

(19) 日本国特許庁(JP)

(12) 特 許 公 報(B2)

(11) 特許番号

特許第4393191号

(P4393191)

(45) 発行日 平成22年1月6日(2010.1.6)

(24) 登録日 平成21年10月23日(2009.10.23)

(51) Int.Cl.

F I

G 0 6 F 12/00 (2006.01)

G 0 6 F 12/00 5 1 7

G 0 6 F 9/445 (2006.01)

G 0 6 F 9/06 6 5 0 C

請求項の数 1 (全 24 頁)

(21) 出願番号	特願2003-537233 (P2003-537233)	(73) 特許権者	500046438
(86) (22) 出願日	平成14年10月17日(2002.10.17)		マイクロソフト コーポレーション
(65) 公表番号	特表2005-539277 (P2005-539277A)		アメリカ合衆国 ワシントン州 9805
(43) 公表日	平成17年12月22日(2005.12.22)		2-6399 レッドモンド ワン マイ
(86) 国際出願番号	PCT/US2002/033145		クロソフト ウェイ
(87) 国際公開番号	W02003/034634	(74) 復代理人	100115624
(87) 国際公開日	平成15年4月24日(2003.4.24)		弁理士 濱中 淳宏
審査請求日	平成17年10月17日(2005.10.17)	(74) 復代理人	100128015
(31) 優先権主張番号	09/982,764		弁理士 堀田 誠
(32) 優先日	平成13年10月18日(2001.10.18)	(74) 代理人	100077481
(33) 優先権主張国	米国 (US)		弁理士 谷 義一
前置審査		(74) 代理人	100088915
			弁理士 阿部 和夫

最終頁に続く

(54) 【発明の名称】 拡張可能ファイル形式

(57) 【特許請求の範囲】

【請求項 1】

コンピュータ上で現在動作しているアクティブバージョン、前記アクティブバージョンの前に作成された旧バージョンおよび前記アクティブバージョンの後に作成された後のバージョン間のアプリケーションプログラムの互換性を保ったまま、前記コンピュータがアプリケーションプログラムにデータファイルをロードする方法であって、

前記アプリケーションプログラムが最高位のバージョンであることを示すデータからなる上位バージョン透かしおよび前記アプリケーションプログラムが最終バージョンであることを示すデータからなる最終バージョン透かしを前記データファイルに入れるステップと、

前記上位バージョン透かしが示す最高位のバージョンと前記アクティブバージョンとを比較して前記最高位バージョンが前記アクティブバージョンよりも新しいかどうかを判別し、新しいと判別された場合には知られているすべての情報をロードし未知のすべての情報をスキップして前記アプリケーションプログラムに前記データファイルをロードし、新しくないと判別された場合には前記データファイルをそのままアプリケーションプログラムにロードするステップと、

前記最終バージョン透かしが示す最終バージョンと前記アクティブバージョンとを比較して前記最終バージョンが前記旧バージョンに対応しているかどうかを判別し、前記最終バージョンが前記旧バージョンに対応していると判別された場合には前記データファイル内の古い情報に基づき新しい情報を更新して前記データファイルを変換するステップと、

10

20

を含むことを特徴とする方法。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、一般に、コンピュータシステムおよびプログラムに関する。より具体的には、本発明は、アプリケーションプログラムの旧バージョン、アクティブバージョン、および将来のバージョンと互換性のあるファイル形式を規定することができるシステムおよび方法に関する。

【背景技術】

【0002】

コンピュータアプリケーションプログラムは、一般に、多数のコンピュータ処理タスクを実行するために使用される。アプリケーションプログラムを使用して、アプリケーションプログラムに関する特定の項目の情報を格納するデータファイルを作成し修正することができる。例えば、デスクトップパブリッシングタイプのアプリケーションプログラムを使用して、ドキュメントを作成することができる。ドキュメントには、ヘッダ、テキスト、グラフィックス、スタイル、フォントなど多数の個別要素を入れることができる。パブリッシングタイプのアプリケーションプログラムは、その特定のドキュメントのすべての情報を収めたデータファイルを作成するために使用できる。

【0003】

ファイルをコンピュータシステムに保存するアプリケーションはすべて、データを保存する形式を決定しなければならない。データを保存する方法としては、データからメモリイメージを保存する方法から、業界標準形式でデータを書き出す方法までさまざまな方法がある。さらに、コンピュータアプリケーションプログラムは、最新の機能および技術を利用できるように更新および変更が常時行われている。更新および変更は、通常、アプリケーションプログラムの更新バージョンをリリースするという形でエンドユーザに提供される。アプリケーションプログラムの更新バージョンで使用されるファイル形式は、旧バージョンおよび将来のバージョンと互換性がなければならない。しかし、更新バージョンの新しい機能では新しいデータをデータファイルに書き込む必要があるため、旧バージョンと異なるファイル形式が更新バージョンに使用されるのがふつうである。ファイル形式が異なると旧バージョンで更新バージョンのファイルを読み込めなくなる可能性があるため、2つの異なるバージョンのアプリケーションプログラムに問題が生じることがある。

【0004】

本開示で使用しているような「オブジェクト」という用語は、特性を持ち、アプリケーションプログラムにより表示されるか、またはアプリケーションプログラムの一部であるか、またはプログラムによって格納／操作されるデータの一部であるエンティティを指す。アプリケーションプログラムにより表示されるオブジェクトの一例は、ユーザがテキストを入力できるアプリケーションプログラムのウインドウに含まれるテキストボックスである。テキストボックスの特性には、その色、テキストのフォント、およびテキストのポイントサイズがある。アプリケーションプログラムの一部であるオブジェクトの一例として、動物のメモリ内表現 (in-memory representation) があり、色、脚の数、および肉食かどうかなどの特性が与えられる。このメモリ内表現はデータ構造体として実装することができ、データ構造の要素に特性を格納する。このようなデータ構造体の一例に、C++クラスデータ構造体がある。オブジェクトの特性を、そのオブジェクトのプロパティと呼ぶ。オブジェクトの各プロパティは、通常、値を持つ。例えば、色のプロパティは赤という値を持つことができる。

【0005】

オブジェクト100の一例を図1に示す。オブジェクト100は、枠線 (border) 102とテキスト104とからなる。オブジェクト100は、枠線102の6つのプロパティとテキスト104とを備えることができる。枠線102は、枠線スタイルと枠線サイズとのプロパティを持つことができる。テキスト104は、フォント、テキストサイズ

10

20

30

40

50

、位置揃え、およびテキストスタイルのプロパティを持つことができる。枠線 1 0 2 とテキスト 1 0 4 とのデフォルト値では、枠線スタイルを実線、枠線サイズを 4 ポイント、テキストフォントを Times New Roman、テキストサイズを 2 0 ポイント、テキスト位置揃えを左寄せ、テキストスタイルを非イタリック体とすることができる。

【0006】

アプリケーションプログラムでは、初期プログラミングの際にオブジェクトおよびプロパティのデフォルト値を設定することができる。デフォルト値は、プログラムの動作中に、変更してデータファイルに格納することができる。例えば、四角形や三角形などのオブジェクトを表示できる描画プログラムでは、ユーザがオブジェクトのプロパティを修正することができる。図 1 の例に戻ると、オブジェクト 1 0 0 のプロパティを変更することで、実線の枠線を破線の枠線にし、テキストスタイルをイタリック体にするることができる。新しいバージョンのプログラムであれば、デフォルト値と、各オブジェクトの変更を読み取れるのがふつうである。しかし、新しいバージョンのプログラムが作成されると、ファイル形式は通常、旧バージョンのファイル形式とは異なる。したがって、旧バージョンでは、新バージョンのファイルを読み取れないことがある。上述のパラグラフからの上記の例を使用すると、新しいバージョンに、オブジェクト 1 0 0 のシャドウ (shadow) などの新しい機能を含めることが可能である。追加機能は、新しいデータをプログラムのファイルに書き込み、そうすることでその形式を変更する必要がある。旧バージョンからでは、ファイル形式が異なるため、新しいデータが含まれる新しいバージョンのファイルを読み込むことはできない。

【0007】

【特許文献 1】米国特許第 5 9 3 3 8 4 2 号明細書

【特許文献 2】米国特許第 5 9 4 6 6 9 6 号明細書

【発明の開示】

【発明が解決しようとする課題】

【0008】

上述の問題に対する従来の解決策では、更新されたバージョンの新しい機能を制限することで、旧バージョンとのファイル形式の互換性を維持する。しかし、このような解決策では、最良の製品をユーザに配布することができず、プログラム全体の機能およびパフォーマンスを制限することになりかねない。

【0009】

上述の問題に対する他の従来の解決策では、旧ファイルを旧バージョンから更新バージョンと互換性のある新しいファイルに変換するアダプタプログラム (adapter program) を作成する。アダプタプログラムはさらに、新しいファイルを旧バージョンと互換性のあるファイルに変換することもできる。しかし、このような解決策は、ユーザにとっては不便なものであり、ファイル変換に大量のメモリと長い処理時間が必要になることがある。さらに、このような解決策では、ユーザは、旧バージョンでファイルが保存されたときに新しいバージョンのプログラムにより必要な重要情報が失われる可能性がある。

【0010】

したがって、旧バージョン、アクティブバージョン、および新しいバージョンのアプリケーションプログラムと互換性のあるファイル形式を用意することができる改良されシステムおよび方法が当技術分野では必要である。また、旧バージョンとの互換性を保持しながら将来のバージョンのアプリケーションプログラムに新しい機能を追加できる拡張可能ファイル形式が当技術分野では必要である。

【課題を解決するための手段】

【0011】

本発明は、一般に、アプリケーションプログラムの旧バージョン、アクティブバージョン、および将来のバージョンとの互換性をとることができる拡張可能ファイル形式を規定するシステムおよび方法に関する。本発明は、旧および／またはアクティブバージョンと

の互換性を維持しながら（修正せず）新しい機能を将来のバージョンのアプリケーションプログラムに追加できるようにする拡張可能ファイル形式を対象とする。したがって、本発明を採用したソフトウェアベンダは将来のバージョンに追加できる機能を制限することなく同じファイルとの複数バージョン互換性を実現することができる。

【0012】

本発明の一態様例では、ファイルバージョン透かし（watermark）をアプリケーションプログラムのオリジナルデータファイルに入れることができる。ファイルバージョン透かしは、オリジナルデータファイルのさまざまなプロパティを示すオリジナルデータファイル内の要素とすることができる。例えば、オリジナルデータファイルに上位バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最高位バージョンであることを示すことができる。オリジナルデータファイルに最終バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最終バージョンであることを示すことができる。オリジナルデータファイルに低位バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最低位バージョンであることを示すことができる。オリジナルデータファイルに作成バージョン透かしを入れて、最初にファイルを作成したアプリケーションプログラムのバージョンであることを示すことができる。オリジナルデータファイルにオブジェクトバージョン透かしを入れて、特定のオブジェクトを書き込むアプリケーションプログラムの最高位バージョンであることを示すことができる。各ファイルバージョン透かしは、ファイルに加えられた変更、追加、または削除を示すことができる。

【0013】

本発明の他の態様例では、1つまたは複数のファイルバージョン透かしで与えられる情報に基づきオリジナルデータファイルをロードまたは保存する方法を決定することを含むことができる。本発明の一態様例では、ファイル内の上位バージョン透かしがファイルをロードまたは保存するアプリケーションプログラムのアクティブバージョンよりも新しいバージョンを表すかどうかを決定することができる。上位バージョン透かしは、オリジナルデータファイルがアクティブバージョンに関してアプリケーションプログラムの、より新しいバージョンにより保存された場合に、より新しいバージョンを表す。その場合、本発明では、オリジナルデータファイルに、アクティブバージョンにとっては不明な情報が含まれているかどうかを判別することができる。不明な情報は、アプリケーションプログラムのアクティブバージョンにより無視され、より新しい（より後のほうの）バージョン用に保持されるようにできる。

【0014】

本発明の他の態様例では、ファイル内の上位バージョン透かしがファイルをロードまたは保存するアプリケーションプログラムのアクティブバージョンよりも古いバージョンを表すかどうかを決定することができる。上位バージョン透かしは、オリジナルデータファイルがアクティブバージョンに関してアプリケーションプログラムの、より古いバージョンにより保存された場合に、より古いバージョンを表す。その場合、本発明では、アクティブバージョンに、オリジナルデータファイル内の古い情報に基づく新しい情報が含まれるかどうかを判別することができる。新しい情報は、古い情報に基づいて更新することができる。さらに、新しい情報は、より古いバージョンの形式でデータを格納するだけでなく、アクティブバージョンおよびより新しいバージョンの形式でもデータを格納することにより、より古い（以前の）バージョン、アクティブバージョン、およびより新しい（後のほうの）バージョンで使用できるように保存することができる。

【0015】

本発明の他の態様例によれば、アプリケーションプログラムのメモリ内構造は、データファイル形式から分離することができる。したがって、アプリケーションプログラムの将来のバージョンは、データファイル形式を変更することなく多くの新しい機能を取り込むことができる。例えば、データファイル形式が異なることから、異なるバージョンとの互換性を失うことなく特定のプロセッサタイプについては効率が高くなるように構造体をメ

10

20

30

40

50

メモリ内でいかようにも移動することができる。

【0016】

本発明はさらに、アプリケーションプログラムの異なるバージョンに対応するデータファイルのロードおよび保存に必要なメモリ容量と処理能力を最小限に抑えることができる。ファイルバージョン透かしにより、異なるバージョンに対応するデータファイル内の特定の情報を識別することができる。したがって、処理または変換は識別された情報に対してしか実行できない。

【0017】

本発明の他の態様例では、オブジェクトプロパティリスト (object property list) (「OPL」) またはオブジェクトプロパティリスト配列 (「OPL配列」) をデータファイル構造体として組み込む。OPLまたはOPL配列を使用すると、未知のデータを保存済みデータファイルに伝搬させるか、または返すことにより将来のバージョンから未知のプロパティデータの「ラウンドトリッピング (round-tripping)」を行うことができる。OPLまたはOPL配列は、メモリにロードすることができる。特定のバージョンのアプリケーションプログラムにより認識されているプロパティすべてを、必要に応じて上書きすることができる。残りのプロパティ (つまり、特定バージョンには認識されないプロパティ) は、より新しいバージョンによって作成されており、ファイル内にそのまま残すことができる。データファイルに保存されるメモリ構造毎に、その構造を保存するため関連するOPLまたはOPL配列を用意することができる。

【0018】

本発明のこれらおよびその他の態様、目的、および特徴は、添付図面に関して、また添付図面を参照して、好ましい実施形態の詳細な説明を読むことにより明白になるであろう。

【発明を実施するための最良の形態】

【0019】

本発明については、付属の図面を参照しながら詳細に説明するが、ただし類似の参照番号は類似の要素を表す。

【0020】

本発明は、アプリケーションプログラムの旧バージョン、アクティブバージョン、および将来のバージョンと互換性のある拡張可能ファイル形式を規定することができる。本発明では、ファイルバージョン透かしをアプリケーションプログラムのオリジナルデータファイルに入れることができる。ファイルバージョン透かしは、オリジナルデータファイルのさまざまなプロパティを示すことができる。ファイルバージョン透かしには、上位バージョン透かし、最終バージョン透かし、低位バージョン透かし、作成バージョン透かし、およびオブジェクトバージョン透かしがある。各ファイルバージョン透かしは、ファイルに加えられた変更、追加、または削除を示すことができる。各ファイルバージョン透かしを、ファイル内の特定のオブジェクトなど、ファイル内の特定の情報について用意し、特定の情報を修正したバージョンを示すことができる。本発明では、ファイルバージョン透かしを使用して、ファイル、またはファイル内の特定の情報がアプリケーションプログラムの旧バージョンに対応するのか、アクティブバージョンに対応するのか、それとも将来のバージョンに対応するのかを判別することができる。したがって、本発明は、判別の実行結果に基づいてオリジナルデータファイルのロードまたは保存を行うことができる。

【0021】

図2は、本発明が動作するように設計されているコンピューティング環境例のさまざまな態様を示している。当業者であれば、図2および関連する説明は、好ましいコンピュータハードウェアおよびプログラムモジュールの簡単な概要説明を行うことを目的としており、また適切なプログラミングマニュアル、ユーザズガイド、および類似の刊行物で補足情報を容易に得られることを理解するであろう。

【0022】

10

20

30

40

50

図2は、本発明の好ましい実施形態の動作をサポートするのに好適な従来のパーソナルコンピュータ10を示す。図2に示されているように、パーソナルコンピュータ10は、リモートサーバ11に論理的に接続しているネットワーク接続環境において動作する。パーソナルコンピュータ10とリモートサーバ11との間の論理接続は、ローカルエリアネットワーク12およびワイドエリアネットワーク13により表されている。当業者であれば、このクライアント／サーバ構成で、リモートサーバ11はファイルサーバまたはコンピュータサーバとして機能できることを理解するであろう。

【0023】

パーソナルコンピュータ10は、Intel Corporation（カリフォルニア州サンタクララ）製の「Pentium（登録商標）」マイクロプロセッサなどの処理ユニット14を備える。パーソナルコンピュータ10は、さらに、読み取り専用メモリ（ROM）16およびランダムアクセスメモリ（RAM）17など、システムバス18によってプロセッサ14に接続されているシステムメモリ15も備える。好ましいコンピュータ10ではROM16に格納されているBIOS19を使用する。当業者であれば、BIOS19はパーソナルコンピュータ10内の要素間の情報伝達を補助する基本ルーチンの集まりであることを理解しているであろう。また当業者であれば、本発明は、BIOSを使用しないコンピュータおよび他のマイクロプロセッサを利用しているコンピュータなど、他のアーキテクチャを備えるコンピュータ上に実装できることも理解するであろう。

【0024】

パーソナルコンピュータ10内には、ローカルハードディスクドライブ20が装備され、ハードディスクドライブインターフェース21を介してシステムバス18に接続されている。フロッピー（登録商標）ディスク23を読み書きするために使用されるフロッピー（登録商標）ディスクドライブ22は、フロッピー（登録商標）ディスクドライブインターフェース24を介してシステムバス18に接続される。CD-ROMまたはDVDディスク26を読み取るために使用されるCD-ROMまたはDVDドライブ25は、CD-ROMまたはDVDインターフェース27を介してシステムバス18に接続される。ユーザは、シリアルポートインターフェース30を介してシステムバス18に接続されている、キーボード28および／またはマウス29などのポインティングデバイスなどの入力デバイスを使用して、コマンドおよび情報をパーソナルコンピュータ10に入力する。他の種類のポインティングデバイス（図2には示されていない）としては、トラックパッド、トラックボール、ペン、ヘッドトラッカ、データグローブ、およびコンピュータのモニタ31上でカーソルの位置を決めるのに好適なその他のデバイスがある。モニタ31またはその他の種類の表示デバイスも、ビデオアダプタ32を介してバス18に接続される。

【0025】

このネットワーク接続環境内のリモートサーバ11は、リモートメモリ記憶デバイス33に接続される。リモートメモリ記憶デバイス33は、通常、ハードディスクドライブ、CD-ROM、またはDVDドライブ、光磁気ドライブなどの大容量デバイスである。当業者であれば、アプリケーションプログラムモジュール37Cおよび37Dなどのプログラムモジュールはコンピュータ可読媒体を介してリモートサーバ11に供給されることを理解するであろう。パーソナルコンピュータ10は、ネットワークインターフェース34によりリモートサーバ11に接続されており、このインターフェースを介して、ローカルエリアネットワーク12上で通信を行う。

【0026】

他の実施形態では、パーソナルコンピュータ10は、さらに、モデム35によりリモートサーバ11にも接続され、このモデムを使用して、インターネットなどのワイドエリアネットワーク13上で通信を行う。モデム35は、シリアルポートインターフェース30を介してシステムバス18に接続される。また、モデム35は、公衆交換電話網（PSTN）または共聴アンテナ式テレビ（CATV）ネットワークに接続することもできる。図2にはパーソナルコンピュータ10の外部にあるように示されているが、当業者であれば、モデム35は、パーソナルコンピュータ10に内蔵することもでき、そうすることで、

システムバス 18 を介して直接通信することを理解することができる。ローカルエリアネットワーク 12 とワイドエリアネットワーク 13 との両方を介したりリモートサーバ 11 との接続は必要なことではなく、パーソナルコンピュータ 10 とリモートサーバ 11 との間の通信経路を提供する他の方法を単に示しているだけであることに注意されたい。

【0027】

パーソナルコンピュータ 10 の他の内部コンポーネントは図に示されていないが、当業者であれば、そのようなコンポーネントおよびコンポーネント間の相互接続は周知であることを理解するであろう。したがって、本発明に関しては、パーソナルコンピュータ 10 の内部構造に関するさらなる詳細を開示する必要はない。

【0028】

当業者であれば、オペレーティングシステム 36、アプリケーションプログラムモジュール 37A、ブラウザプログラムモジュール 37B、その他のプログラムモジュール 37N などのプログラムモジュールおよびデータは、コンピュータ可読媒体を介してパーソナルコンピュータ 10 に供給されることを理解するであろう。好ましいコンピュータ 10 では、コンピュータ可読媒体として、ローカルまたはリモートメモリ記憶デバイスがあり、ローカルハードディスクドライブ 20、フロッピー（登録商標）ディスク 23、CD-ROM または DVD 26、RAM 17、ROM 16、およびリモートメモリ記憶デバイス 33 などがある。好ましいパーソナルコンピュータ 10 では、ローカルハードディスクドライブ 20 を使用して、データおよびプログラムを格納する。

【0029】

プログラムモジュール 37N は、本発明の実施例による拡張可能ファイル形式のファイルを利用できるアプリケーションプログラムを含むことができる。第 1 のアプリケーションプログラムは、さらに拡張可能ファイル形式のオブジェクトプロパティリスト (OPL) を入れることができるオブジェクトを管理することができる。OPL は、システムメモリ 16 の連続するブロック内に格納することができる。第 2 のアプリケーションプログラムは、オブジェクトの OPL にアクセスすることができ、また後述のように、OPL を使用してさまざまな機能を実行することができる。

【0030】

OPL は、データファイルにデータを保存するためのデータ構造体である。OPL についてはさらに詳しく説明されている（例えば、特許文献 1 および 2 参照）。これらの特許の開示は、参照により本明細書に組み込まれている。

【0031】

従来の OPL では、データファイル用にデータ構造体を用意する。そのデータ構造体を使用することで、アプリケーションプログラムの旧バージョンおよび将来のバージョンとの互換性を実現するために使用される共通形式で情報を格納することができる。OPL の形式を使用すると、アプリケーションプログラムで認識されるデータを使用しながら、アプリケーションプログラムで認識されないデータをスキップすることができる。従来の OPL は、新しいオブジェクトおよびプロパティ用に動的に拡張できるため、アプリケーションプログラムの新しいバージョンに新しい情報が加わるとともに拡大することが可能である。

【0032】

図 3 では、従来の OPL 300 の特性を説明する。（OPL の詳細については、（特許文献 1）および（特許文献 2）を参照のこと。）

【0033】

本明細書の開示で使用している用語「オブジェクト」は、アプリケーションプログラムにより表示されるか、またはアプリケーションプログラムの一部である特性を持つエンティティを指す。アプリケーションプログラムにより表示されるオブジェクトの一例は、ユーザがテキストを入力できるアプリケーションプログラムのウィンドウに含まれるテキストボックスである。テキストボックスの特性には、その色、テキストのフォント、およびテキストのポイントサイズがある。アプリケーションプログラムの一部であるオブジェク

10

20

30

40

50

トの一例として、動物のメモリ内表現があり、色、脚の数、および肉食かどうかなどの特性が与えられる。このメモリ内表現はデータ構造体として実装することができ、データ構造の要素に特性を格納する。このようなデータ構造体の一例に、C++クラスデータ構造体がある。オブジェクトの特性を、そのオブジェクトのプロパティと呼ぶ。オブジェクトの各プロパティは、通常、値を持つ。例えば、色のプロパティは赤という値を持つことができる。

【0034】

「背景技術」で説明したように、オブジェクト100の例が図1に示されている。オブジェクト100は、枠線102とテキスト104とからなる。オブジェクト100は、枠線102の6つのプロパティとテキスト104とを備えることができる。枠線102は、枠線スタイルと枠線サイズとのプロパティを持つことができる。テキスト104は、フォント、テキストサイズ、位置揃え、およびテキストスタイルのプロパティを持つことができる。枠線102とテキスト104とのデフォルト値では、枠線スタイルを実線、枠線サイズを4ポイント、テキストフォントをTimes New Roman、テキストサイズを20ポイント、テキスト位置揃えを左寄せ、テキストスタイルを非イタリック体とすることができる。

【0035】

OPL 300では標準形式を使用しているため、オブジェクトのプロパティを内部的に格納するために使用できる。例えば、OPL 300は、オブジェクトプロパティ識別（「opyid」）要素302、オブジェクトプロパティ型要素304、およびオブジェクトプロパティ値要素306を備えることができる。Opyid要素302は、特定のプロパティに関連付けられている数値識別子を含むことができる。システムは、opyidへのすべてのプロパティのマッピングを保持するが、このマッピングはOPL毎に異なってもよい。つまり、各OPLで、opyid値が表すプロパティを定義することができる。他のOPLで、異なるプロパティに同じopyid値を使用することができる。オブジェクトプロパティ型要素304は、短整数（2バイト）、長整数（4バイト）、Unicode文字列などを参照する。オブジェクトプロパティ値要素306は、関連するオブジェクトプロパティ型要素304の適切なデータ型に対応する。例えば、図3に示されているように、opyid「1」は、システムが「色」に対応していると認識する短整数（2バイト）のオブジェクトプロパティ型要素304および、「赤」を示すRGB（赤、緑、青）の符号化である0x00FF0000のオブジェクトプロパティ値要素306を表す。特定のプロパティを参照するときに、opyidを使用する。例えば、OPL 300の枠線プロパティが必要な場合、これは、opyid「2」により参照される。OPLはさらに、他のOPLをプロパティとして含むことができるため、さらに複雑なデータ構造体を作成することができる。

【0036】

16ビットの従来のOPLでは、通常、opyid要素302は11ビットである。このような構成では、11ビットという制約があるためOPL 300は2048項目に制限される。パブリッシングシステムでは、オブジェクトおよびプロパティの数は簡単に2048項目を超えてしまう。パブリッシングドキュメントは複数のページで構成でき、それぞれ多数のオブジェクトを含む。各オブジェクトは、それに関連付けられた多数のプロパティを有することができる。したがって、いくつかの場合にOPL容量を最大にすることが望ましい。このために、本発明では、改良されたOPL型、OPL配列を備え、ほとんど無制限の容量を使えるようにできる。OPL配列には、ドキュメント内のすべてのオブジェクトなど大量のデータを格納することができる。OPLはそれでも、少ない量のプロパティを格納するのに使用することができる。

【0037】

図4では、本発明の一実施形態によるデータ構造体400について説明する。データ構造体400は、ルートOPL 402を含むことができる。ルートOPL 402は、「最大」プロパティ402a、不変プロパティ402b、およびOPL配列プロパティ402c

10

20

30

40

50

を備えることができる。必要というわけではないが、最大プロパティ402aはOPL配列プロパティ402cのサイズを示し、データ構造体400を読み込む際に十分なメモリを簡単に割り当てられる。不変プロパティ402bは、OPL配列プロパティ402c内の項目の配列インデックスが一定となる特徴を定義することができる。つまり、OPL配列プロパティ402c内の項目の配列インデックスは、その項目の存続期間全体を通して変化することがない。したがって、現在使用されていないインデックスに新しい項目を追加できるだけであり、項目を削除するとインデックスロケーションが空になる。

【0038】

OPL配列プロパティ402cは、OPL配列404を参照することができる。OPL配列404は、OPL配列プロパティ402cに格納するか、または別個に格納することができる。OPL配列404は、プロパティ、他のOPL、または他のOPL配列を取得することができる。図4に示されているように、OPL配列404は、サブオブジェクトプロパティリスト(「subopl」)404a~404eを含み、各subopl404a~404eはプロパティ、OPL、または他のOPL配列とすることができる。各subopl404a~404eは、配列要素406を参照することができる。各配列要素406は、プロパティ、OPL、または他のOPL配列とすることができる。例では、配列要素406は、IDおよび変数長文字列を含むOPLである。

【0039】

図4には、関連するOPL配列要素406とともにOPL配列404が1つだけ示されているが、本発明はこのような構造体に限られない。例えば、データ構造体400は、OPL配列404のような複数のOPLを含み、それぞれOPL配列要素406が関連付けられている。その場合、インデックス402cは各OPL配列404を参照することができる。

【0040】

subopls404a~404eはそれぞれ、従来のOPLのopyid302(図3)と同様に、プロパティ、オブジェクト、またはその他のOPLを参照する。しかし、データ構造体400では、OPL配列404内の項目に対しては特定のopyidを与えられない。その代わりに、OPL配列404内の各subopl404a~404eのopyidは「-1」に設定することができる。その後、OPL配列404内の各subopl404a~404eの位置をopyidとして使用することができる。各subopl404a~404eの位置は、図4でそれぞれのインデックス位置指標405a~405eにより表されている。例えば、OPL配列404内の位置「2」にあるものなどの特定のsuboplが必要な場合、インデックス位置指標の項目に対応するsubopl404cが用意される。subopl404cは、OPL配列404内のインデックス位置2に配置される。したがって、データ構造体400は、OPL配列404内のそれぞれのインデックス位置に基づいて特定のOPLを参照する。特定のsuboplのインデックス位置がそのopyidである。データ構造体400を使用することで、各要素の特定のopyidを格納する必要がなくなり、それにより、従来のOPLの欠陥に対処することができる。通常の32ビットシステムでは、データ構造体400は、40億を超える項目を格納することができるが、それは、項目の個数はもはやopyidの範囲による制限を受けることがないからである(位置指標405a~405eは、説明を目的としており、通常、OPL配列404で与えられる)。

【0041】

図4では、さらに、後で参照するためにOPL配列404内の空のスロットをどのように保存すればよいかについて説明されている。インデックス位置405bの空のスロット404bについては、プレースホルダプロパティ「oplempty」で、インデックス位置405bにあるそれぞれの項目が空であることを示すことができる。oplemptyは、単一配列要素が空であり、スキップしなければならないことを示すことができる。複数の隣接する空の配列要素について、プレースホルダプロパティ「oplskip=n」を使用できる。oplskip=nとすると、次の(n)個のエントリが空であること

10

20

30

40

50

を指定できる。例えば、図4に示されているように、インデックス位置405d、つまりOPL配列404内のインデックス位置「3～5」に対応する項目404dはプレースホルダプロパティ「oplskip=3」により埋められ、OPL配列404内の次の3項目が空であり、スキップしなければならないことを示す。

【0042】

データファイルは、オブジェクトの集合として特徴付けることができる。各オブジェクトは、OPLにより定義することができる。各OPLには、オブジェクトハンドル(OH)を関連付けることができる。OHは、オブジェクトOPLへのインデックスとすることができる。したがって、OHによりオブジェクトを識別することができ、OHはオブジェクトの存続期間中変化しない。(図4の不変フィールド402bを参照のこと)各オブジェクトに特有の情報を使用しなくても、各オブジェクトのOHを参照することによりファイルの構造をトラバースすることができる。OHは、ファイルを保存しても変化しないため(不変)、OHを使用してメモリおよび/またはファイル内のオブジェクトを参照することができる。

10

【0043】

通常、OHは、単一オブジェクトに割り当てなければならず、アプリケーションプログラムの後のバージョンでファイルを保存した場合、再利用すべきでない。例えば、追跡テーブルをアプリケーションプログラムのアクティブバージョンに導入し、プログラムによって作成されたすべてのオブジェクトを追跡することも可能である。旧バージョンでファイルをロードし、アクティブバージョンのオブジェクトの1つを削除した場合、アクティブバージョンの追跡テーブルは変更を反映するために更新されない。さらに、削除されたオブジェクトのOHを旧バージョンで作成した異なるオブジェクトについて再利用した場合、アクティブバージョンの追跡テーブルはその再利用されるOHを異なるオブジェクトとして解釈する。本発明では、後のバージョンでファイルを保存したかどうかを判別することにより、OHの再利用を回避し、そうすることで、問題が生じないようにする。次に、後のバージョンでファイルをロードする場合、後のバージョンは削除されているすべてのオブジェクトを見ることができる。それに応じて追跡テーブルを調整することができる。さらに、本発明では、(バージョン透かしにより判別された)そのファイルを使用したことのある最高位バージョンOHを再利用できる。本発明では、最高位バージョンを使用して、使用可能OHを判別することができ、また旧バージョンによりすでに削除されているオブジェクトへのすべての古い参照を削除することにより再利用することができる。

20

30

【0044】

メモリ構造をOPLまたはOPL配列内に格納するために、「OPL辞書」を定義し、各プロパティのデフォルト値および型を示すようにできる。未使用プロパティは領域をいっさい占有しないため、辞書の数を最小限に抑えることができる。したがって、本発明の一実施形態では、単一の辞書を使用することができる。このような実施形態では、すべての共通オブジェクトに、それぞれの同一項目に対する同じopyidまたは配列インデックス位置を持たせることができる。他の実施形態では、(1)すべてのファイル構造体関連プロパティ(つまり、OPLツリーのルート)を収めることができるファイル構造体辞書、(2)パブリッシングドキュメントのすべてのページオブジェクトのプロパティを収めることができるページオブジェクト辞書、および(3)関連するテキストとともにオブジェクトが含まれるテキストを記述するための辞書、または出版などで使用される色を定義するための色記述辞書を収めることができるその他の構造体辞書を用意することができる。プロパティが辞書の中に定義されている場合、これらのプロパティは、どのようなオブジェクトからも参照できる。したがって、共通オブジェクトによって使用される共通プロパティは、同じ辞書が出所なので同じである。

40

【0045】

OPL(およびOPL配列)は、将来のバージョンからの未知のプロパティまたは情報を保存する、つまり「ラウンドトリッピング」する場合に役立つ。特定のバージョンにより、オリジナルのOPLをディスクからメモリにロードをすることができる。特定のバー

50

ジョンにより認識されるO P Lのすべてのプロパティを上書きすることができる。残りのプロパティ（つまり、特定バージョンには認識されない未知のプロパティ）は、将来のバージョンによって作成されており、ファイル内にそのまま残すことができる。したがって、未知のプロパティは、後のバージョンでできるように保存済みファイルに伝搬により戻すことができる。未知のプロパティは、ロードするときに無視し、保存するときに簡単に保持できる。

【0046】

O P LまたはO P L配列をメモリ構造体として使用すると、メモリ内構造をファイル形式から分離することができる。したがって、アプリケーションプログラムの将来のバージョンは、異なるファイル形式に関連する問題を引き起こすことなく多くの新しい機能を取り込むことができる。例えば、ファイル形式を変更せずに特定のプロセッサタイプについては効率が高くなるように、構造体をメモリ内であちこち移動することができる。ファイルに保存されるメモリ構造体毎に、その構造体を保存するため関連するO P Lを用意することができる。O P Lから構造体へのマッピングは、1対1である必要はない。ディスク上の論理オブジェクトは、パフォーマンス上の理由から別のメモリ構造体とすることができる。例えば、オブジェクト上のハイパーリンクプロパティは、逆引きテーブルに格納し、これを使用して、与えられたオブジェクトにリンクされている他のオブジェクトを調べることも可能である。あるいは、複数のディスク上オブジェクトを組み合わせることで単一の構造体をメモリ内に作成することもできる。

【0047】

そこで図5および6を参照すると、本発明による方法は、アクティブバージョン、旧バージョン、および後のバージョンのアプリケーションプログラム間の互換性を実現するものであり、以下ではこれについて説明する。アクティブバージョンのアプリケーションプログラムは、コンピュータシステム200（図2）上で現在動作しているアプリケーションプログラムである。旧バージョンのアプリケーションプログラムは、アクティブバージョンの前に作成されたバージョンのアプリケーションプログラムである。後のバージョンのアプリケーションプログラムは、アクティブバージョンの後に作成されたバージョンのアプリケーションプログラムである。通常、アプリケーションプログラムのバージョンは、番号で示される。例えば、アプリケーションプログラムの第1のバージョンはバージョン1.0と表すことができる。アプリケーションプログラムの後のバージョンは、それよりも高い番号で示すことができる。例えば、アプリケーションプログラムの第2のバージョンはバージョン2.0と表すことができる。バージョン1.0は、バージョン2.0に関して旧バージョンである。バージョン2.0は、バージョン1.0に関して後のバージョンである。

【0048】

本発明は、データファイルの構造体として上で説明したO P L配列および従来のO P Lとともに使用することができる。しかし、当業者であれば、本発明はこれらのファイル構造体に限られないことを理解するであろう。

【0049】

本発明の一実施形態によれば、ファイルバージョン透かしは、オリジナルデータファイルのさまざまなプロパティを示すアプリケーションプログラムのオリジナルデータファイルで与えることができる。ファイルバージョン透かしはファイルヘッダに入れることができ、これによりファイルに関する情報を提供する。上位バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最高位バージョンであることを示すことができる。最終バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最終バージョンであることを示すことができる。低位バージョン透かしを入れて、ファイルを保存するために使用されるアプリケーションプログラムの最低位バージョンであることを示すことができる。作成バージョン透かしを入れて、ファイルを最初に作成したアプリケーションプログラムのバージョンを示すことができる。オブジェクトバージョン透かしをプロパティとして各オブジェクトに入れ、特

定のオブジェクトを書き込むアプリケーションプログラムの最高位バージョンであることを示すことができる。各ファイルバージョン透かしは、ファイルに加えられた変更、追加、または削除を示すことができる。本発明は、ファイルをロードをするときに、低位透かしバージョン以降に追加または修正されたデータ構造体およびプロパティを判別することができる。その後、どのようなバージョンとも互換性を持つように、追加または修正された項目を訂正することができる。

【0050】

動作時に、データファイル内の上位バージョン透かしをアクティブバージョンのアプリケーションプログラムと比較することができる。このような比較により、オリジナルデータファイルがアクティブバージョンに対応しているか、旧バージョンに対応しているか、または後のバージョンのアプリケーションプログラムに対応しているかを判別することができる。アプリケーションプログラムによりファイル全体を保存できるので、ファイル内のオブジェクトは、上位バージョン透かしによって識別されたバージョンで作成されたファイルの特性を含む。アプリケーションプログラムでは、アクティブバージョンに対応しているか、旧バージョンに対応しているか、または後のバージョンに対応しているかに基づいてオリジナルデータファイルをロードし、かつ／または保存する方法を決定できる。オリジナルデータファイルは、オブジェクトプロパティリスト、オブジェクトプロパティ配列、またはその他のファイル構造体を含むことができる。

【0051】

例えば、バージョン2.0（後のバージョン）のアプリケーションプログラムは、バージョン2.0の情報の取り扱いがバージョン1.0（旧バージョン）と比べてどれだけ異なっているかに基づいて情報を変換するかどうかを決定することができる。例えば、バージョン1.0にハイパーリンク用の追跡テーブルがあるが、バージョン2.0では、ハイパーリンクデータをオブジェクト自身に配置し、バージョン1.0によって書き込まれたファイルをロードする際にそのような情報を変換すべきであると判断することができる。しかし、バージョン2.0がファイルを書き込む最古バージョンであった場合、バージョン2.0では、そのような変換が必要でないと判定することができる。

【0052】

低位および上位バージョン透かし（およびオブジェクト透かし）は、前のマークを超えた場合に更新できる。ファイルを保存する最終バージョンは、保存時に更新できる。作成バージョン透かしは、ファイルが最初に作成されても、通常は更新されないときに入れることができる。

【0053】

そこで図5を参照し、本発明の一実施形態によるデータファイルをロードする方法500について説明する。図5は、オリジナルデータファイルとアクティブバージョン、旧バージョン、および後のバージョンとの互換性が維持されるように、アプリケーションプログラムにオリジナルデータファイルをロードする方法500を説明する流れ図である。方法500は、オリジナルデータファイルに上位バージョン透かし、低位バージョン透かし、最終バージョン透かし、作成バージョン透かし、および／またはオブジェクトバージョン透かしを入れるステップ510を含むことができる。ステップ520で、オリジナルデータファイル内の上位バージョン透かしを決定する。この方法ではさらに、ステップ530に示されているように、オリジナルデータファイルの中の低位バージョン透かし、最終バージョン透かし、作成バージョン透かし、および／またはオブジェクトのオブジェクトバージョン透かしを決定することもできる。ステップ540で、上位バージョン透かしをアプリケーションプログラムのアクティブバージョンと比較し、上位バージョンがアクティブバージョンよりも新しいかどうかを判別する。この方法は、その比較に基づいて、オリジナルデータファイルをロードする方法を決定する。

【0054】

ステップ540の比較によって上位バージョン透かしがアクティブバージョンよりも新しくないアプリケーションプログラムのバージョンを表していることが示されると、オリ

ジナルデータファイルはアクティブバージョンに関してアプリケーションプログラムの同じバージョンまたは旧バージョンに対応する。その場合、この方法はステップ560に分岐し、そこで変更することなくオリジナルデータファイルをアプリケーションプログラムにロードすることができる。上位バージョン透かしはアプリケーションプログラムの同じバージョンまたは旧バージョンに対応しているため、アクティブバージョンはオリジナルデータファイル内のすべての情報を読み取り、理解できる。

【0055】

ステップ540の比較によって上位バージョン透かしがアクティブバージョンよりも新しいアプリケーションプログラムのバージョンを表していることが示されると、オリジナルデータファイルはアクティブバージョンに関してアプリケーションプログラムの後のバージョンに対応する。その場合、この方法はステップ550に分岐し、オリジナルデータファイルをロードする。上位バージョン透かしはアプリケーションプログラムの後のバージョンに対応しているため、アクティブバージョンはオリジナルデータファイル内のすべての情報を読み取り、理解することができない場合がある。したがって、ステップ550で、知られているすべての情報をロードし未知のすべての情報をスキップすることで、オリジナルデータファイルがアプリケーションプログラムにロードされる。OPLまたはOPL配列をファイル構造体として使用する場合、オブジェクトに関連付けられているOPLの未知のプロパティを保持するために、未知のすべてのプロパティをロードし、ファイルが保存されるまでそれらのプロパティをメモリブロック内に格納する。あるいは、プログラム側で、ディスク上のオブジェクトに関連付けられているオリジナルのOPLの場所を追跡し、OPLの未知のプロパティを必要に応じてメモリ内に読み込んで戻してからファイルを上書きすることができる。

【0056】

ステップ550または560の後、この方法はステップ570に進み、そこでは、オリジナルのコンピュータファイル内の最終バージョン透かしがアクティブバージョンよりも古いアプリケーションプログラムのバージョンを表しているかどうかを判別する。ステップ570での判別が否定的であれば、この方法は終了する。ステップ570での判別が肯定的であれば、この方法はステップ580に分岐し、そこで、オリジナルデータファイル内の古い情報に基づき新規または修正済み情報を更新することによりオリジナルデータファイルを変換することができる。また古い情報は、変更を加えずに保持することもできる。

【0057】

ステップ570および580は、最終バージョン透かしに基づいてオリジナルデータファイルを変換する動作を行う。最終バージョン透かしは、上位バージョン透かしからの追加情報を示し、オリジナルデータファイルに対してどのような変更が必要かを判別することができる。例えば、バージョン2のコンピュータアプリケーションプログラム（上位バージョン透かし）は、オリジナルデータファイルを書き込んでいるものとする。バージョン2は、バージョン1からの新しいまたは修正済みプロパティを含んでいる。しかし、バージョン1のアプリケーションプログラムがファイルを書き込む最終バージョンであった場合（最終バージョン透かし）、バージョン2の新規または修正済みプロパティに対応するデータの一部が変更されている可能性がある。したがって、バージョン2または後のバージョン（アクティブバージョン）がオリジナルデータファイルを読み込む場合、そのバージョンは、ファイルを書き込む最終バージョンはバージョン1だったのでデータファイル内のデータを変換する必要があるかもしれない（バージョン1により書き込まれたデータは、バージョン2または後のバージョンに対応するように更新しなければならない場合がある）。

【0058】

ファイルバージョン透かしは、ファイル全体の代わりに、またはそれだけでなくファイル全体でも、オブジェクトに入れることができる。その後、方法500（および後述の方法600）をオリジナルデータファイル内のオブジェクトに適用することができる。個々

のオブジェクトについてファイルバージョン透かしを入れると、変換するか、または無視する必要があるファイル内のオブジェクトを判別することによりロードおよび保存の方法を最適化することができる。さらに、ファイルバージョン透かしは、ファイル全体またはオブジェクト全体ではなくプロパティ毎に、またはファイル全体またはオブジェクト全体とともにプロパティ毎に入れることができる。その後、方法500（および後述の方法600）をオリジナルデータファイル内のプロパティに適用することができる。個々のプロパティについてファイルバージョン透かしを入れると、変換するか、または無視する必要があるファイル内のプロパティを判別することによりロードおよび保存の方法を最適化することができる。

【0059】

さらに、アプリケーションプログラムのアクティブバージョンでは、アクティブバージョン透かしを、オブジェクトを改訂、変更、または削除したアプリケーションプログラムのバージョンを表すことができる各オブジェクトに関連付けることができる。方法500（および後述の方法600）では、特定のオブジェクトのオブジェクトバージョン透かしをアクティブバージョンと比較できるが、ただし、アクティブバージョンはアクティブバージョンのその特定のオブジェクトのアクティブバージョン透かしにより表される。したがって、本発明では、アクティブバージョン透かしがオブジェクトバージョン透かしよりも新しい場合のみオブジェクトを更新する必要があるため、オリジナルデータファイル内のオブジェクトを更新する必要があるかどうかの判別を最適化することができる。例えば、オリジナルデータファイル内の特定のオブジェクトのオブジェクトバージョン透かしは、バージョン1.0が特定のオブジェクトを保存する最終バージョンであったことを示すと仮定する。アクティブバージョン2.0がファイルをロードすると、バージョン2.0はバージョン1.0よりも新しいため、特定のオブジェクトに基づくアクティブバージョン内で情報を更新する。しかし、アクティブバージョンのその特定のオブジェクトについてアクティブバージョン透かしが入れられた場合、アクティブバージョン透かしは、バージョン1.0以降、特定のオブジェクトが変化しておらず、バージョン2.0で変更されていないためアクティブバージョンは特定のオブジェクトを更新しない。したがって、ロード方法は、不要な変換ステップを削除することにより最適化することができる。

【0060】

そこで図6を参照し、本発明によるデータファイルを保存する方法600の実施形態について説明する。方法600では、修正済みデータファイルを保存することで、アクティブバージョン、旧バージョン、および後のバージョンと互換性を持つようにできる。アクティブバージョンのアプリケーションプログラムによりオリジナルデータファイルが修正された後、その修正されたオリジナルデータファイルを、アクティブバージョン、旧バージョン、および後のバージョンと互換性を持つように保存することができる。図6は、方法600を説明する流れ図である。方法600は、オリジナルデータファイルに上位バージョン透かし、低位バージョン透かし、最終バージョン透かし、作成バージョン透かし、および／またはオブジェクトバージョン透かしを入れるステップ605を含むことができる。ステップ610で、オリジナルデータファイル内の上位バージョン透かしを決定する。方法600ではさらに、ステップ615に示されているように、オリジナルデータファイルの低位バージョン透かし、最終バージョン透かし、作成バージョン透かし、および／またはオブジェクトバージョン透かしを決定することもできる。ステップ620で、上位バージョン透かしをアクティブバージョンのアプリケーションプログラムと比較する。この方法は、その比較に基づいて、修正されたオリジナルデータファイルをどのように保存するかを決定する。

【0061】

ステップ620の比較によって上位バージョン透かしがアクティブバージョンよりも古いアプリケーションプログラムのバージョンを表していることが示されると、オリジナルデータファイルはアクティブバージョンに関してアプリケーションプログラムの旧バージョンに対応する。その場合、この方法はステップ625に分岐し、修正されたオリジナル

データファイルをアプリケーションプログラムのアクティブバージョンの形式で保存する。上位バージョン透かしはアプリケーションプログラムの旧バージョンに対応しているため、アクティブバージョンのアプリケーションプログラムはさらに、修正されたオリジナルデータファイルをすべての旧バージョンと互換性のある形式で保存することができる。したがって、方法はステップ630に進み、そこで、修正されたオリジナルデータファイル内の項目が旧バージョンの形式に変換され、保存される。すると、修正されたオリジナルデータファイルはアクティブバージョンと旧バージョンとの両方の形式で保存される。旧バージョンからの古い情報は、ステップ630で再構築され、アプリケーションプログラムの旧バージョン、アクティブバージョン、および将来のバージョンによってファイルが使用される。古い情報を再構築する作業が大きくなりすぎた場合、将来のバージョンのうち特定のバージョンでのみファイルを読み込めることを伝えるコマンドを挿入することにより、将来のファイルは切り詰められる。ステップ635で、オリジナルデータファイルの上位バージョン透かしは、アクティブバージョンのアプリケーションプログラムと対応するように更新される。次に、この方法はステップ665に進み、さらにファイルを保存する最終バージョンを将来の使用に備えて更新する。

10

【0062】

ステップ620の比較によって上位バージョン透かしがアクティブバージョンよりも新しいアプリケーションプログラムのバージョンを表していることが示されると、オリジナルデータファイルはアクティブバージョンに関してアプリケーションプログラムの後のバージョンに対応する。その場合、この方法はステップ640に分岐し、オリジナルデータファイルを取得する。ステップ645で、オリジナルデータファイルを修正済みオリジナルデータファイルで上書きする。上位バージョン透かしはアプリケーションプログラムの後のバージョンに対応しているため、修正されたオリジナルデータファイルには、アクティブバージョンには未知の情報またはデータを格納することができる（例えば、図5のステップ580を参照のこと）。したがって、ステップ645では、オリジナルデータファイルの知られているデータのみが修正されたオリジナルデータファイルのデータで上書きされる。未知のプロパティデータは変更されない。この方法により、未知のデータは、そのデータを読み込むことができるアプリケーションプログラムの後のバージョンでそのまま利用できる。未知のデータは、図5を参照して上述したようにアプリケーションプログラムにより保持することができる。その後、この方法はステップ630に進み、そこで、修正されたオリジナルデータファイル内の項目が旧バージョンの形式に変換され、保存される。すると、修正されたオリジナルデータファイルはアクティブバージョンと旧バージョンとの両方の形式で保存される。

20

30

【0063】

次に、この方法では、低位バージョン透かしを更新する必要があるかどうかを判定する。ステップ650で、アプリケーションプログラムのアクティブバージョンをオリジナルデータファイルの低位バージョン透かしと比較する。ステップ650で、アクティブバージョンが低位バージョン透かしにより表されるアプリケーションプログラムのバージョンよりも古いと判断された場合、方法はステップ655に分岐し、そこで、低位バージョン透かしが更新されアクティブバージョンに対応ようになる。次に、この方法はステップ665に進み、さらにファイルを保存する最終バージョンを将来の使用に備えて更新する。ステップ650で、アクティブバージョンが低位バージョン透かしにより表されるアプリケーションプログラムのバージョンと同じか、それよりも新しいと判断された場合、方法はステップ665に分岐し、そこで、最終バージョン透かしが更新されアクティブバージョンに対応ようになる。

40

【0064】

ステップ620の比較によって上位バージョン透かしがアクティブバージョンと同じアプリケーションプログラムのバージョンを表していることが示されると、オリジナルデータファイルはアプリケーションプログラムのアクティブバージョンに対応する。その場合、この方法はステップ660に分岐し、修正されたオリジナルデータファイルは、オリジ

50

ナルデータファイルを上書きすることにより保存される。その後、この方法はステップ 630 に進み、そこで、修正されたオリジナルデータファイル内の項目が旧バージョンの形式に変換され、保存される。そのため、修正されたオリジナルデータファイルはアクティブバージョンと旧バージョンとの両方の形式で保存される。次に、この方法はステップ 665 に進み、ファイルを保存する最終バージョンを将来の使用に備えて更新する。

【0065】

ステップ 630 はステップ 620 から辿る分岐に関係なく実行することができることに注意されたい。したがって、旧バージョンに対応するプロパティは、常に、ファイルが旧バージョンと互換性を持つようにファイルに書き込むことができる。

【0066】

いくつかの場合には、方法 500 および 600 は、ファイルを修正後に保存する OPL 配列要素の場所を追跡する追加ステップ（図に示されていない）を含むことができる。例えば、あるバージョンのデータファイルは、4つの要素からなるテーブルを格納することができる。しかし、データファイルの修正されたバージョンには、3つの要素しか入らない。アプリケーションプログラムがオリジナルデータファイルを上書きしようとする、ディスクバージョンには、オリジナルの4つの要素が入るが、メモリバージョンには、要素は3つしか入らない。追跡ステップがないと、アプリケーションプログラムでは、どの要素が削除されたかを知ることができない。追跡ステップでは、アプリケーションプログラムはすべてのテーブルのオリジナルインデックスを追跡する。ディスクからテーブルが読み込まれると、テーブル内の各要素の位置を示す付加データ項目が追加される。つまり、アプリケーションプログラムは、最初の要素、2番目の要素、というようにマークを付けてゆく。したがって、プログラムでは、存在していないため削除されてしまった要素を判別し、配列要素内に存在するかもしれない将来のバージョンプロパティを保持することができる。

【0067】

ステップ 500 および 600 では、アプリケーションプログラムのアクティブバージョンは、透かしのどれかを使用して、オリジナルデータファイルをロードするときに実行するアクションを決定することができる。上位バージョン透かしの使用については、すでに上述した。低位バージョン透かしは、アプリケーションプログラムに与えることができるヒントとなり、オリジナルデータファイルに対する必要な変換の種類を制限する。前の例を構築して、低位バージョン透かしがそのバージョンによりファイルが書き込まれたことのないことを示しているため、バージョン 1.0 のハイパーリンクテーブルが存在しないことが知られている場合、変換プロセスは、ハイパーリンクテーブルを扱うものをスキップしても安全である。最終バージョン透かしは、ファイルを書き込む最終バージョンによって認識されているデータだけでなく、未知ではあるがファイルで運ばれるデータをも示すことができる。

【0068】

オブジェクトバージョン透かしは、特定のオブジェクトに対する特定の情報を提供することができる。そのため、その情報に基づいて、特定オブジェクトのロード、保存、または変換を行うことができる。オブジェクトバージョン透かしは、特定のオブジェクトの情報を提供することができ、また特定のオブジェクトに含まれるプロパティの最高位ファイルバージョンを示すことができる。例えば、ハイパーリンクプロパティがバージョン 2.0 のオブジェクト X に追加され、バージョン 3.0 のオブジェクト X には新しいプロパティは追加されなかったと仮定する。オブジェクト X に、バージョン 2.0 または後のバージョンで書かれたハイパーリンクが含まれる場合、そのバージョン透かしはバージョン 2.0 に設定される。オブジェクト X がハイパーリンクを含まない（したがって、ハイパーリンクプロパティが書き込まれない）場合、またはバージョン 1.0 によって書かれている場合、バージョン透かしはバージョン 1.0 に設定される。この情報を使用して、各個別オブジェクトおよび、そのオブジェクト内のすべてのプロパティを認識する最高位バージョンに関してロードおよび保存オペレーションをさらに最適化することができる。例えば

、バージョン 3. 0 によって書かれたが、ハイパーリンクを含まない（したがって、バージョン 1. 0 オブジェクト透かしを持つ）オブジェクト X は、バージョン 1. 0 により読み書きすることができ、しかも、ファイルの上位バージョン透かしがバージョン 3. 0 を反映するとしても、将来のプロパティの存在に関して気にする必要がない。

【0069】

一般に、ファイルを保存する最新バージョンのアプリケーションプログラムは、そのバージョンまたは旧バージョンからの情報の取り扱い方法を決定するのに最も適している。最新バージョンでは、ファイル内のすべてを読むことができ、また認識することができる。アプリケーションプログラムの最新バージョンにより、保持する、変換する、または破棄する情報が決定される。したがって、ファイルを書き込む最高位バージョンにより、情報を破棄するのか、それとも変換するのかが決定される。より低いバージョンはすべて、単に、未知の情報を伝搬することができるだけである。

10

【0070】

結局、本発明を動作可能にするために図 5 および 6 に示されている方法 500 および 600 のすべてのステップが必要なわけではないということである。さらに、図に示されているのと違う順序でいくつかのステップを実行することができる。

【0071】

図 7～9 を参照して、本発明による方法 500 および 600 の具体的実施形態について説明する。図 7～9 のそれぞれで、方法 500 または 600 により、最終バージョン透かしがアプリケーションプログラムのアクティブバージョンよりも低い（つまり、オリジナルデータファイルを保存するアプリケーションプログラムの最終バージョンはアクティブバージョンに関してアプリケーションプログラムの旧バージョンに対応している）と判断されている（図 5 のステップ 595 を参照のこと）。さらに、図 7～9 を参照して以下で説明している実施形態は、方法 500 または 600 により、上位バージョン透かしがアプリケーションプログラムのアクティブバージョンよりも低いと判断されたときでも動作可能である。

20

【0072】

図 7 は、アクティブバージョンが旧バージョンに含まれない追加オブジェクトプロパティを持つ、本発明によりオリジナルデータファイルのロードおよび保存を行うための方法 700 を示す流れ図である。方法 700 のステップ 710 では、新しいプロパティのデフォルト値が、アプリケーションプログラムのアクティブバージョンに対する OPL で定義される。プロセスはステップ 725 に進み、そこで、オリジナルデータファイルが変更なしでロードされる。オリジナルデータファイルは、修正された後、ステップ 730 で、保存される。したがって、アプリケーションプログラムのアクティブバージョンおよび将来のバージョンで使用できるように、その新しいプロパティが保存される。旧バージョンは、単に新しいバージョンを無視するだけである。ステップ 740 で、上位バージョン透かしはアクティブバージョンに更新される。

30

【0073】

図 8 は、アクティブバージョンでは旧バージョンで使用されたオブジェクトプロパティを削除してしまっている、本発明によりオリジナルデータファイルのロードおよび保存を行う方法 800 を示す流れ図である。ステップ 820 で、削除されたプロパティは、オリジナルデータファイルがアクティブバージョンにロードされる際に無視される。ステップ 830 で、オリジナルデータファイルの修正バージョンが保存されるときに、ファイルは、古いバージョンから使用できる削除済みプロパティとともに保存される。アクティブバージョンは削除済みプロパティを使用していなかったが、削除済みプロパティは、旧バージョンで使用できるように保存ファイルに戻される。削除済みプロパティは、オリジナルデータファイル内に残すこと、つまり、削除済みプロパティをファイルから書き出さないことにより、保存ファイルに戻すことができる。あるいは、削除済みプロパティを（オリジナルデータファイルから）保存ファイルに伝搬させて戻すという方法で、保存ファイルに戻すことができる。削除済みプロパティはアクティブバージョンに知られているので、

40

50

削除済みプロパティの値を他のプロパティ値から導いたり、または削除済みプロパティを表すデフォルト値を生成し、削除済みプロパティをファイルに戻すようにすることができる。ステップ840で、上位バージョン透かしはアクティブバージョンに更新できる。

【0074】

例えば、旧バージョンは、オブジェクトについてドロップシャドウ(drop shadow)プロパティを含んでいたと仮定する。アクティブバージョンは、このプロパティをもはやサポートできない、つまり、ドロップシャドウプロパティは、アクティブバージョンから削除されてしまっている。アプリケーションプログラムのより新しいバージョンでは、ドロップシャドウプロパティが削除されたことを認識している。しかし、ドロップシャドウプロパティはそれでも、アクティブバージョンによりファイルに書き込むことができる。ステップ820で、オリジナルデータファイルをロードし、削除されたドロップシャドウプロパティを無視することができる。ステップ830で、オリジナルデータファイルの修正バージョンが保存されるときに、ファイルは、古いバージョンから使用できる削除済みドロップシャドウプロパティとともに保存される。アクティブバージョンは削除済みドロップシャドウプロパティを使用していなかったが、削除済みドロップシャドウプロパティは、旧バージョンで使用できるように保存ファイルに戻すことができる。ドロップシャドウプロパティ値は、他のプロパティの値から決定できるか、またはデフォルト値を使用することができる。ステップ840で、上位バージョン透かしはアクティブバージョンに更新できる。

【0075】

図9は、アクティブバージョンでは旧バージョンの古いオブジェクトプロパティを修正してしまっている、本発明によりオリジナルデータファイルのロードおよび保存を行う方法900を示す流れ図である。古いプロパティを修正されたプロパティで置き換えるのではなく、修正されたプロパティを表す新しいプロパティをアクティブバージョンに入れる。方法900のステップ910では、新しいプロパティのデフォルト値が、アプリケーションプログラムのアクティブバージョンのOPLで定義される。古いプロパティは、変更されずそのままである。次にステップ920で、オリジナルデータファイルがアクティブバージョンにロードされ、古いプロパティが新しい(修正された)プロパティに変換される。オリジナルデータファイルは、修正された後、ステップ930で、保存される。ステップ930で、古いプロパティを参照せずに新しいプロパティが保存される。さらに、ステップ930で、新しいプロパティが古いプロパティに変換され、旧バージョンでできるように保存される。ステップ940で、上位バージョン透かしはアクティブバージョンに更新される。

【0076】

例えば、アプリケーションプログラムの旧バージョンでRGB(赤、緑、青)色モデルを使用してオブジェクトの色を定義したとする。アプリケーションプログラムのアクティブバージョンは、新しいプロパティを備え、プロ用プリンタで使用されるCMYK色モデル(シアン、マゼンタ、黄、黒)でオブジェクトの色を定義することができる。新しいCMYK色プロパティは、古いRGB色プロパティに基づき、相当するCMYK値は、RGB値毎に決定することができる。新しいopyidを適切なOPL辞書に追加することができる。例えば、適切な辞書として、ページオブジェクト用のページオブジェクト辞書、他の既存の辞書、または新規辞書を設定できる。ステップ910で、新しいCMYK色プロパティのデフォルト値をOPL辞書で定義することができる。古いRGBプロパティは、変更されずそのままである。ステップ920で、オリジナルデータファイルをアクティブバージョンにロードし、オブジェクトの古いRGBプロパティに基づき新しいCMYK色プロパティを更新することができる。オリジナルデータファイルは、修正された後、ステップ930で、保存できる。ステップ930で、オリジナルデータファイルからの古いRGBプロパティを参照せずに新しいCMYK色プロパティを保存できる。新しいCMYKプロパティは、対応する古いRGBプロパティに変換し、ファイルに保存することができる。したがって、アプリケーションプログラムのアクティブバージョンおよび将来のバ

ージョンで使用できるように、新しいCMYK色プロパティを保存できる。旧バージョンは、単に新しいCMYK色プロパティを無視するだけである。ステップ940で、上位バージョン透かしはアクティブバージョンに更新できる。

【0077】

さらに、本発明の方法では、データファイルに保存されているアプリケーションプログラムの2つのバージョンの間の食い違いを解決することもできる。上述したCMYK-RGBの例では、後のバージョンは新しいCMYK色プロパティを対応するRGBプロパティに下方変換するため、旧バージョンはファイルを読み込むことができる。しかし、ファイルを書き込む最終バージョンがファイルを書き込むアクティブバージョンと上位バージョンの両方よりも古い場合に、食い違いが発生することがある。アクティブバージョンは、ファイルを書き込む上位バージョンと同じバージョンまたは後のバージョンであってよい。最終バージョンでない、ファイルを書き込む上位バージョンが新しいCMYK色プロパティを認識した場合、新しいCMYK色プロパティ値をデータファイルに書き出すだけでなく、対応するRGB色プロパティ値も書き出す。さらに、ファイルを書き出す最終バージョンがRGB色プロパティのみを認識した場合、RGB色プロパティ値をデータファイルに書き出す。データファイルが新しいCMYK色プロパティをサポートするアクティブバージョンにロードされると、データファイルのCMYK色プロパティは古い情報に対応する。上位バージョンはCMYK色プロパティ値をファイルに書き込み、最終バージョンはRGB色プロパティ値をファイルに書き込んだため、色プロパティ情報の2つのバージョンがデータファイル内に存在することになる。ファイルを書き込む最終バージョンはRGB値を更新したが、未知のCMYKプロパティの値を保持していたため、RGBおよびCMYK値は一致しない。ファイルがアクティブバージョンにロードされるときに、ファイルを書き込む最終バージョンを決定できる（図5のステップ570を参照のこと）。その情報に基づき、CMYK色プロパティ値が期限切れになっており、RGB色プロパティ値をロードし、更新されたCMYK色プロパティ値に変換しなければならないと判断することができる。

【0078】

オブジェクトはプロパティの集合である。したがって、アプリケーションプログラムの異なるバージョンにおけるオブジェクトの変更は、オブジェクトに対応するプロパティの変更で示すことができる。オブジェクトを含むデータファイルがロードまたは保存されるときに、方法700、800、および900のステップをオブジェクトの個々のプロパティについて反復することができる。

【0079】

本発明の一実施形態を、本願特許出願人（ワシントン州レッドモンド）が販売しているMicrosoft Publisherというドキュメントパブリッシングシステムの一部として使用することができる。このパブリッシングシステムを使用すると、ユーザは、ドキュメントを編集し、グラフィックスデータ、テキストデータ、スプレッドシートデータ、またはその他の種類のデータを含む各種オブジェクトを挿入することができる。さらに、このシステムでは、オブジェクトのプロパティをユーザ側で修正することができる。システムは、標準値（デフォルト値）に設定されている定義済みプロパティを持つ多数の定義済みオブジェクト型を用意できる。各オブジェクト型は、さまざまなプロパティおよび／またはプロパティ値の集まりを備えることができる。各オブジェクトは、そのオブジェクト型のインスタンスとすることができる。したがって、オリジナルの形式のオブジェクトの設定は容易に取得可能である。ドキュメントパブリッシングシステムを参照して実施形態を説明しているが、当業者であれば、本明細書で説明した手法を実質的に数限りない他の種類のシステムに応用できることを理解するであろう。

【0080】

本発明は、上述の処理機能を実行するコンピュータのハードウェアおよびソフトウェアとともに使用することができる。当業者であれば理解するであろうが、本明細書で説明しているシステム、方法、および手順は、プログラム可能コンピュータ、コンピュータ実行

可能ソフトウェア、またはデジタル回路により実現することができる。ソフトウェアは、コンピュータ可読媒体に格納することができる。例えば、コンピュータ可読媒体には、フロッピー（登録商標）ディスク、RAM、ROM、ハードディスク、取り外し可能媒体、フラッシュメモリ、メモリスティック、光媒体、光磁気媒体、CD-ROMなどがある。デジタル回路には、集積回路、ゲートアレイ、基本的論理回路、フィールドプログラマブルゲートアレイ（FPGA）などがある。

【0081】

上記では本発明の特定の実施形態について説明したが、この説明は単に例示することのみを目的としている。上述のものに加えて、複数の実施形態の開示されている態様のさまざまな修正、およびそれらの態様に対応する相当するステップは、当業者であれば、特許請求の範囲で定められている本発明の精神および範囲を逸脱することなく実施することができ、その範囲はそのような修正および同等の構造を含む最も広い解釈を受け入れるものである。

10

【図面の簡単な説明】

【0082】

【図1】アプリケーションプログラムによって使用されるオブジェクトを説明するブロック図である。

【図2】本発明の例示的な実施形態を実施するのに好適なコンピュータ例を説明するブロック図である。

【図3】従来のOPLの特徴を説明するブロック図である。

20

【図4】本発明の例示的な実施形態によるOPL配列の特徴を説明するブロック図である。

【図5】本発明の例示的な実施形態によりオリジナルデータファイルをロードする方法を示す流れ図である。

【図6】本発明の例示的な実施形態によりオリジナルデータファイルを保存する方法を示す流れ図である。

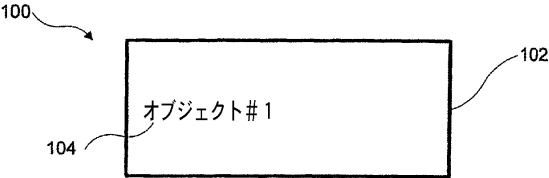
【図7】アクティブバージョンは旧バージョンに含まれない追加オブジェクトプロパティを持つ、本発明によりオリジナルデータファイルのロードおよび保存を行う方法を示す流れ図である。

【図8】アクティブバージョンでは旧バージョンで使用されたオブジェクトプロパティを削除してしまっている、本発明によりオリジナルデータファイルのロードおよび保存を行う方法を示す流れ図である。

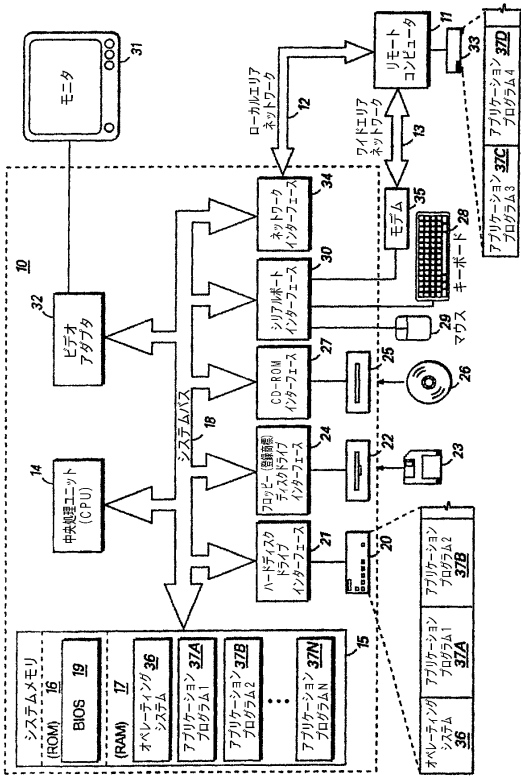
30

【図9】アクティブバージョンでは旧バージョンの古いオブジェクトプロパティを修正してしまっている、本発明によりオリジナルデータファイルのロードおよび保存を行う方法を示す流れ図である。

【図 1】



【図 2】



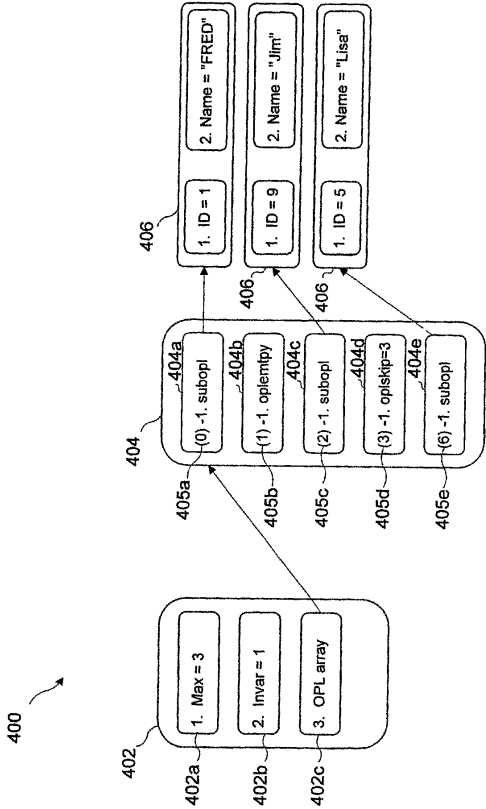
【図 3】

300

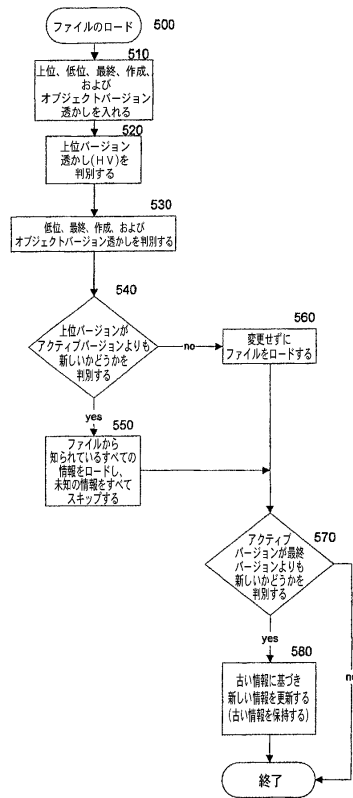
302	304	306
名前 11ビット	型 5ビット	値 可変
1	短整数 (2バイト) 色	0x00FF0000 赤
2	短整数 (2バイト) 枠線	0x00000001 実線

従来技術

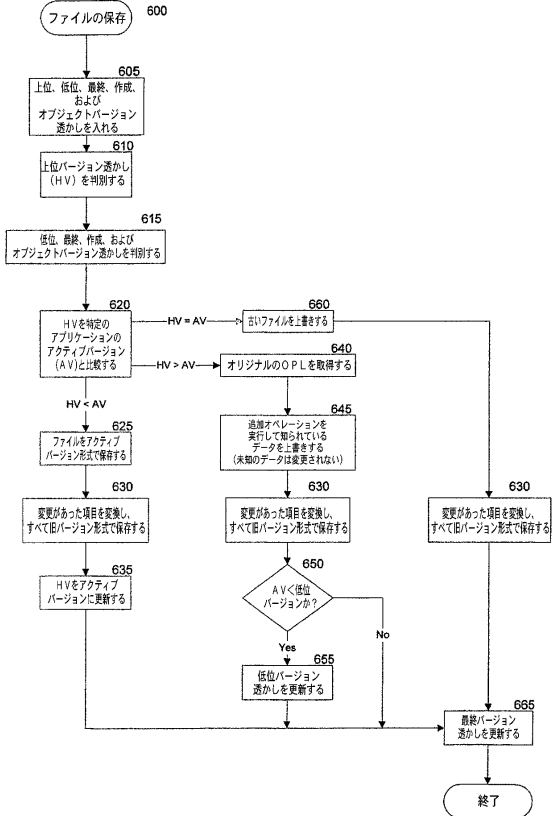
【図 4】



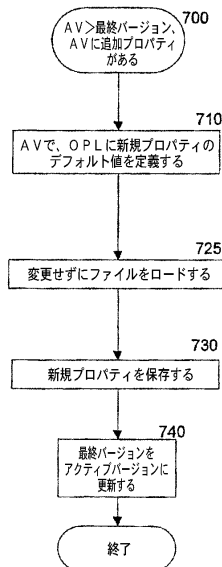
【図 5】



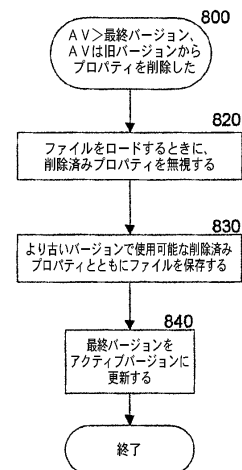
【図 6】



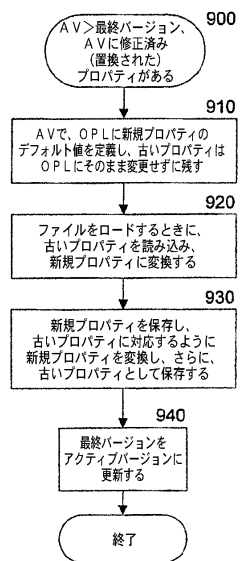
【図 7】



【図 8】



【図 9】



フロントページの続き

(72)発明者 アダム クライド エバーソール
アメリカ合衆国 98052 ワシントン州 レッドモンド ノース イースト 142 コート
6929

(72)発明者 ベンジャメン エリック ロス
アメリカ合衆国 98115 ワシントン州 シアトル ノース イースト 23 アベニュー
7018

審査官 田川 泰宏

(56)参考文献 株式会社ユニゾン, Windows版 これならわかるファイルの互換・変換・活用, 株式会社
ディー・アート, 2000年 2月25日, 初版, p.178-185

(58)調査した分野(Int.Cl., DB名)
G06F 12/00
G06F 9/445
CA/REGISTRY(STN)