ブロックからリトリーブされる（ステップ８１３１）。ステップ８１３２では、その要求に対して共通の前処理動作が行われ、その要求はステップ８１１２、８１１３でそれぞれ上述したようにエージェントへ送られる（ステップ８１３３）。

ステップ８１３５では、イタレータを異なるボリュームへ移動する必要があるかどうかが判定される。そうである場合は、共用アクセスのためにマスタおよびエージェント制御ブロックセマフォが獲得され（ステップ８１３６）、イタレータ制御ブロックは新たなボリ

ューム制御ブロックへ移動される（ステップ８１３７）。一実施例では、これは最初にイタレータが移動される前に指していたボリュームを制御するフォーマットエージェントにメッセージを送り、ついでイタレータが移動された後で指しているボリュームを制御するエージェントにメッセージを送ることにより達成される。そしてマスターセマフォが開放され（ステップ８１３８）、ステップ８１３３で要求が送られる。ステップ８１３５でイタレータを移動する必要がない場合は、ディスパッチャは図８ｄのステップ８１１４～８１１７とそれぞれ同じである、図８ｇのステップ８１３９～８１４２を実行する。

ステップ８１５０で、その要求がマルチプルボリューム要求である場合は、ルーチンは図８ｈのステップ８１６０へ進む。ディスパッチャに受信されたメッセージに基づいて要求により影響される複数のボリュームが決定される（ステップ８１６０）。それらのボリュームに対するボリューム制御ブロックを指すポインタがリトリーブされ（ステップ８１６１）、それらのボリューム制御ブロックに対するエージェント制御ブロックを指すポインタがリトリーブされる（ステップ８１６２）。

ステップ８１６３では、それらのボリュームへアクセスするよう要求されてりるフォーマットエージェントが同じであるかどうかが判定される。そうである場合は、ステップ８１１２～８１１７にそれぞれ対応するステップ８１６４～８１６９が実行される。

エージェントが同じでない場合は、処理は図８ｉのステップ８１７０へ進む。ステップ８１７０で共通の前処理動作が行われる。ある要求が第１のボリュームに対して作成され（ステップ８１７１）、その要求は第１のボリュームに対するフォーマットエージェントへ送られ（ステップ８１７２）、ある要求が第２のボリュームに対して作成され（ステップ８１７３）、その要求が第２のボリュームに対するフォーマットエージェントへ送られる（ステップ８１７４）。ステップ８１７５では、その要求が完全かどうかが判定される。そうでない場合は、ルーチンはステップ８１７１へループする。要求が完全である場合は、ステップ８１６５～８１６９に対応するステップ８１７６～８１８０が実行される。

ステップ８１５１で、その要求がディスパッチャに単独で扱われるタイプである場合は、処理は図８ｊのステップ８１８１へ進む。特にある要求については、ディスパッチャはエージェントからの応答を受信することなく、その要求を扱うのに十分な情報を入手することができる。この場合、前処理動作が行われ（ステップ８１８１）、メッセージに対する応答が作成される（ステップ８１８２）。図８ｄのステップ８１１４～８１１７に対応するステップ８１８３～８１８６が実行される。

ステップ８１５２で、その要求がマウント要求である場合は、ルーチンは図８ｋのステップ８１９０へ進む。ステップ８１９０で共通の前処理動作が行われ、「マウント可能か」というメッセージが作成され（ステップ８１９１）フォーマットエージェントへ送られる。ステップ８１９２では、メッセージが第１のフォーマットエージェントへ送られる。ステップ８１９３では、フォーマットエージェントがボリュームをマウントできるかどうかが判定される。そうでない場合は、最後のメッセージが送られてからエージェントのリストが変更されたかどうかが判定される（ステップ８１９４）。このテストは、ディスパッチャがその要求を処理している間に、あるエージェントが削除または付加された場合に生ずる可能性がある、希な状況を扱うものである。ステップ８１９３の判定が肯定である場合、ルーチンはステップ８１９２へループバックして再びポーリングを開始する。そうでない場合は、ルーチンはステップ８１９５へループしてメッセージを次のエージェントへ送る。

ステップ８１９３でエージェントがボリュームをマウント可能であった場合は、ステップ８１９７が実行される。図８ｄのステップ８１１３～８１１７はステップ８１９７～８２０１に対応する。

ステップ８１５３で、その要求が分解(resolve)パスネーム要求である場合は、ルーチンは図８ｍのステップ８２１０へ進む。ステップ８２１０では共通の前処理動作が行われる。分解パスネームの場合は、ディスパッチャによって、アクセスされているボリュームに対して不完全で部分的なスペシフィケーションが受信される。ディスパッチャは、その部分的なスペシフィケーションからボリュームを決定する（ステップ８２１１）。エージェ

ント制御ブロックがボリューム制御ブロックからリトリーブされ（ステップ８２１２）、その要求がエージェント制御ブロック内のObjectIDを使用してそのエージェントへ送られる（ステップ８２１３）。ステップ８２１５では、エージェントがスペシフィケーションを分解するのに成功して、そのボリュームに対して有効なスペシフィケーションを作成したかどうかが判定される。そうでない場合は、フォーマットエージェントからの応答で返されたスペシフィケーションから決定されるパス内の次のボリュームを処理するために、ルーチンはステップ８２１１へループする。スペシフィケーションがエージェントによって十分に分解された場合は、図８ｄのステップ８１１４～８１１７に対応するステップ８２１５～８２１８が実行される。

メッセージ構造

本発明の一実施例によれば、メッセージはインタフェースモジュールとディスパッチャ間、ディスパッチャとフォーマットエージェント間で通信を行うのに使用される。メッセージは処理されている要求に応じて変化する所定の構造を使用して作成される。この構造の定義に関する一実施例は付録(Appendix)Ａ（「FSRequests.h」）で提供されている。さらに、いくつかの構造を以下に説明する。付録Ｂ（「FSTypes.h」）には、付録Ａに記載のメッセージ構造で使用されている定数とタイプの定義が含まれている。

定義

次の説明ではメッセージ構造で使用されているいくつかのオブジェクトとコンセプトに関する定義を挙げることにする。これらの定義は説明の目的で提供され、本発明の開示を理解すれば当業者であれば変更することが可能であろう。

FSSpecificationは特定のボリューム、ファイル、またはディレクトリのスペシフィケーションである。FSSpecificationはボリュームID、ディレクトリIDおよびネーム（名称）を含んでいる。１つの例外を除くと、特定のボリューム、ファイル、またはディレクトリを必要とする全てのファイルマネージャ要求は、正確にこの形式（パスネームが入らない）でFSSpecificationを参照する。この１つの例外とは、FSSpecificationの有効性要求に従わない処理に対するスペシフィケーションを受け入れるResolvePathname要求である。

FSIteratorは、オブジェクト上、特にボリューム、ファイルおよびディレクトリ上を反復（イタレート：iterate）するためのメカニズムである。FSIteratorはいずれのボリューム、ファイルまたはディレクトリにも位置することが可能で、そして現在参照しているいずれの親オブジェクト、次の兄弟オブジェクト、先の兄弟オブジェクト、第一の子供オブジェクト、次の子供のオブジェクトにも再度位置することが可能な構成である。またFSIteratorは現在参照しているオブジェクトに対するFSSpecificationを含んでおり、FSIteratorを必要とするいずれのコール（要求元）へも送ることが可能である。

FSIteratorはより精巧なFSSpecificationとして使用することができ、そこではポインタFSIteratorはFSSpecification構造を記述するのに用いられるが、ポインタの実際の価値は、ポインタが指すエンティティ間の関係に基づいてファイルシステム内を移動することができるという事実から生じている。その最も簡潔な形式で、ポインタは、現在指しているエンティティといくらかの関係があるファイルシステム内で、最初および／または最後のエンティティへと移動することができる。従って、あるディレクトリを指定しているFSIteratorが与えられると、それは現在のディレクトリを親ディレクトリとして有する最初／最後のファイルまたはディレクトリを指定するよう移動することができる。同様に、FSIteratorはある他のエンティティと同じ関係を持つ、前後のエンティティを指定するよう移動することができる。こうして、あるファイルまたはディレクトリを指定しているFSIteratorが与えられると、それは同じ親ディレクトリを有する次／先のファイルまたはディレクトリを指定するよう移動することができる。

ファイルシステム内でFSIteratorを段階的に移動することはしばしば極めて有益であるが、より強力なFSIteratorのコールは、単一のコールでFSIteratorを移動のパターン内を移動させ、処理に際して通過する各エンティティからの情報を（属性の形式で）収集するのに使用することができる。こうして、単一のコールで、同じ親ディレクトリを有する全てのファイルから属性を収集するのにFSIteratorを使用することができる。

アイデンティティはファイルマネージャ特権の主題（サブジェクト）である。アイデンティティは個別、または個々のアイデンティティのセット又は別々のセットとなり得る。アイデンティティは特定ファイルおよびディレクトリ上の特定動作を行う許可を与えられる。

特権はアイデンティティがファイルまたはディレクトリに対して何をすることができるかを特定するものである。各ファイル及びディレクトリは、アイデンティティのリストや、そのアイデンティティに関連した特権を持っている。PrivilegeIteratorはファイルまたはディレクトリに対するアイデンティティ／特権ペア上で反復するメカニズムである。全てのボリューム、ディレクトリおよびファイルは、PrivilegeIteratorに関するリストを有し、これは、そのボリューム、ディレクトリまたはファイル上のどんな要求が特定のアイデンティティに代わって実行を許可されているかを規定している。これらのアクセス制御リストは、ファイルシステムに対するアイデンティティに許可されている特権を規定する。

APIは、コーラが常に１度にいくつかの属性値をグループで獲得し、セットしていると想定している。単一の属性を獲得してセットするのは退化したケースである。コーラが属性のグループを識別して獲得またはセットするよう要求される際はいつでも、BufferDescriptor（データを保持するためのもの）、そして転送される属性と、データが格納された、あるいはこれから格納されるバッファの位置の両方を特定するFSAttributeDescriptor列を指すポインタを提供することが求められる。

コーラが標準ストリームI/Oアクセス方法を使用してアクセスパスをオープンすると、その作成されたアクセスパスで実行される動作の種類と、その新たなパスがオープンしたままである場合に他のアクセスパスで実行が許可される動作の種類とを記述するAccessConstraint構造を提供するよう求められる。他のアクセス方法はこの構造あるいは、これと同等の構造を使用することができる。また、時にはアクセス方法自身によってアクセス制限が示される。

要求（リクエスト）

このセクションでは本発明の一実施例に基づく全ての要求を説明する。一実施例では、これらの要求の全ては同期及び非同期の形式を有している。

ファイルシステム・エンティティ要求

次の要求はFSSpecificationsによって識別可能なエンティティを操作(manipulate)するのに使用される。すなわちファイル、ディレクトリ、そしてボリュームである。FileCreateは新たなファイルを作成し、DirectoryCreateは新たなディレクトリを作成するものである。AttributesGet、AttributesSet、AttributesGetSizeはファイル、ディレクトリおよびボリュームに関連していた属性を操作するのに使用される。FileFlushはメモリにバッファされているファイルに関する全ての情報を、そのファイルが存在するボリューム上に強制的に書き出すのに使用される。これにはその属性、格納割り当て等の全てのメタ情報が含まれている。TestEntityは、FSSpecificationとFSSearchCriteriaを受けて、スペシフィケーションにより参照されるものが基準に合うかどうかについてのブーリアン(boolean)表現を返す。EntityTypeは、FSSpecificationによって参照されているものの種類（ファイル、ディレクトリまたはボリューム）と、それがシステムに対しある特別な重要性を持つことを示すいくつかのフラグを単に返す。

ボリューム要求

次の要求はボリュームを操作するのに使用される。VolumeMountは、既知のデバイス上のボリュームをマウントするための通常の要求である。先にマウントされたボリュームは、VolumeRemount要求を使用して再度マウントすることができる。VolumeRemountを使用するには、コーラはまずVolumeGetRemountDataをコールし、再マウントを行うのに必要な情報を収集させなければならず、VolumeGetRemoutSizeを使用して、VolumeGetRemountDataが返すデータのサイズを得なければならない。このVolumeRemountはデバイスに対するボリュームの関係を想定するため特に有用である。

ボリュームは、通常ボリュームの全情報をファイルマネージャから削除するVolumeUnmoun

t要求を使用してアンマウントされる。VolumeEjectは同様の要求であるが、デバイスをシステムから排除させるものでもある。VolumeOffilineは、ファイルマネージャがボリュームの存在（その上のファイルはオープンの状態のままである、等）を記憶しているという点でわずかに異なっているが、しかしファイルマネージャは、ある期間利用不可能なボリュームに対して準備をする。オフラインのボリュームがアクセスされる場合、ファイルマネージャはボリュームを再び利用可能にしようと試みる。

VolumeFlushは、メモリにバッファされているあるボリュームに関する全てのメタ情報（ディレクトリ構造等）を、そのボリュームが存在するデバイスへ強制的に書き出すのに使用される。VolumeCreateは、デバイスに空のボリュームを作成する(initialize)のに使用される。VolumeGetCapabilityは、ある特定のボリュームがサポートすることができる要求を知るのに使用される。

フォーク要求

次の要求はファイル内のフォークを操作するのに使用される。ForkCreateはファイルにフォークを付加するのに使用され、ForkDeleteはフォークを削除するのに使用される。ForkAttributesGet、ForkAttributesSet、ForkAttributesGetSizeはフォークに関連した属性を操作するのに使用される。最後に、ForkFlushはメモリにバッファされているフォークからの全ての情報を、そのフォークが存在するボリュームへ強制的に書き出すのに使用される。

ファイルシステムイタレータ要求

次の要求は、ファイルシステム内のファイル、ディレクトリおよびボリューム上の情報を発見して収集するようにファイルシステムのイタレータを操作するのに使用される。FSIteratorCreateはイタレータを作成するのに使用され、FSIteratorDisposeは、１つのイタレータを処分(dispose)するのに使用される。Iterateは、ファイルシステムでFSIteratorを１度に１ステップ移動するのに使用される。Searchは、与えられた基準に適合するファイル、ディレクトリ、またはボリュームを探しながら、ステップのパターンを通じてFSIteratorを移動するのに使用される。AttributesGetBulkは、通過する各ファイル、ディレクトリ、またはボリュームに対する属性値のセットの値を収集しながら、ステップのパターンを通じてFSIteratorを移動するのに使用される。AttributesGetBulkFilteredも同様の要求であるが、与えられた基準に合うそれらエントリーから属性値を収集するだけである。

ストリームアクセス方法要求

次の要求は標準ストリームI/Oアクセス方法のために、アクセスパスをオープン、クローズおよびアクセスパスを用いるのに使用される。PathOpenはアクセスパスをオープンするのに使用され、PathCloseはアクセスパスをクローズするのに使用される。PathReadとPathWriteはフォークデータの部分を読み書きするのに使用される。

ストリームI/Oアクセス方法を使用してオープンされた各アクセスパスは、そのステートの中にフォークの現在の位置を含んでいる。そのためForkPositionDescriptorは、その位置に関連付けることができる。この現在位置は、PathReadおよびPathWriteによって読み出された、あるいは書き込まれたセクションの末尾に暗黙的にセットされるか、あるいはPathSetPositionコールを使用して明示的にセットされる。この位置の現在値は、PathGetPositionコールを使用して得ることができる。

コーラはストリームI/Oアクセス方法を使用してオープンしたいずれのフォーク内の範囲をロックすることも可能である。この範囲は排他的（どのオーバーラップする範囲もロックされないようにする）アクセスのために、あるいは共用（オーバーラップする範囲が排他的アクセスのためにロックされないようにする）アクセスのためにロックすることが可能である。PathLockRangeは、この範囲をロックするのに使用され、PathUnlockRangeは、その範囲のロックを解除するのに使用される。

最後の要求のいくつかは、そのフォークに割り当てられた記憶装置を操作するのに使用される。PathGetEOFはフォークの現在の長さをバイトで返し、PathSetEOFはその長さをセットする。EOFを通じてフォークに割り当てられたいずれの記憶内容は、フォークへの全て

のアクセスパスがクローズされればただちに解放および再利用することができる。PathAllocateは記憶装置をフォークに割り当てるのに使用され、PathReleaseは記憶装置の割り当てを解除するのに使用される。これらの要求の両者とも範囲を決めるための引数を取り、これによりメモリ空間において間を空けて割り当てることができるファイルシステム上で、非連続的な割り当てを形成するのに使用することができる。PathWriteも、書き込みが割り当てられていない領域になされた場合に、そのフォークに記憶装置を割り当てるようにし、上述のように記憶装置はそのフォークへの全てのアクセスパスがクローズされた時に解放される。

バッキングストアアクセス方法要求

次の要求はメモリにマップファイルアクセス方法に関するアクセスパスを操作するのに使用される。BackingStoreCreateは、アクセスパスをオープンするのに使用され、BackingStoreDisposeは、アクセスパスをクローズするのに使用される。メモリマップアクセスパスは、メモリの読み出しおよび書き込みによって、そしてオペレーティングシステムのカーネルメモリマネージャに対する要求によって操作されるため、メモリマップアクセスパスを使用するための要求は存在しない。

ファイルクラス要求

次の要求はボリュームに対するファイルクラス情報を操作するのに使用される。FileClassAttributesGet、FileClassAttributesSet、およびFileClassAttributesGetSizeは、ファイルクラスに関連した属性にアクセスするのに使用される。FileClassIteratorCreateは、ボリュームに知られている全てのファイルクラスを移動できるイタレータを作成するのに使用され、FileClassIteratorDisposeはそのようなイタレータを処分するのに使用される。FileClassIterateは、イタレータを、ボリュームに知られているファイルクラスのリスト中を実際に移動するのに使用される。

特権要求

次の要求はファイル、ディレクトリ、およびファイルシステム内のボリュームに関連した特権を操作するのに使用される。PrivilegeGetは、アイデンティティが与えられると、そのアイデンティティが、与えられたファイルや、ディレクトリまたはボリュームに関して有している特権を返す、基本的な要求である。PrivilegeGetは、与えられたアイデンティティが、異なる特権を持つ可能性のあるグループのアイデンティティのメンバであるかどうか見るためにグループを評価することはしない。PrivilegeGetNativeは、ある特定のファイル、ディレクトリ、またはボリュームに対するアクセス制御リストの表現に依存するボリューム形式を得るのに使用される。そしてPrivilegeSetNativeは、そのデータを異なるファイル、ディレクトリまたはボリュームに適用するのに使用される。

PrivilegeIteratorCreateは、ある特定のファイル、ディレクトリまたはボリュームに対するアクセス制御リストを移動することが可能なイタレータを作成するのに使用される。そしてPrivilegeIteratorDisposeは、そのイタレータを処分する。PrivilegesIterateは、イタレータを１度に１ステップずつアクセス制御リストを通して移動するのに使用され、PrivilegeSearchはある特定のアイデンティティに合うエントリを探しながらイタレータをリストを通じて移動するのに使用される。あるイタレータが与えられると、PrivilegeAddは、新たなエントリをアクセス制御リストに挿入し、PrivilegeDeleteはエントリを削除し、PrivilegeChangeは現在のエントリを変更する。

その他の要求

次の要求はファイルマネージャ上でその他の動作を行うのに使用される。GetFileSystemVersionは、システム上で動いているファイルマネージャのバージョン数を返すものである。PathnameResolveは、関係するパスネームを取りそれを標準のFSSpecificationに分解する。FSDeleteはファイルまたはディレクトリを削除する。FSMoveRenameはファイルまたはディレクトリの親ディレクトリとネームを変更する。FSCopyRenameは、ファイルのコピーを作成し、その処理においてオプションで別の名前を付ける。

発明の利点

エンドユーザは本発明に係るファイルマネージャから少なくとも２つの大きな利益を得る

であろう。第１は、全体のシステムで認識できる性能の向上である。同時実行可能によって、特にそれを利用するアプリケーションに関して、ファイルマネージャのスループットが向上する。そしてファイルマネージャをコールするバックグラウンド処理がフォアグラウンド性能に及ぼす影響はより小さいものとなる。
第２に、システム全体がより強固なものになる。本発明に係るファイルマネージャは、メッセージングシステムを利用してそのデータ構造を保護し、そのアプリケーションとシステム自体の間をより良く隔離させる。さらに、提供されたアーキテクチャ上の特徴によって、異なるファイルシステムや他のコードに対し、それらの機能性を提供するためにいくらかの危険をおかさなければならない、現在の必要性を排除することができる。
一実施例によれば、メッセージを使用してファイルマネージャ要求を作成するファイルマネージャに新たなインタフェースが提供される。このインタフェースは２つの点で有用である。第１には、それはトラップよりもメッセージングを使用するということ、第２には、要求の構造は従来のパラメータブロックに類似していないということである。メッセージプロトコルは、ファイルマネージャと通信を行うのに使用される本発明に従って規定される。共用ライブラリは、これらのメッセージをフォーマットしたり送ったり、また応答を扱う機能を持つ。メッセージインタフェースは、本発明に係るファイルマネージャによって、また他のシステムソフトウェアに関連したファイルマネージャによって内部的に使用されるサービスである。特に、本発明に係るファイルマネージャは、特定のディスクファイルシステムフォーマット／サーバプロトコルを実行するフォーマットエージェントと通信するのに、この同じインタフェースを使用する。
ファイルマネージャの一実施例によれば、Power PCマシン上のネイティブモードで動作することができる。この実施例によると、ファイルマネージャはアセンブリ言語コードをほとんど含まず、非RISCプロセッサ上で動作しているという想定を行うことがない。コードはバイトの順に依存せずに作成される。
ファイルマネージャの一実施例に基づきアクセスが解除される内部インタフェース／構造のほとんどは、本質的に危険なものである。従って本発明に係るファイルマネージャの利点の一つは、開発者の使用のために、これらのインタフェースをより安全なインタフェースに置き換えることである。こうして、本発明に係るファイルマネージャは、システムの利用可能な機能性を減少させることなく、むしろ、より安全に、そして他に動作しているソフトウェアに影響を及ぼすことなく使用できる再設計されたインタフェースを提供するものである。
本発明に係るファイルマネージャは、多数の異なるファイルシステム（現在と未来の）とのインターフェースを行い、これらの各ファイルシステムからの、合理的に可能な限り多くの機能性へのアクセスを提供するのに使用できる、高性能なアブストラクト・モデルを提供するよう設計されたファイルシステムモデルに基づいて構築される。
本発明の一実施例に係るファイルマネージャは同時性を提供する。既知のファイルマネージャの多くは１度に１つのファイルシステム要求しか処理することができない。ファイルシステム要求は、ファイルシステムに関するデータにアクセスする要求である。これは通常、特定のボリュームフォーマットに基づいて組織された記憶媒体上のデータへのアクセスを必要としている。先の要求が処理されている間に提出される他の要求は、キューに配列され後、１度に１つずつ処理される。これは特に後に控える要求が速度の遅い媒体に対するものである場合には煩わしいものである。
１度に１つのアプリケーションしか動作せず、一般に単一のフォアグラウンド・アプリケーションによってしかファイルマネージャ要求がされなかった初期のコンピュータシステムにおいては、このアプローチは合理的である。しかし、今日多くのコンピュータ上では、アプリケーションや初期化（起動）ルーチンなどが同時に動作する多くのサービスが存在しており、これらのサービスの多くはバックグラウンドでファイルマネージャ要求を発行する。フォアグラウンド・アプリケーション内で、いくつものタスクにファイルマネージャ要求を並行して発行することを許可する、NuKERNEL（TM）などのメッセージングシステムの出現によって、複雑性が増している。

これらの理由により、本発明に係るファイルマネージャは、１度に１つ以上の要求で同時発生する処理をサポートする。このサポートを達成するために、コードは完全に再エントリ可能であり、マルチタスキングを利用するものである。
従来のファイルマネージャは、初期の世代のパーソナルコンピュータの比較的単純な要求を扱うように設計されていた。しかし、それ以来市場価格が大幅に変化し、現在重要となっているが、当時は一時的な利点であったいくつかの特徴を実現するのが困難となっている。ディスクドライブははるかに大きくなり、競合するファイルシステムを搭載することがより重要であり、ネットワーキングは不可欠であり、そして先行技術のAPIは混乱して複雑である。
上述の特定の実施例の説明は本発明の一般的な特質を十分に明らかにしているため、他者が現在の知識を適用することにより、容易に変更が可能であり、そして／または、そうした特定の実施例を様々なアプリケーションに対して総括的な概念から離れることなく適用することが可能である。従って、そうした適用および変更は開示された実施例に対応する意味と範囲内で理解されるよう意図されるべきであり、またそのように意図されるものである。ここで使用される語句の表現は説明のためのものであり限定する目的で使用されているものではないことが理解されるであろう。

APPENDIX A

"FSRequests.h"

```
1  /*
2    File:       FSRequests.h
3
4    Contains:   Contains definitions for all request structures which
5                are part of the File Manager Request Interface
6
7    Copyright:  © 1992-1994 by Apple Computer, Inc., all rights reserved.
8
9    */
10
11 #ifndef __FSREQUESTS__
12 #define __FSREQUESTS__
13
14 #ifndef __FSTYPES__
15 #include <FSTypes.h>
16 #endif
17
18 #ifndef __FSREQUESTPRIV__
19 #include <FSRequestPriv.h>
20 #endif
21
22 /*--------------------------------------------------------------------------*/
23 /* CONSTANTS:                                                               */
24 /*--------------------------------------------------------------------------*/
25
```

```
/*
 * FSRequestAttributes are used to specify variations
 * or other attributes of requests.  These values should be OR'ed
 * into the prmCommandID field in the request's header.
 */
enum {
  kFRAAsynchronous          = 0x40000000,    // an asynchronously request
                                             // Applies to all requests

  kFRAPathOpen_LoadMap      = 0x01000000,    // preload and lock in memory
                                             // all data for file to volume
                                             // block mapping.
                                             // Applies to kPathOpen requests,

  kFSRequestVariantMask     = 0x00ffffff     // used to clear off the above bits.
};

/*
 * The Message type(s) to use when sending a FileSystem request
 */
enum {
  kUserFSRequest            = 0x00000001     // requests from sources external to the filesystem
};
```

```
/*
 * The actual list of request which can be made.
 */
enum {

  kPathOpen                          = 0x0001,
  kPathClose                         = 0x0002,
  kPathRead                          = 0x0011,
  kPathWrite                         = 0x0012,
  kPathLockRange                     = 0x0013,
  kPathUnlockRange                   = 0x0014,
  kPathGetPosition                   = 0x0015,
  kPathSetPosition                   = 0x0016,
  kPathGetEOF                        = 0x0017,
  kPathSetEOF                        = 0x0018,
  kPathAllocate                      = 0x0019,
  kPathRelease                       = 0x001A,

  kForkCreate                        = 0x0101,
  kForkDelete                        = 0x0102,
  kForkAttributesGet                 = 0x0105,
  kForkAttributesSet                 = 0x0106,
  kForkAttributesGetSize             = 0x0107,
  kForkFlush                         = 0x0108,

  kFileCreate                        = 0x0201,
  kDirectoryCreate                   = 0x0202,

  kAttributesGet                     = 0x0305,
  kAttributesSet                     = 0x0306,
  kAttributesGetSize                 = 0x0307,
  kFileFlush                         = 0x0308,
  kTestEntity                        = 0x0311,
  kEntityType                        = 0x0312,

  kVolumeMount                       = 0x0401,
  kVolumeUnmount                     = 0x0402,
  kVolumeFlush                       = 0x0408,
  kVolumeCreate                      = 0x0411,
  kVolumeGetCapability               = 0x0412,
  kVolumeEject                       = 0x0413,
  kVolumeOffline                     = 0x0414,
  kVolumeGetRemountSize              = 0x0415,
  kVolumeGetRemountData              = 0x0416,
  kVolumeRemount                     = 0x0417,

  kFileClassIteratorCreate           = 0x0501,
  kFileClassIteratorDispose          = 0x0502,
  kFileClassAttributesGet            = 0x0505,
  kFileClassAttributesSet            = 0x0506,
```

```
    kFSNotificationGetEvent     = 0x0813,   kFileClassAttributesGetSize = 0x0507,
                                            kFileClassIterate       = 0x0509,
    kExtendedVolumeMount        = 0x0901,
    kExtendedVolumeUnmount      = 0x0902,   kFSIteratorCreate       = 0x0601,
    kExtendedVolumeRequest      = 0x090A,   kFSIteratorDispose      = 0x0602,
    kExtendedRequest            = 0x090B,   kIterate                = 0x0609,
    kExtendedPathOpen           = 0x0911,   kSearch                 = 0x060A,
    kExtendedPathClose          = 0x0912,   kAttributesGetBulk      = 0x0611,
    kExtendedPathRequest        = 0x091A,   kAttributesGetBulkFiltered = 0x0612,

    kBackingStoreCreate         = 0x0A01,   kPermissionIteratorCreate = 0x0701,
    kBackingStoreDispose        = 0x0A02,   kPermissionIteratorDispose = 0x0702,
                                            kPermissionsIterate     = 0x0709,
    kFSDelete                   = 0x0F02,   kPermissionSearch       = 0x070A,
    kFSExchange                 = 0x0F11,   kPermissionAdd          = 0x0711,
    kFSMoveRename               = 0x0F12,   kPermissionDelete       = 0x0712,
    kFSCopyRename               = 0x0F13,   kPermissionChange       = 0x0713,
    kDirectoryGetCurrent        = 0x0F14,   kPermissionGet          = 0x0714,
    kDirectorySetCurrent        = 0x0F15,   kPermissionGetNative    = 0x0715,
    kFileGetIDFromFSSpec        = 0x0F16,   kPermissionSetNative    = 0x0716,
    kFileGetFSSpecFromID        = 0x0F17,
    kPathnameResolve            = 0x0F18,   kFSNotificationQueueCreate = 0x0801,
                                            kFSNotificationQueueDispose = 0x0802,
                                            kFSNotificationSubscribe = 0x0811,
    kGetFilesystemVersion       = 0x0FFF    kFSNotificationUnsubscribe = 0x0812,
};
```

```
151 typedef uint32 RequestCode;
152
153 #define kMaxRequestCode        (kGetFilesystemVersion)
154
155 /*-------------------------------------------------------------------------*/
156 /* TYPES:
157 /*-------------------------------------------------------------------------*/
158
159 /*
160  * Request Header
161  * The first 16 bytes of all File Manager Requests are taken up with the
162  * request header and have the same structure:
163  *
164  * The first 4 bytes are a message header which defines the version of the
165  * message structure used (essentially what these paragraphs describe); and
166  * the length of the entire message.
167  *
168  * The next 4 bytes are a header for the request specific parameter list which
169  * defines the version of the parameter list (the part of the message format
170  * which is different for each request) and the length of that list. The length
171  * is specified by the prmEnd field which gives the offset from the top of the
172  * message to the end of the parameter list (which is also the start of the
173  * Trailer structure).
174  *
175  * The third longword in the request header is the command ID for the request
```

```
176  * which indicates what the user wants to do and defines the structure of the
177  * following parameter list. The upper 8 bits of the CommandID are reserved
178  * for flags and variant indicators.
179  *
180  * The fourth longword is a pointer to a ProgressReport structure. If this value
181  * is not nil, the File Manager used the indicated ProgressReport to indicate
182  * its progress completing the request.
183  *
184  * The fifth longword is reserved for use by the File Manager. It essentially
185  * serves as a 'unique' identifier for the specific request within the system.
186  *
187  * Request Parameters
188  * The middle of each File Manager Request structure is taken up by the actual
189  * parameters for the request This list is different for each kind of request.
190  *
191  * Request Trailer
192  * At the end of every File Manager Requests is the Request Trailer which
193  * has the same format for every request. It consists of 16 bytes which are
194  * reserved for use by the File Manager, followed by a variable length array
195  * of bytes where all of the variable length parameters for the request are
196  * stored.
197  */
198
199 enum {
200   kRequestVersion                 = 0x01,
```

```
    kParameterVersion               = 0x01
};

/*
 * The standard Request Header
 */
typedef struct {
    // the message header
    uint8       msgReserved;        // reserved byte for message, MUST BE 0
    uint8       msgVersion;         // format of the Header and Trailer structs
    uint16      msgLength;          // total length of the message

    // the parameter list header
    uint8       prmReserved;        // reserved byte for parameters, MUST BE 0
    uint8       prmVersion;         // format of the Parameters for this request
    uint16      prmEnd;             // offset from the begining of the MESSAGE
                                    // to the end of the parameter list

    RequestCode prmCommandID;       // the command ID of the request
//** cannot be a pointer in multi-address space world
    ProgressReportPtr   prmProgress;    // if non-nil, how to report progress on request
//**
    uint32      prmRequestID;       // reserved for internal use, MUST BE SET TO 0
} FSRequestHeader, *FSRequestHeaderPtr;
```

```
226 /*
227  * FSRequestPrivate is the private data in a request structure which is
228  * used by the File Manager to complete requests. All of these fields
229  * MUST BE SET TO 0 by the normal user.
230  */
231 typedef struct {
232   void*                 privReserved1;
233   void*                 privReserved2;
234   void*                 privReserved3;
235   void*                 privReserved4;
236   void*                 privReserved5;
237   void*                 privReserved6;
238   void*                 privReserved7;
239   } FSRequestPrivate, *FSRequestPrivatePtr;
240
241 /*
242  * The standard Request Trailer
243  */
244 typedef struct {
245   FSRequestPrivate      msgPrivate;     // the private reserved space in the request
246   } FSRequestTrailer, *FSRequestTrailerPtr;
247
248 /*------------------------------------------------------------------------*/
249 /* REQUESTS:                                                              */
250 /*------------------------------------------------------------------------*/
```

```
251
252 /*
253  *    Some of our requests are variable length. The data in variable portion of
254  *    the message buffer is refered to by an input field in the static portion.
255  *    The input field contains an offset from the beginning of the variable portion
256  *    of the message buffer (i.e. the first datum in the variable portion has an
257  *    offset of 0. The macro M_VaribleDataPtr is used to compute a pointer to
258  *    a specific variable datum (the computed pointer is a Ref, so it needs to be
259  *    cast to the appropriate type.
260  */
261
262 /*------------------------------------------------------------------------------
263  * Basic IO (Path Based Functions)
264  */
265
266 /*
267  * FSReqPathOpen opens a new Access Path to the specified Fork of the
268  * specified File, and return the RefNum of the Path. The Path will be
269  * constructed to have the indicated access rights.
270  */
271 typedef struct FSReqPathOpen {
272     FSRequestHeader     header;     // header.ioCommandID = kPBPathOpen
273     // the target & output
274     struct {
275         PathObjID       thePath;
```

```
276     }                               theReply;
277     // the outputs
278     // the inputs
279     FSSpecification                 theFile;
280     ForkNum                         theFork;
281     IOPermission                    thePermissions;
282     //
283     FSRequestTrailer                trailer;
284     } FSReqPathOpen, *FSReqPathOpenPtr;
285
286
287     /*
288      * FSReqPathClose closes the given Access Path.
289      */
290     typedef struct FSReqPathClose {
291       FSRequestHeader               header; // header.prmCommandID = kPathClose
292       // the target
293       PathObjID                     thePath;
294       // the outputs
295       // the inputs
296       //
297       FSRequestTrailer              trailer;
298     } FSReqPathClose, *FSReqPathClosePtr;
299
300
```

```
301 /*
302  * FSReqPathRead reads data from a Fork into the supplied buffer. The number of
303  * bytes requested is indicated by the buffer description, and the number of
304  * bytes actually read is returned. The argument newPosition is filled in with
305  * the position in the Fork following the last byte read.
306  */
307 typedef struct FSReqPathRead {
308     FSRequestHeader             header; // header.prmCommandID = kPathRead
309     // the target
310     PathObjID                   thePath;
311     // the outputs
312     struct {
313         uint32                  actualLength;
314         ForkPositionDescriptor  newPosition;
315     }
316     BufferDescriptor            theReply;
317     // the inputs
318     ForkPositionDescriptor      theBuffer;      // filled in with InterspaceBlockCopy and/or DMA from device
319     IOOptions                   thePosition;
320     //                          theOptions;
321     FSRequestTrailer            trailer;
322 } FSReqPathRead, *FSReqPathReadPtr;
323
324
325 /*
```

```
 * FSReqPathWrite writes data to a Fork from the supplied buffer. The number of
 * bytes requested is indicated by the buffer description, and the number of
 * bytes actually written is returned. The argument newPosition is filled in
 * with the position in the Fork following the last byte written.
 */
typedef struct FSReqPathWrite {
    FSRequestHeader             header; // header.prmCommandID = kPathWrite
    // the target
    PathObjID                   thePath;
    // the outputs
    struct {
        uint32                  actualLength;
        ForkPositionDescriptor  newPosition;
    }                           theReply;
    // the inputs
    BufferDescriptor            theBuffer;      // moved with InterspaceBlockCopy and/or DMA to device
    ForkPositionDescriptor      thePosition;
    IOOptions                   theOptions;
    //
    FSRequestTrailer            trailer;
} FSReqPathWrite, *FSReqPathWritePtr;

/*
 * FSReqPathLockRange locks the range of bytes starting at the given position
```

```
 * and running for the specified length. It fills in actualStart with the position
 * at which the locked range starts.
 */
typedef struct FSReqPathLockRange {
    FSRequestHeader             header; // header.prmCommandID = kPathLockRange
    // the target
    PathObjID                   thePath;
    // the outputs
    struct {
        ForkPositionDescriptor  actualStart;
    }                           theReply;
    // the inputs
    ForkPositionDescriptor      thePosition;
    int64                       theLength;
    uint32                      options;
    //
    FSRequestTrailer            trailer;
} FSReqPathLockRange, *FSReqPathLockRangePtr;


/*
 * FSReqPathUnlockRange unlocks the range of bytes starting at the given position
 * and running for the specified length.
 */
typedef struct FSReqPathUnlockRange {
```

```
    FSRequestHeader         header; // header.primCommandID = kPathUnlockRange
    // the target
    PathObjID               thePath;
    // the outputs
    // the inputs
    ForkPositionDescriptor  thePosition;
    int64                   theLength;
    uint32                  options;
    //
    FSRequestTrailer        trailer;
} FSReqPathUnlockRange, *FSReqPathUnlockRangePtr;


/*
 * FSReqPathGetPosition gets the current position of the Path in its Fork.
 */
typedef struct FSReqPathGetPosition {
    FSRequestHeader         header; // header.primCommandID = kPathGetPosition
    // the target
    PathObjID               thePath;
    // the outputs
    struct {
        ForkPositionDescriptor  currentPosition;
    }                       theReply;
    // the inputs
```

```
401  //
402  FSRequestTrailer          trailer;
403  } FSReqPathGetPosition, *FSReqPathGetPositionPtr;
404
405
406  /*
407   * FSReqPathSetPosition sets the current position of the Path in its Fork.
408   *
409   */
410  typedef struct FSReqPathSetPosition {
411      FSRequestHeader           header; // header.prmCommandID = kPathSetPosition
412      // the target
413      PathObjID                 thePath;
414      // the outputs
415      struct {
416          ForkPositionDescriptor    actualPosition;    //  New position relative to start of file
417          theReply;
418      }
419      // the inputs
420      ForkPositionDescriptor    newPosition;
421      //
422      FSRequestTrailer          trailer;
423  } FSReqPathSetPosition, *FSReqPathSetPositionPtr;
424
425  /*
```

```
426  * FSReqPathGetEOF gets the logical length of the Fork.
427  */
428  typedef struct FSReqPathGetEOF {
429    FSRequestHeader                header;  // header.prmCommandID = kPathGetEOF
430    // the target
431    PathObjID                      thePath;
432    // the outputs
433    struct {
434      ForkPositionDescriptor       theEOF;
435    }                              theReply;
436    // the inputs
437    //
438    FSRequestTrailer               trailer;
439  } FSReqPathGetEOF, *FSReqPathGetEOFPtr;
440
441
442  /*
443   * FSReqPathSetEOF sets the logical length of the Fork. If the new length is longer
444   * than the old one, the new disk storage is allocated according to theMode.
445   */
446  typedef struct FSReqPathSetEOF {
447    FSRequestHeader                header;  // header.prmCommandID = kPathSetEOF
448    // the target
449    PathObjID                      thePath;
450    // the outputs
```

```
451   // the inputs
452   ForkPositionDescriptor      theEOF;
453   AllocationMode              theMode;
454   uint16                      filler1;    //** because AllocationMode is a uint16
455   //
456   FSRequestTrailer            trailer;
457   } FSReqPathSetEOF, *FSReqPathSetEOFPtr;
458
459
460   /*
461    * FSReqPathAllocate Allocates storage for the Fork access by thePath starting at
462    * theStart and running for at least theLength bytes. The storage is allocated based
463    * on the given allocation mode. This call does not changes the end of the fork, even
464    * if the space allocated in beyond the current end. However any excess storage beyond
465    * the end of the fork is automatically released when the last path on the fork is
466    * closed. The user may also manually release excess storage using the PathRelease request.
467    */
468   typedef struct FSReqPathAllocate {
469     FSRequestHeader           header;  // header.prmCommandID = kPathAllocate
470     // the target
471     PathObjID                 thePath;
472     // the outputs
473     struct {
474       int64                   actualLength;   // actually amount of storage allocated
475     }                         thePathy;
```

```
476   // the inputs
477   ForkPositionDescriptor      theStart;
478   int64                       theLength;
479   AllocationMode              theMode;
480   uint16                      fillerU;    // • because AllocationMode is a uint16
481   //
482   FSRequestTrailer            trailer;
483   ) FSReqPathAllocate, *FSReqPathAllocatePtr;
484
485   /*
486    * FSReqPathRelease allows the system to releases storage allocated to the given
487    * range of a fork. This is only an advisory call, the system may on fact release
488    * less, or even no storage, depending on the actually allocation structure. Note
489    * that while this call is in some respects an inverse of the PathAllocate request,
490    * the two are NOT symetrical. PathAllocate will tend to round allocation up (if they
491    * are not aligned on convenient boundaries), while PathRelease will tend to round
492    * things down under the same circumstances. Therefore it is advisable that
493    * PathRelease requests NOT be made piecemeal, always release the largest range
494    * possible, even if you have alreday released parts of it on previous calls.
495    */
496   typedef struct FSReqPathRelease {
497     FSRequestHeader           header; // header.prmCommandID = kPathRelease
498     // the target
499     PathObjID                 thePath;
500     // the outputs
```

```
    // the inputs
    ForkPositionDescriptor          theStart;
    int64                           theLength;
    //
    FSRequestTrailer                trailer;
    } FSReqPathRelease, *FSReqPathReleasePtr;


/*------------------------------------------------------------------------
 * Cache Manipulation
 */

/*
 * FSReqForkFlush writes out any cache entries associated with the given
 * Path's Fork. This include block which contain Filesystem structural
 * information about the fork (Catalog and Extent Blocks, etc).
 */
typedef struct FSReqForkFlush {
    FSRequestHeader                 header;  // header.pmCommandID = kFSForkFlush
    // the target
    PathObjID                       thePath;
    // the outputs
    // the inputs
    //
    FSRequestTrailer                trailer;
```

```
526   ) FSReqForkFlush, *FSReqForkFlushPtr;
527
528
529   /*
530    * FSReqFileFlush writes out any cache entries associated with the given File.
531    */
532   typedef struct FSReqFileFlush {
533     FSRequestHeader      header; // header.pmCommandID = kFileFlush
534     // the target
535     FSSpecification      theFile;
536     // the outputs
537     // the inputs
538     FSRequestTrailer     trailer;
539   ) FSReqFileFlush, *FSReqFileFlushPtr;
540
541
542   /*
543    * FSReqVolumeFlush writes out any cache entries associated with the given Volume.
544    */
545   typedef struct FSReqVolumeFlush {
546     FSRequestHeader      header; // header.pmCommandID = kVolumeFlush
547     // the target
548     FSSpecification      theVolume;
549     // the outputs
550     // the inputs
```

```
551  //
552  FSRequestTrailer          trailer;
553  } FSReqVolumeFlush, *FSReqVolumeFlushPtr;
554
555
556  #if 0          //** kFSCacheAdvise is not defined, is this still a request?
557  /*
558   * FSReqCacheAdvise performs the specified operation on any cache entries
559   * which are associated with the portion of the Fork indicated by the given
560   * range.
561   */
562  typedef struct FSReqCacheAdvise {
563      FSRequestHeader           header; // header.prmCommandID = kFSCacheAdvise
564      // the target
565      PathObjID                 thePath;
566      // the outputs
567      // the inputs
568      ForkPositionDescriptor    thePosition;
569      int64                     theLength;
570      CacheOperation            theOperation;
571      uint16                    fill[1];        //** CacheOperation is a uint16
572      //
573      FSRequestTrailer          trailer;
574  } FSReqCacheAdvise, *FSReqCacheAdvisePtr;
575  #endif
```

```
576
577
578 /*-----
579  * Filesystem Browsing
580  */
581
582 /*
583  * FSReqIteratorCreate initializes theIterator. If theSymbolicEntity is equal to
584  * one of the 'absolute' iteration operations, theIterator will be created such
585  * that it points to the symbolic entity. For symbolic entities which can reside
586  * on any volume, the volume object ID in theEntity is used if provided,
587  * otherwise the boot volume's object ID is used. For symbolic entities which
588  * only occur once in a system (e.g. kFSIterFontsFolder), theEntity is ignored.
589  * When theSymbolicEntity is NULL, theIterator will be created such that it
590  * points to theEntity. E_MissingParameter is returned when both
591  * theSymbolicEntity and theEntity are NULL. E_ParamErr is returned when
592  * theSymbolicEntity is neither NULL nor an 'absolute' iteration operation.
593  */
594 typedef struct FSReqIteratorCreate {
595     FSRequestHeader             header; // header.prmCommandID = kFSIteratorCreate
596     // the target & output
597     struct {
598         FSIterator              theIterator;
599         theReply;
600     // the outputs
```

```
601   // the inputs
602   FSSpecification              theEntity;
603   FSIterationOperation         theSymbolicEntity;
604   uint16                       filler1;        //** FSIterationOperation is a uint16
605   //
606   FSRequestTrailer             trailer;
607   } FSReqIteratorCreate, *FSReqIteratorCreatePtr;
608
609
610   /*
611    * FSReqIteratorDispose disposes of any internal storage associated with
612    * the given FSIterator.
613    */
614   typedef struct FSReqIteratorDispose {
615     FSRequestHeader            header; // header.ptmCommandID = kFSIteratorDispose
616     // the target
617     FSIterator                 theIterator;
618     // the outputs
619     // the inputs
620     //
621     FSRequestTrailer           trailer;
622   } FSReqIteratorDispose, *FSReqIteratorDisposePtr;
623
624
625   /*
```

```
626  * FSReqIterate moves from oldPosition according to operation.
627  *
628  * When the new position is valid, newPosition is set to the new entity;
629  * otherwise, oldPosition is copied to newPosition. NewPosition can point at the
630  * same iterator passed as oldPosition.
631  */
632 typedef struct FSReqIterate {
633     FSRequestHeader         header; // header.pmCommandID = kIterate
634     // the target
635     FSIterator              oldPosition;
636     // the outputs
637     struct {
638         FSIterator          newPosition;
639         }                   theReply;
640     // the inputs
641     FSIterationOperation    operation;
642     uint16                  filler1;        //** FSIterationOperation is a uint16
643     //
644     FSRequestTrailer        trailer;
645 } FSReqIterate, *FSReqIteratePtr;
646
647
648 /*
649  * FSReqSearch, starting at oldPosition searches for the next entity
650  * which matches the given criteria. The exact route throught the
```

```
651  * filesystem (and spoce of that route) is defined by the operation
652  * parameter. The enity refered to by oldPosition is included in the
653  * search unless the operation is kFSIterContinue, in which case it is
654  * skipped (and the route is specified by the previous call with the
655  * same iterator). Absolute operations are meaningless do to the iterative
656  * nature of search, so an error is returned.
657  *
658  * When the new position is valid (i.e. a match), newPosition is set to the new
659  * entity; otherwise, oldPosition is copied to newPosition. NewPosition can
660  * point at the same iterator passed as oldPosition.
661  */
662 typedef struct FSReqSearch {
663     FSRequestHeader     header;  // header.prmCommandID = kSearch
664     // the target
665     FSIterator          oldPosition;
666     // the outputs
667     struct {
668         FSIterator      newPosition;
669     }                   theReply;
670     // the inputs
671     FSSearchCriteriaPtr criteria;
672     FSIterationOperation operation;
673     uint16              fillerU;
674     //
675     FSRequestTrailer    trailer;
```

```
676  ) FSReqSearch, *FSReqSearchPtr;
677
678
679  /*
680   * FSReqAttributesGet gets a list of attributes of a Filesystem entity.
681   * The field theAttributes is a pointer to a nul terminated array of
682   * FSAttributeDescriptor's. The values of each listed attribute are
683   * copied into the buffer as indicated by the FSAttributeDescriptor
684   * If a value is longer than the give length, a leading subset of the
685   * value is copied and an error is flagged. If the value is sorter,
686   * it is 0-padded at the end and a different error is logged. Locations
687   * in the buffer intended for Attributes which do not exist are left
688   * unchanged by the operation, but an appropriate error is logged.
689   */
690
691  typedef struct FSReqAttributesGet {
692    FSRequestHeader      header; // header.pmCommandID = kAttributesGet
693    // the target
694    FSSpecification      theEntity;
695    // the outputs
696    BufferDescriptor     theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
697    // the inputs
698    FSAttributeDescriptorPtr theAttributes;  // actually an offset into variable portion of message
699    //
700    FSRequestTrailer     trailer;
```

```
} FSReqAttributesGet, *FSReqAttributesGetPtr;

/*
 * FSReqAttributesSet sets a list of attributes for a Filesystem entity.
 * The field theAttributes is a pointer to a nul terminated array of
 * FSAttributeDescriptor's. The values of each listed attribute are
 * copied from the buffer as indicated by the FSAttributeDescriptor
 */

typedef struct FSReqAttributesSet {
  FSRequestHeader       header; // header.pmCommandID = kAttributesSet
  // the target
  FSSpecification       theEntity;
  // the outputs
  // the inputs
  BufferDescriptor      theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
  FSAttributeDescriptorPtr theAttributes;   // actually an offset into variable portion of message
  //
  FSRequestTrailer      trailer;
} FSReqAttributesSet, *FSReqAttributesSetPtr;

/*
 * FSReqAttributesGetSize gets the sizes of a list of attributes for a particular
 * Filesystem entity. theAttributes, describes what attributes to get the sizes of,
 * where in theBuffer to put the values, and how many bytes are allocated to the
```

```
 * values. Any attributes which don't exist are considered to have a size of 0.
 */

typedef struct FSReqAttributesGetSize {
    FSRequestHeader         header;     // header.primCommandID = kAttributesGetSize
    // the target
    FSSpecification         theEntity;
    // the outputs
    BufferDescriptor        theBuffer;      // FSDispatch does CreateAreaForRange and DeleteArea
    // the inputs
    FSAttributeDescriptorPtr    theAttributes;      // actually an offset into variable portion of message
    //
    FSRequestTrailer        trailer;
} FSReqAttributesGetSize, *FSReqAttributesGetSizePtr;

/*
 * FSReqAttributesGetBulk sequences through the filesystem, starting at
 * oldPosition, collecting attributes from each filesystem entity
 * encountered until there are no more entities along the specified route.
 * The exact route throught the filesystem used is defined
 * by the operation parameter.
 *
 * The count parameter is filled in with the number of entities from which
 * Attributes were collected before the request either ran out of room in
 * theBuffer or ran out of entities.
```

```
751  *
752  * This is equivalent to iteratively calling either AttributesGet followed
753  * by a call to Iterate(oldPosition, newPosition, operation). The advantage
754  * is that it saves a lot of overhead work, so throughput is higher.
755  *
756  * TheAttributes is a list of FSAttributeDescriptor's which is terminated with
757  * a descriptor of the form: theAttribute == NULL, theOffset == NUL, and
758  * theLength == sizeof( yourAttributeStructure ). TheBuffer is viewed as an
759  * array of elements. Each element is the size of yourAttributeStructure. This
760  * allows space at the end of yourAttributeStructure which is not described in
761  * theAttributes.
762  *
763  * NewPosition is set to the last entity which was added to theBuffer, or
764  * oldPosition if no entities were found.
765  * NewPosition can point at the same iterator passed as oldPosition.
766  */
767  typedef struct FSReqAttributesGetBulk {
768      FSRequestHeader         header; // header.pubCommandID = kAttributesGetBulk
769      // the target
770      FSIterator              oldPosition;
771      // the outputs
772      struct {
773          FSIterator          newPosition;
774          int32               count;
775                              //** should there be an end of iteration flag
```

```
}
BufferDescriptor                theReply;
// the inputs                   theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
FSAttributeDescriptorPtr        theAttributes;      // actually an offset into variable portion of message
FSIterationOperation            operation;
uint16                          filler1;        //** FSIterationOperation is a uint16
//
FSRequestTrailer                trailer;
} FSReqAttributesGetBulk, *FSReqAttributesGetBulkPtr;


/*
 * FSReqAttributesGetBulkFiltered sequences through the filesystem, starting
 * at oldPosition, collecting attributes from each filesystem entity
 * encountered which meet the given criteria until ther are no more entities
 * along the specified route. The exact route throught the
 * filesystem used is defined by the operation parameter.
 *
 * The count parameter is filled in with the number of entities from which
 * Attributes were collected before the request either ran out of room in
 * theBuffer or ran out of entities.
 *
 * This is equivalent to iteratively calling either AttributesGet followed
 * by a call to Search(oldPosition, newPosition, criteria, operation). The
 * advantage is that it saves a lot of overhead work, so throughput is higher.
```

```
 *
 * TheAttributes is a list of FSAttributeDescriptor's which is terminated with
 * a descriptor of the form: theAttribute == NULL, theOffset == NUL, and
 * theLength == sizeof( yourAttributeStructure ). TheBuffer is viewed as an
 * array of elements. Each element is the size of yourAttributeStructure. This
 * allows space at the end of yourAttributeStructure which is not described in
 * theAttributes.
 *
 * NewPosition is set to the last entity which was added to theBuffer, or
 * oldPosition if no entities were found.
 * NewPosition can point at the same iterator passed as oldPosition.
 */
typedef struct FSReqAttributesGetBulkFiltered {
  FSRequestHeader                  header; // header.prmCommandID = kAttributesGetBulkFiltered
  // the target
  FSIterator                       oldPosition;
  // the outputs
  struct {
      FSIterator                   newPosition;
      int32                        count;
                                                    //** should there be an end of iteration flag
  }                                theReply;
  BufferDescriptor                 theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
  // the inputs
  FSSearchCriteriaPtr              criteria;           // actually an offset into variable portion of message
```

```
826  FSAttributeDescriptorPtr      theAttributes;      // actually an offset into variable portion of message
827  FSIterationOperation          operation;
828  uint16                        filler1;            // FSIterationOperation is a uint16
829  //
830  FSRequestTrailer              trailer;
831  ) FSReqAttributesGetBulkFiltered, *FSReqAttributesGetBulkFilteredPtr;
832
833  /*
834   * FSReqTestEntity compares theEntity against criteria. E_NoError is returned
835   * for a match; otherwise, an appropriate error is returned. This is
836   * essentially a non-iterative version of what is performed by Search and
837   * AttributesGetBulkFiltered
838   */
839  typedef struct FSReqTestEntity {
840    FSRequestHeader             header; // header.prmCommandID = kTestEntity
841    // the target
842    FSSpecification             theEntity;
843    // the outputs
844    // the inputs
845    FSSearchCriteriaPtr         criteria;           // actually an offset into variable portion of message
846    //
847    FSRequestTrailer            trailer;
848  ) FSReqTestEntity, *FSReqTestEntityPtr;
849
850  /*
```

```
851  * FSReqEntityType returns what kind of entity the given FSSpecification refers
852  * to.
853  */
854 typedef struct FSReqEntityType {
855     FSRequestHeader             header; // header.pmCommandID = kEntityType
856     // the target
857     FSSpecification             theEntity;
858     // the outputs
859     struct {
860         FSObjectInfo            theInfo;
861     }                           theReply;
862     // the inputs
863     //
864     FSRequestTrailer            trailer;
865 } FSReqEntityType, *FSReqEntityTypePtr;
866
867
868 /*------------------------------------------------------------------------
869  * Volume Manipulation
870  */
871
872 /*
873  * FSReqVolumeMount mounts the volume on the logical DeviceID, thereafter,
874  */
875 typedef struct FSReqVolumeMount {
```

```
FSRequestHeader                    header;      // header.prmCommandID = kVolumeMount
struct {
    // the target & output
    FSSpecification                theVolume;
    // the outputs
    FSObjectInfo                   theInfo;
}                                  theReply;
// the inputs
ObjectID                           theDevice;  //•• type this when DeviceInterface.h/DeviceTypes.h are stable
MountPermission                    thePermissions;
//•• remove theDrive and partitionIndex when we switch to NuIO
int16                              driveNum;
int16                              driveRefNum;
uint32                             theDrive;
uint32                             partitionIndex;
//
FSRequestTrailer                   trailer;
} FSReqVolumeMount, *FSReqVolumeMountPtr;


/*
 * FSReqVolumeUnmount unmounts the given volume.
 */
typedef struct FSReqVolumeUnmount {
  FSRequestHeader                  header; // header.prmCommandID = kVolumeUnmount
```

```
    // the target
    FSSpecification                 theVolume;
    // the outputs
    // the inputs
    //
    FSRequestTrailer                trailer;
} FSReqVolumeUnmount, *FSReqVolumeUnmountPtr;


/*
 * FSReqVolumeCreate initializes a new Volume. The amount of work
 * which is done is controlled by the operationMask field. The
 * parameter kind indicate the type of filesystem to be created
 * on the volume, and variant indicates which sub-choice of
 * filesystem is to be used. The new volume is mounted after it is
 * created.
 */
typedef struct FSReqVolumeCreate {
    FSRequestHeader                 header; // header.prmCommandID = kVolumeCreate
    struct {
        // the target & output
        FSSpecification             theVolume;
        // the outputs
        FSObjectInfo                theInfo;
    }                               theReply;
```

```
926   // the inputs
927   ObjectID                theDevice;  //** type this when DeviceInterface.h/DeviceTypes.h are stable
928   FileSystemKind          kind;
929   int32                   variant;
930   FSName                  volName;
931   FormatOperationMask     operationMask;
932   uint16                  filler1;        //** FormatOperationMask is a uint16
933   MountPermission         thePermissions;
934   //** remove theDrive and partitionIndex when we switch to HwIO
935   uint32                  theDrive;
936   uint32                  partitionIndex;
937   //
938   FSRequestTrailer        trailer;
939   } FSReqVolumeCreate, *FSReqVolumeCreatePtr;
940
941
942   /*
943    * FSReqVolumeGetCapability fills in the provided structure with
944    * a description of what functionality is actually supported by
945    * the given volume.
946    */
947   typedef struct FSReqVolumeGetCapability {
948     FSRequestHeader       header; // header.prmCommandID = kVolumeGetCapability
949     // the target
950     FSSpecification       theVolume;
```

```
951     // the outputs
952     struct {
953         VolumeFunctionality         theInfo;
954     }                               theReply;
955     // the inputs
956     //
957     FSRequestTrailer                trailer;
958     )FSReqVolumeGetCapability, *FSReqVolumeGetCapabilityPtr;
959
960
961     /*
962      * FSReqVolumeEject ejects the given volume.
963      */
964     typedef struct FSReqVolumeEject {
965         FSRequestHeader                 header; // header.prmCommandID = kVolumeEject
966         // the target
967         FSSpecification                 theVolume;
968         // the outputs
969         // the inputs
970         //
971         FSRequestTrailer                trailer;
972         } FSReqVolumeEject, *FSReqVolumeEjectPtr;
973
974
975     /*
```

```
976   * FSReqVolumeOffline puts the given volume off-line.
977   */
978  typedef struct FSReqVolumeOffline {
979    FSRequestHeader      header;  // header.prmCommandID = kVolumeOffline
980    // the target
981    FSSpecification      theVolume;
982    // the outputs
983    // the inputs
984    //
985    FSRequestTrailer     trailer;
986    } FSReqVolumeOffline, *FSReqVolumeOfflinePtr;
987
988
989  /*
990   * FSReqVolumeGetRemountSize returns the size of the data block which is
991   * required to store enough information to mount the given volume
992   * again in the future. This call is primarily used to enable
993   * remounting of network volumes.
994   */
995  typedef struct FSReqVolumeGetRemountSize {
996    FSRequestHeader      header; // header.prmCommandID = kVolumeGetRemountSize
997    // the target
998    FSSpecification      theVolume;
999    // the outputs
1000   struct {
```

```
    uint32                  theSize;
    )                       theReply;
    // the inputs
    //
    FSRequestTrailer        trailer;
) FSReqVolumeGetRemountSize, *FSReqVolumeGetRemountSizePtr;


/*
 * FSReqVolumeGetRemountData fills in the provided buffer with the data
 * which is required to mount the given volume again in the future.
 */
typedef struct FSReqVolumeGetRemountData {
    FSRequestHeader         header; // header.prmCommandID = kVolumeGetRemountData
    // the target
    FSSpecification         theVolume;
    // the outputs
    BufferDescriptor        remountData;        // FSDispatch does CreateAreaForRange and DeleteArea
    // the inputs
    //
    FSRequestTrailer        trailer;
) FSReqVolumeGetRemountData, *FSReqVolumeGetRemountDataPtr;


/*
```

```
1026  * FSReqVolumeRemount remounts a volume based on information which
1027  * was previously obtained by a GetRemountData call.
1028  */
1029 typedef struct FSReqVolumeRemount {
1030     FSRequestHeader         header;  // header.pbCommandID = kVolumeRemount
1031     struct {
1032         // the target & output
1033         FSSpecification     theVolume;
1034         // the outputs
1035         FSObjectInfo        theInfo;
1036         theReply;
1037     }
1038     // the inputs
1039     BufferDescriptor        remountData;        // FSDispatch does CreateAreaForRange and DeleteArea
1040     //
1041     FSRequestTrailer        trailer;
1042 } FSReqVolumeRemount, *FSReqVolumeRemountPtr;
1043
1044 /*------------------------------------------------------------------------
1045  * Directory Manipulation
1046  * These calls are used to manipulate and control Directories. They can also
1047  * be used on Volumes, in which case they act on the root directory of the
1048  * Volume.
1049  */
1050
```

```
1051 /*
1052  * FSReqDirectoryCreate creates a new Directory whose name and location
1053  * are described by the given FSSpecification.
1054  */
1055 typedef struct FSReqDirectoryCreate {
1056     FSRequestHeader         header; // header.prmCommandID = kDirectoryCreate
1057     // the target
1058     FSSpecification         theDirectory;
1059     // the outputs
1060     struct {
1061         DirectoryNum        theDirectoryNum;
1062     }                       theReply;
1063     // the inputs
1064     //
1065     FSRequestTrailer        trailer;
1066 } FSReqDirectoryCreate, *FSReqDirectoryCreatePtr;
1067
1068
1069 /*------------------------------------------------------------------------
1070  * File Manipulation
1071  */
1072
1073 /*
1074  * FSReqFileCreate creates a new File whose name and location are
1075  * described by the given FSSpecification.
```

```
1076  */
1077  typedef struct FSReqFileCreate {
1078    FSRequestHeader     header;  // header.pbCommandID = kFileCreate
1079    // the target
1080    FSSpecification     theFile;
1081    // the outputs
1082    // the inputs
1083    OSType              fileType;
1084    OSType              fileCreator;
1085    //
1086    FSRequestTrailer    trailer;
1087    } FSReqFileCreate, *FSReqFileCreatePtr;
1088
1089
1090  /*------------------------------------------------------------------------
1091   * Fork Manipulation
1092   */
1093
1094  /*
1095   * FSReqForkCreate create a new fork with the given ID in the indicated File.
1096   */
1097  typedef struct FSReqForkCreate {
1098    FSRequestHeader     header;  // header.pbCommandID = kForkCreate
1099    // the target
1100    FSSpecification     theFile;
```

```
    ForkNum                         theFork;
    // the outputs
    // the inputs
    //
    FSRequestTrailer                trailer;
} FSReqForkCreate, *FSReqForkCreatePtr;

/*
 * FSReqForkDelete disposes of the indicated Fork in the File
 */
typedef struct FSReqForkDelete {
    FSRequestHeader                 header; // header.prmCommandID = kForkDelete
    // the target
    FSSpecification                 theFile;
    ForkNum                         theFork;
    // the outputs
    // the inputs
    //
    FSRequestTrailer                trailer;
} FSReqForkDelete, *FSReqForkDeletePtr;

/*
 * FSReqForkAttributesGet gets a list of attributes of a fork.
 * The field theAttributes is a pointer to a nul terminated array of
```

```
1126  * FSAttributeDescriptor's. The values of each listed attribute are
1127  * copied into the buffer as indicated by the FSAttributeDescriptor
1128  * If a value is longer than the give length, a leading subset of the
1129  * value is copied and an error is flagged. If the value is sorter,
1130  * it is 0-padded at the end and a different error is logged. Locations
1131  * in the buffer intented for Attributes which do not exist are left
1132  * unchanged by the operation, but an appropriate error is logged.
1133  */
1134
1135 typedef struct FSReqForkAttributesGet {
1136     FSRequestHeader     header; // header.pimCommandID = kForkAttributesGet
1137     // the target
1138     FSSpecification     theFile;
1139     ForkNum             theFork;
1140     // the outputs
1141     BufferDescriptor    theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
1142     // the inputs
1143     FSAttributeDescriptorPtr    theAttributes;      // actually an offset into variable portion of message
1144     //
1145     FSRequestTrailer    trailer;
1146 } FSReqForkAttributesGet, *FSReqForkAttributesGetPtr;
1147
1148
1149 /*
1150  * FSReqForkAttributesSet sets a list of attributes for a fork.
```

```
1151  * The field theAttributes is a pointer to a nul terminated array of
1152  * FSAttributeDescriptor's. The values of each listed attribute are
1153  * copied from the buffer as indicated by the FSAttributeDescriptor
1154  */
1155
1156 typedef struct FSReqForkAttributesSet {
1157   FSRequestHeader          header;  // header.prmCommandID = kForkAttributesSet
1158   // the target
1159   FSSpecification          theFile;
1160   ForkNum                  theFork;
1161   // the outputs
1162   // the inputs
1163   BufferDescriptor         theBuffer;
1164   FSAttributeDescriptorPtr theAttributes;     // FSDispatch does CreateAreaForRange and DeleteArea
1165   //                                          // actually an offset into variable portion of message
1166   FSRequestTrailer         trailer;
1167 } FSReqForkAttributesSet, *FSReqForkAttributesSetPtr;
1168
1169
1170 /*
1171  * FSReqForkAttributesGetSize gets the sizes of a list of attributes for a
1172  * particular fork. ttheAttributes, describes what attributes to get the sizes of,
1173  * where in theBuffer to put the values, and how many bytes are allocated to the
1174  * values. Any attributes which don't exist are considered to have a size of 0.
1175  */
```

```
1176
1177 typedef struct FSReqForkAttributesGetSize {
1178   FSRequestHeader              header; // header.pmCommandID = kFSForkAttributesGetSize
1179   // the target
1180   FSSpecification              theFile;
1181   ForkNum                      theFork;
1182   // the outputs
1183   BufferDescriptor             theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
1184   // the inputs
1185   FSAttributeDescriptorPtr     theAttributes;      // actually an offset into variable portion of message
1186   //
1187   FSRequestTrailer             trailer;
1188 } FSReqForkAttributesGetSize, *FSReqForkAttributesGetSizePtr;
1189
1190 /*--------------------------------------------------------------------------
1191  * File Class requests
1192  */
1193
1194
1195 /*
1196  * FSReqFileClassIteratorCreate fills in the referenced FileClassIterator
1197  * with a valid iterator.
1198  */
1199 typedef struct FSReqFileClassIteratorCreate {
1200   FSRequestHeader              header; // header.pmCommandID = kFileClassIteratorCreate
```

```
    // the target
    FSSpecification                 theVolume;
    // the outputs
    struct {
        FileClassIterator           theIterator;
    }                               theReply;
    // the inputs
    FileClass                       initialPoint;   // iterator set great than or equal to initial point
    //
    FSRequestTrailer                trailer;
    } FSReqFileClassIteratorCreate, *FSReqFileClassIteratorCreatePtr;


/*
 * FSReqFileClassIteratorDispose disposes of any internal storage associated with
 * the given FileClassIterator
 */
typedef struct FSReqFileClassIteratorDispose {
    FSRequestHeader                 header; // header.prmCommandID = kFileClassIteratorDispose
    // the target
    FileClassIterator               theIterator;
    // the outputs
    // the inputs
    //
    FSRequestTrailer                trailer;
```

```
1226  ) FSReqFileClassIteratorDispose, *FSReqFileClassIteratorDisposePtr;
1227
1228
1229  /*
1230   * FSReqFileClassAttributesGet gets a list of attributes of a file class.
1231   * The field theAttributes is a pointer to a nul terminated array of
1232   * FSAttributeDescriptor's. The values of each listed attribute are
1233   * copied into the buffer as indicated by the FSAttributeDescriptor
1234   * If a value is longer than the give length, a leading subset of the
1235   * value is copied and an error is flagged. If the value is sorter,
1236   * it is 0-padded at the end and a different error is logged. Locations
1237   * in the buffer intented for Attributes which do not exist are left
1238   * unchanged by the operation, but an appropriate error is logged.
1239   */
1240  typedef struct FSReqFileClassAttributesGet {
1241    FSRequestHeader          header; // header.prmCommandID = kFileClassAttributesGet
1242    // the target
1243    FSSpecification          theVolume;
1244    FileClass                theClass;
1245    // the outputs
1246    BufferDescriptor         theBuffer;      // FSDispatch does CreateAreaForRange and DeleteArea
1247    // the inputs
1248    FSAttributeDescriptorPtr theAttributes;  // actually an offset into variable portion of message
1249    //
1250    FSRequestTrailer         trailer;
```

```
1251  ) FSReqFileClassAttributesGet, *FSReqFileClassAttributesGetPtr;
1252
1253
1254
1255  /*
1256   * FSReqFileClassAttributesSet sets a list of attributes for a file class.
1257   * The field theAttributes is a pointer to a nul terminated array of
1258   * FSAttributeDescriptor's. The values of each listed attribute are
1259   * copied from the buffer as indicated by the FSAttributeDescriptor
1260   */
1261  typedef struct FSReqFileClassAttributesSet {
1262      FSRequestHeader         header; // header.prmCommandID = kFileClassAttributesSet
1263      // the target
1264      FSSpecification         theVolume;
1265      FileClass               theClass;
1266      // the outputs
1267      // the inputs
1268      BufferDescriptor        theBuffer;
1269      FSAttributeDescriptorPtr theAttributes;     // FSDispatch does CreateAreaForRange and DeleteArea
1270      //                                          // actually an offset into variable portion of message
1271      FSRequestTrailer        trailer;
1272  ) FSReqFileClassAttributesSet, *FSReqFileClassAttributesSetPtr;
1273
1274
1275  /*
```

```
1276  * FSReqFileClassAttributesGetSize gets the sizes of a list of attributes for a
1277  * particular FileClass. theAttributes, describes what attributes to get the sizes of,
1278  * where in theBuffer to put the values, and how many bytes are allocated to the
1279  * values. Any attributes which don't exist are considered to have a size of 0.
1280  */
1281
1282 typedef struct FSReqFileClassAttributesGetSize {
1283     FSRequestHeader         header; // header.pmCommandID = kFileClassAttributesGetSize
1284     // the target
1285     FSSpecification         theVolume;
1286     FileClass               theClass;
1287     // the outputs
1288     BufferDescriptor        theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
1289     // the inputs
1290     FSAttributeDescriptorPtr theAttributes;     // actually an offset into variable portion of message
1291     //
1292     FSRequestTrailer        trailer;
1293 } FSReqFileClassAttributesGetSize, *FSReqFileClassAttributesGetSizePtr;
1294
1295
1296 /*
1297  * FSReqFileClassIterate makes theNewClass refer to the same class as
1298  * theOldClass and then moves it according to whichWay. If whichWay
1299  * is equal to kFSClassOpThis, then theNewClass is positioned to refer
1300  * to the class indicated by initialPoint.
```

```
1301  */
1302  typedef struct FSReqFileClassIterate {
1303      FSRequestHeader          header; // header.prmCommandID = kFileClassIterate
1304      // the target
1305      FileClassIterator        oldFileClass;
1306      // the outputs
1307      struct {
1308          FileClassIterator    newFileClass;
1309          theReply;
1310      }
1311      // the inputs
1312      ClassIterationOperation  whichWay;
1313      uint16                   filler1;          //** ClassIterationOperation is a uint16
1314      FileClass                initialPoint;     // only used for kFSClassOpThis
1315      //
1316      FSRequestTrailer         trailer;
1317  } FSReqFileClassIterate, *FSReqFileClassIteratePtr;
1318
1319  /*------------------------------------------------------------------------
1320   * Miscellaneous Filesystem requests
1321   */
1322
1323  /*
1324   * FSReqDelete deletes a filesystem entity (file or directory).
1325   */
```

```
1326 typedef struct FSReqDelete {
1327     FSRequestHeader     header; // header.pmCommandID = kFSDelete
1328     // the target
1329     FSSpecification     theEntity;
1330     // the outputs
1331     // the inputs
1332     //
1333     FSRequestTrailer    trailer;
1334     } FSReqDelete, *FSReqDeletePtr;
1335
1336 /*
1337  * FSReqExchange exchanges the identity of two files.
1338  */
1339 typedef struct FSReqExchange {
1340     FSRequestHeader     header; // header.pmCommandID = kFSExchange
1341     // the target
1342     // the outputs
1343     // the inputs
1344     FSSpecification     File1;
1345     FSSpecification     File2;
1346     //
1347     FSRequestTrailer    trailer;
1348     } FSReqExchange, *FSReqExchangePtr;
1349
1350
```

```
1351 /*
1352  * FSReqMoveRename moves and optionally rename a File or Directory.
1353  * If newName is the "empty" FSName, the old name is preserved.
1354  */
1355 typedef struct FSReqMoveRename {
1356   FSRequestHeader                header; // header.prmCommandID = kFSMoveRename
1357   // the target
1358   FSSpecification                source;
1359   // the outputs
1360   // the inputs
1361   FSSpecification                theNewDirectory;
1362   FSName                         newName;
1363   //
1364   FSRequestTrailer               trailer;
1365 } FSReqMoveRename, *FSReqMoveRenamePtr;
1366
1367
1368 /*
1369  * FSReqCopyRename copies and optionally rename a File or Directory.
1370  * If newName is nil, the old name is preserved.
1371  */
1372 typedef struct FSReqCopyRename {
1373   FSRequestHeader                header; // header.prmCommandID = kFSCopyRename
1374   // the target
1375   FSSpecification                source;
```

```
    // the outputs
    // the inputs
    FSSpecification         theNewDirectory;
    FSName                  newName;
    //
    FSRequestTrailer        trailer;
    } FSReqCopyRename, *FSReqCopyRenamePtr;


    /*
     * FSReqDirectoryGetCurrent gets the current default directory
     * for the application.
     */
    typedef struct FSReqDirectoryGetCurrent {
        FSRequestHeader         header; // header.pmCommandID = kDirectoryGetCurrent
        struct {
            // the target & output
            FSSpecification     directory;
            // the outputs
            FSObjectInfo        theInfo;
        }                       theReply;
        // the inputs
        //
        FSRequestTrailer        trailer;
    } FSReqDirectoryGetCurrent, *FSReqDirectoryGetCurrentPtr;
```

```
/*
 * FSReqDirectorySetCurrent ets the current default directory
 * for the application.
 */
typedef struct FSReqDirectorySetCurrent {
  FSRequestHeader                  header; // header.prmCommandID = kDirectorySetCurrent
  // the target
  FSSpecification                  directory;
  // the outputs
  // the inputs
  //
  FSRequestTrailer                 trailer;
  } FSReqDirectorySetCurrent, *FSReqDirectorySetCurrentPtr;


/*
 * FSReqGetFilesystemVersion gets the version of the currently
 * operating filesystem.
 */
//••
//•• Perhaps this message should be eliminated, and the version is put in the
//•• NameSpace as an attribute of FileManager, and each FileSystem.
//••
```

```
typedef struct FSReqGetFilesystemVersion {
    FSRequestHeader     header; // header.pmCommandID = kGetFilesystemVersion
    // the target
    // the outputs
    struct {
        int32           version;        // needs to be defined
    }                   theReply;
    // the inputs
    //
FSRequestTrailer        trailer;
} FSReqGetFilesystemVersion, *FSReqGetFilesystemVersionPtr;

/*------------------------------------------------------------------------
 * Permissions
 * This section is probably the one which is the most radically different
 * from the current model. The problem is that we anticipate substantial
 * improvements to the Macintosh permissions model in the future; but we
 * don't want to take the time we would need to be able to commit to specific
 * changes now. Therefore we need to come up with an API which presents a
 * permissions model which covers the current functionality, as well as all
 * of the possible models we may employ in the future. This approach does
 * have one advantage: it will make integration with other foreign filesystem
 * more straightforward.
 *
```

```
1451  * Since virtually any permissions scheme can be modeled by Access Control Lists,
1452  * we have chosen to use ACLs as the model used by the API. This does not mean
1453  * we are committed to providing ACLs in the future, just that we are sure we
1454  * can access whatever we do in the future underneath an ACL API.
1455  *
1456  * The current permission structure under AppleShare is easily modeled by a
1457  * list of two entries, one an individual (who has ChangePermissions access)
1458  * and a group; plus the usual default permissions entry to cover everyone else.
1459  * The Add and Delete Permission calls will not be functional on standard HFS
1460  * volumes.
1461  */
1462
1463 /*
1464  * FSReqPermissionIteratorCreate fills in theIterator with information
1465  * necessary to iterate over the permissions associated with the given
1466  * File or Directory. As a default, the iterator is left referring to
1467  * the default permissions (i.e. Everyone).
1468  */
1469 typedef struct FSReqPermissionIteratorCreate {
1470     FSRequestHeader             header; // header.prmCommandID = kPermissionIteratorCreate
1471     // the target & output
1472     struct {
1473         PermissionIterator      theIterator;
1474     }                           theReply;
1475     // the outputs
```

```
1476    // the inputs
1477    FSSpecification                 whose Permissions;
1478    //
1479    FSRequestTrailer                trailer;
1480    ) FSReqPermissionIteratorCreate, *FSReqPermissionIteratorCreatePtr;
1481
1482
1483    /*
1484     * FSReqPermissionIteratorDispose disposes of any internal storage associated
1485     * with the iterator.
1486     */
1487    typedef struct FSReqPermissionIteratorDispose {
1488        FSRequestHeader             header; // header.prmCommandID = kPermissionIteratorDispose
1489        // the target
1490        PermissionIterator          theIterator;
1491        // the outputs
1492        // the inputs
1493        //
1494        FSRequestTrailer            trailer;
1495    ) FSReqPermissionIteratorDispose, *FSReqPermissionIteratorDisposePtr;
1496
1497
1498    /*
1499     * FSReqPermissionsIterate iterates over the permissions based on the
1500     * give operation
```

```
1501  */
1502  typedef struct FSReqPermissionsIterate {
1503      FSRequestHeader          header; // header.prmCommandID = kPermissionsIterate
1504      // the target
1505      PermissionIterator       oldIterator;
1506      // the outputs
1507      struct {
1508          PermissionIterator   newIterator;
1509      }                        theReply;
1510      // the inputs
1511      PermissionOperation      operation;
1512      uint16                   filler1;        //** PermissionOperation is a uint16
1513      //
1514      FSRequestTrailer         trailer;
1515  } FSReqPermissionsIterate, *FSReqPermissionsIteratePtr;
1516
1517
1518  /*
1519   * FSReqPermissionSearch finds the permission entry which is associated
1520   * with the given identity.
1521   */
1522  typedef struct FSReqPermissionSearch {
1523      FSRequestHeader          header; // header.prmCommandID = kPermissionSearch
1524      // the target
1525      PermissionIterator       oldIterator;
```

```
1526    // the outputs
1527    struct {
1528        PermissionIterator      newIterator;
1529    }                           theReply;
1530    // the inputs
1531    FSAccessIdentity            identity;
1532    //
1533    FSRequestTrailer            trailer;
1534    } FSReqPermissionSearch, *FSReqPermissionSearchPtr;
1535
1536
1537    /*
1538     * FSReqPermissionAdd adds a permission specification to the File or Directory.
1539     */
1540    typedef struct FSReqPermissionAdd {
1541        FSRequestHeader             header; // header.prmCommandID = kPermissionAdd
1542        // the target
1543        PermissionIterator          oldIterator;
1544        // the outputs
1545        struct {
1546            PermissionIterator      newIterator;
1547        }                           theReply;
1548        // the inputs
1549        PermissionSpec              newPermission;
1550        //
```

```
1551    FSRequestTrailer                trailer;
1552    ) FSReqPermissionAdd, *FSReqPermissionAddPtr;
1553
1554
1555    /*
1556     * FSReqPermissionChange changes the permission currently indicated by the iterator
1557     * to what is specified by the given PermissionSpec.
1558     */
1559    typedef struct FSReqPermissionChange {
1560        FSRequestHeader             header; // header.prmCommandID = kPermissionChange
1561        // the target
1562        PermissionIterator          oldIterator;
1563        // the outputs
1564        struct {
1565            PermissionIterator      newIterator;
1566        }                           theReply;
1567        // the inputs
1568        PermissionSpec              newPermission;
1569        //
1570        FSRequestTrailer            trailer;
1571    ) FSReqPermissionChange, *FSReqPermissionChangePtr;
1572
1573
1574    /*
1575     * FSReqPermissionDelete deletes the permission specification referred
```

```
1576  * to by the iterator from the File or Directory
1577  */
1578 typedef struct FSReqPermissionDelete {
1579   FSRequestHeader              header; // header.prmCommandID = kPermissionDelete
1580   // the target
1581   PermissionIterator           oldIterator;
1582   // the outputs
1583   struct {
1584     PermissionIterator         newIterator;
1585   }                            theReply;
1586   // the inputs
1587   //
1588   FSRequestTrailer             trailer;
1589 } FSReqPermissionDelete, *FSReqPermissionDeletePtr;
1590
1591
1592 /*
1593  * FSReqPermissionGet gets the permissions associated with the given
1594  * identity for the given File or Directory.
1595  */
1596 typedef struct FSReqPermissionGet {
1597   FSRequestHeader              header; // header.prmCommandID = kPermissionGet
1598   // the target
1599   FSSpecification              theFileOrDirectory;
1600   // the outputs
```

```
1601    struct {
1602        FSPermission            thePermission;
1603    }                           theReply;
1604    // the inputs
1605    FSAccessIdentity            identity;
1606    //
1607    FSRequestTrailer            trailer;
1608    } FSReqPermissionGet, *FSReqPermissionGetPtr;
1609
1610
1611    /*
1612     * FSReqPermissionGetNative fills into PermissionBuffer a description
1613     * of the native permissions for the given File or Directory. The format
1614     * of this data is dependant on the type of filesystem supported by the
1615     * volume on which the file resides. The actualLength field will be updated
1616     * to give the actual amount of data returned back in theBuffer.
1617     */
1618    typedef struct FSReqPermissionGetNative {
1619        FSRequestHeader             header; // header.prmCommandID = kPermissionGetNative
1620        // the target
1621        FSSpecification             theFileOrDirectory;
1622        // the outputs
1623        struct {
1624            uint32                  actualLength;
1625        }                           theReply;
```

```
    BufferDescriptor        theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
    // the inputs
    //
    FSRequestTrailer        trailer;
} FSReqPermissionGetNative, *FSReqPermissionGetNativePtr;


/*
 * FSReqPermissionSetNative gets the native permissions for the given
 * File or Directory based on the contents of PermissionBuffer.
 * The format of this data is dependant on the type of filesystem
 * supported by the volume on which the file resides.
 */
typedef struct FSReqPermissionSetNative {
    FSRequestHeader         header; // header.prmCommandID = kPermissionSetNative
    // the target
    FSSpecification         theFileOrDirectory;
    // the outputs
    // the inputs
    BufferDescriptor        theBuffer;          // FSDispatch does CreateAreaForRange and DeleteArea
    //
    FSRequestTrailer        trailer;
} FSReqPermissionSetNative, *FSReqPermissionSetNativePtr;
```

```
1651 /*------------------------------------------------------------------------
1652  * FSSpecification Utilities
1653  */
1654
1655 /*
1656  * FSReqFileGetIDFromFSSpec returns the FileNum for the indicated File.
1657  * If the Filesystem does not currently have a public ID for the file,
1658  * it creates one.
1659  */
1660 typedef struct FSReqFileGetIDFromFSSpec {
1661     FSRequestHeader             header; // header.pmCommandID = kFileGetIDFromFSSpec
1662     // the target
1663     FSSpecification             theFile;
1664     // the outputs
1665     struct {
1666         FileNum                 theID;
1667     }                           theReply;
1668     // the outputs
1669     //
1670     FSRequestTrailer            trailer;
1671 } FSReqFileGetIDFromFSSpec, *FSReqFileGetIDFromFSSpecPtr;
1672
1673
1674 /*
1675  * FSReqFileGetFSSFromID resolves which File is referred to by the given
```

```
1676  * ID and Volume, and Fills in the FSSpecification so that it refers to
1677  * the file.
1678  */
1679 typedef struct FSReqFileGetFSSFromID {
1680   FSRequestHeader              header; // header.prmCommandID = kFileGetFSSpecFromID
1681   struct {
1682     // the target & output
1683       FSSpecification            theFile;
1684     // the outputs
1685       FSObjectInfo               theInfo;
1686                                  theReply;
1687   } // the inputs
1688   FSSpecification                theVolume;
1689   FileNum                        theID;
1690   //
1691   FSRequestTrailer               trailer;
1692 } FSReqFileGetFSSFromID, *FSReqFileGetFSSFromIDPtr;
1693
1694
1695 /*
1696  * FSReqPathnameResolve fills in theFile with an FSSpecification for the file
1697  * referred to by the given path. This call also validates the existance of
1698  *  such a File. */
1699 typedef struct FSReqPathnameResolve {
1700   FSRequestHeader              header; // header.prmCommandID = kPathnameResolve
```

```
1701  struct {
1702      // the target & output
1703      FSSpecification      theFile;
1704      // the outputs
1705      FSObjectInfo         theInfo;
1706                           theReply;
1707  }   // the inputs
1708  PathSpecification        thePath;        //.tname is actually an offset into variable portion of message
1709  //
1710  FSRequestTrailer         trailer;
1711  ) FSReqPathnameResolve, *FSReqPathnameResolvePtr;
1712
1713
1714  /*-----------------------------------------------------------------
1715   * Notification
1716   * This section describes a mechanism to allow applications to be notified
1717   * when objects in the filesystem change. It is intended for use by the Finder
1718   * and by applications like OnLocation which need to know what other
1719   * applications are doing to the Filesystem.
1720   */
1721
1722  /*
1723   * FSReqNotificationQueueCreate creates a new Notification Queue with no
1724   * subscriptions. The parameter maxQueueLength defines the maximum number
1725   * of NTChangeReport's which will be stored for the Queue. The overrunAction
```

```
1726   * parameter is evoked whenever the queue is full.
1727   */
1728  typedef struct FSReqNotificationQueueCreate {
1729      FSRequestHeader              header; // header.prmCommandID = kFSNotificationQueueCreate
1730      // the target & output
1731      struct {
1732          NotificationObjID        theQueue;
1733      }                            theReply;
1734      // the outputs
1735      // the inputs
1736      uint32                       maxQueueLength;
1737      EventNotification            overrunAction;
1738      //
1739      FSRequestTrailer             trailer;
1740  } FSReqNotificationQueueCreate, *FSReqNotificationQueueCreatePtr;
1741
1742
1743  /*
1744   * FSReqNotificationQueueDispose removes all of the given Notification
1745   * Queue's subscriptions and then disposes of the Queue itself.
1746   */
1747  typedef struct FSReqNotificationQueueDispose {
1748      FSRequestHeader              header; // header.prmCommandID = kFSNotificationQueueDispose
1749      // the target
1750      NotificationObjID            theQueue;
```

```
1751    // the outputs
1752    // the inputs
1753    //
1754    FSRequestTrailer                trailer;
1755    ) FSReqNotificationQueueDispose, *FSReqNotificationQueueDisposePtr;
1756
1757
1758    /*
1759     * FSReqNotificationSubscribe subscribes the given Notification Queue to the type of
1760     * change indicated, and queue the reports with the given scope. If collapse
1761     * is true, then identical reports on the queue are collapsed together into
1762     * a single report Thus if theChange equals NTFileCreated and theScope
1763     * equals NTReportOnDirectory and collapse is true, multiple reports of files
1764     * being created in the samedirectory will be collapsed into a single report
1765     * if they appear on the queue at the same time. All events are filtered by
1766     * the given volume and directory ID's.
1767     */
1768    typedef struct FSReqNotificationSubscribe {
1769        FSRequestHeader             header; // header.pbCommandID = kFSNotificationSubscribe
1770        // the target
1771        NotificationObjID           theQueue;
1772        // the outputs
1773        // the inputs
1774        FSChangeEvent               theChange;
1775        FSChangeReportScope         theScope;
```

```
1776    VolumeObjID             theVolume;
1777    DirectoryNum            theDirectory;
1778    uint32                  options;
1779    //
1780    FSRequestTrailer        trailer;
1781    ) FSReqNotificationSubscribe, *FSReqNotificationSubscribePtr;
1782
1783
1784    /*
1785     * FSReqNotificationUnsubscribe remove any subscription which the given Notification
1786     * Queue might have to thetype of change indicated which have the given scope.
1787     * If theScope is nil, thenall subscriptions to the type of change are removed.
1788     * If theChange is nil, thenall subscriptions are removed.
1789     */
1790    typedef struct FSReqNotificationUnsubscribe {
1791        FSRequestHeader         header; // header.prmCommandID = kFSNotificationUnsubscribe
1792        // the target
1793        NotificationObjID       theQueue;
1794        // the outputs
1795        // the inputs
1796        FSChangeEvent           theChange;
1797        FSChangeReportScope     theScope;
1798        VolumeObjID             theVolume;
1799        DirectoryNum            theDirectory;
1800        //
```

```
1801    FSRequestTrailer                trailer;
1802    ) FSReqNotificationUnsubscribe, *FSReqNotificationUnsubscribePtr;
1803
1804
1805    /*
1806     * FSReqNotificationGetEvent removes the top change report from the queue, and returns a
1807     * pointer to the next change report (which is now at the top).
1808     */
1809    typedef struct FSReqNotificationGetEvent {
1810        FSRequestHeader             header; // header.prmCommandID = kFSNotificationGetEvent
1811        // the target
1812        NotificationObjID           theQueue;
1813        // the outputs
1814        struct {
1815            FSChangeReport          theChange;
1816        }                           theReply;
1817        // the inputs
1818        //
1819        FSRequestTrailer            trailer;
1820    ) FSReqNotificationGetEvent, *FSReqNotificationGetEventPtr;
1821
1822    /*-----------------------------------------------------------------------------
1823     * Access Methods Interfaces
1824     * While the new File Manager interface is fairly rich, inevitably Apple and Third
1825     * Party developers will want to add new functionality to the interface which is
```

```
1826  * availble under specific circumstances. This may include, but is not limited to,
1827  * providing access to special features of particular volume formats, and new
1828  * access methods. The following requests provide half of the mechanism to do this.
1829  * They essentially allow new kinds of requests to be defined and sent through the
1830  * Filer Manager. The other half of the mechanism allows code to be installed in
1831  * the file manager to handle requests which are not handled by FSAgents.
1832  *
1833  * While the details of the contents of these messages are under the control of the
1834  * service developer, these messages become a part of the File Manager Interface
1835  * and therefore will be processed by FSDispatch. It is neccessary that they follow
1836  * the same rules as all other File Manager requests. While violations of these
1837  * rules may appear to work for any specific version, Apple makes no guarantee that
1838  * they will containue to work for all versions of the file manager. The rules are:
1839  *
1840  * 1) All requests must begin with a FSRequestHeader structure, and end with a
1841  *    FSRequestTrailer structure, followed by any variable length data.
1842  *
1843  * 2) All data (input and output) which is neccessary for processing the request
1844  *    must be able to be identified by FSDispatch. For this reason, all of these
1845  *    requests make use of arrays of BufferDescriptor's to identify all data
1846  *    related to the request which is NOT in the message proper. The use of data
1847  *    outside of the message which is not referenecs by one of these descriptors
1848  *    is strictly forbidden.
1849  *
1850  * 3) All fixed length input data, and all small variable length input data should
```

```
1851  *     be included in the message itself (this is actually a performance issue).
1852  *
1853  * 4) All variable length data which is a part of the message should be stored
1854  *     after the FSRequestTrailer, but refered to (by offset) by fields before the
1855  *     trailer using M_VaribleDataPtr.
1856  *
1857  * 5) No mixed input-and-output parameters are allowed. If a given parameter's
1858  *     value is changes by a request, then its value must not be used as input as
1859  *     well. In cases where this is desired, have two parameters which may refer to
1860  *     the same pieces of data, but are clearly separate parameters.
1861  *
1862  * In order to make the use of these rules more reasonable, all of these requests
1863  * have the same basic structure for their parameters. The first two parameters
1864  * always indicate the Extension the request is to be directed to (as identified
1865  * by a 4 character code), and what specific Action that Extension is being
1866  * requested to perform.
1867  *
1868  * Following that, there is the ID of the File Manager entity to which the request
1869  * is directed, the "Target" of the request. This can be a Path, or a Volume or (if
1870  * nil) the File Manager as a whole.
1871  *
1872  * Following that, there is a single pointer to a longword which is used as a
1873  * simple output parameter. This is provided because most requests only require a
1874  * sangle 4 byte output, and the use of a BufferDescriptor for it would be overkill.
1875  *
```


30 20 10

```
1876  * Following that, there is a descrition of all of the data, outside of the message
1877  * itself, which is used to process the request. All of this data is described by
1878  * BufferDescriptor's. This list is further divided into those pieces of data which
1879  * are changed by the request (the "outputs"), and those which are only read by the
1880  * request (the "inputs"). A count of each type is given, followed by an array of
1881  * BufferDescriptor's. In the array, all output parameters come first, followed by
1882  * any input parameters.
1883  *
1884  * Following that, there are the other fixed length inputs which are included in
1885  * the message itself. If there are any varible length input fields which are to be
1886  * included in the message, their values should be stored after the trailer, but
1887  * offsets to their locations in the variable portion of message buffer array should be stored here.
1888  *
1889  * Follwoing that is the FSRequestTrailer, and the variable data array which form
1890  * the end of the message.
1891  */
1892
1893 /*
1894  * FSReqExtendedRequest, an extended request for the File Manager as a whole.
1895  * Note: while theTarget is usually 0, if the sender knows the ObjectID of a
1896  * specific FSAgent, that value can be passed here, and the request will be
1897  * dispatched to that Agent rather than having the File Manager handle it.
1898  */
1899 typedef struct FSReqExtendedRequest {
1900     FSRequestHeader             header; // header.prmCommandID = kExtendedRequest
```

```
1901    OSType                          extensionID;            // What extension should handle this
1902    uint32                          actionCode;             // What should it do
1903    // the target
1904    ObjectID                        theTarget;              // usually 0
1905    // the output
1906    uint32*                         theOutput;              // the one, cheap, output
1907    // the extenal inputs and outputs (those not in the message proper)
1908    uint16                          outputCount;            // the number of outputs
1909    uint16                          inputCount;             // the number of inputs not in message
1910  //**  BufferDescriptor                  externalParams[0];  // a list of the remaining external outputs and inputs
1911    // the other fixed length inputs
1912  //**  uint32                            inputParams[0];     // the fixed length input parameters
1913    //
1914    FSRequestTrailer                trailer;                // MUST be 32 bit aligned!
1915    ) FSReqExtendedRequest, *FSReqExtendedRequestPtr;
1916
1917  /*
1918   * FSReqExtendedVolumeMount, an extended request to mount a volume. Actually, a
1919   * more correct interpretation of this request is an "extended request to generate
1920   * a VolumeObjID". It is not neccessary that a volume actually be mounted in a
1921   * traditional sense. For an FSAgent to handle this request, it must have registered
1922   * itself with the given extensionID.
1923   */
1924  typedef struct FSReqExtendedVolumeMount {
1925    FSRequestHeader                 header; // header.prmCommandID = kExtendedVolumeMount
```

```
1926    OSType                  extensionID;        // What extension should handle this
1927    uint32                  actionCode;         // What should it do
1928    // the target
1929    VolumeObjID*            theVolume;
1930    // the output
1931    uint32*                 theOutput;          // the one, cheap, output
1932    // the external inputs and outputs (those not in the message proper)
1933    uint16                  outputCount;        // the number of outputs
1934    uint16                  inputCount;         // the number of inputs not in message
1935  //** BufferDescriptor      externalParams[0];  // a list of the remaining external outputs and inputs
1936    // the other fixed length inputs
1937  //** uint32                inputParams[0];     // the fixed length input parameters
1938    //
1939    FSRequestTrailer        trailer;            // MUST be 32 bit aligned!
1940    ) FSReqExtendedVolumeMount, *FSReqExtendedVolumeMountPtr;
1941
1942  /*
1943   * FSReqExtendedVolumeUnmount, an extended request to unmount a volume. Actually, just
1944   * dispose of and clean up after a VolumeObjID created by a FSReqExtendedVolumeMount
1945   * request.
1946   */
1947  typedef struct FSReqExtendedVolumeUnmount {
1948    FSRequestHeader         header; // header.prmCommandID = kExtendedVolumeUnmount
1949    OSType                  extensionID;        // What extension should handle this
1950    uint32                  actionCode;         // What should it do
```

```
1951    // the target
1952    VolumeObjID                 theVolume;
1953    // the output
1954                                theOutput;          // the one, cheap, output
1955    // the extenal inputs and outputs (those not in the message proper)
1956    uint16                      outputCount;        // the number of outputs
1957    uint16                      inputCount;         // the number of inputs not in message
1958    //** BufferDescriptor        externalParams[0];  // a list of the remaining external outputs and in
1959     // the other fixed length inputs
1960    //** uint32                  inputParams[0];     // the fixed length input parameters
1961     //
1962    FSRequestTrailer            trailer;            // MUST be 32 bit aligned!
1963    ) FSReqExtendedVolumeUnmount, *FSReqExtendedVolumeUnmountPtr;
1964
1965    /*
1966     * FSReqExtendedVolumeRequest, an extended request for a volume. The given VolumeObjID
1967     * may or may not have been created with a FSReqExtendedVolumeMount request, depending
1968     * on the specific Extension involved.
1969     */
1970    typedef struct FSReqExtendedVolumeRequest {
1971     FSRequestHeader             header; // header.prmCommandID = kExtendedVolumeRequest
1972     OSType                      extensionID;        // What extension should handle this
1973     uint32                      actionCode;         // What should it do
1974     // the target
1975     VolumeObjID                 theVolume;
```

```
1976    // the output
1977    uint32*                 theOutput;              // the one, cheap, output
1978    // the extenal inputs and outputs (those not in the message proper)
1979    uint16                  outputCount;            // the number of outputs
1980    uint16                  inputCount;             // the number of inputs not in message
1981    //** BufferDescriptor   externalParams[0];      // a list of the remaining external outputs and input
1982    // the other fixed length inputs
1983    //** uint32              inputParams[0];         // the fixed length input parameters
1984    //
1985    FSRequestTrailer        trailer;                // MUST be 32 bit aligned!
1986    ) FSReqExtendedVolumeRequest, *FSReqExtendedVolumeRequestPtr;
1987
1988    /*
1989     * FSReqExtendedPathOpen, an extended request to open a path. Actually, a
1990     * more correct interpretation of this request is an *extended request to generate
1991     * a PathObjID*. It is not neccessary that a path actually be mounted in a
1992     * traditional sense.
1993     */
1994    typedef struct FSReqExtendedPathOpen {
1995      FSRequestHeader       header; // header.prmCommandID = kExtendedPathOpen
1996      OSType                extensionID;            // What extension should handle this
1997      uint32                actionCode;             // What should it do
1998      // the target
1999      PathObjID*            thePath;
2000      // the output
```

```
2001    uint32*                 theOutput;              // the one, cheap, output
2002    // the extenal inputs and outputs (those not in the message proper)
2003    uint16                  outputCount;            // the number of outputs
2004    uint16                  inputCount;             // the number of inputs not in message
2005    //** BufferDescriptor   externalParams[0];      // a list of the remaining external outputs and inputs
2006    // the other fixed length inputs
2007    //** uint32              inputParams[0];         // the fixed length input parameters
2008    //
2009    FSRequestTrailer        trailer;                // MUST be 32 bit aligned!
2010    } FSReqExtendedPathOpen, *FSReqExtendedPathOpenPtr;
2011
2012    /*
2013     * FSReqExtendedPathClose, an extended request to close a path. Actually, just
2014     * dispose of and clean up after a PathObjID created by a FSReqExtendedPathOpen
2015     * request.
2016     */
2017    typedef struct FSReqExtendedPathClose {
2018     FSRequestHeader            header; // header.prmCommandID = kExtendedPathClose
2019     OSType                     extensionID;            // What extension should handle this
2020     uint32                     actionCode;             // What should it do
2021     // the target
2022     PathObjID                  thePath;
2023     // the output
2024     uint32*                    theOutput;              // the one, cheap, output
2025     // the extenal inputs and outputs (those not in the message proper)
```

```
2026    uint16                  outputCount;            // the number of outputs
2027    uint16                  inputCount;             // the number of inputs not in message
2028  //** BufferDescriptor      externalParams[0];      // a list of the remaining external outputs and inputs
2029    // the other fixed length inputs
2030  //** uint32                inputParams[0];         // the fixed length input parameters
2031    //
2032    FSRequestTrailer        trailer;                // MUST be 32 bit aligned!
2033    } FSReqExtendedPathClose, *FSReqExtendedPathClosePtr;
2034
2035  /*
2036   * FSReqExtendedPathRequest an extended request for a path. The given PathObjID
2037   * may or may not have been created with a FSReqExtendedPathOpen request, depending
2038   * on the specific Extension involved.
2039   */
2040  typedef struct FSReqExtendedPathRequest {
2041    FSRequestHeader         header; // header.prmCommandID = kExtendedPathRequest
2042    OSType                  extensionID;            // What extension should handle this
2043    uint32                  actionCode;             // What should it do
2044    // the target
2045    PathObjID               thePath;
2046    // the output
2047    uint32*                 theOutput;              // the one, cheap, output
2048    // the extenal inputs and outputs (those not in the message proper)
2049    uint16                  outputCount;            // the number of outputs
2050    uint16                  inputCount;             // the number of inputs not in message
```

```
2051 //**  BufferDescriptor          externalParams[0];  // a list of the remaining external outputs and inputs
2052  // the other fixed length inputs
2053 //**  uint32                    inputParams[0];     // the fixed length input parameters
2054  //
2055  FSRequestTrailer              trailer;
2056  ) FSReqExtendedPathRequest, *FSReqExtendedPathRequestPtr;   // MUST be 32 bit aligned!
2057
2058 /*-----------------------------------------------------------------------
2059  * Backing Store Requests
2060  */
2061
2062 /*
2063  * FSReqBackingStoreCreate creates a BackingObjectID which can be used to map
2064  * the indicated fork.
2065  */
2066 typedef struct FSReqBackingStoreCreate {
2067   FSRequestHeader               header; // header.prmCommandID = kBackingStoreCreate
2068   // the target & output
2069   BackingObjectID*              theStore;
2070   // the inputs
2071   FSSpecification               theFile;
2072   ForkNum                       theFork;
2073   uint32                        options;
2074   //
2075   FSRequestTrailer              trailer;
```

```
2076    } FSReqBackingStoreCreate, *FSReqBackingStoreCreatePtr;
2077
2078    /*
2079     * FSReqBackingStoreDispose Disposes of a BackingObjectID which was provided by the
2080     * File Manager.
2081     */
2082    typedef struct FSReqBackingStoreDispose {
2083      FSRequestHeader          header; // header.prmCommandID = kBackingStoreDispose
2084      // the target & output
2085      BackingObjectID          theStore;
2086      //
2087      FSRequestTrailer         trailer;
2088    } FSReqBackingStoreDispose, *FSReqBackingStoreDisposePtr;
2089
2090
2091    /*
2092     *
2093     */
2094
2095    typedef union {
2096      FSRequestHeader          header;
2097
2098      FSReqPathOpen            PathOpen;
2099      FSReqPathClose           PathClose;
2100      FSReqPathRead            PathRead;
```

```
2101  FSReqPathWrite                  PathWrite;
2102  FSReqPathLockRange              PathLockRange;
2103  FSReqPathUnlockRange            PathUnlockRange;
2104  FSReqPathGetPosition            PathGetPosition;
2105  FSReqPathSetPosition            PathSetPosition;
2106  FSReqPathGetEOF                 PathGetEOF;
2107  FSReqPathSetEOF                 PathSetEOF;
2108  FSReqPathAllocate               PathAllocate;
2109  FSReqPathRelease                PathRelease;
2110
2111  FSReqForkFlush                  ForkFlush;
2112  FSReqFileFlush                  FileFlush;
2113  FSReqVolumeFlush                VolumeFlush;
2114  //--  FSReqCacheAdvise              CacheAdvise;
2115
2116  FSReqIteratorCreate             IteratorCreate;
2117  FSReqIteratorDispose            IteratorDispose;
2118  FSReqIterate                    Iterate;
2119  FSReqSearch                     Search;
2120  FSReqAttributesGet              AttributesGet;
2121  FSReqAttributesSet              AttributesSet;
2122  FSReqAttributesGetSize          AttributesGetSize;
2123  FSReqAttributesGetBulk          AttributesGetBulk;
2124  FSReqAttributesGetBulkFiltered  AttributesGetBulkFiltered;
2125  FSReqTestEntity                 TestEntity;
```

```
2126 FSReqEntityType                    EntityType;
2127
2128 FSReqVolumeCreate                  VolumeCreate;
2129 FSReqVolumeGetCapability           VolumeGetCapability;
2130 FSReqVolumeMount                   VolumeMount;
2131 FSReqVolumeUnmount                 VolumeUnmount;
2132 FSReqVolumeEject                   VolumeEject;
2133 FSReqVolumeOffline                 VolumeOffline;
2134 FSReqVolumeGetRemountSize          VolumeGetRemountSize;
2135 FSReqVolumeGetRemountData          VolumeGetRemountData;
2136 FSReqVolumeRemount                 VolumeRemount;
2137
2138 FSReqDirectoryCreate               DirectoryCreate;
2139
2140 FSReqFileCreate                    FileCreate;
2141
2142 FSReqForkAttributesGet             ForkAttributesGet;
2143 FSReqForkAttributesSet             ForkAttributesSet;
2144 FSReqForkAttributesGetSize         ForkAttributesGetSize;
2145 FSReqForkCreate                    ForkCreate;
2146 FSReqForkDelete                    ForkDelete;
2147
2148 FSReqFileClassAttributesGet        FileClassAttributesGet;
2149 FSReqFileClassAttributesSet        FileClassAttributesSet;
2150 FSReqFileClassAttributesGetSize    FileClassAttributesGetSize;
```

```
FSReqFileClassIteratorCreate    FileClassIteratorCreate;
FSReqFileClassIteratorDispose   FileClassIteratorDispose;
FSReqFileClassIterate           FileClassIterate;

FSReqDelete                     Delete;
FSReqExchange                   Exchange;
FSReqMoveRename                 MoveRename;
FSReqCopyRename                 CopyRename;
FSReqDirectoryGetCurrent        DirectoryGetCurrent;
FSReqDirectorySetCurrent        DirectorySetCurrent;
FSReqGetFilesystemVersion       GetFilesystemVersion;

FSReqPermissionIteratorCreate   PermissionIteratorCreate;
FSReqPermissionIteratorDispose  PermissionIteratorDispose;
FSReqPermissionsIterate         PermissionsIterate;
FSReqPermissionSearch           PermissionSearch;
FSReqPermissionAdd              PermissionAdd;
FSReqPermissionChange           PermissionChange;
FSReqPermissionDelete           PermissionDelete;
FSReqPermissionGet              PermissionGet;
FSReqPermissionGetNative        PermissionGetNative;
FSReqPermissionSetNative        PermissionSetNative;

FSReqFileGetIDFromFSSpec        FileGetIDFromFSSpec;
FSReqFileGetFSSFromID           FileGetFSSFromID;
```

```
    FSReqPathnameResolve    PathnameResolve;

    FSReqNotificationQueueCreate    NotificationQueueCreate;
    FSReqNotificationQueueDispose   NotificationQueueDispose;
    FSReqNotificationSubscribe      NotificationSubscribe;
    FSReqNotificationUnsubscribe    NotificationUnsubscribe;
    FSReqNotificationGetEvent       NotificationGetEvent;

    FSReqExtendedRequest        ExtendedRequest;
    FSReqExtendedVolumeMount    ExtendedVolumeMount;
    FSReqExtendedVolumeUnmount  ExtendedVolumeUnmount;
    FSReqExtendedVolumeRequest  ExtendedVolumeRequest;
    FSReqExtendedPathOpen       ExtendedPathOpen;
    FSReqExtendedPathClose      ExtendedPathClose;
    FSReqExtendedPathRequest    ExtendedPathRequest;

    FSReqBackingStoreCreate     BackingStoreCreate;
    FSReqBackingStoreDispose    BackingStoreDispose;

} FSRequest, *FSRequestPtr;
#endif //__FSREQUESTS__
```

APPENDIX B

"FSTypes.h"

```
/*
  File:       FSTypes.h

  Contains:   All types and constants which are part of the public File Manager API

  Copyright:  © 1992-1994 by Apple Computer, Inc., all rights reserved.

*/

#ifndef __FSTYPES__
#define __FSTYPES__

#ifndef __MORETYPES__
#include <MoreTypes.h>
#endif
#ifndef __KernelTypes__
#include <KernelTypes.h>
#endif

/*------------------------------------------------------------------------------*/
/* CONSTANTS:
/*------------------------------------------------------------------------------*/
// The following are all of the constant which are used in this API design.

/*
```

```
 * PositionMode is used do describe how the posOffset field in a
 * ForkPositionDescriptor is to be interpreted.
 */
typedef enum {
  kFSAtMark                 = 0,        // ignore offset, use current position
  kFSFromStart              = 1,        // offset relative to start of file
  kFSFromLEOF               = 2,        // offset relative to end of file
  kFSFromMark               = 3         // offset relative to current position
  };
typedef uint16 PositionMode;

/*
 *    Path lock options
 */
enum {
  kFSRangeLockMask          = 0x00000001, // 1 = write lock, 0 = read lock
      kFSRangeReadLock      = 0x00000000,
      kFSRangeWriteLock     = 0x00000001
};

/*
 * AllocationMode is used to indicate how new disk space should be
 * allocated to a Fork when it grows.
 */
typedef enum {
```

```
  kFSAllocAny                     = 0,        // allocate new blocks anywhere
  kFSAllocContiguous              = 1         // allocate blocks only if contiguous
  };
typedef uint16 AllocationMode;


/*
 * CacheOperation is used to indicate what action is to be performed
 * on a set of cache entries and/or to indicate how the user expects
 * to use a range of entries in the future.
 */
typedef enum {
  kFSCacheWrite                   = 0,        // write entry out at earliest convenience
  kFSCacheInvaldate,                          // invalidate entry without writing
  kFSCacheRelease,                            // entries will not be used again soon
  kFSCacheMaintain                            // entries will be used again soon
  };
typedef uint16 CacheOperation;

/*
 * CachePriority is used to indicate how important it is for the Filesystem
 * to keep specific data in memory. There are three steps, based on how
 * likely it is that the data will be accessed again.
 */
typedef enum {
```

```
    kFSCacheWillNeedAgain           = 0x01,     // The data is likely to be needed again
    kFSCacheMayNeedAgain            = 0x00,     // It is undetermined if the data will be needed again.
    kFSCacheWillNotNeedAgain        = 0x11      // The data is not expected to be needed again
    };
typedef uint16 CachePriority;

/*
 * Used to indicate the type of an entity in the filesystem.
 */
typedef enum {
    kFSIsAFile                      = 0x1,      // Entity is a File
    kFSIsADirectory                 = 0x2,      // Entity is a Directory
    kFSIsAVolume                    = 0x3,      // Entity is a Mounted Volume
    kFSIsTheMetaRoot                = 0x4,      // Entity is the Metaroot
    kFSEntityMask                   = 0x7       // Entity is a 3 bit datum merged into other datums
    };
typedef uint16 FSEntityType;

/*
 * FSIterationOperation is used to indicate how a Filesystem Iterator
 * is to be moved within the filesystem.
 */
typedef enum {
    //
    // Absolute locations
```

```
101    //
102    kFSIterFMRoot                    = 0xFF01,    // go to the filesystem "meta" root
103
104    kFSIterBootVolume                = 0xFF11,    // go to Boot Volume
105    kFSIterSystemFolder              = 0xFF12,    // go to Boot Volume's System Folder
106    kFSIterAppleMenuFolder           = 0xFF13,    // go to Boot Volume's Apple Menu Folder
107    kFSIterControlPanelFolder        = 0xFF14,    // go to Boot Volume's Control Panel Folder
108    kFSIterExtensionFolder           = 0xFF15,    // go to Boot Volume's Extension Folder
109    kFSIterFontsFolder               = 0xFF16,    // go to Boot Volume's Fonts Folder
110    kFSIterPreferencesFolder         = 0xFF17,    // go to Boot Volume's Preferences Folder
111    kFSIterPrintMonitorDocsFolder    = 0xFF18,    // go to Boot Volume's Print Monitor Docs Folder
112    kFSIterStartupFolder             = 0xFF19,    // go to Boot Volume's Startup Folder
113
114    kFSIterVolumeRoot                = 0xFF31,    // go to this volume's root
115    kFSIterDesktopFolder             = 0xFF32,    // go to this volume's Desktop Folder
116    kFSIterTemporaryFolder           = 0xFF33,    // go to this volume's Temporary Folder
117    kFSIterTrashFolder               = 0xFF34,    // go to this volume's Trash Folder
118    kFSIterWhereToEmptyTrashFolder   = 0xFF35,    // go to this volume's "Where To Empty Trash" Folder
119    //
120    // "Upward" movement
121    //
122    kFSIterParent                    = 0x0100,    // go to parent of current entity
123    //
124    // "Downward" movement
125    //
```

```
126   kFSIterFirstChild             = 0x1100,   // go to first child of current entity
127   kFSIterFirstChildFile         = (kFSIterFirstChild | kFSIsAFile),          // first file
128   kFSIterFirstChildDir          = (kFSIterFirstChild | kFSIsADirectory),     // first directory
129   kFSIterLastChild              = 0x1200,   // go to last child of current entity
130   kFSIterLastChildFile          = (kFSIterLastChild | kFSIsAFile),           // last file
131   kFSIterLastChildDir           = (kFSIterLastChild | kFSIsADirectory),      // last directory
132   //
133   // Lateral movement
134   //
135   kFSIterNextSibling            = 0x2100,   // go to next entity in same directory
136   kFSIterNextFile               = (kFSIterNextSibling | kFSIsAFile),         // next file
137   kFSIterNextDir                = (kFSIterNextSibling | kFSIsADirectory),    // next directory
138   kFSIterPrevSibling            = 0x2200,   // go to previous entity in same directory
139   kFSIterPrevFile               = (kFSIterPrevSibling | kFSIsAFile),         // previous file
140   kFSIterPrevDir                = (kFSIterPrevSibling | kFSIsADirectory),    // previous directory
141   //
142   // Agregate movement
143   //
144   kFSIterChildren               = 0x8100,   // Start iteration over all objects with a child relationship to
145                                             // the current node. The order is not guaranteed.
146   kFSIterChildrenFile           = (kFSIterChildren | kFSIsAFile),            // only file entities
147   kFSIterChildrenDir            = (kFSIterChildren | kFSIsADirectory),       // only directory entities
148
149   kFSIterSubtree                = 0x8200,   // Start iteration over all objects in the subtree below
150                                             // the current node. The order is not guaranteed.
```

```
151     kFSIterSubtreeFile              = (kFSIterSubtree | kFSIsAFile),        // only file entities
152     kFSIterSubtreeDir               = (kFSIterSubtree | kFSIsADirectory),   // only directory entities
153
154     kFSIterContinue                 = 0x0001    // move to next entity along the route specified by the previous
155 operation
156     };
157 typedef uint16 FSIterationOperation;
158
159 /*
160  * ClassIterationOperation is use to indicate how a FileClass Iterator
161  * is to be moved through the database of FileClass'
162  */
163
164 typedef enum {
165     kFSClassOpThis                  = 1,        // go here
166     kFSClassOpFirst,                            // go to first class which has any attributes
167     kFSClassOpNext,                             // go to next class
168     kFSClassOpNextSameType,                     // go to next class with same type
169     kFSClassOpNextSameCreator                   // go to next class with same creator
170     };
171 typedef uint16 ClassIterationOperation;
172
173 /*
174  * PermissionOperation is used to indicate how a Permission
175  * Iterator is to be moved within a list of permissions.
```

```
 */
typedef enum {
  kFSPermOpThis                    = 1,        // stay on current entry
  kFSPermOpDefault,                            // go to the default permissions entry
  kFSPermOpFirst,                              // go to the first entry in the list
  kFSPermOpFirstUser,                          // go to the first individual user entry
  kFSPermOpFirstGroup,                         // go to the first group entry
  kFSPermOpNext,                               // go to the next entry in the list
  kFSPermOpNextUser,                           // go to the next individual user entry
  kFSPermOpNextGroup                           // go to the next group entry
  };
typedef uint16 PermissionOperation;

/*
 * FormatOperationMask is used to indicate which operations
 * are to be performed by a NewVolume call.
 */
typedef enum {
  kFSFormatDisk                    = 0x01,     // do a raw format of the volume
  kFSVerifyDisk                    = 0x02,     // verify the format of a volume
  kFSZeroDisk                      = 0x04,     // erase the contents of a volume
  kFSInitializeDisk                = 0xff      // do a complete initialization
  };
typedef uint16 FormatOperationMask;
```

```
enum {
  kFMMetaRootID                    = 0,        // Metaroot's volume object ID
  kFMMetaRoot                      = 1,        // Metaroot's directory number
  kFSPathNameSeparator             = 0x003A    // ':'
  };


/*
 * FSAttribute Constants:
 * Used to indicate an attribute of a filesystem object. Attributes
 * include both the standard information (available by GetInfo calls)
 * and any extended information which has been added to the filesystem.
 * Values which contain lower case letters are reserved by Apple,
 * and values which contain nil characters are illegal.
 *
 * The structure of an FSAttribute is two OSTypes. The "high order"
 * OSType defines the service which "owns" the attribute, and the
 * low order OSType specifies which of that services attributes it is.
 */

/* FSAttribute Services */
typedef enum {
  kSpecialAttribute                = '\0\0\0\1',   // chosen for its illegality for real attributes
  kAnyServiceID                    = '****',   // Wild card to match any service ID
  kFileManagerAttribute            = 'fmgr',
```

```
226   kFinderAttribute          = 'fndr',
227   kHFSAttribute             = 'hfs ',
228   kPermissionsAttribute     = 'perm',
229   kFileShareAttribute       = 'fshr'    // File Share (i.e. AppleShare PDS)
230   };
231
232 /* FSAttribute Attributes, by service */
233 typedef enum {
234   /* kSpecialAttribute attributes */
235   /* the following three "attributes" are not attributes at all, but can be */
236   /* requested like attributes to get detailed information about errors */
237   /* encountered getting a list of attributes. Each is a bit map, starting with */
238   /* the msb, of information about each attribute requested in the list */
239   kAttrOverrun              = 'over',   // Bit Map of Attributes which are
240                                         // longer than buffer
241   kAttrUnderflow            = 'undr',   // Bit Map of Attributes which are
242                                         // shorter than buffer
243   kAttrNotFound             = 'nfnd',   // Bit Map of Attributes which are
244                                         // not found
245   kAttrReadOnly             = 'ronl',   // Bit Map of Attributes which
246                                         // attempted to change "read-only"
247                                         // attribute.
248
249   /* kFileManagerAttribute ('fmgr') attributes */
250   kAnyServiceAttribute      = '****',   // Wild card to match any service attribute
```

```
kDesktopFolderDirNum                = 'desk',
kExtensionFolderDirNum              = 'extn',
kPreferencesFolderDirNum            = 'pref',
kPrintMonitorDocsFolderDirNum       = 'prnt',
kWhereToEmptyTrashFolderDirNum      = 'empt',
kTrashFolderDirNum                  = 'trsh',
kStartupFolderDirNum                = 'strt',
kSystemFolderDirNum                 = 'macs',
kTemporaryFolderDirNum              = 'temp',
kFSSpecification                    = 'fspc',
kFSVersionNumber                    = 'Ver#',

/* kFinderAttribute ('fndr') attributes */
kFSComment                          = 'Cmnt',
kFSApplInfo                         = 'ApIn',
kFSIconTypes                        = 'ITps',
kFSIconList                         = 'icn#',
kFS4BitIcon                         = 'icl4',
kFS8BitIcon                         = 'icl8',
kFSSmallIconList                    = 'ics#',
kFS4BitSmallIcon                    = 'ics4',
kFS8BitSmallIcon                    = 'ics8',
kFSCommentID                        = 'CtID',
kFSIconID                           = 'IcID',
kFSPutAwayDirectory                 = 'PtDr',
kFSDriveID                          = 'DrvN',
kFSPartitionID                      = 'PrtN',
kFSVolumeBlockSize                  = 'VBSz',
kFSTotalFiles                       = 'TFil',
kFSTotalDirectories                 = 'TDir',
kFSNumberOfFiles                    = 'NFil',
kFSNumberOfDirectories              = 'NDir',
kFSCreationDate                     = 'CrDt',
kFSModificationDate                 = 'MdDt',
kFSBackupDate                       = 'BkDt',
kFSName                             = 'Name',
kFSParentID                         = 'PrID',
kFSDirectoryNum                     = 'DrID',
kFSFileNum                          = 'FlID',
kFSType                             = 'Type',
kFSCreator                          = 'Crtr',
kFSForkType                         = 'FkTy',
kFirstAllocationBlock               = 'FBlk',
kFSLogicalSize                      = 'Size',
kFSPhysicalSize                     = 'Stor',
kFSKind                             = 'kind',
kFSLoginName                        = 'logn',
kFSLoginMethod                      = 'lmtd',
kAppleMenuFolderDirNum              = 'amnu',
kControlPanelFolderDirNum           = 'ctrl',
```

```
kFSLocation                = 'Locn',
kFSDirectoryRect           = 'DrRt',
kFSDirectoryView           = 'View',
kFSFinderFlags             = 'FFlg',
kFSExtendedFinderFlags     = 'XFFg',
kFSScriptFlag              = 'sflg',
kFSDirectoryScrollPos      = 'DrSP',
kFSDirectoryOpenChain      = 'DrOC',

/* kHFSAttribute ('hfs ') attributes */
kFSVolumeBitmapBlock       = 'VBbl',   // Returns back a short
kFSNextAllocationBlock     = 'NAbl',   // Returns back a short
kFSNumAllocationBlocks     = 'NAnb',   // Returns back a long
kFSAllocationBlockSize     = 'AblS',   // Returns back a long
kFSDefaultClumpSize        = 'CpSz',   // Returns back a long
kFSVolumeBlockmapBlock     = 'VBmb',   // Returns back a short
kFSNextCatalogNodeID       = 'NCNd',   // Returns back a long
kFSFreeAllocationBlocks    = 'NUAb',   // Returns back a short
kFSVolumeSignature         = 'VlSg',   // Returns back a short
kFSDriveNumber             = 'DvNm',   // Returns back a short
kFSDriverReferenceNum      = 'DrvN',   // Returns back a short
kFSFileSystemID            = 'FSID',   // Returns back a short
kFSVolumeSequenceNum       = 'VSqN',   // Returns back a short
kFSWriteCount              = 'WrCt',   // Returns back a long
kFSVolumeAttribute         = 'VAtr'    // Returns back a short
```

```
};

/*
 * Currently defined ForkNum's
 */
typedef enum {
  kFSResourceFork                 = 'rsrc',
  kFSDataFork                     = 'data'
  };

/*
 * Currently defined Fork Formats
 */
typedef enum {
  kFSResourceFormat               = 'rsrc',
  kFSStreamFormat                 = 'strm',
  kFSBTreeFormat                  = 'btre'
  };

/*
 * FSSearchOperator
 */

typedef enum {
  kFSNot                          = 0x8000,    // Unary Operator
```

```
    kFSAnd                  = 0x0001,
    kFSNand                 = 0x8001,
    kFSOr                   = 0x0002,
    kFSNor                  = 0x8002,
    kFSEqual                = 0x0011,
    kFSNotEqual             = 0x8011,
    kFSGreaterThan          = 0x0012,
    kFSLessOrEqual          = 0x8012,
    kFSLessThan             = 0x0013,
    kFSGreaterOrEqual       = 0x8013,
    kFSContains             = 0x0024,
    kFSDoesntContain        = 0x8024,
    kFSStartsWith           = 0x0025,
    kFSDoesntStartWith      = 0x8025,
    kFSEndsWith             = 0x0026,
    kFSDoesntEndsWith       = 0x8026,
    kFSExists               = 0x0031,   // Unary Operator, FSAttribute as paramter
    kFSDoesntExist          = 0x8031    // Unary Operator, FSAttribute as paramter
    };
/*
 * modifiers for FSSearchOperators
 */

typedef enum {
    kIgnoreCase             = 0x0800,
```

```
    kIgnoreDiacriticals                 = 0x0400
    };

typedef uint16 FSSearchOperator;

/*
 * FSDataType
 */
enum {
    kFSNoDataType                       = 0,

    //** bitfields would be nice, but what grammar

    kFSDTint8,
    kFSDTuint8,
    kFSDTint16,
    kFSDTuint16,
    kFSDTint32,
    kFSDTuint32,
    kFSDTint64,
    kFSDTuint64,
    kFSDTPStr,                          // 255 is maximum length
    kFSDTCStr,
    kFSDTUniStr,
```

```
401    kFSDTDirectoryNum,
402    kFSDTFileNum,
403    kFSDTForkNum,
404    kFSDTFSName,
405    kFSDTFSDate,
406    kFSDTFileSystemKind,
407    kFSDTFSPermission,  //** needed? How would I Search for all files I can delete older than some date?
408
409    kFSDataTypeMask                 = 0x7F,
410    kFSOperandIsAttribute           = 0x80
411  };
412  typedef uint8 FSDataType;
413
414  /*
415   * FSChangeEvent defines the types of changes which can happen to the filesystem.
416   */
417  typedef enum {
418    kFSSubscriptionChanged          = 0x00,
419    kFSFileCreated                  = 0x01,
420    kFSFileDeleted                  = 0x02,
421    kFSFileMoved                    = 0x03,
422    kFSFileRenamed                  = 0x04,
423    kFSFileMoveRenamed              = 0x05,
424    kFSFileRead                     = 0x06,
425    kFSFileWritten                  = 0x07,
```


40 30 20 10

```
426     kFSFileSizeChanged      = 0x08,
427     kFSFilePermChanged      = 0x09,
428     kFSFileInfoChanged      = 0x0A,
429     kFSDirCreated           = 0x11,
430     kFSDirDeleted           = 0x12,
431     kFSDirMoved             = 0x13,
432     kFSDirRenamed           = 0x14,
433     kFSDirMoveRenamed       = 0x15,
434     kFSDirContentsSeen      = 0x16,
435     kFSDirContentsChanged   = 0x17,
436     kFSDirPermChanged       = 0x19,
437     kFSDirInfoChanged       = 0x1A,
438     kFSVolMounted           = 0x21,
439     kFSVolUnmounted         = 0x22,
440     kFSVolInfoChanged       = 0x2A,
441     kFSVolOnline            = 0x2B,
442     kFSVolOffline           = 0x2C
443     };
444 #define kMaxChangeEvent kFSVolOffline
445 typedef uint16 FSChangeEvent;
446
447 /*
448  * NTReportScope defines the scope of how changed can be reported.
449  * A change to a file or a directory can be reported on the object
450  * itself, on the directory which contains the object, or on the
```

```
 * volume which contains the object.
 */
typedef enum {
  kFSReportOnObject                 = 0x1000,
  kFSReportOnDirectory              = 0x2000,
  kFSReportOnVolume                 = 0x4000
  };
typedef uint16 FSChangeReportScope;

/*
 *    Notification subscribe options
 */
enum {
  kCollapseNotification             = 0x00000001    // 1 = collapse, 0 = all events
};

/*-----------------------------------------------------------------------------
 * BTree Access Constants
 */

/*
 * BTSetRecordMode is used do describe how a record is to be set in a BTree.
 * Currently, the only options control how the call reacts to the presence
 * or absense of a record already in the BTree with the same key. If the
 * requested state is NOT true, then an excetion is raised.
```

```
476  */
477 typedef enum {
478   kFSRecordMustExist              = 0x01,
479   kFSRecordMayExist               = 0x02,
480   kFSRecordMustNotExist           = 0x03
481   };
482 typedef uint16 BTSetRecordMode;
483
484 /*-----------------------------------------------------------------------------
485  * Access Control Constants
486  * The following types are used to control access to objects in the filesystem.
487  */
488
489 /*
490  * CommonPermissions is used to describe those permissions which are common
491  * to all entities in the filesystem.
492  */
493 typedef   enum {
494   kFSVisible                      = 0x01,     // can be seen
495   kFSCanDelete                    = 0x02,     // can be deleted
496   kFSChangePath                   = 0x04,     // can change its location
497   kFSChangeName                   = 0x08,     // can change its name
498   kFSChangePerm                   = 0x10,     // can change permissions
499   kFSChangeInfo                   = 0x20      // can change its other info
500 );
```


30 20 10

```
typedef   uint8 CommonPermissions;

/*
 * FilePermissions is used to describe those permissions which are
 * particular to files. It is employed both to maintain the persistent
 * permissions of a file and the access rights associated with a Path.
 */
typedef   enum {
  kFSRead                         = 0x01,     // can read data in file
  kFSWrite                        = 0x02,     // can write data to file
  kFSExtend                       = 0x04,     // can increase allocation of file
  kFSTruncate                     = 0x08,     // can decrease allocation of file
  kFSExecute                      = 0x10      // can execute code from file
};
typedef   uint8 FilePermissions;

/*
 * DirectoryPermissions is used to describe those permissions which are
 * particular to directories. In general, these are transitive permissions
 * which are applied to the contents of the directory. Thus to move a file
 * from one directory to another, you must have kFSChangePath permission
 * on the file itself, kFSRemoveEntity on its current directory, and
 * kFSAddEntity on the directory you are moving it to.
 */
typedef   enum {
```

```
526     kFSSeeFiles                     = 0x01,     // can see files inside
527     kFSSeeDirectories               = 0x02,     // can see folders inside
528     kFSAddEntity                    = 0x04,     // can add new entities
529     kFSRemoveEntity                 = 0x08,     // can remove existing entities
530     kFSRenameEntity                 = 0x10      // can rename an entity
531 };
532 typedef   uint8 DirectoryPermissions;
533
534 /*
535  * VolumePermissions is used to describe those permissions which are
536  * particular to volumes. It is only employed to maintain access rights
537  * associated with a mounted volume (i.e.there is no persistent meaning of
538  * volume permissions). Some explaination of volume structures is needed. These
539  * are the structures which makes a datastore a volume format. For HFS, this
540  * would be information stored in the MDB, Volume BitMap, Catalog File, Extents
541  * File, etc.
542  */
543 //.......
544 //....... I don't like the suffix 'VolumeStructures'. Does anyone have a better suffix?
545 //.......
546 typedef   enum {
547     kFSReadEntities                 = 0x01,     // allow non-distructive entity messages
548     kFSWriteEntities                = 0x02,     // allow change entity messages
549     kFSReadVolumeStructures         = 0x04,     // allow messages which read structures that makes a device a volume
550     kFSWriteVolumeStructures        = 0x08,     // allow messages which modify structures that makes a device a
```

```
volume
  kFSVolUnavailable                = 0x10      // allow messages which makes volume unavailable
};
typedef   uint8 VolumePermissions;


/*-----------------------------------------------------------------------------*/
/* TYPES:
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------
 * Foundation Types
 */

typedef EventNotification *EventNotificationPtr;  // get NuKernel to add this

/*
 * FileSystemKind is used to indicate a type of filesystem.
 * Apple reserves values with all lower case letters, and
 * those whose first two bytes are nil (for compatibility
 * with old FSID's)
 */
typedef OSType FileSystemKind;

/*
```

```
576  * FSAgentObjID is used to indicate a specific registered FS Agent.
577  * This value is not persistent across rebooting the machine or
578  * re-registering the given agent. It is in fact an object ID to
579  * which requests for the given agent should be sent.
580  */
581 typedef ObjectID FSAgentObjID;
582
583 /*
584  * VolumeObjID is used to indicate a specific mounted volume.
585  * This value is not persistent across rebooting the machine
586  * or remounting a given volume. It is in fact an object ID to
587  * which requests for the given volume should be sent.
588  */
589 typedef ObjectID VolumeObjID;
590
591 /*
592  * PathObjID is used to indicate a specific access path to a
593  * specific fork of a file. This value is session relative and
594  * not persistent between invocations of an application. It is
595  * in fact an object ID to which requests for the given path
596  * should be sent.
597  */
598 typedef ObjectID PathObjID;
599
600 /*
```

```
601   * BTreeObjID is used to indicate a access path using the
602   * BTree access method to a specific fork of a file. This value
603   * is session relative and not persistent between invocations of
604   * an application. It is in fact an object ID to which requests
605   * for the given path should be sent.
606   */
607  typedef ObjectID BTreeObjID;
608
609  /*
610   * FSAccessIdentity is used to indicate the persistent identity of
611   * a specific user or of a defined group of users.
612   */
613  typedef   uint32 FSAccessIdentity;
614
615  /*
616   * DirectoryNum is used to indicate a specific Directory within a
617   * particular volume. This is a persistent value, relative to
618   * the particular volume.
619   */
620  typedef   uint32 DirectoryNum;
621
622  /*
623   * FileNum is used to indicate a specific File within a particular
624   * volume. This is a persistent value, relative to the particular
625   * volume.
```

```
626  */
627  typedef   uint32 FileNum;
628
629  /*
630   * ForkNum is used to indicate a specific Fork within a
631   * particular File. This is a persistent value, relative to the
632   * particular file.
633   */
634  typedef   OSType ForkNum;
635
636  //
637  //··  This dates need to be thought out. What is the correct size of the
638  //··  datum? Should it be scalar? What is the epoch of the date? What is
639  //··  the precision of the date (e.g. seconds since epoch)?
640  //
641  typedef uint32 FSDate;
642
643  /*
644   * NotificationObjID is the ID by which a NotificatioQueue is referenced
645   */
646  typedef ObjectID NotificationObjID;
647
648  /*
649   * FSAttribute is used to indicate an attribute of a filesystem object.
650   * The structure of an FSAttribute is two OSTypes. The "high order"
```

```
651  * OSType defines the service which 'owns' the attribute, and the
652  * low order OSType specifies which of that services attributes it is.
653  */
654
655 typedef struct FSAttribute{
656     OSType          service;
657     OSType          attribute;
658     }FSAttribute, *FSAttributePtr;
659
660 /*
661  * FSAttributeDescriptor is used to describe a single attribute which is
662  * to be transferred into or out of a part of a user's buffer.
663  */
664
665 typedef struct FSAttributeDescriptor {
666     FSAttribute             attribute;
667     uint32                  length;
668     uint32                  offset;
669     }FSAttributeDescriptor, *FSAttributeDescriptorPtr;
670
671 /*--------------------------------------------------------------------------
672  *  File Specification Types
673  */
674
675 /*
```

```
676  * FSObjectInfo is used to get generic information about an object in
677  * a filesystem.
678  */
679
680 typedef struct {
681   unsigned int      entityType:3;      // use FSEntityType constants
682   unsigned int      isVolumeRoot:1;    // true if it is the root of a volume
683   unsigned int      isFMRoot:1;        // true if it is the FileManager's "meta" root
684   unsigned int      isBoot:1;          // true if it is (or is on) the Boot Volume
685   unsigned int      reserved:26;
686   } FSObjectInfo;
687
688 /*
689  * FSName is used to indicate the name of a File or Directory.
690  */
691 typedef UniChar FSName[32], *FSNamePtr;
692
693 /*
694  * FSPath is used to indicate a complete or partial pathname of
695  * a File or Directory.
696  */
697 typedef UniChar* FSPath;
698
699 /*
700  * FSSpecification is used to indicate a particular File or Directory.
```

```
701  * A proper FSSpecification has only a single name in its name field.
702  */
703 typedef struct FSSpecification {
704   VolumeObjID                volume;
705   DirectoryNum               parentDirectory;
706   FSName                     name;
707   } FSSpecification, *FSSpecificationPtr;
708
709 /*
710  * PathSpecification is used to indicate a particular File or Directory using
711  * a partial or complete pathname. A PathSpecification must be converted into a
712  * FSSpecification before it can be used by most requests. The 512 character
713  * limit implied by the definition of an FSPath, is a limit only to the size
714  * of this datatype. Users may in fact allocate larger structures with longer
715  * paths and the filesystem will handle them correctly.
716  */
717 typedef struct PathSpecification{
718   VolumeObjID                volume;
719   DirectoryNum               parentDirectory;
720   FSPath                     name;
721   } PathSpecification, *PathSpecificationPtr;
722
723 /*
724  * FSIterator is used to iterate over Files and Directories in the filesystem.
725  * Since the first element of this structure is an FSSpecification, it can be
```

```
726  * used in all places where FSSpecification's can be used.
727  */
728 typedef struct {
729   FSSpecification      specification;
730   FSObjectInfo         info;
731   void *               reserved;    // the iterator object ID
732   } FSIterator, *FSIteratorPtr;
733
734 /*
735  * FileClass is used to specify a class of files which may share some common
736  * attributes.
737  */
738 typedef struct {
739   OSType               type;
740   OSType               creator;
741   } FileClass, *FileClassPtr;
742
743 /*
744  * FileClassIterator is used to iterate over those FileClasses which have
745  * defined attributes. Other file classes may exist in the system which are
746  * unreachable by a FileClassIterator because they are undistibguished.
747  */
748 typedef struct {
749   FileClass            class;
750   void*                reserved;
```

```
751    } FileClassIterator, *FileClassIteratorPtr;
752
753
754 /*------------------------------------------------------------------------
755  * Permission and Access Types
756  */
757
758 /*
759  * IOPermission is used to describe the requested and/or actual access rights
760  * of an Access Path to a Fork. The thisPath field specifies what permissions
761  * are associated with the Path itself; and the other fields specify what
762  * access other Paths associated with the same Fork) are allowed to have at
763  * the same time.
764  */
765 typedef   struct {
766   FilePermissions              thisPath;    // what this path can do
767   FilePermissions              thisTeam;    // what other paths in same team can do
768   FilePermissions              thisAddressSpace; // what other paths in same address space can do
769   FilePermissions              others;      // what anyone else can do.
770 } IOPermission;
771
772 /*
773  * MountPermission is used to describe the requested and/or actual access rights
774  * of a mounted volume.
775  */
```

```
776 typedef   struct {
777   uint8                 reserved;     // set to NULL
778   VolumePermissions     thisTask;     // what this task can do
779   VolumePermissions     thisTeam;     // what other tasks in same team can do
780   VolumePermissions     others;       // what anyone else can do.
781 } MountPermission;
782
783 /*
784  * FSPermission is used to describe what kinds of actions a particular user or
785  * defined group of users can perform on a particular filesystem object.
786  */
787 typedef   struct {
788   CommonPermissions     comPerms;
789   union {
790       FilePermissions       filePerms;
791       DirectoryPermissions  dirPerms;
792   }                         entity;
793 } FSPermission;
794
795 /*
796  * PermissionSpecPtr is used to describe a particular permission which is
797  * associated with a File or Directory. The permission field defines what
798  * permission is granted, and the identity field defines what individual user
799  * or defined group received that permission.
800  */
```

```
typedef struct PermissionSpec {
  FSAccessIdentity          identity;
  FSPermission              permission;
} PermissionSpec, *PermissionSpecPtr;

/*
 * PermissionIterator is used to iterate over the permissions which have been
 * associated with a File or Directory. Since the first element of this
 * structure is a PermissionSpec, it can be used in all places where
 * PermissionSpec's can be used.
 */
typedef struct PermissionIterator {
  PermissionSpec            permission;
  uint32                    options;        /** need enum for isGroup
  void *                    reserved;
} PermissionIterator, *PermissionIteratorPtr;

/*---------------------------------------------------------------------------
 * Basic I/O Types
 */

/*
 * ForkPositionDescriptor is used to describe a particular position in a Fork.
 * In order to support files larger than 4GigaByte, the field posOffsetHigh is
 * the high order 32 bits of the offset, and posOffsetLow is the low order 32
```

```
826   * bits.
827   */
828  typedef struct ForkPositionDescriptor {
829    int64                          posOffset;
830    PositionMode                   posMode;
831    uint16                         filler1;        // because PositionMode is uint16
832    } ForkPositionDescriptor, *ForkPositionDescriptorPtr;
833
834  /*
835   * BufferDescriptor is used to describe an application's buffer for the purposes
836   * of reading and writing
837   */
838  typedef struct BufferDescriptor {
839    uint32                         length;
840    void*                          buffer;
841    } BufferDescriptor, *BufferDescriptorPtr;
842
843  /*
844   * FSElementDescriptor is used to describe an element within a buffer.
845   */
846  typedef struct {                     //** need a better name
847    uint32                         length;
848    uint32                         offset;
849    } FSElementDescriptor, *FSElementDescriptorPtr;
850
```

```
/*
 * IOOptions is used to define variations on read and write requests. Each option is
 * commented (in brackets) to indicate if it is applicable to Reads [R], Writes [W],
 * or Both [RW].
 */
typedef struct IOOptions {
  unsigned int              cachePriority:2;    // [RW] the priority of the data after the operation
  unsigned int              forceFlush:1;       // [W] force the data to be written out of cache
  unsigned int              verifyWrite:1;      // [W] force the flush and then read it back and compare
  unsigned int              terminatorMode:1;   // [R] reads are terminated by a terminator character
  unsigned int              unicodeTerminator:1;// [R] The terminator character is 16 bits
  unsigned int              reserved:10;        // reserved for future use
  unsigned int              terminatorChar:16;  // [R] The terminator character (lower 8 bits if not unicode)
  } IOOptions, *IOOptionsPtr;

/*
 * ProgressReport is used by Agents to provide feedback to the user on the
 * Agent's progress completing a request. There is an optional pointer to one
 * of these structures in every request header.
 */
typedef struct ProgressReport {
  uint32                    totalEffort;    // a unitless quantity set to how much
                                            // effort the Agent thinks this request
                                            // will take. This should be intialized
                                            // to 0 when the request is sent to the
```

```
876                                                 // File Manager, and the Agent should set
877                                                 // it to the appropriate value as soon as
878                                                 // it gets the request. A value of 0xffffffff
879                                                 // indicates that the request is of unknown
880                                                 // duration, and therefore progress can not
881                                                 // reported
882
883   uint32                    effortComplete;     // relative to totalEffort, how much of
884                                                 // the request has been completed.
885
886   EventNotification         updateEvent;        // an event which is invoked when the Agent
887                                                 // first sets totalEffort, and whenever it
888                                                 // changes effortComplete. If updateEvent
889                                                 // is all nil's, no action is taken (it is
890                                                 // assumed that the user is polling the
891                                                 // effortComplete locations.
892
893   } ProgressReport, *ProgressReportPtr;
894
895 /*-------------------------------------------------------------------------
896  * BTree Access Method Types
897  */
898
899 /*
900  * BTIIint is used to indicate a starting point for BTree Searches.
```

```
901  */
902  typedef struct BTHint {
903    uint32                reserved[4];
904    } BTHint, *BTHintPtr;
905
906  /*
907   * BTreeKeyDescriptor is used to indicate how keys for a particular B*Tree
908   * are to be compared.
909   */
910  typedef char BTreeKeyDescriptor[26], *BTreeKeyDescriptorPtr;
911
912  /*
913   * BTreeInformation is used tq describe the public information about a BTree
914   */
915
916  typedef struct BTreeInformation{
917    uint16                NodeSize;
918    uint16                MaxKeyLength;
919    uint16                Depth;
920    uint32                NumRecords;
921    uint32                NumNodes;
922    uint32                NumFreeNodes;
923    uint32                ClumpSize;
924    BTreeKeyDescriptor    KeyDescriptor;
925    } BTreeInformation, *BTreeInformationPtr;
```

```
926
927 /*-----
928  * Information Types
929  * The following types are used to convey information about
930  * objects in the filesystem.
931  */
932
933 /*
934  * VolumeFunctionality is used to describe what functionality is supported by
935  * a particular volume. This includes such things as number of forks,
936  * permissions model, and the like.
937  */
938 typedef struct VolumeFunctionality {
939     unsigned int        supportsNotification    :1;
940     unsigned int        supportsAFPPermissions  :1;
941     unsigned int        supportsACLPermissions  :1;
942     unsigned int        supportsBTree           :1;
943     unsigned int        supportsRangeLocking    :1;
944     unsigned int        supportsFileNums        :1;
945     unsigned int        supportsAttributes      :1;
946     unsigned int        supportsUserAttributes  :1;
947     unsigned int        supportsShortNames      :1;
948     unsigned int        supportsMoveRename      :1;
949     unsigned int        supportsCopyRename      :1;
950     unsigned int        supportsAccessDeny      :1;
```

```
    unsigned int        supportsBooting     :1;
    unsigned int        supportsSystem      :1;
    unsigned int        supportsDesktop     :1;
    unsigned int        reserved            : 16;
    } VolumeFunctionality, *VolumeFunctionalityPtr;

/*
 * VolumeROInformation is used to describe the read only characteristics of a
 * particular Volume.
 */
typedef struct VolumeROInformation {
    int32               driveID;
    int32               partitionID;
    int32               volumeBlockSize;
    int32               numFiles;
    int32               numDirectories;
    } VolumeROInformation, *VolumeROInformationPtr;

/*
 * VolumeRWInformation is used to describe the read/write characteristics of a
 * particular Volume.
 */
typedef struct VolumeRWInformation {
    int16               attributes;
    FSDate              createDate;
```

```
    FSDate          modifyDate;
    FSDate          backupDate;
    FSName          volumeName;
} VolumeRWInformation, *VolumeRWInformationPtr;

/*
 * DirectoryROInformation is used to describe the read only characteristics of
 * a particular Directory.
 */
typedef struct DirectoryROInformation {
    int32           parentID;
    int32           DirectoryNum;
} DirectoryROInformation, *DirectoryROInformationPtr;

/*
 * DirectoryRWInformation is used to describe the read/write characteristics
 * of a particular Directory.
 */
typedef struct DirectoryRWInformation {
    FSName          fileName;
    FSDate          createDate;
    FSDate          modifyDate;
    FSDate          backupDate;
} DirectoryRWInformation, *DirectoryRWInformationPtr;
```

```
1001 /*
1002  * FileROInformation is used to describe the read only characteristics of
1003  * a particular File.
1004  */
1005 typedef struct FileROInformation {
1006     int32               parentID;
1007     int32               FileNum;
1008 } FileROInformation, *FileROInformationPtr;
1009
1010 /*
1011  * FileRWInformation is used to describe the read/write characteristics of
1012  * a particular File.
1013  */
1014 typedef struct FileRWInformation {
1015     FSName              fileName;
1016     FSDate              createDate;
1017     FSDate              modifyDate;
1018     FSDate              backupDate;
1019     OSType              fileType;
1020     OSType              fileCreator;
1021 } FileRWInformation, *FileRWInformationPtr;
1022
1023 /*
1024  * ForkROInformation is used to describe the read only characteristics of
1025  * a particular Fork.
```

```
1026  */
1027  typedef struct ForkROInformation {
1028      int32                forkType;
1029      int64                logicalEOF;
1030      int64                physicalEOF;
1031  } ForkROInformation, *ForkROInformationPtr;
1032
1033  /*
1034   * ForkRWInformation is used to describe the read/write characteristics of
1035   * a particular Fork.
1036   */
1037  typedef struct ForkRWInformation {
1038      OSType               format;
1039  } ForkRWInformation, *ForkRWInformationPtr;
1040
1041  /*
1042   * PathROInformation is used to describe the read only characteristics of
1043   * a particular Path. This includes such things as cache characteristics
1044   * and permissions.
1045   */
1046  typedef struct PathROInformation {
1047      TaskID               taskID;
1048      int32                userID;
1049      FSPermission         accessPrivs;
1050  } PathROInformation, *PathROInformationPtr;
```

```
1051
1052 /*
1053  * PathRWInformation is used to describe the read/write characteristics of
1054  * a particular Path. This includes such things as cache characteristics
1055  * and permissions.
1056  */
1057 typedef struct PathRWInformation {
1058   int64                         mark;
1059   } PathRWInformation, *PathRWInformationPtr;
1060
1061 /*
1062  * Following are Finder file and directory info structs.
1063  */
1064
1065 #if defined(powerc) || defined (__powerc)
1066   #pragma options align=mac68k
1067 #endif
1068 struct FinderFileInfo {
1069   OSType                        fdType;         /*the type of the file*/
1070   OSType                        fdCreator;      /*file's creator*/
1071   unsigned short                fdFlags;        /*flags ex. hasbundle,invisible,locked, etc.*/
1072   Point                         fdLocation;     /*file's location in folder*/
1073   short                         fdFldr;         /*folder containing file*/
1074 };
1075 #if defined(powerc) || defined (__powerc)
```

```
1076   #pragma options align=reset
1077 #endif
1078
1079 typedef struct FinderFileInfo FinderFileInfo;
1080
1081 #if defined(powerc) || defined (__powerc)
1082   #pragma options align=mac68k
1083 #endif
1084 struct FinderXFileInfo {
1085   short                fdIconID;       /*Icon ID*/
1086   short                fdUnused[3];    /*unused but reserved 6 bytes*/
1087   char                 fdScript;       /*Script flag and number*/
1088   char                 fdXFlags;       /*More flag bits*/
1089   short                fdComment;      /*Comment ID*/
1090   long                 fdPutAway;      /*Home Dir ID*/
1091 };
1092 #if defined(powerc) || defined (__powerc)
1093   #pragma options align=reset
1094 #endif
1095
1096 typedef struct FinderXFileInfo FinderXFileInfo;
1097
1098 #if defined(powerc) || defined (__powerc)
1099   #pragma options align=mac68k
1100 #endif
```

```
1101 struct FinderDirInfo {
1102   Rect          frRect;          /*folder rect*/
1103   unsigned short frFlags;        /*Flags*/
1104   Point         frLocation;      /*folder location*/
1105   short         frView;          /*folder view*/
1106 };
1107 #if defined(powerc) || defined (__powerc)
1108   #pragma options align=reset
1109 #endif
1110
1111 typedef struct FinderDirInfo FinderDirInfo;
1112
1113 #if defined(powerc) || defined (__powerc)
1114   #pragma options align=mac68k
1115 #endif
1116 struct FinderXDirInfo {
1117   Point         frScroll;        /*scroll position*/
1118   long          frOpenChain;     /*DirID chain of open folders*/
1119   char          frScript;        /*Script flag and number*/
1120   char          frXFlags;        /*More flag bits*/
1121   short         frComment;       /*comment*/
1122   long          frPutAway;       /*DirID*/
1123 };
1124 #if defined(powerc) || defined (__powerc)
1125   #pragma options align=reset
```

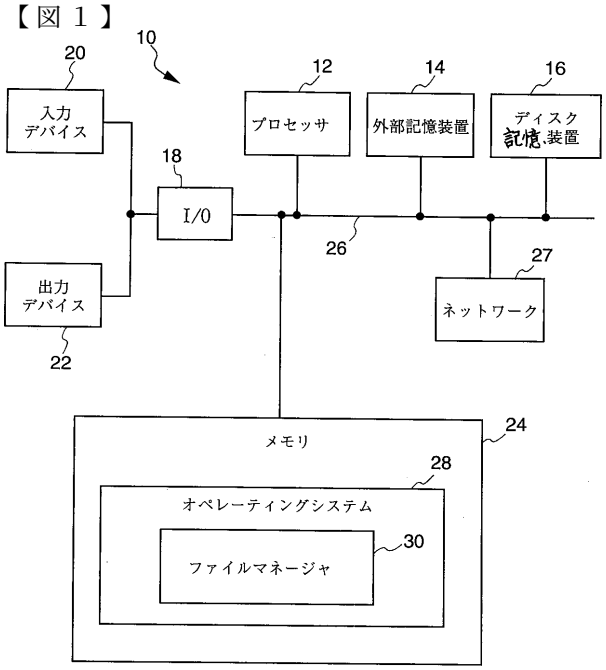
```
#endif

typedef struct FinderXDirInfo FinderXDirInfo;

/*-----------------------------------------------------------------------------
 * Search Types
 */

/*
 * FSComparisonDescriptor is used to describe a comparison between an attribute
 * and value or two attributes.
 *
 * Operand1Type, Operand2Type, operand1Length, and operand2Length are NULL when
 * there is no comparision required for operator (e.g. kFSAnd, kFSExists).
 */
typedef struct {
  FSSearchOperator            operator;
  FSDataType                  operand1Type;
  FSDataType                  operand2Type;   //** so we could support mixed types in the future
  uint16                      operand1Length; // how many instances of "type"
  uint16                      operand2Length; //** so we could support mixed types in the future
  } FSComparisonDescriptor, *FSComparisonDescriptorPtr;

/*
 * FSSearchElement is used to describe part of a FSSearchCriteria. A Search
```

```
 * Criteria is essentially a prefix expression stack. Therefore, each element
 * of the expression is either an operator, or an attribute specification
 * followed by a pointer to a buffer descriptor which contains a value for
 * that attribute.
 */
typedef union FSSearchElement {
  FSComparisonDescriptor      op;
  FSAttribute                 attr;
  FSElementDescriptor         val;            // describes position and length within valBuffer
  } FSSearchElement, *FSSearchElementPtr;

/*
 * FSSearchCriteria is used to describe what kinds of files the user is
 * searching for. As noted above it is a prefix expression stack.
 */
typedef struct FSSearchCriteria {
  int32                       count;          // the number of elements in the operator
  FSSearchElement             criteria[1];    // THIS A VARIABLE LENGTH ARRAY
//**  uint32                      valBuffer[0];   // WHICH IS FOLLOWED BY THIS VARIABLE LENGTH ARRAYthe data for
search elements
  } FSSearchCriteria, *FSSearchCriteriaPtr;

/*
 * FSSimpleSearchCriteria is a simplified structure for the most simple
 * and most common sorts of searches. It still needs to be cast into a
```

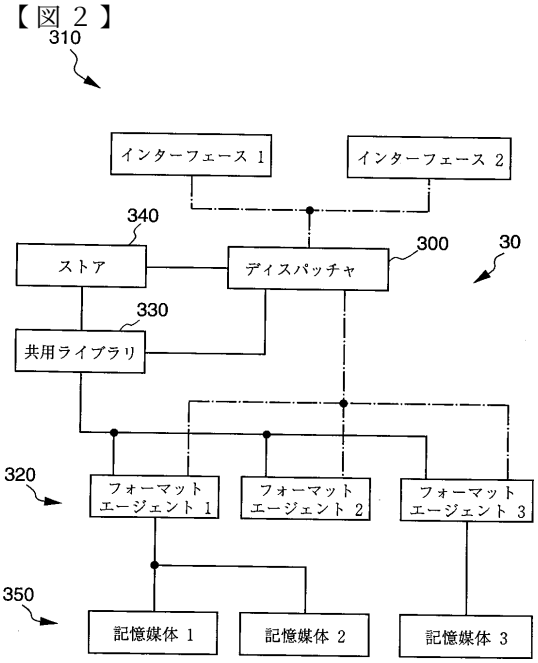
```
 * FSSearchCriteria before it can be passed to FSReqSearch.
 */
typedef struct FSSimpleSearchCriteria {
  int32                    count;          // MUST BE = 3
  FSComparisonDescriptor   op;             // usually = kFSEqual
  FSAttribute              attr;
  FSElementDescriptor      val;
//**  uint32                   valBuffer[0];   // the data for 'val' search elements
  } FSSimpleSearchCriteria, *FSSimpleSearchCriteriaPtr;

/*-----------------------------------------------------------------------------
 * Notification Types
 * The following types are used to control notification of events
 * in the filesystem.
 */

/*
 * NTChangeReport is used to describe specific change to the filesystem.
 * If the scope of the change is on the Directory of Volume, the 'it' field
 * is nil and should be ignored. If the scope of the change is on the
 * Volume, the oldDir and newDir fields are nil and should be ignored.
 */
typedef struct {
  int16                    count;      // the number of reports
                                       // which have been colapsed
```

```
1201                                    // into this message
1202    FSChangeEvent       change;     // What happened to 'it'?
1203    VolumeObjID         oldVol;     // What volume was 'it' on?
1204    VolumeObjID         newVol;     // What volume IS 'it' on now?
1205    DirectoryNum        oldDir;     // What dir WAS 'it' in?
1206    DirectoryNum        newDir;     // What dir IS 'it' in now?
1207
1208    uint32              oldIt;      // 'it's' ID before the change
1209                                    // (either a FileNum for a file, or a
1210                                    // DirectoryNum for a dir or a
1211                                    // VolumeObjID for a volume.
1212                                    // You can tell which by the
1213                                    // value of the change field
1214
1215    uint32              newIt;      // 'it's' ID after the change
1216    ) FSChangeReport , *FSChangeReportPtr;
1217
1218
1219 #endif //__FSTYPES__
1220
```

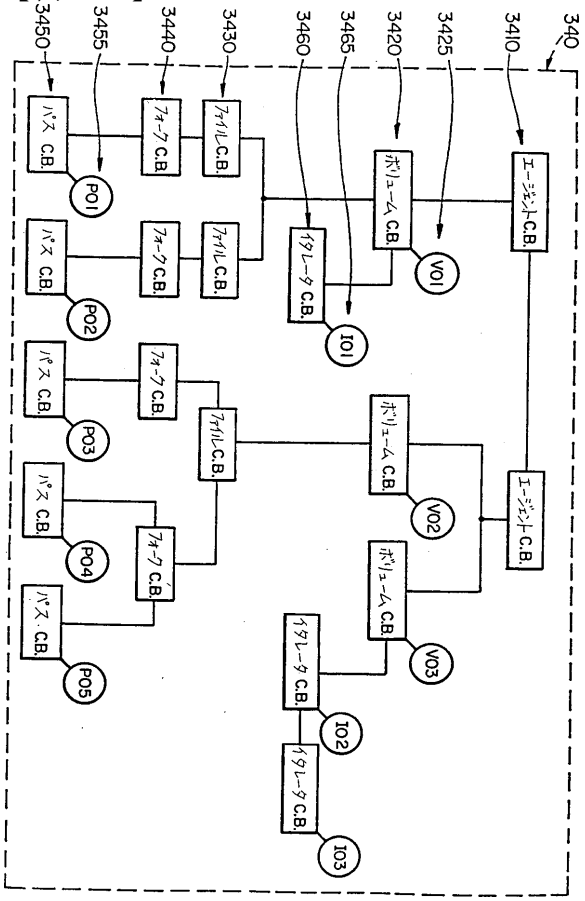
【図１】



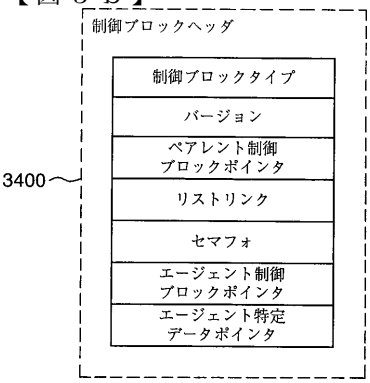
【図２】



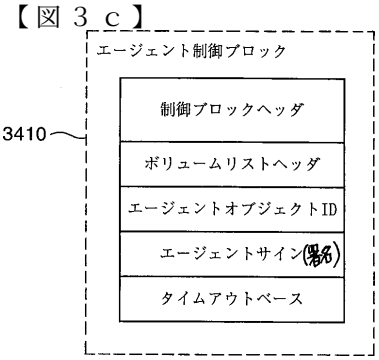
【図３ａ】



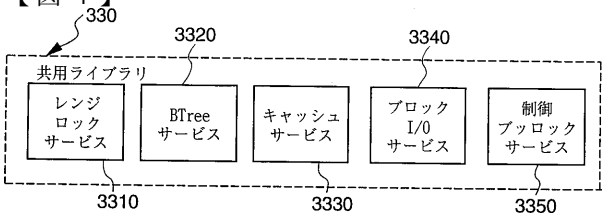
【図３ｂ】



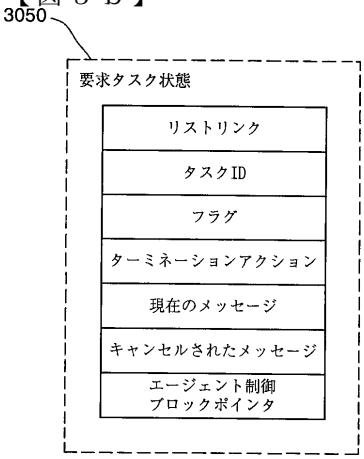
【図３ｃ】



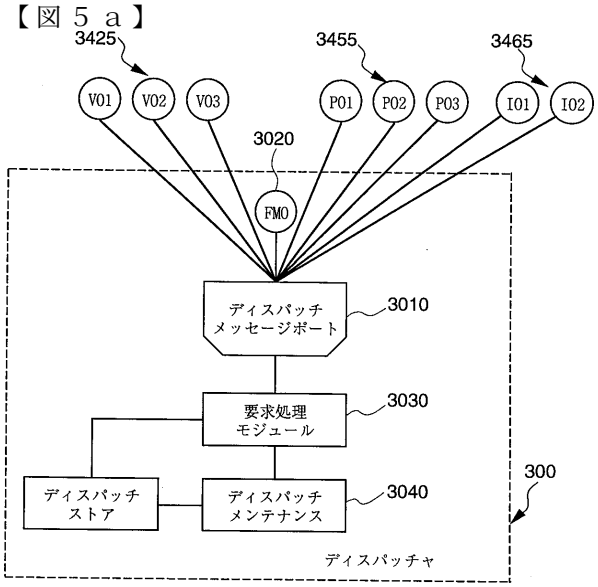
【図４】



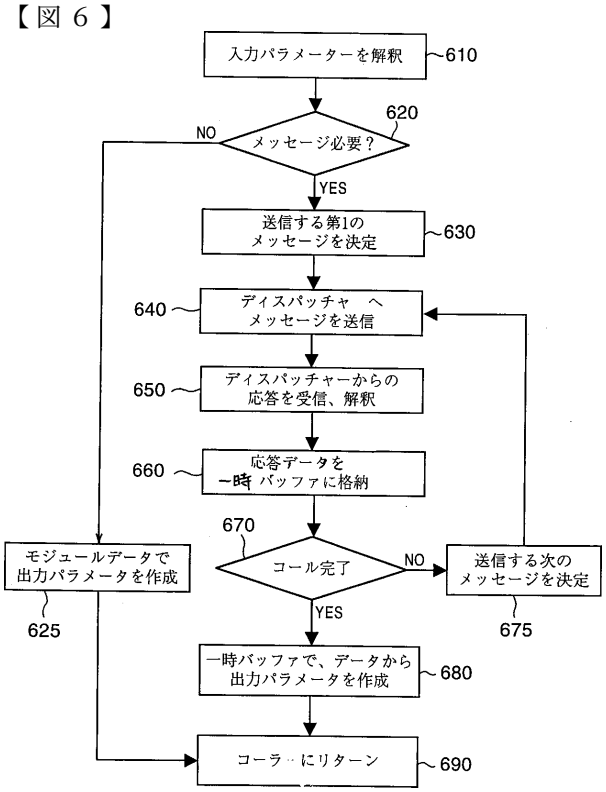
【図５ｂ】



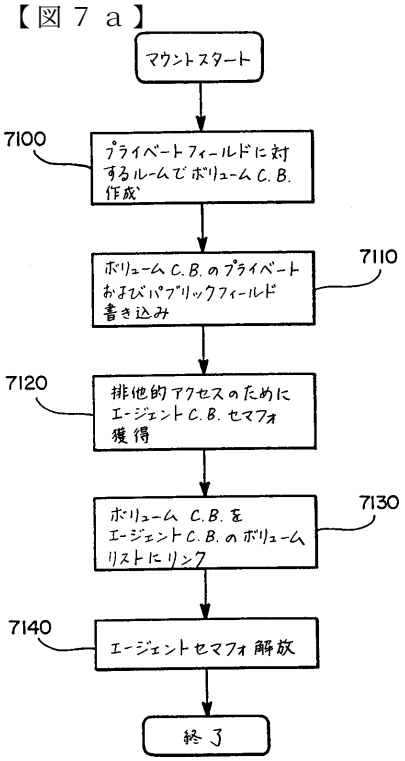
【図５ａ】



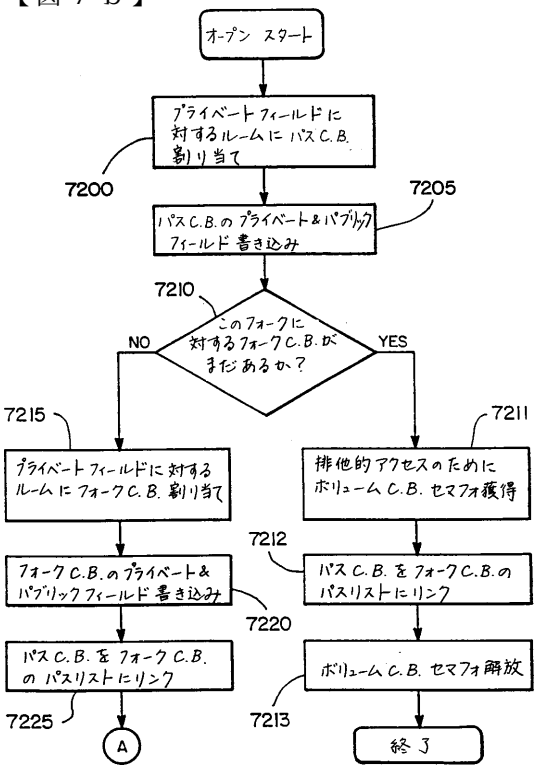
【図６】



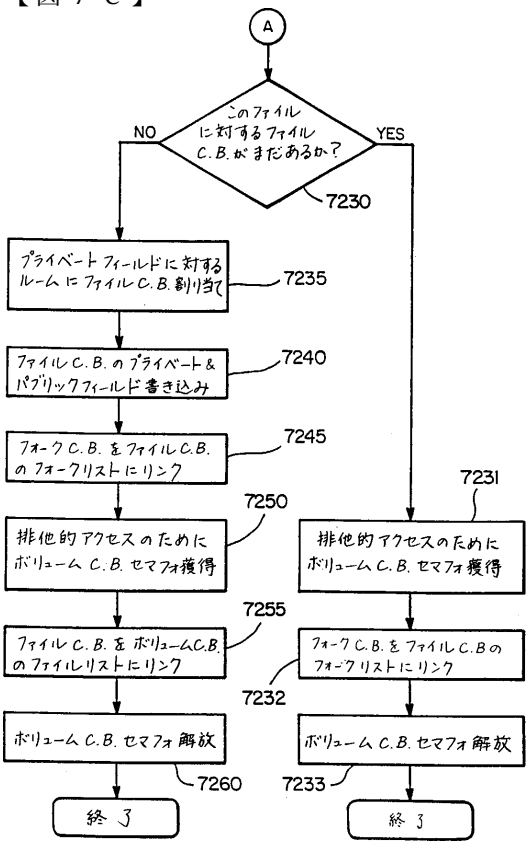
【図７ａ】



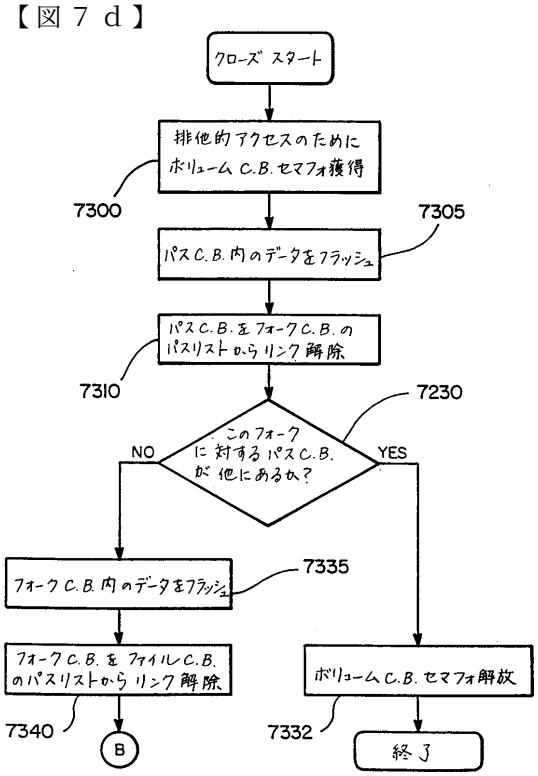
【図７ｂ】



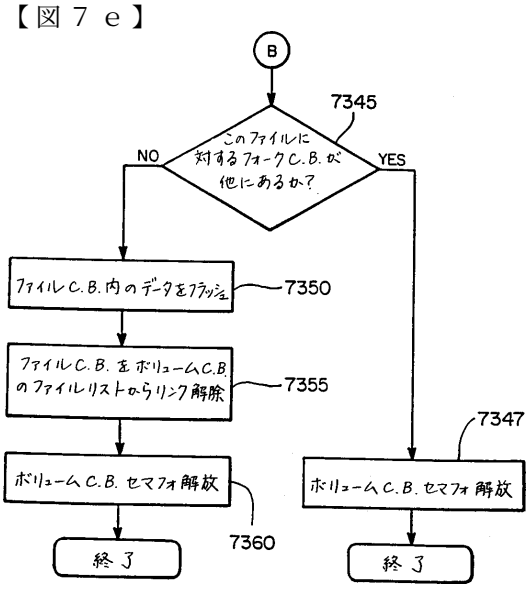
【図７ｃ】



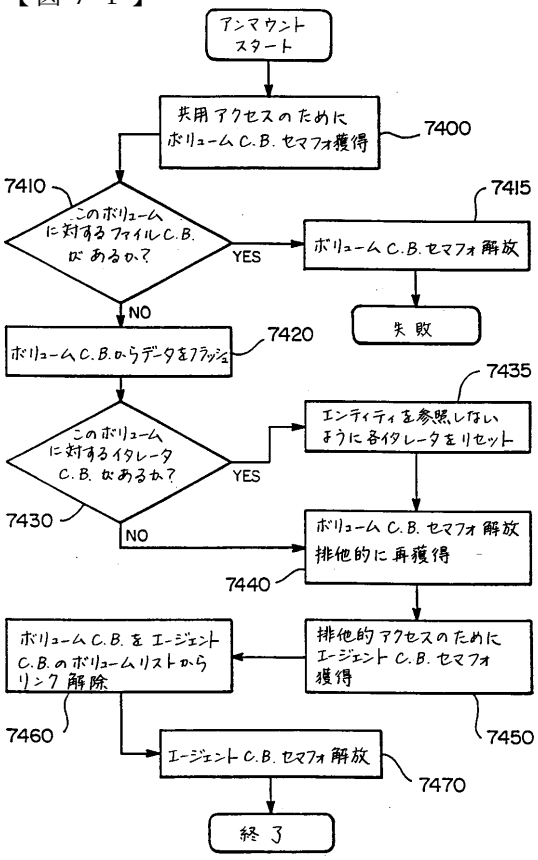
【図７ｄ】



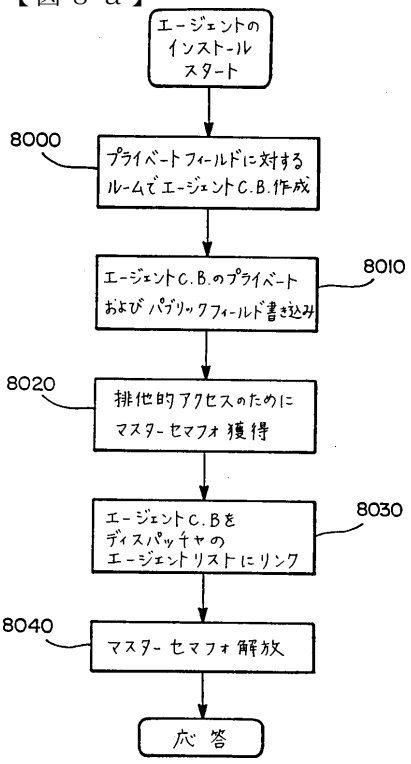
【図７ｅ】



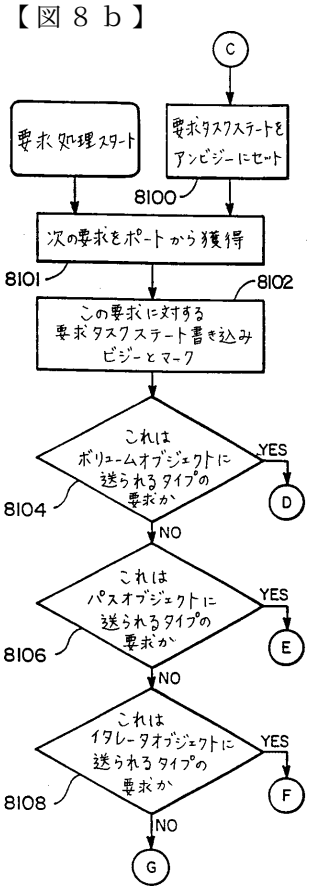
【図７ｆ】



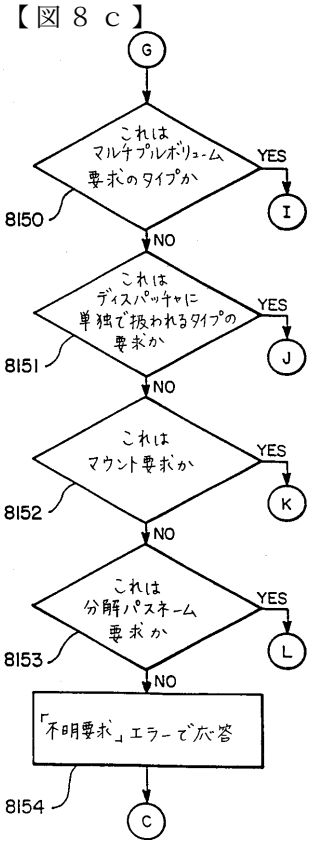
【図８ａ】



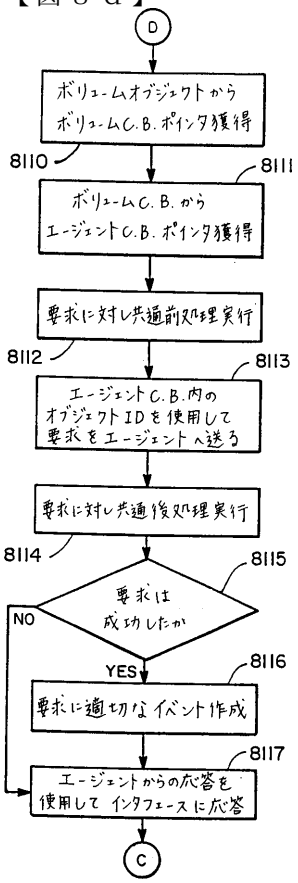
【図８ｂ】



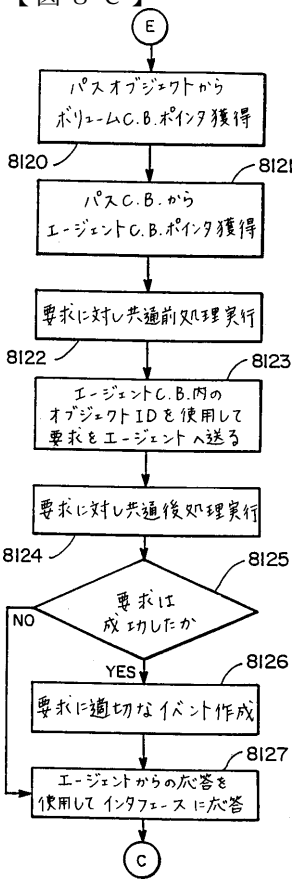
【図８ｃ】



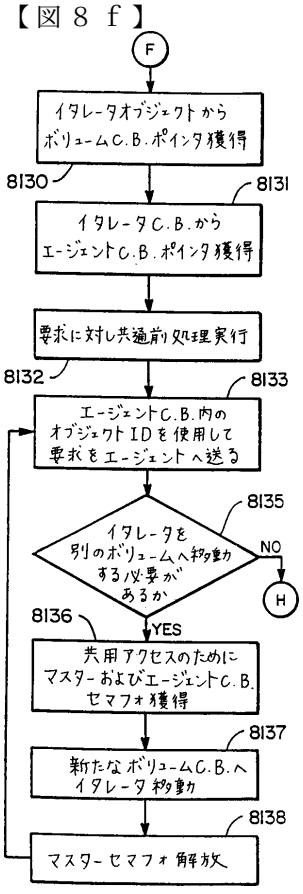
【図８ｄ】



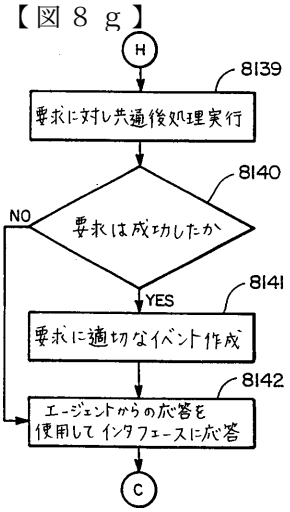
【図８ｅ】



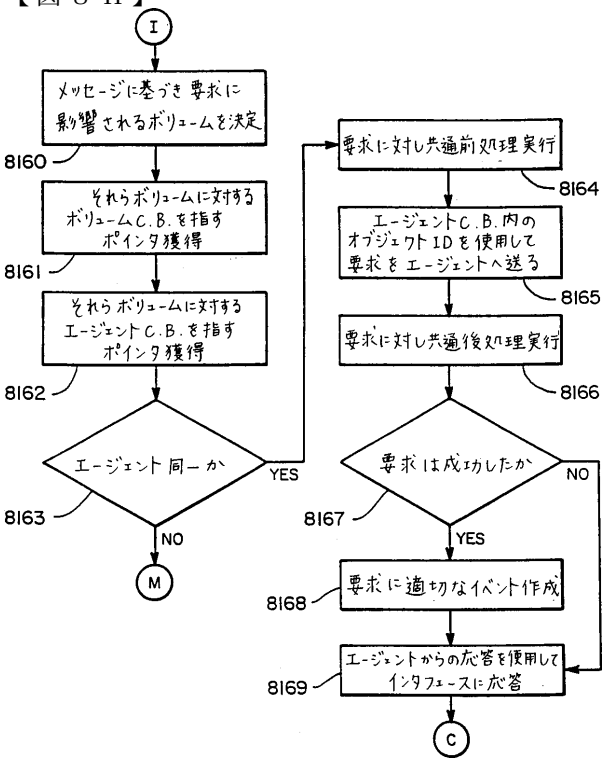
【図８ｆ】



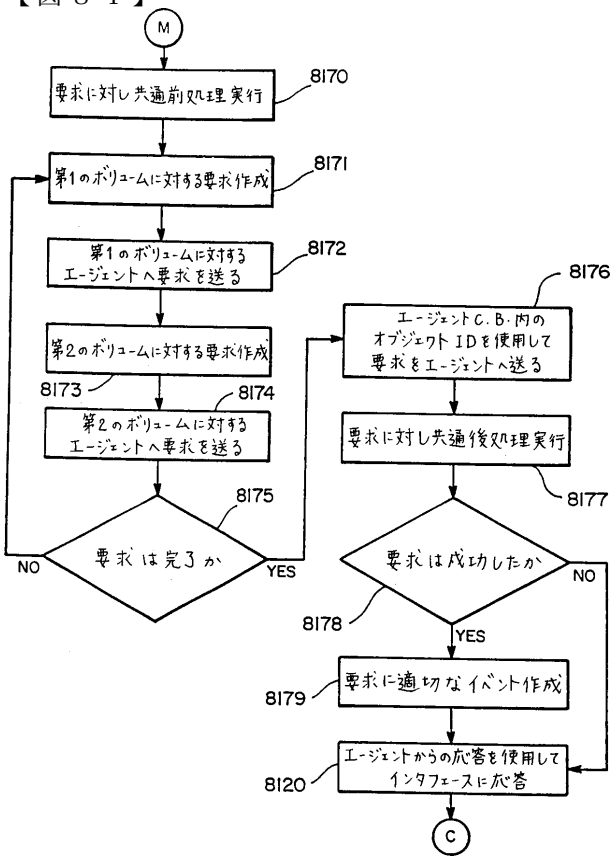
【図８ｇ】



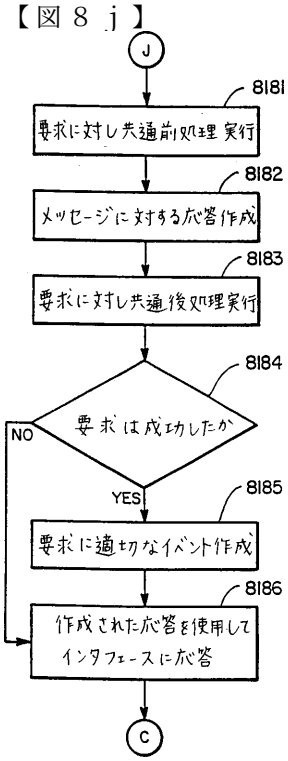
【図８ｈ】



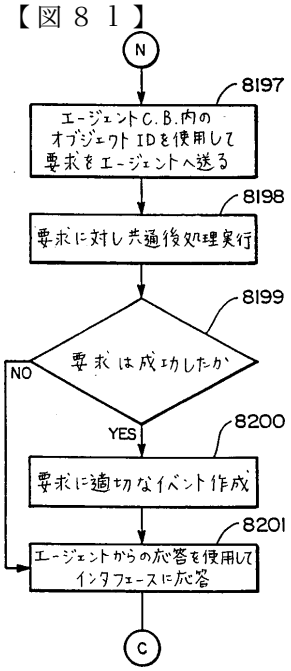
【図８ｉ】



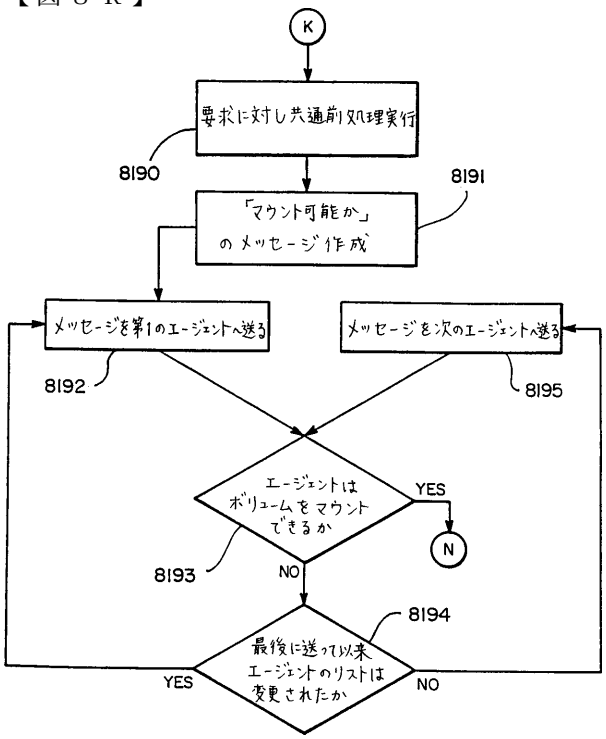
【図８ｊ】



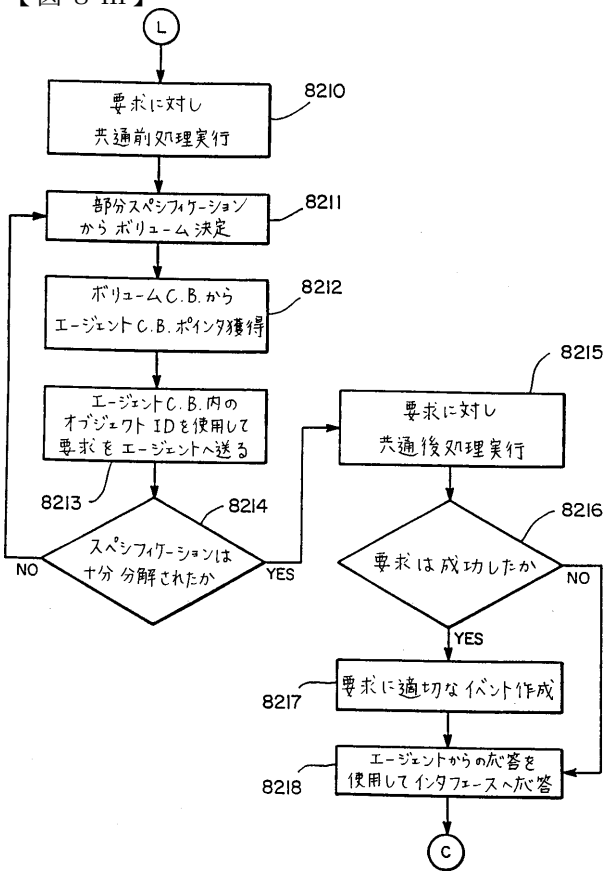
【図８ｌ】



【図８ｋ】


K
要求に対し共通前処理実行
8190
「マウント可能か」
のメッセージ作成
8191
メッセージを第1のエージェントへ送る
メッセージを次のエージェントへ送る
8192
8195
エージェントは
ボリュームをマウント
できるか
YES
N
8193
NO
8194
最後に送って以来
エージェントのリストは
変更されたか
YES
NO

【図８ｍ】


L
要求に対し
共通前処理実行
8210
部分スペシフィケーション
からボリューム決定
8211
ボリュームC.B.から
エージェントC.B.ポインタ獲得
8212
エージェントC.B.内の
オブジェクトIDを使用して
要求をエージェントへ送る
8213
スペシフィケーションは
十分分解されたか
8214
NO
YES
8215
要求に対し
共通後処理実行
要求は成功したか
8216
NO
YES
8217
要求に適切なイベント作成
8218
エージェントからの応答を
使用してインタフェースへ応答
C

フロントページの続き

(72)発明者 ズィマンスキー, スティーブン, ジェイムズ
アメリカ合衆国 カリフォルニア州 ９５０１４ クッパチーノ, ロドリーグス アベニュー ２００６０ アパートメント ディー （番地なし）
(72)発明者 ブラッフィー, ビル, モンロー
アメリカ合衆国 カリフォルニア州 ９５０１４ クッパチーノ, ミラ ビスタ ロード １０３７１

審査官 金子 幸一

(56)参考文献 特開平０６－０７５８４６（ＪＰ，Ａ）
S. R. Kleiman, Vnodes: An Architecture for Multiple File System Types in Sun UNIX, USENIX Association Summer Conference Proceedings, 米国, USENIX Assoc., １９８６年 ６月, p. 238-247

(58)調査した分野(Int.Cl., ＤＢ名)
G06F 12/00 501
G06F 12/00 511
G06F 12/00 514