

(19) 日本国特許庁(JP)

(12) 公表特許公報(A)

(11) 特許出願公表番号

特表2007-509407**(P2007-509407A)**(43) 公表日 **平成19年4月12日(2007.4.12)**

(51) Int. Cl.

G06F 9/455 (2006.01)

F I

G06F 9/44 310A

テーマコード (参考)

審査請求 未請求 予備審査請求 未請求 (全 51 頁)

(21) 出願番号	特願2006-536563 (P2006-536563)	(71) 出願人	500046438
(86) (22) 出願日	平成16年7月22日 (2004. 7. 22)		マイクロソフト コーポレーション
(85) 翻訳文提出日	平成17年5月19日 (2005. 5. 19)		アメリカ合衆国 ワシントン州 9805
(86) 国際出願番号	PCT/US2004/023608		2-6399 レッドモンド ワン マイ
(87) 国際公開番号	W02005/045565		クロソフト ウェイ
(87) 国際公開日	平成17年5月19日 (2005. 5. 19)	(74) 代理人	100077481
(31) 優先権主張番号	10/693, 409		弁理士 谷 義一
(32) 優先日	平成15年10月24日 (2003.10.24)	(74) 代理人	100088915
(33) 優先権主張国	米国 (US)		弁理士 阿部 和夫
		(72) 発明者	ジェフリー ビー. スノーバー
			アメリカ合衆国 98052 ワシントン
			州 レッドモンド ワン マイクロソフト
			ウェイ マイクロソフト コーポレーシ
			ョン内

最終頁に続く

(54) 【発明の名称】 コマンドライン命令への拡張機能を提供する機構

(57) 【要約】

本機構は、コマンドライン動作環境においてコマンドラインで入力されるコマンドに、第1の実行モードまたは代替実行モードで実行する能力を与える。コマンドは、代替実行モードで実行する命令を含む場合、代替実行モードで実行される。代替実行モードは、動作環境によって提供され、コマンドに拡張機能を提供する。代替実行モードは、コマンドの実行結果を視覚表示し、コマンドの実行をシミュレートした結果を視覚表示し、コマンドの実行前に検証を促し、実行を要求しているユーザが、コマンドを実行するのに十分な特権をもっているかどうか判定するためにセキュリティチェックを実施することなどができる。

【特許請求の範囲】

【請求項 1】

コマンドライン動作環境において、

第 1 の実行モードまたは代替実行モードで、コマンドライン上の各コマンドを実行することであって、前記代替実行モードでの前記コマンドの実行は、前記コマンドが、前記代替実行モードで実行する命令を含むときに起こり、前記代替実行モードは、前記動作環境によって提供されることを含むことを特徴とするコンピュータ実行可能方法。

【請求項 2】

前記代替実行モードは、前記コマンドの実行結果を視覚表示することを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

10

【請求項 3】

前記代替実行モードは、前記コマンドの実行をシミュレートした結果を視覚表示することを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

【請求項 4】

前記代替実行モードは、前記コマンドを実行する前に前記コマンドの実行の検証を促すことを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

【請求項 5】

前記代替実行モードは、前記コマンドの実行を要求しているユーザが、前記コマンドを実行するのに十分な特権をもっているかどうか判定するためのセキュリティチェックを実施することを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

20

【請求項 6】

前記代替実行モードでの前記コマンドの実行は、前記コマンドラインが、前記代替実行モードを指すスイッチを含むときにさらに起こることを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

【請求項 7】

前記スイッチは、「what if」を含み、前記代替実行モードは、前記コマンドの実行をシミュレートした結果を視覚表示することを特徴とする請求項 6 に記載のコンピュータ実行可能方法。

【請求項 8】

前記命令は、前記動作環境によって提供されるメソッドのコールを含むことを特徴とする請求項 1 に記載のコンピュータ実行可能方法。

30

【請求項 9】

コンピュータ実行可能命令を有する少なくとも 1 つのコンピュータ可読媒体であって、前記命令は、

タスクの性能を指示するように動作可能なコマンドラインを受け取ること、

シミュレーションモードに関連づけられる前記コマンドラインにパラメータが存在するかどうか判定すること、

前記パラメータが存在する場合、前記タスクの性能をシミュレートすること、および

前記シミュレーションの結果を報告することを含む方法を実施することを特徴とするコンピュータ可読媒体。

40

【請求項 10】

前記パラメータは、スイッチを含むことを特徴とする請求項 9 に記載のコンピュータ可読媒体。

【請求項 11】

前記スイッチは、「what if」を含むことを特徴とする請求項 10 に記載のコンピュータ可読媒体。

【請求項 12】

前記タスクは、スタンドアロン型実行可能コマンドを含むことを特徴とする請求項 9 に記載のコンピュータ可読媒体。

【請求項 13】

50

前記タスクは、実行可能コマンドからなるパイプラインを含み、各実行可能コマンドは、別個のプロセス中で動作することを特徴とする請求項 9 に記載のコンピュータ可読媒体。

【請求項 14】

前記タスクは、実行可能コマンドからなるパイプラインを含み、各実行可能コマンドは、同じプロセス中で動作することを特徴とする請求項 9 に記載のコンピュータ可読媒体。

【請求項 15】

各実行可能コマンドは、インスタンス化されたクラスを含むことを特徴とする請求項 14 に記載のコンピュータ可読媒体。

【請求項 16】

コマンドライン動作環境を提供するシステムであって、
プロセッサと、
メモリとを備え、前記メモリは、前記プロセッサによる実行のために前記メモリにロードされる複数のコンピュータ実行可能命令用に割り振られ、前記コンピュータ実行可能命令は、

コマンドライン上で入力される各コマンドを実行することであって、前記コマンドが、前記動作環境によって提供される拡張機能を用いて、前記コマンドを実行させる命令を含む場合、前記コマンドの実行は、前記拡張機能を用いることを含む方法を実施することを特徴とするシステム。

【請求項 17】

前記拡張機能は、前記コマンドの実行結果を視覚表示することを含むことを特徴とする請求項 16 に記載のシステム。

【請求項 18】

前記拡張機能は、前記コマンドの実行をシミュレートした結果を視覚表示することを含むことを特徴とする請求項 16 に記載のシステム。

【請求項 19】

前記拡張機能は、前記コマンドを実行する前に検証を促すことを含むことを特徴とする請求項 16 に記載のシステム。

【請求項 20】

前記拡張機能は、前記コマンドの実行を要求しているユーザが、前記コマンドを実行するのに十分な特権をもっているかどうか判定するためのセキュリティチェックを実施することを含むことを特徴とする請求項 16 に記載のシステム。

【発明の詳細な説明】

【技術分野】

【0001】

本明細書において開示される主題は、コマンドライン環境に関し、より詳細には、コマンドライン環境におけるコマンドの処理に関する。

【背景技術】

【0002】

コマンドライン環境において、コマンドラインインターフェイスは、コマンドを入れることによって、ユーザにタスクを直接実施させる。たとえば、プロンプト（たとえば、「C: ¥>」）を表示するウィンドウを提供するコマンドラインインターフェイスが呼び出され得る。ユーザは、コマンドを実施するために、プロンプトで「dir」などのコマンドをタイプ入力することができる。いくつかのコマンドが、より複雑なタスクを実施するために一緒にパイプライン処理され得る。こうしたパイプライン化されるコマンドが非常に複雑なコマンドライン命令をもつことは一般的である。

【0003】

コマンドラインインターフェイスに伴う 1 つの欠点は、有用な情報がコマンドラインインターフェイスによって示されないので、ユーザが、入力する正確なコマンドライン命令を知らなければならないことである。コマンドライン命令の 1 つに対して、タイプミスな

10

20

30

40

50

どの不注意によるエラーが入力された場合、タスクは、ユーザの期待に沿わない方法で実施され得る。

【発明の開示】

【発明が解決しようとする課題】

【0004】

したがって、コマンドライン命令を入力するユーザを補助する機構が必要である。

【課題を解決するための手段】

【0005】

本機構は、コマンドライン動作環境においてコマンドラインで入力されるコマンドに、第1の実行モードまたは代替実行モードで実行する能力を与える。コマンドは、代替実行モードで実行する命令を含む場合、代替実行モードで実行される。代替実行モードは、動作環境によって提供され、コマンドに拡張機能を提供する。代替実行モードは、コマンドの実行結果を視覚表示し、コマンドの実行をシミュレートした結果を視覚表示し、コマンドの実行前に検証を促し、実行を要求しているユーザが、コマンドを実行するのに十分な特権をもっているかどうか判定するためにセキュリティチェックを実施することなどができる。したがって、動作環境によって提供される拡張機能は、コマンドライン命令を入力するユーザを補助するが、開発者がコマンド中に拡張コードを書くことを要求しない。

【発明を実施するための最良の形態】

【0006】

簡潔に述べると、本機構は、コマンドライン命令への拡張機能を提供し、コマンドライン命令を入力するユーザを補助する。本機構は、望まれる拡張機能を指定するコマンドライン文法を提供する。拡張機能は、実行前の命令の確認を可能にし、実行される命令の視覚表現をもたらし、シミュレートされた命令の視覚表現をもたらし、または命令を実行する前に特権を検証することができる。コマンドライン文法は、他の機能を提供するように拡張され得る。

【0007】

以下の説明は、本機構が作用する、具体的、例示的な管理用ツール環境を述べる。他の例示的な環境は、コマンドライン命令を入力するユーザを補助することを目指す、この具体的な実施形態の特徴および／または他の特徴を含み得る。

【0008】

以下の詳細な説明は、いくつかのセクションに分けられる。第1のセクションは、管理用ツール環境が動作し得る例示的な計算機環境について述べる。第2のセクションは、管理用ツール環境のための例示的なフレームワークについて述べる。それ以降のセクションは、例示的なフレームワークの個々のコンポーネントおよびこうしたコンポーネントの動作について述べる。たとえば、「cmdletを実行する例示的なプロセス」に関するセクションは、図6とともに、コマンドライン命令に拡張機能を提供する例示的な機構を説明する。

【0009】

(例示的な計算機環境)

図1は、例示的な管理用ツール環境において使用され得る、例示的なコンピューティングデバイスを示す。ごく基本的な構成において、コンピューティングデバイス100は通常、少なくとも1つの処理ユニット102およびシステムメモリ104を含む。コンピューティングデバイスの正確な構成およびタイプによると、システムメモリ104は、揮発性（たとえばRAM）でも、不揮発性（たとえばROM、フラッシュメモリなど）でも、この2つの何らかの組合せでもよい。システムメモリ104は通常、オペレーティングシステム105、1つまたは複数のプログラムモジュール106を含み、プログラムデータ107を含み得る。オペレーティングシステム105は、コンポーネント（プロパティおよびイベントを含む）、オブジェクト、継承、多相性、リフレクションをサポートする、コンポーネントベースのフレームワーク120を含み、ワシントン州レッドモンドのマイクロソフトコーポレーションによって作成された、NET（商標）フレームワークのAPI

10

20

30

40

50

など、オブジェクト指向コンポーネントに基づくアプリケーションプログラミングインターフェイス（API）を提供する。オペレーティングシステム105は、管理用ツール（図示せず）の開発をサポートするためにコンポーネントベースのフレームワーク120と対話する管理用ツールフレームワーク200も含む。この基本構成が、破線108内の構成要素によって図1に示される。

【0010】

コンピューティングデバイス100は、追加の特徴も機能も有することができる。たとえば、コンピューティングデバイス100は、たとえば磁気ディスク、光ディスク、またはテープなど、追加のデータ記憶装置（着脱式および／または固定式）を含み得る。このような追加記憶装置が、図1に着脱式記憶装置109および固定式記憶装置110で示される。コンピュータ記憶媒体は、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータなどの情報を記憶するためのどの方法でも技術でも実装される揮発性媒体および不揮発性媒体、着脱式媒体および固定式媒体を含み得る。システムメモリ104、着脱式記憶装置109および固定式記憶装置110はすべて、コンピュータ記憶媒体の例である。コンピュータ記憶媒体は、RAM、ROM、EEPROM、フラッシュメモリまたは他のメモリ技術、CD-ROM、DVD（デジタル多用途ディスク）または他の光学記憶装置、磁気カセット、磁気テープ、磁気ディスク記憶装置または他の磁気記憶装置、あるいは、所望の情報を記憶するのに使われ得るとともにコンピューティングデバイス100によってアクセスされ得る他のどの媒体も含むが、それに限定されない。このようなどのコンピュータ記憶媒体も、装置100の一部となり得る。コンピューティングデバイス100は、たとえばキーボード、マウス、ペン、音声入力デバイス、タッチ式入力デバイスなどの入力デバイス（群）112も有することができる。たとえばディスプレイ、スピーカ、プリンタなどの出力デバイス（群）114も含められ得る。こうしたデバイスは、当該分野において公知であり、ここで詳細に説明される必要はない。

【0011】

コンピューティングデバイス100は、たとえばネットワークを介して装置を他のコンピューティングデバイス118と接続させる通信接続116も含み得る。通信接続116は、通信媒体の一例である。通信媒体は一般に、コンピュータ可読命令、データ構造、プログラムモジュール、または他のデータによって、変調データ信号、たとえば搬送波や他の移送機構として具体化されることができ、どの情報配信媒体も含む。「変調データ信号」という用語は、信号中の情報を符号化するようなやり方で設定されまたは変更される信号特性の1つまたは複数を有する信号を意味する。限定ではなく例として、通信媒体は、有線ネットワークや直接有線接続などの有線媒体、ならびに音響、RF、赤外線、および他の無線媒体などの無線媒体を含む。本明細書で使用されるコンピュータ可読媒体という用語は、記憶媒体および通信媒体両方を含む。

【0012】

（例示的な管理用ツールフレームワーク）

図2は、例示的な管理用ツールフレームワーク200の概観を全体的に示すブロック図である。管理用ツールフレームワーク200は、1つまたは複数のホストコンポーネント202、ホスト固有コンポーネント204、ホスト非依存コンポーネント206、およびハンドラコンポーネント208を含む。ホスト非依存コンポーネント206は、他のコンポーネントそれぞれ（すなわち、ホストコンポーネント202、ホスト固有コンポーネント204、およびハンドラコンポーネント208）と通信することができる。こうしたコンポーネントはそれぞれ、後で簡潔に説明され、必要に応じて、以降のセクションでさらに詳しく説明される。

【0013】

（ホストコンポーネント）

ホストコンポーネント202は、ユーザまたは他のプログラムに、関連するアプリケーション用の自動化特徴を公開する1つまたは複数のホストプログラム（たとえば、ホストプログラム210～214）を含む。各ホストプログラム210～214は、その独自の

スタイルで、たとえば、コマンドライン、グラフィカルユーザインターフェイス（GUI）、音声認識インターフェイス、アプリケーションプログラミングインターフェイス（API）、スクリプト言語、ウェブサービスなどを介して、こうした自動化特徴を公開することができる。ただし、ホストプログラム210～214はそれぞれ、管理用ツールフレームワークによって提供される機構を介して、1つまたは複数の自動化特徴を公開する。

【0014】

この例では、機構は、関連するホストプログラム210～214のユーザに対して管理用ツールの機能を明らかにするのに、`cmdlet`を用いる。さらに、機構は、ホストによって利用可能にされる1組のインターフェイスを用いて、対応するホストプログラム210～214に関連するアプリケーションに、管理用ツール環境を組み込む。以下の説明を通して、「`cmdlet`」という用語は、図2～23を参照して説明される例示的な管理用ツール環境において使われるコマンドを指すのに用いられる。

10

【0015】

`cmdlet`は、従来の管理環境におけるコマンドに対応する。しかし、`cmdlet`は、こうした従来のコマンドとは大きく異なる。たとえば、`cmdlet`は通常、それに相対するコマンドよりもサイズが小さい。というのは、`cmdlet`は、たとえば解析、データ認証、エラー報告など、管理用ツールフレームワークによって提供される共通機能を利用し得るからである。このような共通機能は、一度だけ実装され、一度だけテストされればよいので、管理用ツールフレームワーク全体に及ぶ`cmdlet`の使用は、アプリケーション固有機能に関連するインクリメンタル開発およびテストのコストを、従来の環境に比べてかなり低くさせる。

20

【0016】

さらに、従来の環境とは対照的に、`cmdlet`は、スタンドアロン型の実行可能プログラムでなくてよい。そうではなく、`cmdlet`は、管理用ツールフレームワーク内部の同じプロセス中で実行され得る。このことは、`cmdlet`が、互いの間で「ライブ」オブジェクトを交換することを可能にする。「ライブ」オブジェクトを交換するこの機能は、`cmdlet`が、こうしたオブジェクトにおいてメソッドを直接呼び出すことを可能にする。`cmdlet`の作成および使用の詳細は、後でさらに詳しく説明される。

【0017】

概して、各ホストプログラム210～214は、管理用ツールフレームワーク内部で、ユーザと他のコンポーネントの間の対話を管理する。こうした対話は、パラメータを求めるプロンプト、エラーの報告などを含み得る。通常、各ホストプログラム210～214は、独自の具体的なホスト`cmdlet`（たとえば、ホスト`cmdlet`218）の組を提供することができる。たとえば、ホストプログラムがeメールプログラムの場合、ホストプログラムは、メールボックスおよびメッセージと対話するホスト`cmdlet`を提供することができる。図2は、ホストプログラム210～214を示すが、ホストコンポーネント202は、既存または新規作成アプリケーションに関連する他のホストプログラムを含み得ることを当業者は理解するであろう。こうした他のホストプログラムは、それと関連するアプリケーションに、管理用ツール環境によって提供される機能も組み込む。ホストプログラムによって提供される処理は、後で図8とともに詳しく説明される。

30

40

【0018】

図2に示される例において、ホストプログラムは、コンピューティングデバイスのハードウェア、ソフトウェア、およびネットワークコンポーネントを管理する管理用ツールをユーザが作成し、保存し、開くための、シンプルで一貫性がある管理ユーザインターフェイスを提供する管理コンソール（すなわち、ホストプログラム210）でよい。こうした機能を遂行するために、ホストプログラム210は、管理用ツールフレームワークの上に管理GUIを構築する1組のサービスを提供する。GUIとの対話は、管理用ツール環境によって提供されるスクリプト作成能力をユーザに教えるのを補助する、ユーザが見ることのできるスクリプトとして公開されることもできる。

【0019】

50

別の例では、ホストプログラムは、コマンドライン対話シェル（すなわち、ホストプログラム 2 1 2）でよい。コマンドライン対話シェルは、コマンドライン上でシェルメタデータ 2 1 6 を入力させて、コマンドラインの処理に影響を与え得る。

【0020】

さらに別の例では、ホストプログラムは、複数のプラットフォーム、プログラミング言語、およびアプリケーションに及ぶ、分散コンピューティングおよび相互運用性のための業界標準仕様を用いるウェブサービス（すなわち、ホストプログラム 2 1 4）でよい。

【0021】

こうした例に加えて、サードパーティは、そのホストプログラムまたは他のホストプログラムとともに用いられる「サードパーティ」すなわち「プロバイダ」インターフェイスおよびプロバイダ `cmdlet` を作成することによって、独自のホストコンポーネントを追加することができる。プロバイダインターフェイスは、アプリケーションまたはインフラストラクチャが管理用ツールフレームワークによって操作され得るように、アプリケーションまたはインフラストラクチャを公開する。プロバイダ `cmdlet` は、ナビゲーション、診断、構成、ライフサイクル、動作などの自動化をもたらす。プロバイダ `cmdlet` は、全く異質な 1 組のデータストアに対して、多様型 (polymorphic) `cmdlet` 行動を示す。管理用ツール環境は、他の `cmdlet` クラスと同じ優先度で、プロバイダ `cmdlet` に作用する。プロバイダ `cmdlet` は、他の `cmdlet` と同じ機構を用いて作成される。プロバイダ `cmdlet` は、管理用ツールフレームワークに、アプリケーションまたはインフラストラクチャの詳細な機能を公開する。したがって、`cmdlet` を使うことにより、製品開発者は、1 つのホストコンポーネントを作成するだけでよく、このホストコンポーネントは次いで、製品開発者の製品を多くの管理用ツールを用いて動作させる。たとえば、例示的な管理用ツール環境を用いると、システムレベルのグラフィカルユーザインターフェイスのヘルプメニューが、既存のアプリケーションに統合され移植され得る。

【0022】

（ホスト固有コンポーネント）

ホスト固有コンポーネント 2 0 4 は、計算機システム（たとえば、図 1 のコンピューティングデバイス 1 0 0）が、フレームワークが実行されているプラットフォームの各部から管理用ツールフレームワークを切り離すのに用いるサービスの集合体を含む。したがって、プラットフォームの各タイプごとに、1 組のホスト固有コンポーネントが存在する。ホスト固有コンポーネントは、異なるオペレーティングシステム上で同じ管理用ツールをユーザが使うことを可能にする。

【0023】

一時的に図 3 に移ると、ホスト固有コンポーネント 2 0 4 は、インテリセンス (Intelli Sense) / メタデータアクセスコンポーネント 3 0 2、ヘルプ `cmdlet` コンポーネント 3 0 4、構成 / 登録コンポーネント 3 0 6、`cmdlet` セットアップコンポーネント 3 0 8、および出力インターフェイスコンポーネント 3 0 9 を含み得る。コンポーネント 3 0 2 ~ 3 0 8 は、データベースストア 3 1 4 に関連づけられたデータベースストアマネージャ 3 1 2 と通信する。パーサ 2 2 0 およびスクリプトエンジン 2 2 2 は、インテリセンス / メタデータアクセスコンポーネント 3 0 2 と通信する。コアエンジン 2 2 4 は、ヘルプ `cmdlet` コンポーネント 3 0 4、構成 / 登録コンポーネント 3 0 6、`cmdlet` セットアップコンポーネント 3 0 8、および出力インターフェイスコンポーネント 3 0 9 と通信する。出力インターフェイスコンポーネント 3 0 9 は、ホストによってアウト `cmdlet` に提供されるインターフェイスを含む。こうしたアウト `cmdlet` は次いで、ホストの出力オブジェクトをコールして、レンダリングを実施することができる。ホスト固有コンポーネント 2 0 4 は、コアエンジン 2 2 4 がロギングおよび監査能力をもたらすホスト固有（すなわち、プラットフォーム固有）サービスと通信するのに用いるロギング / 監査コンポーネント 3 1 0 も含み得る。

【0024】

例示的な一管理用ツールフレームワークにおいて、インテリセンス／メタデータアクセスコンポーネント302は、コマンド、パラメータ、およびパラメータ値の自動補完をもたらす。ヘルプcmdletコンポーネント304は、ホストユーザインターフェイスに基づいて、カスタマイズされたヘルプシステムを提供する。

【0025】

(ハンドラコンポーネント)

図2に戻ると、ハンドラコンポーネント208は、レガシーユーティリティ230、管理cmdlet232、非管理cmdlet234、リモートイングcmdlet236、およびウェブサービスインターフェイス238を含む。管理cmdlet232（プラットフォームcmdletとも呼ばれる）は、コンピューティングデバイスに関連する構成情報を照会し、または操作するcmdletを含む。管理cmdlet232は、システムタイプ情報を操作するので、ある特定のプラットフォームに依存する。しかし、各プラットフォームは通常、他のプラットフォーム上で管理cmdlet232と同様のアクションを提供する管理cmdlet232を有する。たとえば、各プラットフォームは、システム管理属性（たとえば、get/process、set/IPAddress）を入手し設定する管理cmdlet232をサポートする。ホスト非依存コンポーネント206は、ホスト非依存コンポーネント206内で生成されたcmdletオブジェクトを介して管理cmdletと通信する。cmdletオブジェクトの例示的なデータ構造は、後で図5～7とともに詳しく説明される。

10

【0026】

非管理cmdlet234（基本cmdletと呼ばれる場合もある）は、管理cmdlet232によって提供されるオブジェクトに対して他の処理をグループ化し、ソートし、フィルタリングし、実施するcmdletを含む。非管理cmdlet234は、パイプラインオブジェクトに関連づけられたデータをフォーマットし出力するcmdletも含み得る。データ駆動コマンドライン出力をもたらす例示的な機構が、後で図19～23とともに説明される。非管理cmdlet234は、各プラットフォームにおいて同じでよく、cmdletオブジェクトを介してホスト非依存コンポーネント206と対話する1組のユーティリティを提供し得る。非管理cmdlet234とホスト非依存コンポーネント206の間の対話は、オブジェクトのリフレクションを可能にし、その（オブジェクト）タイプに依存しない、リフレクトされたオブジェクトに対する処理を可能にする。したがって、こうしたユーティリティは、開発者が、非管理cmdletを書くと、こうした非管理cmdletを計算機システム上でサポートされる全クラスのオブジェクトに渡って適用することを可能にする。従来、開発者は、処理されるデータ形式を最初に把握し、次いで、そのデータのみを処理するようなアプリケーションを書かなければならなかった。その結果、従来のアプリケーションは、非常に限られた範囲のデータを処理することしかできなかった。オブジェクトタイプに依存しない、オブジェクトを処理する例示的な機構が、後で図18とともに説明される。

20

30

【0027】

レガシーユーティリティ230は、cmd.exeの下で実行されるwin32実行可能プログラムなど、既存の実行可能プログラムを含む。各レガシーユーティリティ230は、オブジェクトフレームワークにおいてあるタイプのオブジェクトであるテキストストリーム（すなわち、stdinおよびstdout）を使って管理用ツールフレームワークと通信する。レガシーユーティリティ230は、テキストストリームを使用するので、管理用ツールフレームワークによって提供されるリフレクションベースの動作は利用可能でない。レガシーユーティリティ230は、管理用ツールフレームワークとは異なるプロセスで実行される。図示されていないが、他のcmdletも、プロセス外部で動作することができる。

40

【0028】

リモートイングcmdlet236は、ウェブサービスインターフェイス238と組み合わせられて、インターネットやイントラネット（たとえば、図2に示されるインターネッ

50

ト／イントラネット 240)などの通信媒体を介して、他のコンピューティングデバイス上の、対話型でありプログラマティック(programmatic)管理用ツール環境にアクセスするためのリモーティング機構を提供する。例示的な一管理用ツールフレームワークにおいて、リモーティング機構は、複数の独立コントロールドメインに及ぶインフラストラクチャに依存する統合サービスをサポートする。リモーティング機構は、リモートコンピューティングデバイス上でスクリプトを実行させる。スクリプトは、1つまたは複数のリモートシステム上で実行され得る。スクリプトの結果は、個々の各スクリプトが完了したときに処理されることもでき、様々なコンピューティングデバイスの上のスクリプトすべてが完了した後で集約され、まとめて処理されることもできる。

【0029】

たとえば、ホストコンポーネント 202 の 1 つとして示されるウェブサービス 214 は、リモートエージェントでよい。リモートエージェントは、ターゲットシステム上のパーサおよび管理用ツールフレームワークへのリモートコマンド要求の提出を操作する。リモーティング `cmdlet` は、リモートエージェントへのアクセスを提供するためのリモートクライアントとして働く。リモートエージェントおよびリモーティング `cmdlet` は、解析されたストリームを介して通信を行う。解析されたこのストリームが、プロトコル層で保護されることもでき、解析されたストリームを暗号化し、次いで解読するために追加 `cmdlet` が使われることもできる。

【0030】

(ホスト非依存コンポーネント)

ホスト非依存コンポーネント 206 は、パーサ 220、スクリプトエンジン 222、およびコアエンジン 224 を含む。ホスト非依存コンポーネント 206 は、複数の `cmdlet` をグループ化し、`cmdlet` の動作を調整し、`cmdlet` との他のリソース、セッション、およびジョブの対話を調整するための機構およびサービスを提供する。

【0031】

(例示的なパーサ)

パーサ 220 は、様々なホストプログラムから入力要求を受け取り、コアエンジン 224 内部など、管理用ツールフレームワーク内で用いられる一様な `cmdlet` オブジェクトに入力要求をマップする機構を提供する。さらに、パーサ 220 は、受け取られた入力に基づいてデータ処理を実施することができる。入力に基づいてデータ処理を実施する例示的な一方法が、後で図 12 とともに説明される。本発明による管理用ツールフレームワークのパーサ 220 は、異なる言語またはシンタクスを容易に公開する能力を提供し、ユーザに同じ能力をもたらす。たとえば、パーサ 220 は、入力要求の解釈を担当するので、予想される入力シンタクスに影響を与える、パーサ 220 内部でのコードへの変更は、管理用ツールフレームワークの各ユーザに本質的に影響を与える。したがって、システム管理者は、異なるコンピューティングデバイス上で、異なるシンタクスをサポートする異なるパーサを提供することができる。ただし、同じパーサを使って操作している各ユーザは、各 `cmdlet` に対して一貫性があるシンタクスを経験する。対照的に、従来の環境では、各コマンドは、独自のシンタクスを実装していた。したがって、各環境は、数千のコマンドを使って、通常その多くは互いと一貫性がなかった異なるいくつかのシンタクスをサポートした。

【0032】

(例示的なスクリプトエンジン)

スクリプトエンジン 222 は、スクリプトを用いて複数の `cmdlet` を結合するための機構およびサービスを提供する。スクリプトとは、厳格な継承規則の下でセッション状態を共有するコマンドラインの集約である。スクリプト中の複数のコマンドラインは、入力要求中で提供されるシンタクスに基づいて、同期または非同期で実行され得る。スクリプトエンジン 222 は、ループおよび条件節などの制御構造を処理し、スクリプト中の変数を処理することができる。スクリプトエンジンはまた、セッション状態を管理し、ポリシー(図示せず)に基づいて、セッションデータへのアクセスを `cmdlet` に与える。

10

20

30

40

50

【0033】

(例示的なコアエンジン)

コアエンジン224は、パーサ220によって識別されるcmdletの処理を担当する。一時的に図4に移ると、管理用ツールフレームワーク200における例示的なコアエンジン224が示されている。例示的なコアエンジン224は、パイプラインプロセッサ402、ローダ404、メタデータプロセッサ406、エラー&イベントハンドラ408、セッションマネージャ410、および拡張タイプマネージャ412を含む。

【0034】

(例示的なメタデータプロセッサ)

メタデータプロセッサ406は、メタデータにアクセスし、図3に示されるデータベースストア314などのメタデータストアにメタデータを格納するように構成される。メタデータは、cmdletクラス定義などにおいて、コマンドラインを介して与えられ得る。管理用ツールフレームワーク200内部の様々なコンポーネントは、その処理を実施するとき、メタデータを要求し得る。たとえば、パーサ220は、コマンドライン上で供給されたパラメータを検証するためにメタデータを要求し得る。

【0035】

(例示的なエラー&イベントプロセッサ)

エラー&イベントプロセッサ408は、コマンドラインの処理中にエラーのそれぞれの出現についての情報を格納するためのエラーオブジェクトを提供する。本発明による管理用ツールフレームワークに特に適している特定の一エラーおよびイベントプロセッサについての追加情報については、たとえば、米国特許出願第10/413,054号明細書「System and Method for Persisting Error Information in a Command Line Environment」を参照されたい。

【0036】

(例示的なセッションマネージャ)

セッションマネージャ410は、管理用ツールフレームワーク200内部の他のコンポーネントに、セッションおよび状態情報を供給する。セッションマネージャによって管理される状態情報は、プログラミングインターフェイスを介して、どのcmdlet、ホスト、またはコアエンジンによってもアクセスされ得る。こうしたプログラミングインターフェイスは、状態情報の作成、修正、および削除を可能にする。

【0037】

(例示的なパイプラインプロセッサおよびローダ)

ローダ404は、パイプラインプロセッサ402がcmdletを実行するために、各cmdletをメモリにロードするように構成される。パイプラインプロセッサ402は、cmdletプロセッサ420およびcmdletマネージャ422を含む。cmdletプロセッサ420は、個々のcmdletをディスパッチする。cmdletが、リモートでの、または1組のリモートマシンでの実行を要求した場合、cmdletプロセッサ420は、図2に示されるリモートイングcmdlet236を用いて実行を調整する。cmdletマネージャ422は、cmdletの集約の実行を操作する。cmdletマネージャ422、cmdletプロセッサ420、およびスクリプトエンジン222(図2)は、ホストプログラム210~214から受け取られた入力に対する処理を実施するために、相互に通信する。通信は、本質的に再帰的であり得る。たとえば、ホストプログラムがスクリプトを与えた場合、スクリプトは、それ自体がスクリプトであり得るcmdletを実行するために、cmdletマネージャ422を呼び出し得る。スクリプトは次いで、スクリプトエンジン222によって実行され得る。コアエンジン用の例示的な一プロセスフローが、後で図14とともに詳しく説明される。

【0038】

(例示的な拡張タイプマネージャ)

上述したように、管理用ツールフレームワークは、オブジェクトのリフレクションを可能にし、その(オブジェクト)タイプに依存しない、リフレクトされたオブジェクトに対

する処理を可能にする1組のユーティリティを提供する。管理用ツールフレームワーク200は、このリフレクションを実施するために、計算機システム上のコンポーネントフレームワーク(図1のコンポーネントフレームワーク120)と対話する。当業者は理解するであろうが、リフレクションは、オブジェクトを照会し、そのオブジェクトのタイプを入手し、次いで、様々なオブジェクトと、そのタイプのオブジェクトに関連するプロパティとをリフレクトして、他のオブジェクトおよび/または所望の値を入手する能力をもたらす。

【0039】

リフレクションは、管理用ツールフレームワーク200に、オブジェクトについての相当量の情報を提供するが、本発明者は、リフレクションが、オブジェクトのタイプに焦点を当てていることを認識した。たとえば、データベースのデータテーブルがリフレクトされると、返される情報は、データテーブルが2つのプロパティ、すなわち列プロパティおよび行プロパティを有するというものである。こうした2つのプロパティは、データテーブル中の「オブジェクト」に関する十分な詳細を提供しない。リフレクションが拡張マークアップ言語(XML)および他のオブジェクトにおいて用いられると、同様の問題が起こる。

【0040】

したがって、本発明者は、タイプの使用法に焦点を当てる拡張タイプマネージャ412を考案した。この拡張タイプマネージャにとって、オブジェクトのタイプは重要でない。そうではなく、拡張タイプマネージャは、必要な情報を入手するためにオブジェクトが使用され得るかどうかに関心を示す。上記データテーブルの例を用いて続けると、本発明者は、データテーブルが列プロパティおよび行プロパティを有することがわかってあまり興味深くないと認識したが、ある1列が興味深い情報を含むことを認識した。使用法に注目すると、各行を「オブジェクト」に関連づけ、各列をその「オブジェクト」の「プロパティ」に関連づけることができよう。したがって、拡張タイプマネージャ412は、精密解析可能などのタイプの入力からも「オブジェクト」を作成するための機構を提供する。この機構の提供において、拡張タイプマネージャ412は、コンポーネントベースのフレームワーク120によって提供されるリフレクション機能を補い、「リフレクション」を、精密解析可能などのタイプの入力にも拡張する。

【0041】

概して、拡張タイプマネージャは、精密解析可能な入力(図示せず)にアクセスし、精密解析可能な入力を、要求されたデータ型に相関させるように構成される。拡張タイプマネージャ412は次いで、パイプラインプロセッサ402やパーサ220などの要求コンポーネントに、要求された情報を提供する。以下の説明において、精密解析可能な入力は、プロパティおよび値が見分けられ得る入力として定義される。精密解析可能な一部の例示的入力は、WMI(Windows(登録商標) Management Instrumentation)入力、ActiveXデータオブジェクト(ADO)入力、拡張マークアップ言語(XML)入力、および、NETオブジェクトなどのオブジェクト入力を含む。他の精密解析可能な入力は、サードパーティのデータ形式を含み得る。

【0042】

一時的に図18に移ると、管理用ツールフレームワークでの使用に適した例示的な拡張タイプマネージャを示す機能ブロック図が示されている。説明のために、拡張タイプマネージャによって提供される機能(丸で囲まれた番号「3」で示される)は、従来の密束縛システム(tightly bound system)によって提供される機能(丸で囲まれた番号「1」で示される)、およびリフレクションシステムによって提供される機能(丸で囲まれた番号「2」で示される)と対比される。従来の密束縛システムでは、アプリケーションにおけるコール側1802は、オブジェクトAの中の情報(たとえば、プロパティP1およびP2、メソッドM1およびM2)に直接アクセスする。上述したように、コール側1802は、コンパイル時にオブジェクトAによって与えられるプロパティ(たとえば、プロパティP1およびP2)ならびにメソッド(たとえば、メソッドM1およびM2)を先験的に知

10

20

30

40

50

っていなければならない。リフレクションシステムにおいて、ジェネリックコード1820（どのデータ型にも依存しない）は、要求されたオブジェクトに対してリフレクション1810を実施するとともにオブジェクト（たとえば、オブジェクトA）についての情報（たとえば、プロパティP1およびP2、メソッドM1およびM2）をジェネリックコード1820に返すシステム1808に問合せを行う。オブジェクトAには示されていないが、返される情報は、たとえばベンダ、ファイル、日付などの付加情報を含み得る。したがって、リフレクションにより、ジェネリックコード1820は少なくとも、密束縛システムが提供する同じ情報を入手する。リフレクションシステムは、コール側1802がシステムに問合せを行うこと、およびパラメータについて先験的に何も知らなくても付加情報を入手することも可能にする。

10

【0043】

密束縛システムおよびリフレクションシステム両方において、新規データ型は、動作環境に容易に組み込まれることができない。たとえば、密束縛システムにおいて、動作環境が引き渡されると、動作環境は、新規データ型をサポートするために再構築されなければならないので、新規データ型を組み込むことができない。同様に、リフレクションシステムでは、各オブジェクトクラスについてのメタデータが固定される。したがって、新規データ型の組込みは通常、行われない。

【0044】

しかし、本発明による拡張タイプマネージャを用いると、新規データ型が、オペレーティングシステムに組み込まれ得る。拡張タイプマネージャ1822を用いると、ジェネリックコード1820は、要求されたオブジェクトをリフレクトして、様々な外部ソース、たとえばサードパーティオブジェクト（たとえば、オブジェクトAおよびB）、セマンティックウェブ1832、オントロジー（ontology）サービス1834などによって与えられる拡張データ型（たとえば、オブジェクトA'）を入手することができる。説明されたように、サードパーティオブジェクトは、既存のオブジェクト（たとえばオブジェクトA'）を拡張することもでき、全く新しいオブジェクト（たとえばオブジェクトB）を作成することもできる。

20

【0045】

こうした外部ソースはそれぞれ、タイプメタデータ1840に一義的な構造を登録し、コード1842を提供することができる。オブジェクトが照会されると、拡張タイプマネージャは、タイプメタデータ1840を見直して、オブジェクトが登録されているかどうか判定する。オブジェクトがタイプメタデータ1840に登録されていない場合、リフレクションが実施される。登録されている場合、拡張リフレクションが実施される。コード1842は、リフレクトされるタイプに関連づけられた追加プロパティおよびメソッドを返す。たとえば、入力タイプがXMLの場合、コード1842は、XMLドキュメントからオブジェクトを作成するためのXMLの使われ方を記述する記述ファイルを含み得る。したがって、タイプメタデータ1840は、その具体的な入力タイプ向けのオブジェクトを作成する所望のプロパティを入手するために、拡張タイプマネージャ412が様々なタイプの精密解析可能入力（たとえば、サードパーティオブジェクトAおよびB、セマンティックウェブ1832）をどのようにして照会するべきかを記述し、コード1842は、こうした所望のプロパティを入手するための命令を提供する。その結果、拡張タイプマネージャ412は、全タイプのオブジェクトの「リフレクション」を可能にする間接指定層を提供する。

30

40

【0046】

拡張タイプの提供に加えて、拡張タイプマネージャ412は、たとえばプロパティパス機構、キー機構、比較機構、変換機構、グロバ（globber）機構、プロパティセット機構、関係機構など、追加クエリ機構を提供する。こうしたクエリ機構はそれぞれ、後で「例示的な拡張タイプマネージャ処理」というセクションで説明されるが、コマンド文字列を入力するとき、システム管理者に柔軟性をもたらす。拡張タイプマネージャ用のセマンティクスを実装するのに、様々な技術が用いられ得る。3つの技術が後で説明される。ただ

50

し、こうした技術の変形形態は、特許請求の範囲に記載されている本発明の範囲から逸脱することなく利用され得ることを当業者は理解するであろう。

【0047】

ある技術では、静的メソッド（たとえば、`getProperty()`）を有する一連のクラスが提供される。オブジェクトは、静的メソッドに入力され（たとえば、`getProperty(object)`）、静的メソッドは、1組の結果を返す。別の技術では、動作環境は、オブジェクトをアダプタで包む。このようにして、入力を与えられない。アダプタの各インスタンスは、包まれたオブジェクトに作用するとともに包まれたオブジェクト用のプロパティを返す `getProperty` メソッドを有する。以下は、この技術を示す擬似コードである。

10

【0048】

【数1】

Class Adaptor

```
{
    Object X;

    getProperties();
}
```

20

【0049】

さらに別の技術では、アダプタクラスが、オブジェクトを下位分類する。従来、下位分類は、コンパイルの前に起きた。しかし、特定の動作環境を用いると、下位分類は、動的に起こり得る。こうしたタイプの環境に対して、以下が、この技術を示す擬似コードである。

【0050】

【数2】

Class Adaptor : A

```
{
    getProperties()
    {
        return data;
    }
}
```

30

40

【0051】

したがって、図18に示されるように、拡張タイプマネージャは、開発者が新規データ型を作成し、データ型を登録することを可能にし、他のアプリケーションおよび `cmdlet` に、新規データ型を使用させる。対照的に、従来の管理環境では、各データ型は、そのデータ型からインスタンス化されたオブジェクトに関連づけられたプロパティまたはメソッドが直接アクセスされ得るように、コンパイル時に知られていなければならなかった。したがって、管理環境によってサポートされる新規データ型の追加は、従来、ほとんど行われなかった。

50

【0052】

図2に戻ると、概して、管理用ツールフレームワーク200は、ユーザによって入力されるコマンドの実行を調整するシェルに依拠するのではなく、（たとえば、ホストcmdletにより）機能を処理部分（たとえば、ホスト非依存コンポーネント206）およびユーザ対話部分に分割する。さらに、本発明による管理用ツール環境は、解析およびデータ認証に要求されるコードが、各コマンドに含まれなくなり、管理用ツールフレームワーク内部のコンポーネント（たとえば、パーサ220）によって提供されるので、管理用ツールのプログラミングを大幅に簡略化する。管理用ツールフレームワークにおいて実施される例示的な処理は、後で説明される。

【0053】

10

（例示的な動作）

図5～7は、管理用ツール環境において使われる例示的なデータ構造を、図表により示す。図8～17は、管理用ツール環境における例示的な処理フローを、図表により示す。特定の処理は、後で説明されるコンポーネントとは異なるコンポーネントによって、本発明の範囲から逸脱することなく実施され得ることを当業者は理解するであろう。管理用ツールフレームワークのコンポーネント内で実施される処理を説明する前に、管理用ツールフレームワークにおいて用いられる例示的なデータ構造が説明される。

【0054】

（cmdletオブジェクトの例示的なデータ構造）

図5は、図2に示される管理用ツールフレームワークでの使用に適したcmdletを指定する例示的なデータ構造である。cmdletは、完了されると、管理cmdlet、非管理cmdlet、ホストcmdlet、プロバイダcmdletなどになり得る。以下の説明は、システム管理者の視点（すなわち、プロバイダcmdlet）を参照して、cmdletの作成について述べる。ただし、各タイプのcmdletは、同じやり方で作成され、同じように動作する。cmdletは、C#など、どの言語でも書かれ得る。さらに、cmdletは、スクリプト言語などを使って書かれ得る。管理用ツール環境が、NETフレームワークとともに動作する場合、cmdletは、.NETオブジェクトでよい。

20

【0055】

プロバイダcmdlet500（これ以降、cmdlet500と呼ばれる）は、cmdletクラス名（たとえば、StopProcess504）を有するパブリッククラスである。cmdlet500は、cmdletクラス506から派生する。cmdletクラス506用の例示的なデータ構造が、後で図6とともに説明される。各cmdlet500は、名前（たとえば、Stop/Process）をcmdlet500に関連づけるコマンド属性502に関連づけられる。名前は、管理用ツール環境内部で登録される。後で説明されるように、名前（たとえば、Stop/Process）をもつコマンド文字列が、コマンドラインで、またはスクリプト中で入力として与えられると、パーサは、cmdletレジストリ内部を調べて、cmdlet500を識別する。

30

【0056】

cmdlet500は、cmdletに対して予想される入力パラメータ用の文法を定義する文法機構に関連づけられる。文法機構は、直接または間接的にcmdletに関連づけられ得る。たとえば、cmdlet500は、直接的な文法の関連づけを示す。このcmdlet500において、1つまたは複数のパブリックパラメータ（たとえば、ProcessName510およびPID512）が宣言される。パブリックパラメータの宣言は、cmdlet500への入力オブジェクトの解析を駆動する。あるいは、パラメータの記述は、XMLドキュメントなど、外部ソースに現れ得る。この外部ソースにおけるパラメータの記述は次いで、cmdletへの入力オブジェクトの解析を駆動する。

40

【0057】

各パブリックパラメータ510、512は、それに関連づけられた1つまたは複数の属性（すなわち、ディレクティブ）をもち得る。ディレクティブは、解析ディレクティブ5

50

20、データ認証ディレクティブ521、データ生成ディレクティブ522、処理ディレクティブ523、エンコードディレクティブ524、およびドキュメンテーションディレクティブ525というカテゴリーのいずれでもよい。ディレクティブは、角括弧で囲まれ得る。各ディレクティブは、予想される次の入力パラメータに対して実施される動作を記述する。ユーザ対話タイプのディレクティブなど、ディレクティブのいくつかは、クラスレベルでも適用され得る。ディレクティブは、`cmdlet`に関連づけられたメタデータに格納される。こうした属性の適用が、後で図12とともに説明される。

【0058】

こうした属性は、`cmdlet`内で宣言されるパラメータの投入にも影響を与え得る。こうしたパラメータを投入する例示的な一プロセスが、後で図16とともに説明される。10
コアエンジンは、準拠性を保証するためにこうしたディレクティブを適用し得る。`cmdlet 500`は、第1のメソッド530（これ以降、同義的に`Start Processing`メソッド530と呼ばれる）および第2のメソッド540（これ以降、同義的に`processRecord`メソッド540と呼ばれる）を含む。コアエンジンは、第1および第2のメソッド530、540を用いて、`cmdlet 500`の処理を指示する。たとえば、第1のメソッド530が一度実行され、セットアップ機能を実施する。第2のメソッド540中のコード542は、`cmdlet 500`によって処理される必要がある各オブジェクト（たとえば`record`）に対して実行される。`cmdlet 500`は、`cmdlet 500`の後でクリーンアップを行う第3のメソッド（図示せず）も含み得る。

【0059】

したがって、図5に示されるように、第2のメソッド540中のコード542は通常、非常に短く、たとえば解析コード、データ認証コードなど、従来の管理用ツール環境において要求される機能を含まない。したがって、システム管理者は、複雑なプログラミング言語を学習することなく、複雑な管理タスクを開発することができる。20

【0060】

図6は、図5に示される`cmdlet`が派生される`cmdlet`基本クラス602を詳述する例示的なデータ構造600である。`cmdlet`基本クラス602は、`cmdlet`がフックステートメントを含み、対応するスイッチがコマンドラインでまたはスクリプト中で入力される（併せてコマンド入力と呼ばれる）ときはいつでも、付加機能を提供する命令を含む。30

【0061】

例示的なデータ構造600は、ブーリアンパラメータ`verbose 610`、`what if 620`、および`confirm 630`などのパラメータを含む。後で説明されるように、こうしたパラメータは、コマンド入力で入力され得る文字列に対応する。例示的なデータ構造600は、実行を要求されているタスクが許可されるかどうかを判定するセキュリティメソッド640も含み得る。

【0062】

図7は、`cmdlet`を詳述する、別の例示的なデータ構造700である。概して、データ構造700は、管理用ツールフレームワークと`cmdlet`の間のコントラクトを明白に表現する手段を提供する。データ構造500と同様、データ構造700は、`cmdlet` 40
`クラス704`から派生するパブリッククラスである。ソフトウェア開発者は、「`get/process`」および「`format/table`」などの名詞／動詞ペアを`cmdlet 700`に関連づける`cmdlet Declaration 702`を指定する。名詞／動詞ペアは、管理用ツール環境内で登録される。動詞も名詞も、`cmdlet`名において暗黙的でよい。また、データ構造500と同様、データ構造700は、データ構造500とともに説明される1つまたは複数のディレクティブ510～520に関連づけられ得る1つまたは複数のパブリックメンバ（たとえば、`Name 730`、`Recursive 732`）を含み得る。

【0063】

ただし、この例示的なデータ構造700において、予想される入力パラメータ730お 50

よび732はそれぞれ、入力属性731および733それぞれに関連づけられる。入力属性731および733は、それぞれのパラメータ730および732用のデータがコマンドラインから取得されるべきであると指定する。したがって、この例示的なデータ構造700において、別のcmdletによって発せられたパイプラインオブジェクトから投入される、予想される入力パラメータが存在しない。したがって、データ構造700は、cmdlet基本クラスによって提供される第1のメソッド（たとえば、StartProcessing）も第2のメソッド（たとえば、ProcessRecord）もオーバーライドしない。

【0064】

データ構造700は、入力パラメータとして認識されないプライベートメンバ740も含み得る。プライベートメンバ740は、ディレクティブの1つに基づいて生成されるデータの格納に使われ得る。 10

【0065】

したがって、データ構造700に示されるように、具体的なcmdletクラスにおいてパブリックプロパティおよびディレクティブの宣言を使用することにより、cmdlet開発者は、そのcmdletに対して予想される入力パラメータ用の文法を容易に指定し、cmdlet開発者が基底論理のいずれかをも生成する必要なく、予想される入力パラメータに対して実施されるべき処理を指定することができる。データ構造700は、cmdletと文法機構の間の直接的な関連づけを示す。上述したように、この関連づけは、XMLドキュメントなどの外部ソースにおいて、予想されるパラメータ定義を明記する 20
などして、間接的でもよい。

【0066】

管理用ツール環境における例示的なプロセスフローが次に説明される。

【0067】

（例示的なホスト処理フロー）

図8は、図2に示される管理用ツールフレームワーク内部で実施されるホスト処理のための例示的なプロセスを示す論理フロー図である。プロセス800は、ブロック801で始まり、ここで、具体的なアプリケーション向けの管理用ツール環境を開始するための要求が受け取られている。要求は、アプリケーションアイコンの選択などのキーボード入力を介してローカルに、または異なるコンピューティングデバイスのウェブサービスインターフェイスを介してリモートに送られている可能性がある。いずれのシナリオでも、処理は、ブロック802へ続く。 30

【0068】

ブロック802で、「ターゲット」コンピューティングデバイス上の具体的なアプリケーション（たとえばホストプログラム）が、その環境をセットアップする。このセットアップは、cmdletのどのサブセット（たとえば、管理cmdlet232、非管理cmdlet234、およびホストcmdlet218）がユーザに対して使用可能にされるかという判定を含む。通常、ホストプログラムは、非管理cmdlet234すべてを使用可能にし、独自のホストcmdlet218を使用可能にする。さらに、ホストプログラムは、プロセス、ディスクなどを扱うcmdletなど、管理cmdlet232の一部を使用可能にする。したがって、ホストプログラムがcmdletの一部を使用可能にすると、管理用ツールフレームワークは、対応するアプリケーションに有効に組み込まれる。処理は、ブロック804へ続く。 40

【0069】

ブロック804で、特定のアプリケーションを介して入力が入手される。上述したように、入力は、たとえばコマンドライン、スクリプト、音声、GUIなど、いくつかの形をとり得る。たとえば、コマンドラインを介して入力が入手されると、入力は、キーボードで入れられるキーストロークから取得される。GUIホストに対しては、GUIに基づいて文字列が組み立てられる。処理は、ブロック806で継続する。

【0070】

ブロック 806 で、入力 は、処理のために管理用ツールフレームワーク内部の他のコンポーネントに与えられる。ホストプログラムは、パーサなど、他のコンポーネントに入力を直接転送することができる。あるいは、ホストプログラムは、そのホスト `cmdlet` の 1 つを介して入力を転送することもできる。ホスト `cmdlet` は、その具体的なタイプの入力（たとえば音声）を、管理用ツールフレームワークによって認識されるタイプの入力（たとえば、テキスト列、スクリプト）に変換することができる。たとえば、音声入力は、音声入力の内容に応じて、スクリプトまたはコマンドライン文字列に変換され得る。各ホストプログラムは、そうしたタイプの入力を、管理用ツールフレームワークによって認識される入力に変換する責任があるので、管理用ツールフレームワークは、任意の数の様々なホストコンポーネントからの入力を受諾することができる。さらに、管理用ツールフレームワークは、入力がその `cmdlet` の 1 つを介して転送されるときにデータ型の間の変換を実施する、豊富な 1 組のユーティリティを提供する。他のコンポーネントによって入力に対して実施される処理が、後で他のいくつかの図とともに説明される。ホスト処理は、判断ブロック 808 で継続する。

10

20

30

40

50

【0071】

判断ブロック 808 で、追加入力の要求が受け取られたかどうか、判定が行われる。この判定は、入力の処理を担当する他のコンポーネントの 1 つが、その処理を完了するためにユーザからの付加情報を必要とする場合に起こり得る。たとえば、特定のデータにアクセスするのにパスワードが要求され、具体的なアクションの確認が必要とされる場合などがある。特定のタイプのホストプログラム（たとえば音声メール）に対して、このような要求は適切でない場合がある。したがって、ユーザに付加情報を問い合わせるのではなく、後で状態が再開され、入力の実行が継続され得るようにホストプログラムが状態をシリアル化し、状態を一時停止し、通知を送ることができる。別の変形形態では、ホストプログラムは、所定の期間の後でデフォルト値を提供することができる。追加入力の要求が受け取られた場合、処理は、ブロック 804 にループバックし、ここで、追加入力が入手される。処理は次いで、上述されたようにブロック 806 および 808 を経由して継続する。追加入力要求が受け取られず、入力が処理されている場合、処理は、ブロック 810 へ続く。

【0072】

ブロック 810 で、管理用ツールフレームワーク内部の他のコンポーネントから結果が受け取られる。結果は、エラーメッセージ、状況などを含み得る。結果は、管理用ツールフレームワーク内部のホスト `cmdlet` によって認識され処理されるオブジェクトの形である。後で説明されるように、各ホスト `cmdlet` 用に書かれるコードは、非常に小さい。したがって、開発コストへの莫大な投資を要せずに豊富な 1 組の出力が表示され得る。処理は、ブロック 812 で継続する。

【0073】

ブロック 812 で、結果が閲覧され得る。ホスト `cmdlet` は、ホストプログラムによってサポートされる表示スタイルに結果を変換する。たとえば、返されたオブジェクトは、アイコン、ほえる犬 (barking dog) など、図形表示を用いて、GUI ホストプログラムによって表示され得る。ホスト `cmdlet` は、データのデフォルト形式および出力を提供する。デフォルト形式および出力は、後で図 19 ~ 23 とともに説明される例示的な出力処理 `cmdlet` を使用し得る。結果が任意選択で表示された後、ホスト処理が完了する。

【0074】

（入力を操作する例示的なプロセスフロー）

図 9 は、図 2 に示される管理用ツールフレームワーク内部で実施される入力操作のための例示的なプロセスを示す論理フロー図である。処理は、ブロック 901 で始まり、ここで、入力が、ホストプログラムを介して入れられ、管理用ツールフレームワーク内部の他のコンポーネントに転送されている。処理は、ブロック 902 で継続する。

【0075】

ブロック902で、入力、ホストプログラムから受け取られる。例示的な一管理用ツールフレームワークにおいて、入力は、パーサによって受け取られ、このパーサは、入力を解読し、入力をさらに処理するように向ける。処理は、判断ブロック904で継続する。

【0076】

判断ブロック904で、入力がスクリプトであるかどうか判定が行われる。入力は、スクリプト、またはコマンドラインを表す文字列（これ以降、「コマンド文字列」と呼ばれる）の形をとり得る。コマンド文字列は、まとめてパイプライン処理される1つまたは複数のcmdletを表し得る。管理用ツールフレームワークは、異なるいくつかのホストをサポートするが、各ホストは、処理用のスクリプトまたはコマンド文字列として入力を提供する。後で示されるように、スクリプトとコマンド文字列の間の対話は、本質的に再帰的である。たとえば、スクリプトは、cmdletを呼び出す行を有し得る。cmdlet自体は、スクリプトでよい。

10

【0077】

したがって、判断ブロック904で、入力がスクリプトの形の場合、処理は、ブロック906で継続し、ここで、スクリプトの処理が実施される。スクリプトの形でない場合、処理は、ブロック908で継続し、ここで、コマンド文字列の処理が実施される。ブロック906または908で実施される処理が完了されると、入力の処理が完了する。

【0078】

（スクリプトの例示的な処理）

20

図10は、図9に示される入力操作プロセスでの使用に適したスクリプトを処理するプロセスを示す論理フロー図である。プロセスは、ブロック1001で始まり、ここで、入力は、スクリプトとして識別されている。スクリプトエンジンおよびパーサは、相互に通信して、後続の機能を実施する。処理は、ブロック1002で継続する。

【0079】

ブロック1002で、スクリプトに対して前処理が実施される。一時的に図11に移ると、スクリプト処理プロセス1000での使用に適したスクリプト前処理プロセス1100を示す論理フロー図が示されている。スクリプトの前処理は、ブロックで1101始まり、判断ブロック1102へ続く。

【0080】

30

判断ブロック1102で、スクリプトが初めて実行されているのかどうか、判定が行われる。この判定は、レジストリまたは他の記憶機構から入手される情報に基づき得る。スクリプトは、記憶機構内部から識別され、関連するデータが検討される。スクリプトが実行されたことがない場合、処理は、ブロック1104で継続する。

【0081】

ブロック1104で、スクリプトがレジストリに登録される。この登録は、管理用ツールフレームワーク内部のコンポーネントによるその後のアクセスのために、スクリプトについての情報を格納させる。処理は、ブロック1106で継続する。

【0082】

ブロック1106で、ヘルプおよびドキュメンテーションがスクリプトから抽出され、レジストリに格納される。やはり、この情報も、管理用ツールフレームワーク内部のコンポーネントによって後でアクセスされ得る。スクリプトはこの時点で、処理の準備が整い、図10のブロック1004に戻る。

40

【0083】

判断ブロック1102に戻ると、スクリプトが前に実行されたことがあるとプロセスが結論づけた場合、処理は判断ブロック1108へ続く。判断ブロック1108で、スクリプトが処理中に障害を起こしたかどうか、判定が行われる。この情報は、レジストリから入手され得る。スクリプトが障害を起こしていない場合、スクリプトは、処理の準備が整い、図10のブロック1004に戻る。

【0084】

50

しかし、スクリプトが障害を起こした場合、処理は、ブロック1110で継続する。ブロック1110で、スクリプトエンジンは、スクリプトが障害を起こしたことを、ホストプログラムを介してユーザに通知し得る。この通知は、スクリプトを進めるか、それともスクリプトを終了させるか、ユーザに判断させる。図8とともに上述されたように、ホストプログラムは、この要求を、入力スタイル（たとえば、音声、コマンドライン）に応じて様々なやり方で扱うことができる。ユーザから追加入力が受け取られると、スクリプトは、処理のために図10のブロック1004に戻るか、またはスクリプトが破棄される。

【0085】

図10のブロック1004に戻ると、スクリプト中の行が取得される。処理は、判断ブロック1006で継続する。判断ブロック1006で、行が何らかの制約を含むかどうか、判定が行われる。制約は、予め定義された開始キャラクタ（たとえば、括弧「[」）および対応する終了キャラクタ（たとえば、閉じ括弧「]」）で検出される。行が制約を含む場合、処理は、ブロック1008へ続く。

【0086】

ブロック1008で、行に含まれる制約が適用される。概して、制約は、スクリプトに入れられたパラメータ用のタイプを指定するとともにパラメータに対して実施されるべき認証論理を指定するための機構を、管理用ツールフレームワーク内部で提供する。制約は、パラメータに対して適用可能なだけでなく、変数など、スクリプトに入れられたどのタイプの構文(construct)にも適用可能である。したがって、制約は、データ型を指定し、パラメータを検証するための機構を、解釈環境内部で提供する。従来の環境では、システム管理者は、スクリプトに入れられたパラメータを正式にテストすることができない。制約を課す例示的なプロセスは、図12に示される。

【0087】

判断ブロック1010で、スクリプトの行が組込み機能を含むかどうか、判定が行われる。組込み機能は、コアエンジンによって実施されない機能である。組込み機能は、`cmdlet`を用いて処理されることも、インライン関数など、他の機構を用いて処理されることもできる。行が組込み機能をもたない場合、処理は、判断ブロック1014で継続する。組込み機能をもつ場合、処理は、1012ブロックで継続する。

【0088】

ブロック1012で、スクリプトの行で与えられた組込み機能が処理される。組込み機能の例は、たとえば「`if`」ステートメント、「`for`」ループ、スイッチなど、コントロール構造の実行を含み得る。組込み機能は、割当てタイプのステートメント（たとえば、`a = 3`）も含み得る。組込み機能が処理されると、処理は、判断ブロック1014へ続く。

【0089】

判断ブロック1014で、スクリプトの行がコマンド文字列を含むかどうか、判定が行われる。判定は、登録されているコマンド文字列と、起こり得る`cmdlet`呼出しのシンタクスとに行中のデータが関連づけられているかどうかに基づく。上述したように、コマンド文字列およびスクリプトの処理は、本質的に再帰的であり得る。というのは、スクリプトは、コマンド文字列を含むことができ、コマンド文字列は、それ自体がスクリプトである`cmdlet`を実行することができるからである。行がコマンド文字列を含まない場合、処理は、判断ブロック1018で継続する。コマンド文字列を含む場合、処理は、ブロック1016で継続する。

【0090】

ブロック1016で、コマンド文字列が処理される。概して、コマンド文字列の処理は、パーサによって`cmdlet`クラスを識別すること、および対応する`cmdlet`オブジェクトを実行のためにコアエンジンに渡すことを含む。コマンド文字列は、いくつかの別個の`cmdlet`オブジェクトに解析されるとともにコアエンジンによって個別に処理されるパイプライン型コマンド文字列も含み得る。コマンド文字列を処理する例示的な一プロセスが、後で図14とともに説明される。コマンド文字列が処理されると、処理は、

判断ブロック1018で継続する。

【0091】

判断ブロック1018で、スクリプトにさらに行があるかどうか、判定が行われる。スクリプトにさらに行がある場合、処理は、ブロック1004にループバックし、ブロック1004～1016で上述されたように進行する。行がない場合、処理は完了する。

【0092】

ブロック1008で制約を課す例示的なプロセスが、図12に示されている。プロセスは、ブロック1201で始まり、ここで、スクリプト中の、またはコマンドライン上のコマンド文字列中の制約が検出される。制約がスクリプト中にある場合、制約および関連する構文は、同じ行にも別個の行にも現れ得る。制約がコマンド文字列中にある場合、制約および関連する構文は、行末インジケータ（たとえば、エンターキー）の前に現れる。処理は、ブロック1202へ続く。

10

【0093】

ブロック1202で、解釈環境から制約が入手される。例示的な一管理用ツール環境において、パーサは、入力を復号し、制約の出現を判定する。制約は、述語ディレクティブ(predicate directive)、解析ディレクティブ、データ認証ディレクティブ、データ生成ディレクティブ、処理ディレクティブ、エンコードディレクティブ、およびドキュメンテーションディレクティブのカテゴリの1つでよい。例示的な一解析シンタックスの場合、ディレクティブは、角括弧で囲まれ、その後続く構文を記述する。構文は、関数、変数、スクリプトなどでよい。

20

【0094】

後で説明されるように、ディレクティブを使用することにより、スクリプト作成者は、スクリプト作成者が基底論理のいずれを生成することも要せずに、スクリプトまたはコマンドライン（すなわち、解釈環境）におけるパラメータに対する処理を容易にタイプし実施することを可能にされる。処理は、ブロック1204へ続く。

【0095】

ブロック1204で、入手された制約が、関連する構文用のメタデータに格納される。関連する構文は、1つまたは複数の属性表示(attribution)トークン(制約を示すトークン)が遭遇された後、第1の非属性表示トークンであると識別される。処理は、ブロック1206へ続く。

30

【0096】

ブロック1206で、スクリプト中またはコマンド文字列中で構文が遭遇されると常に、メタデータ内部で定義された制約が、構文に適用される。制約は、データ型、述語ディレクティブ1210、ドキュメンテーションディレクティブ1212、解析ディレクティブ1214、データ生成ディレクティブ1216、データ認証ディレクティブ1218、ならびにオブジェクト処理およびエンコードディレクティブ1220を含み得る。データ型を指定する制約は、管理用ツールフレームワークが実行されているシステム上でサポートされるどのデータ型も指定し得る。述語ディレクティブ1210は、処理が起こるべきかどうかを示すディレクティブである。したがって、述語ディレクティブ1210は、実行に対して環境が適正であることを保証する。たとえば、スクリプトは、次の述語ディレクティブを含み得る。

40

【0097】

```
[PredicateScript("isInstalled","ApplicationZ")]
```

この述語ディレクティブは、スクリプトを実行する前に正しいアプリケーションがコンピューティングデバイスにインストールされることを保証する。通常、システム環境変数は、述語ディレクティブとして指定され得る。ディレクティブタイプ1212～1220にある例示的なディレクティブが、テーブル1～5に示される。スクリプトの処理は次いで、完了する。

【0098】

したがって、解釈環境においてタイプおよび制約を適用する本プロセスは、システム管

50

理者が、この処理を実施する基底論理を書くことなく、タイプを容易に指定し、認証要件を指定することなどを可能にする。以下は、次のように指定されたコマンド文字列に対して実施される制約処理の例である。

【0099】

【数3】

`[Integer][ValidationRange(3,5)]$a=4.`

【0100】

「[]」で示される属性表示トークンにより指定される2つの制約がある。第1の属性表示トークンは、変数が整数型であることを示し、第2の属性表示トークンは、変数\$aの値が3と5を含むその間でなければならないことを示す。このコマンド文字列の例は、後続コマンド文字列または行の中で変数\$aが割り当てられている場合、変数\$aは、2つの制約と突き合わせて調べられることを保証する。したがって、次のコマンド文字列が、エラー中でそれぞれ現れるはずである。

【0101】

【数4】

`$a = 231`

`$a = "apple"`

`$a = $(get/location).`

【0102】

制約は、管理用ツールフレームワークにおいて様々な段階で適用される。たとえば、使用可能性ディレクティブ、ドキュメンテーションディレクティブ、および解析ガイドラインディレクティブは、非常に早い段階にパーサ内で処理される。データ生成ディレクティブおよび認証ディレクティブは、パーサが入力パラメータすべての解析を終えるとエンジン内で処理される。

【0103】

以下のテーブルは、ディレクティブに応答して管理用ツール環境によって実施されるプロセスの説明とともに、様々なカテゴリー用の代表的なディレクティブを示す。

【0104】

10

20

30

【表 1】

名称	記述
PrerequisiteMachineRoleAttribute	特定のマシン役割(例:ファイルサーバ、メールサーバ)においてのみ要素が用いられるかどうかを、シェルに通知する
PrerequisiteUserRoleAttribute	特定のユーザ役割(例:ドメイン管理者、バックアップオペレータ)においてのみ要素が用いられるかどうかを、シェルに通知する
PrerequisiteScriptAttribute	実際のコマンドまたはパラメータを実行する前にこのスクリプトが実行されることをシェルに通知する。パラメータ認証に用いられ得る
PrerequisiteUITypeAttribute	実行前に使用可能なユーザインターフェイスを調べるのに用いられる

10

テーブル1 使用可能性ディレクティブ

20

【0105】

【表 2】

名称	記述
ParsingParameterPositionAttribute	位置に基づいて、制限なしのパラメータをマップする
ParsingVariableLengthParameterListAttribute	ParsingParameterPosition属性をもたないパラメータをマップする
ParsingDisallowInteractionAttribute	要求される数よりパラメータ数が少ないときのアクションを指定する
ParsingRequireInteractionAttribute	パラメータが対話により入手されることを指定する
ParsingHiddenElementAttribute	エンドユーザに対してパラメータを見えないようにする
ParsingMandatoryParameterAttribute	パラメータが必要とされることを指定する
ParsingPasswordParameterAttribute	パラメータの特殊操作を要求する
ParsingPromptStringAttribute	パラメータを促すプロンプトを指定する
ParsingDefaultAnswerAttribute	パラメータに対するデフォルトの回答を指定する
ParsingDefaultAnswerScriptAttribute	パラメータに対するデフォルトの回答を得るためのアクションを指定する
ParsingDefaultValueAttribute	パラメータのデフォルト値を指定する
ParsingDefaultValueScriptAttribute	パラメータのデフォルト値を得るためのアクションを指定する
ParsingParameterMappingAttribute	パラメータをグループ化する方法を指定する
ParsingParameterDeclarationAttribute	フィールドがパラメータであると定義する
ParsingAllowPipelineInputAttribute	パイプラインからパラメータが投入され得ることを定義する

10

20

テーブル2 解析ガイドラインディレクティブ

30

【0106】

40

【表 3】

名称	記述
DocumentNameAttribute	対話またはヘルプ用の要素を指すNameを与える
DocumentShortDescriptionAttribute	要素を簡単に記述する
DocumentLongDescriptionAttribute	要素を詳細に記述する
DocumentExampleAttribute	要素の例を挙げる
DocumentSeeAlsoAttribute	関連要素を列挙する
DocumentSynopsisAttribute	要素のドキュメンテーション情報を提供する

テーブル3 ドキュメンテーションディレクティブ

10

【0107】

【表 4】

名称	記述
ValidationRangeAttribute	パラメータが特定の範囲内でなければならないことを指定する
ValidationSetAttribute	パラメータが特定のコレクション中でなければならないことを指定する
ValidationPatternAttribute	パラメータが特定のパターンに適合しなければならないことを指定する
ValidarionLengthAttribute	文字列がサイズ範囲内でなければならないことを指定する
ValidationTypeAttribute	パラメータが特定のタイプでなければならないことを指定する
ValidationCountAttributue	入力項目が特定の個数でなければならないことを指定する
ValidationFileAttribute	ファイル用の特定のプロパティを指定する
ValidationFileAttributesAttribute	ファイル用の特定のプロパティを指定する
ValidationFileSizeAttribute	ファイルが指定された範囲内でなければならないことを指定する
ValidationNetworkAttribute	与えられたNetworkEntityが特定のプロパティをサポートすることを指定する
ValidationScriptAttribute	要素を使う前に、評価条件を指定する
ValidationMethodAttribute	要素を使う前の評価条件を指定する

テーブル4 データ認証ディレクティブ

20

30

40

【0108】

【表 5】

名称	記述
ProcessingTrimStringAttribute	文字列のサイズ制限を指定する
ProcessingTrimCollectionAttribute	コレクションのサイズ制限を指定する
EncodingTypeCoercionAttribute	オブジェクトがエンコードされるべきTypeを指定する
ExpansionWildcardsAttribute	グロブリング(globbering)を可能にするための機構を提供する

テーブル5 処理およびエンコードディレクティブ

10

【0109】

NET (商標) フレームワーク内部で例示的な管理用ツールフレームワークが動作しているとき、各カテゴリは、基本カテゴリークラス (たとえば、CmdAttribute) から派生される基本クラスを有する。基本カテゴリークラスは、System.Attribute クラスから派生する。各カテゴリは、カテゴリ処理中にパーサによってコールされる、予め定義された関数 (たとえば、attrib.func()) を有する。スクリプト作成者は、カスタムカテゴリークラス (たとえば、CmdCustomAttribute) から派生されるカスタムカテゴリーを作成することができる。スクリプト作成者は、そのカテゴリー用の基本カテゴリークラスからディレクティブクラスを派生することによって既存のカテゴリークラスを拡張し、予め定義された関数を実装でオーバーライドすることもできる。スクリプト作成者は、ディレクティブをオーバーライドし、予め定義されたディレクティブの組に新規ディレクティブを追加することもできる。

20

【0110】

こうしたディレクティブの処理順序は、パーサによってアクセス可能な外部データストアに格納され得る。管理用ツールフレームワークは、登録されたカテゴリーを探し、そのカテゴリー中のディレクティブそれぞれに対して関数 (たとえば、ProcessCustomDirective) をコールする。したがって、カテゴリー処理の順序は、カテゴリー実行情報を永続ストアに格納することによって、動的になり得る。様々なプロセス段階で、パーサは、永続ストアの中を調べて、いずれかのメタデータカテゴリーが、その時点で実行される必要があるかどうか判定する。この判定は、永続ストアからカテゴリーエントリを削除することによって、カテゴリーを容易に廃止させる。

30

【0111】

(コマンド文字列の例示的な処理)

コマンド文字列を処理する例示的な一プロセスが次に説明される。図13は、図2に示される管理用ツールフレームワークにおける、パーサ220およびコアエンジン224を介したコマンド文字列1350の処理を図表により示す機能フロー図である。例示的なコマンド文字列1350は、いくつかのコマンド (すなわち、process コマンド1360、where コマンド1362、sort コマンド1364、およびtable コマンド1366) をパイプライン処理する。コマンドライン1350は、こうしたコマンドのいずれにも入力パラメータを渡し得る (たとえば、「handlecount > 400」が、where コマンド1362に渡される)。process コマンド1360は、関連する入力パラメータを何ももたないことに留意されたい。

40

【0112】

従来、各コマンドは、コマンドに関連づけられた入力パラメータの解析、入力パラメータが有効であるかどうかの判定、および入力パラメータが有効でない場合のエラーメッセージの発行を担当した。コマンドは通常、様々なプログラマによって書かれたので、コマンドライン上の入力パラメータ用のシンタクスは、あまり一貫していなかった。さらに、

50

エラーが起きた場合、エラーメッセージは、同じエラーに対してさえも、コマンドの間であまり一貫していなかった。

【0113】

たとえば、UNIX（登録商標）環境では、「ls」コマンドおよび「ps」コマンドの間には多くの不一致がある。両方ともオプション「-w」を受諾するが、「-w」オプションは、「ls」コマンドではページ幅を示すのに使われ、「-w」オプションは、「ps」コマンドでは幅のある出力の印刷を示すのに使われる（本質的に、ページ幅を無視する）。「ls」および「ps」コマンドに関連するヘルプページにもいくつかの不一致があり、たとえば、あるページではオプションをボールド体にしてあるが他のページではそうでなく、あるページではオプションをアルファベット順にソートしてあるが他のページではそうでなく、一部のオプションにはダッシュをつけさせるが一部のオプションにはつけない。

10

【0114】

本管理用ツールフレームワークは、より一貫性がある手法を提供し、各開発者が書かなければならない重複コードの量を最小限にする。管理用ツールフレームワーク200は、開発者が管理用ツールフレームワーク200によって提供される共通機能を容易に利用することを可能にするためのシンタクス（たとえば文法）、対応するセマンティクス（たとえば辞書）、および参照モデルを提供する。

【0115】

本発明をさらに説明する前に、本明細書を通して現れる追加用語の定義が与えられる。入力パラメータは、cmdlet用の入力フィールドを指す。引数は、argvアレイ中のただ1つの文字列の等価物であるcmdletに渡され、またはcmdletオブジェクト中のただ1つの要素として渡される入力パラメータを指す。後で説明されるように、cmdletは、文法を指定する機構を提供する。機構は、直接的にも間接的にも提供され得る。引数は、オプション、オプション引数、またはコマンド名に続くオペランドの1つである。引数の例が、次のコマンドラインに基づいて与えられる。

20

【0116】

【数5】

```
findstr /i /d:\winnt;\winnt\system32 aa*b *.ini.
```

30

【0117】

上記のコマンドラインにおいて、「findstr」は引数0であり、「/i」は引数1であり、「/d:¥winnt;¥winnt¥system32」は引数2であり、「aa*b」は引数3であり、「*.ini」は引数4である。「オプション」は、概してプログラムのデフォルト動作に対する変更を指定するのに使われる、cmdletへの引数である。上のコマンドラインの例で続けると、「/i」および「/d」がオプションである。オプション引数は、特定のオプションに続く入力パラメータである。一部の 경우에는、オプション引数は、オプションと同じ引数文字列に含まれる。それ以外の 경우에는、オプション引数は、次の引数として含まれる。再度上記のコマンドラインを参照すると、「winnt;¥winnt¥system32」がオプション引数である。「オペランド」は、概して、プログラム処理を完了するのに必要な情報をプログラムに供給するオブジェクトとして使われるcmdletへの引数である。オペランドは概して、コマンドライン中でオプションの後に続く。再度上のコマンドラインの例を参照すると、「aa*b」および「*.ini」がオペランドである。「解析可能ストリーム」は、引数を含む。

40

【0118】

図13を参照すると、パーサ220は、解析可能ストリーム（たとえば、コマンド文字列1350）を構成要素1320～1326（たとえば、where部分1322）に解析する。各部分1320～1326は、cmdlet1330～1336の1つに関連づけられる。パーサ220およびエンジン224は、解析、パラメータ認証、データ生成、

50

パラメータ処理、パラメータエンコーディング、およびパラメータドキュメンテーションなど、様々な処理を実施する。パーサ220およびエンジン224は、コマンドライン上の入力パラメータに対して共通機能を実施するので、管理用ツールフレームワーク200は、一貫したエラーメッセージをユーザに発行することができる。

【0119】

本発明による管理用ツールフレームワークに従って書かれた実行可能 `cmdlet` 1330～1336は、従来の管理環境におけるコマンドより少ないコードを必要とすることが認識されよう。各実行可能 `cmdlet` 1330～1336は、それぞれの構成要素1320～1326を使って識別される。さらに、（矢印1340、1342、1344、および1346で表される）各実行可能 `cmdlet` 1330～1336は、オブジェクトを出力し、こうした出力オブジェクトは、（矢印1341、1343、および1345で表される）入力オブジェクトとして次のパイプライン型 `cmdlet` に入力される。こうしたオブジェクトは、オブジェクトに参照（たとえばハンドル）を渡すことによって入力され得る。実行可能 `cmdlet` 1330～1336は次いで、ハンドルを渡されたオブジェクトに対して追加処理を実施することができる。

【0120】

図14は、図9に示される入力を操作するプロセスでの使用に適したコマンド文字列の処理をより詳しく示す論理フロー図である。コマンド文字列処理は、ブロック1401で始まり、ここで、パーサまたはスクリプトエンジンが、入力中のコマンド文字列を識別している。概して、コアエンジンは、`cmdlet` のデータフローに対してセットアップおよび順序付けを実施する。1つの `cmdlet` 用のセットアップおよび順序付けは、後で説明されるが、パイプライン中の各 `cmdlet` に適用可能である。処理は、ブロック1404で継続する。

【0121】

ブロック1404で、`cmdlet` が識別される。`cmdlet` の識別は、登録を介し得る。コアエンジンは、`cmdlet` がローカルであるか、それともリモートであるか判定する。`cmdlet` は、1) 管理用ツールフレームワークのアプリケーションドメイン内部、2) 管理用ツールフレームワークと同じプロセスの別のアプリケーションドメイン内部、3) 同じコンピューティングデバイス上の別のプロセス中、または4) リモートコンピューティングデバイス内部で実行され得る。同じプロセス中で動作している `cmdlet` の間の通信は、オブジェクトを介する。異なるプロセス中で動作している `cmdlet` の間の通信は、直列構造データ形式による。例示的な一直列構造データ形式は、拡張マークアップ言語（XML）に基づく。処理は、ブロック1406で継続する。

【0122】

ブロック1406で、`cmdlet` オブジェクトのインスタンスが作成される。`cmdlet` のインスタンスを作成する例示的なプロセスが、後で図15とともに説明される。`cmdlet` オブジェクトが作成されると、処理は、ブロック1408で継続する。

【0123】

ブロック1408で、`cmdlet` オブジェクトに関連づけられたプロパティが投入される。上述したように、開発者は、`cmdlet` クラスの中、または外部ソースの中でプロパティを宣言する。簡潔には、管理用ツールフレームワークは、プロパティに対して宣言される名称およびタイプに基づいて、受信オブジェクト（群）を解読し、`cmdlet` クラスからインスタンス化された `cmdlet` にする。タイプが異なる場合、タイプは、拡張データ型マネージャを介して強制的に変換され得る。前述されたように、パイプライン型コマンド文字列において、各 `cmdlet` の出力は、オブジェクトへのハンドルのリストとなり得る。次の `cmdlet` は、このオブジェクトハンドルリストを入力し、処理を実施し、オブジェクトハンドルの別のリストを次の `cmdlet` に渡す。さらに、図7に示されるように、入力パラメータは、コマンドラインから届くものとして指定され得る。`cmdlet` に関連するプロパティを投入する例示的な一方法が、後で図16とともに説明される。`cmdlet` が投入を行われると、処理は、ブロック1410で継続する。

【0124】

ブロック1410で、`cmdlet`が実行される。概して、`cmdlet`によって提供される処理は、少なくとも一度実施され、これは、`cmdlet`への各入力オブジェクトの処理を含む。したがって、`cmdlet`がパイプライン型コマンド文字列中の第1の`cmdlet`である場合、処理は一度実行される。後続`cmdlet`については、処理は、`cmdlet`に渡される各オブジェクトごとに実行される。`cmdlet`を実行する例示的な一方法が、後で図5とともに説明される。入力パラメータが、コマンドラインからのみ届いている場合、`cmdlet`の実行は、基本`cmdlet`ケースによって提供されるデフォルトメソッドを用いる。`cmdlet`が実行を終了されると、処理はブロック1412へ進む。

10

【0125】

ブロック1412で、`cmdlet`がクリーンアップされる。クリーンアップは、メモリの割振り解除などを担当する、関連する`cmdlet`オブジェクト用デストラクタ(`destructor`)のコールを含む。コマンド文字列の処理は次いで、完了する。

【0126】

(`cmdlet`オブジェクトを作成する例示的なプロセス)

図15は、図14に示されるコマンド文字列処理での使用に適した、`cmdlet`を作成する例示的なプロセスを示す論理フロー図である。この時点で、`cmdlet`データ構造が開発されており、属性および予想される入力パラメータが指定されている。`cmdlet`は、コンパイルされ登録されている。登録の間、クラス名(すなわち、`cmdlet`名)が登録ストアに書き込まれ、`cmdlet`に関連づけられたメタデータが格納されている。プロセス1500は、ブロック1501で始まり、ここで、パーサは、`cmdlet`を示す入力(たとえばキーストローク)を受け取っている。パーサは、レジストリ中から入力をルックアップし、登録された`cmdlet`の1つに入力を関連づけることによって、入力を`cmdlet`として認識し得る。処理はブロック1504へ進む。

20

【0127】

ブロック1504で、`cmdlet`オブジェクトクラスに関連づけられたメタデータが読み取られる。メタデータは、`cmdlet`に関連づけられたディレクティブのいずれをも含む。ディレクティブは、`cmdlet`自体にも、パラメータの1つまたは複数にも適用し得る。`cmdlet`の登録中、登録コードは、メタデータを永続ストアに登録する。メタデータは、シリアル化形式のXMLファイル、外部データベースなどに格納され得る。スクリプト処理中のディレクティブの処理と同様に、ディレクティブの各カテゴリーは、様々な段階で処理される。各メタデータディレクティブは、独自のエラー操作を扱う。処理は、ブロック1506で継続する。

30

【0128】

ブロック1506で、識別された`cmdlet`クラスに基づいて`cmdlet`オブジェクトがインスタンス化される。処理は、ブロック1508で継続する。

【0129】

ブロック1508で、`cmdlet`についての情報が入手される。情報入手は、リフレクションまたは他の手段を介して起こり得る。情報は、予想される入力パラメータに関するものである。上述したように、パブリック宣言されるパラメータ(たとえば、`public string Name730`)は、コマンドライン上のコマンド文字列中で指定されることも入力ストリーム中で与えられることもできる、予想される入力パラメータに対応する。図18で説明される、拡張タイプマネージャによる管理用ツールフレームワークは、(必要に応じて)コール側に情報を返す共通インターフェイスを提供する。処理は、ブロック1510で継続する。

40

【0130】

ブロック1510で、使用可能性ディレクティブ(たとえば、テーブル1)が適用される。使用可能性ディレクティブは、クラスが特定のマシン役割および/またはユーザ役割において使われることを保証する。たとえば、特定の`cmdlet`は、ドメイン管理者に

50

よってのみ使われ得る。使用可能性ディレクティブの1つにおいて指定された制約が満たされない場合、エラーが起こる。処理は、ブロック1512で継続する。

【0131】

ブロック1512で、メタデータが、インテリセンスを提供するのに使われる。処理のこの時点で、コマンド文字列全体は、まだ入力されていない。しかし、管理用ツールフレームワークは、使用可能な `cmdlet` を知っている。`cmdlet` が判別されると、管理用ツールフレームワークは、`cmdlet` オブジェクトをリフレクトすることによって、許可される入力パラメータを知る。したがって、管理用ツールフレームワークは、`cmdlet` 名のあいまい性排除部分が与えられると `cmdlet` を自動補完し、次いで、入力パラメータのあいまい性排除部分がコマンドライン上でタイプされると入力パラメータを自動補完することができる。自動補完は、入力パラメータの部分が入力パラメータの1つを明確に識別できるとすぐに起こり得る。さらに、自動補完は、`cmdlet` 名およびオペランドに対しても起こり得る。処理は、ブロック1514で継続する。

【0132】

ブロック1514で、プロセスは、`cmdlet` 用の入力パラメータが入力されるまで待機する。この待機は、リターンキーを打つことなどによって、ユーザがコマンド文字列の終わりを示すと起こり得る。スクリプト中では、新規行が、コマンド文字列の終わりを示す。この待機は、パラメータに関する付加情報のユーザからの入手、および他のディレクティブの適用を含み得る。`cmdlet` がパイプライン型パラメータの1つである場合、処理は直ちに始まり得る。必要なコマンド文字列および入力パラメータが与えられると、処理は完了する。

【0133】

(`cmdlet` に投入を行う例示的なプロセス)

`cmdlet` に投入を行う例示的なプロセスが、図16に示され、図5とともに次に説明される。例示的な一管理用ツールフレームワークにおいて、コアエンジンは、`cmdlet` 用パラメータを投入するための処理を実施する。処理は、`cmdlet` のインスタンスが作成された後、ブロック1601で始まる。処理は、ブロック1602へ続く。

【0134】

ブロック1602で、`cmdlet` 内部で宣言されたパラメータ（たとえば、`ProcessName`）が取得される。`cmdlet` を用いた宣言に基づいて、コアエンジンは、受信入力オブジェクトが、「`ProcessName`」という名前のプロパティを提供すると認識する。受信プロパティのタイプが、パラメータ宣言で指定されたタイプと異なる場合、タイプは、拡張タイプマネージャにより強制的に変換される。データ型を強制的に変換するプロセスは、後で「例示的な拡張タイプマネージャ処理」というサブセクションで説明される。処理は、ブロック1603へ続く。

【0135】

ブロック1603で、パラメータに関連づけられた属性が取得される。属性は、パラメータ用の入力ソースがコマンドラインであるかどうか、または入力がパイプラインからであるかどうかを識別する。処理は、判断ブロック1604へ続く。

【0136】

判断ブロック1604で、属性が入力ソースをコマンドラインとして指定しているかどうか、判定が行われる。入力ソースがコマンドラインの場合、処理は、ブロック1609で継続する。コマンドラインでない場合、処理は、判断ブロック1605で継続する。

【0137】

判断ブロック1605で、宣言において指定されたプロパティ名が使われるべきかどうか、またはプロパティ名に対するマッピングが使われるべきかどうか判定が行われる。この判定は、コマンド入力が、パラメータに対してマッピングを指定したかどうかに基づく。次の行は、受信オブジェクトの「`foo`」メンバへの、パラメータ「`ProcessName`」の例示的なマッピングを示す。

```
$ get/process | where han* -gt 500 | stop/process -ProcessName<-foo
```

10

20

30

40

50

【0138】

処理は、ブロック1606で継続する。

ブロック1606で、マッピングが適用される。マッピングは、予想されるパラメータの名称を、「ProcessName」から「foo」に置き換え、置き換えられた後の名称は次いで、コアエンジンによって、受信オブジェクトを解析し、予想される正しいパラメータを識別するのに使われる。処理は、ブロック1608で継続する。

【0139】

ブロック1608で、拡張タイプマネージャは、パラメータ用の値を受信オブジェクト中に配置するために問合せを受ける。拡張タイプマネージャとともに説明したように、拡張タイプマネージャは、パラメータ名を受け取り、リフレクションを用いて、パラメータ名を有する受信オブジェクト中でパラメータを識別する。拡張タイプマネージャは、必要な場合、パラメータに対して他の処理も実施することができる。たとえば、拡張タイプマネージャは、上述された変換機構により、データのタイプを、予想されるデータ型に強制的に変換することができる。処理は、判断ブロック1610へ続く。

10

【0140】

ブロック1609に戻ると、属性が、入力ソースがコマンドラインであると指定している場合、コマンドラインからのデータが入手される。コマンドラインからのデータの入手は、拡張タイプマネージャを介して実施され得る。処理は次いで、判断ブロック1610へ続く。

【0141】

判断ブロック1610で、予想される別のパラメータがあるかどうか、判定が行われる。予想される別のパラメータがある場合、処理は、ブロック1602にループバックし、上述されたように進行する。別のパラメータがない場合、処理は完了し、戻る。

20

【0142】

したがって、説明されたように、cmdletは、受信データを分解して、予想されるパラメータを入手するテンプレートとして作用する。さらに、予想されるパラメータは、予想されるパラメータの値を提供する受信オブジェクトのタイプを知らなくても入手される。これは、従来の管理環境と大きく異なる。従来の管理環境は、密束縛され、オブジェクトのタイプがコンパイル時に知られていることを要求する。さらに、従来の環境では、予想されるパラメータは、値または参照によって関数に渡されている。したがって、本発明による解析（たとえば「分解」）機構は、プログラマが、こうしたパラメータの値がどのようにして取得されるかを具体的に知ることを要せずに、パラメータのタイプを指定することを可能にする。

30

【0143】

たとえば、cmdlet Fooに対して次の宣言が与えられる。

【0144】

【数6】

```
class Foo : Cmdlet
```

```
{
```

```
    string Name;
```

```
    Bool Recurse;
```

```
}
```

40

【0145】

コマンドラインシンタクスは、次のいずれでもよい。

50

【0146】

【数7】

\$ Foo -Name: (string) -Recurse: True

\$ Foo -Name <string> -Recurse True

\$Foo -Name (string).

【0147】

規則の組は、所望のシンタクスを提供するために、システム管理者によって修正され得る。さらに、パーサは、複数のシンタクスがユーザによって使われ得るように、複数の規則の組をサポートすることができる。本質的に、cmdlet 構造（たとえば、string Name および Bool Recurse）に関連づけられた文法が、パーサを駆動する。

10

【0148】

概して、解析ディレクティブは、コマンド文字列として入力されたパラメータが、cmdlet オブジェクト中で識別される、予想されるパラメータにどのようにしてマップすべきかを記述する。入力パラメータタイプは、正しいかどうか判定するために調べられる。入力パラメータタイプが正しくない場合、入力パラメータは、正しくなるように強制的に変換され得る。入力パラメータタイプが正しくなく、強制的に変換されることもできない場合、使用上のエラーが印字される。使用上のエラーは、期待される正しいシンタクスをユーザに気づかせる。使用上のエラーは、ドキュメンテーションディレクティブから、シンタクスを記述する情報を入手することができる。入力パラメータタイプがマッピングされるか、または認証されると、cmdlet オブジェクトインスタンス中の対応するメンバが投入される。メンバが投入されるとき、拡張タイプマネージャは、入力パラメータタイプの処理をもたらす。簡潔には、処理は、プロパティパス機構、キー機構、比較機構、変換機構、グローバル機構、関係機構、およびプロパティセット機構を含み得る。こうした機構はそれぞれ、後で、説明のための例も含む「拡張タイプマネージャ処理」というセクションで詳しく説明される。

20

30

【0149】

(cmdlet を実行する例示的なプロセス)

cmdlet を実行する例示的なプロセスが図17に示され、次に説明される。例示的な一管理用ツール環境において、コアエンジンは、cmdlet を実行する。上述したように、第2のメソッド540内部のコード542が、各入力オブジェクトに対して実行される。処理は、ブロック1701で始まり、ここで、cmdlet は既に投入を受けている。処理は、ブロック1702で継続する。

【0150】

ブロック1702で、コード542にあるステートメントが、実行のために取得される。処理は、判断ブロック1704で継続する。

40

【0151】

判断ブロック1704で、ステートメント中にフックが含まれるかどうか、判定が行われる。一時的に図5に移ると、フックは、コアエンジンによって提供されるAPIのコールを含み得る。たとえば、図5のcmdlet 500のコード542中のステートメント550は、必要なパラメータ、第1の文字列（たとえば、「PID=」）、およびパラメータ（たとえば、PID）を指定するconfirmprocessing APIをコールする。図17に戻ると、ステートメントがフックを含む場合、処理は、ブロック1712へ続く。したがって、confirmprocessing APIをコールする命令が指定された場合、cmdlet は、動作環境によって提供される代替実行モードで動作する。それ以外の場合、処理は、ブロック1706で継続し、実行は、「ノーマル」モ

50

ードで継続する。

【0152】

ブロック1706で、ステートメントが処理される。処理は次いで、判断ブロック1708へ進む。ブロック1708で、コードが別のステートメントを含むかどうか、判定が行われる。別のステートメントがある場合、処理は、ブロック1702にループバックして、次のステートメントを入手し、上述されたように進行する。別のステートメントがない場合、処理は判断ブロック1714へ続く。

【0153】

判断ブロック1714で、処理すべき別の入力オブジェクトがあるかどうか、判定が行われる。別の入力オブジェクトがある場合、処理は、ブロック1716へ続き、ここで、`cmdlet`は、次のオブジェクトからのデータを投入される。図16で説明される投入プロセスが、次のオブジェクトで実施される。処理は次いで、ブロック1702にループバックし、上述されたように進行する。オブジェクトすべてが処理されると、`cmdlet`実行プロセスは完了し、戻る。

10

【0154】

判断ブロック1704に戻ると、ステートメントがフックを含む場合、処理は、ブロック1712へ続く。ブロック1712で、管理用ツール環境によってもたらされる追加特徴が処理される。処理は、判断ブロック1708で継続し、上述されたように継続する。

【0155】

ブロック1712において実施される追加処理が、次に、図6に示される例示的なデータ構造600とともに説明される。上で説明されたように、コマンド基本クラス600中には、予想される追加入力パラメータ（たとえばスイッチ）に対応すると宣言したパラメータが存在し得る。

20

【0156】

スイッチは、所定の文字列を含み、認識されると、`cmdlet`に付加機能を提供するよう、コアエンジンに指図する。パラメータ`verbose610`が、コマンド入力中で指定された場合、`verbose`ステートメント614が実行される。以下は、`verbose`スイッチを含むコマンドラインの例である。

【0157】

【数8】

```
$ get/process | where "han* -gt 500" | stop/process -verbose.
```

30

【0158】

概して、「-verbose」が、コマンド入力中で指定された場合、コアエンジンは、各入力オブジェクトに対してコマンドを実行し、表示のために各入力オブジェクトごとにホストプログラムに対して実行された実際のコマンドを転送する。以下は、上記のコマンドラインが例示的な管理用ツール環境において実行されたときに生成される出力の例である。

【0159】

【数9】

```
$ stop/process PID=15
```

40

```
$ stop/process PID=33.
```

【0160】

パラメータ`whatif620`がコマンド入力中で指定された場合、`whatif`ステートメント624が実行される。以下は、`whatif`スイッチを含むコマンドラインの例である。

【0161】

【数 1 0】

```
$ get/process | where "han* -gt 500" | stop/process --whatif.
```

【0 1 6 2】

概して、「-whatif」が指定されると、コアエンジンは、実際にはコード 5 4 2 を実行するのではなく、表示のためにホストプログラムに対して実行されているはずのコマンドを送る。以下は、本発明の例示的な管理用ツール環境において上記のコマンドラインが実行されたときに生成される出力の例である。

【0 1 6 3】

【数 1 1】

```
#$ stop/process PID=15
```

```
#$ stop/process PID=33.
```

【0 1 6 4】

パラメータ confirm 6 3 0 がコマンド入力中で指定された場合、confirm ステートメント 6 3 4 が実行される。以下は、confirm スイッチを含むコマンドラインの例である。

【0 1 6 5】

【数 1 2】

```
$ get/process | where "han* -gt 500" | stop/process --confirm.
```

【0 1 6 6】

概して、「-confirm」が指定されると、コアエンジンは、コマンドを進めるかどうかに関して、追加ユーザ入力を要求する。以下は、本発明の例示的な管理用ツール環境において上記のコマンドラインが実行されたときに生成される出力の例である。

【0 1 6 7】

【数 1 3】

```
$ stop/process PID 15
```

```
Y/N Y
```

```
$ stop/process PID 33
```

```
Y/N N.
```

【0 1 6 8】

上述したように、例示的なデータ構造 6 0 0 は、実行に対して要求されているタスクが許可されるべきかどうか判定するセキュリティメソッド 6 4 0 も含み得る。従来の管理環境では、各コマンドは、コマンドを実行している人がコマンドを実施するのに十分な特権を有しているかどうかのチェックを担当する。このチェックを実施するために、いくつかのソースからの情報にアクセスするのに、拡張コードが必要とされる。こうした複雑さのせいで、多くのコマンドは、セキュリティチェックを実施しなかった。本管理用ツール環境の発明者は、タスクがコマンド入力中で指定されると、セキュリティチェックの実施に必要な情報が管理用ツール環境において利用可能であると認識した。したがって、管理用ツールフレームワークは、ツール開発者に対して複雑なコードを要求せずにセキュリティチェックを実施する。セキュリティチェックは、cmdlet 中でフックを定義するなどの cmdlet に対しても実施され得る。あるいは、フックは、上述された verbose パラメータと同様に、コマンド入力中で指定され得る任意選択の入力パラメータでもよい。

【0 1 6 9】

10

20

30

40

50

セキュリティチェックは、概してユーザの役割に基づいて、どのユーザがリソースへのアクセス権をもつかをコントロールするシステムとして定義される、役割ベースの認証をサポートするように実装される。したがって、各役割は、異なるリソースへの特定のアクセス権を割り当てられる。ユーザは次いで、1つまたは複数の役割に割り当てられる。概して、役割ベースの認証は、3つの項目、すなわち原理、リソース、およびアクションに注目する。リソースに対してアクションが実施されるよう要求したのは誰かを、原理が識別する。

【0170】

本発明の発明者は、要求されている `cmdlet` が、実施されるべきアクションに対応することを認識した。さらに、本発明者は、管理用ツールフレームワークが実行されているプロセスの所有者が、原理に対応することを認識した。さらに、本発明者は、リソースが `cmdlet` 中で指定されることを認識した。したがって、管理用ツールフレームワークは、こうした項目へのアクセス権をもっているので、本発明者は、セキュリティチェックは、ツール開発者がセキュリティチェックを実装することを要せずに管理用ツールフレームワーク内部から実施され得ることを認識した。

10

【0171】

セキュリティチェックの動作は、`confirmprocessing API`などのフックを用いることによって、`cmdlet` 内部で付加機能が要求されるといつでも実施され得る。あるいは、セキュリティチェックは、`verbose`、`whatif`、および `confirm`と同様にコマンドライン上でセキュリティスイッチが入れられたかどうか調べることによって、実施され得る。いずれの実装の場合でも、`checkSecurity`メソッドは、誰が許可されるかを判定する1組のAPIを提供するセキュリティプロセス（図示せず）によって提供されるAPIをコールする。セキュリティプロセスは、管理用ツールフレームワークによって提供される情報を受け取り、タスクが完了され得るかどうかを示す結果をもたらす。管理用ツールフレームワークは次いで、エラーをもたらしても、単にタスクの実行を停止してもよい。

20

【0172】

したがって、`cmdlet` 中でフックを提供することによって、開発者は、管理用ツールフレームワークによって提供される追加処理を使うことができる。

【0173】

30

（例示的な拡張タイプマネージャ処理）

図18とともに上で簡潔に述べられたように、拡張タイプマネージャは、供給されるオブジェクトに対して追加処理を実施することができる。追加処理は、パーサ220、スクリプトエンジン222、またはパイプラインプロセッサ402からの要求時に実施され得る。追加処理は、プロパティパス機構、キー機構、比較機構、変換機構、グローバル機構、関係機構、およびプロパティセット機構を含む。拡張タイプマネージャは、特許請求の範囲に記載されている本発明の範囲から逸脱することなく、他の処理を用いても拡張され得ることを当業者は理解するであろう。追加処理機構がそれぞれ、次に説明される。

【0174】

最初に、プロパティパス機構が、文字列に、オブジェクトのプロパティをナビゲートさせる。現在のリフレクションシステムでは、クエリは、オブジェクトのプロパティを照会することができる。しかし、本発明による拡張タイプマネージャでは、オブジェクトの一連のプロパティへのナビゲーションパスを提供する文字列が指定され得る。プロパティパスの例示的なシンタクスは、`P1.P2.P3.P4`である。

40

【0175】

各コンポーネント（たとえば、`P1`、`P2`、`P3`、および`P4`）は、プロパティ、パラメータを有するメソッド、パラメータをもたないメソッド、フィールド、`XPATH`などを表し得る文字列を含む。`XPATH`は、要素（たとえば、「`/FOO@=13`」）を検索するためのクエリ文字列を指定する。文字列には、コンポーネントのタイプを具体的に示すための特殊文字が含まれ得る。文字列が特殊文字を含まない場合、拡張タイプマネー

50

ジャは、コンポーネントのタイプを判定するためのルックアップを実施し得る。たとえば、コンポーネント P 1 がオブジェクトの場合、拡張タイプマネージャは、P 2 が、オブジェクトのプロパティ、オブジェクトのメソッド、オブジェクトのフィールド、またはプロパティセットであるか照会し得る。拡張タイプマネージャが、P 2 のタイプを識別すると、そのタイプに従った処理が実施される。コンポーネントが、上記のタイプの 1 つでない場合、拡張タイプマネージャは、拡張ソースをさらに照会して、P 1 タイプを P 2 タイプに変換するための変換関数が存在するかどうか判定することができる。こうしたおよび他のルックアップが、例示的なコマンド文字列を使い、それぞれの出力を示して次に説明される。

【0176】

以下は、プロパティパスを含む例示的な文字列である。

```
$ get/process | /where hand* -gt 500 | format/table name.toupper, ws.kb, exe*.ver*.description.tolower.trunc(30)
```

【0177】

上記の例示的な文字列中には、3つのプロパティパス、すなわち (1) 「name.toupper」、(2) 「ws.kb」、および (3) 「exe*.ver*.description.tolower.trunc(30)」がある。こうしたプロパティパスについて述べる前に、「name」、「ws」、および「exe」は、テーブルのプロパティを指定することに留意すべきである。さらに、こうしたプロパティはそれぞれ、最初に「get/process」によって生成され、次いで、様々なcmdletを介してパイプライン処理される受信オブジェクトの直接プロパティであることに留意すべきである。3つのプロパティパスそれぞれに関わる処理が次に説明される。

【0178】

第1のプロパティパス（すなわち、「name.toupper」）において、nameは、受信オブジェクトの直接プロパティであり、オブジェクト自体でもある。拡張タイプマネージャは、上述された優先度ルックアップを用いてシステムを照会して、toupperのタイプを判定する。拡張タイプマネージャは、toupperがプロパティでないことを発見する。ただし、toupperは、文字列タイプによって継承される、文字列中で小文字を大文字に変換するためのメソッドであり得る。あるいは、拡張タイプマネージャは、名前オブジェクトを大文字に変換し得るサードパーティコードが存在するかどうか判定するために、拡張メタデータを照会している可能性もある。コンポーネントタイプがわかると、そのコンポーネントタイプに従って処理が実施される。

【0179】

第2のプロパティパス（すなわち、「ws.kb」）において、「ws」は、受信オブジェクトの直接プロパティであり、オブジェクト自体でもある。拡張タイプマネージャは、「ws」が整数であると判定する。次いで、拡張タイプマネージャは、kbが整数プロパティであるかどうか、kbが整数のメソッドであるかどうかを照会し、最後に、どのようにして整数を受け取り、整数をkbタイプに変換するかをいずれかのコードが知っているかを照会する。サードパーティのコードが、この変換を実施するために登録され、変換が実施される。

【0180】

第3のプロパティパス（すなわち、「exe*.ver*.description.tolower.trunc(30)」）には、いくつかのコンポーネントがある。第1のコンポーネント（「exe*」）は、受信オブジェクトの直接プロパティであり、オブジェクトでもある。やはり、拡張タイプマネージャは、第2のコンポーネント（「ver*」）を処理するためにルックアップクエリを進む。「exe*」オブジェクトは、「ver*」プロパティもメソッドも持たないので、拡張タイプマネージャは、拡張メタデータを照会して、実行ファイル名をバージョンに変換するために登録されているコードがあるかどうか、判定する。この例では、このようなコードが存在する。このコードは、実行ファイル名文字列を受け取り、ファイルを開くのに使うことができ、次いで、バージョンブ

10

20

30

40

50

ロックオブジェクトにアクセスし、バージョンブロックオブジェクトの記述プロパティ（第3のコンポーネント（「description」）を返す。拡張タイプマネージャは次いで、第4のコンポーネント（「tolower」）および第5のコンポーネント（「trunc(30)」）に対して、この同じルックアップ機構を実施する。したがって、説明されたように、拡張タイプマネージャは、管理者が具体的なコードを何も書く必要なく、コマンド文字列に対して極めて入念な処理を実施することができる。テーブル1は、例示的な文字列に対して生成される出力を示す。

【0181】

【表6】

Name.toupper ws.kb exe*.ver*.description.tolower.trunc(30)

ETCLIENT	29,964 etclient
CSRSS	6,944
SVCHOST	28,944 generic host process for win32
OUTLOOK	18,556 office outlook
MSMSGs	13,248 messenger

テーブル 1

【0182】

別のクエリ機構1824は、キーを含む。キーは、データ型のインスタンスを一義的にする1つまたは複数のプロパティを識別する。たとえば、データベース中で、ある列が、各行を一義的に識別し得るキー（たとえば、社会保障番号）として識別され得る。キーは、データ型に関連づけられたタイプメタデータ1840に格納される。このキーは次いで、そのデータ型のオブジェクトを処理するときに拡張タイプマネージャによって使われ得る。データ型は、拡張データ型でも、既存のデータ型でもよい。

【0183】

別のクエリ機構1824は、比較機構を含む。比較機構は、2つのオブジェクトを比較する。2つのオブジェクトが、比較関数を直接サポートする場合、直接サポートされる比較関数が実行される。しかし、どちらのオブジェクトも比較関数をサポートしない場合、拡張タイプマネージャは、タイプメタデータの中で、2つのオブジェクトの比較をサポートするために登録されているコードを探し得る。比較機構を伴う例示的な一連のコマンドライン文字列が、テーブル2の中の対応する出力とともに下に示されている。

【0184】

10

20

30

【表 7】

\$ \$a = \$(get/date)

\$ start/sleep 5

\$ \$b = \$(get/date

compare/time \$a \$b

Ticks : 51196579

Days : 0

Hours : 0

Milliseconds : 119

Minutes : 0

Seconds : 5

TotalDays : 5.92552997685185E-05

TotalHours : 0.00142212719444444

TotalMilliseconds : 5119.6579

TotalMinutes : 0.0853276316666667

TotalSeconds : 5.1196579

10

20

テーブル 2

【0185】

`compare/time cmdlet` は、2つの `datetime` オブジェクトを比較するために書かれる。この場合、`Date Time` オブジェクトは、`IComparable` インターフェイスをサポートする。 30

【0186】

別のクエリ機構 1824 は、変換機構を含む。拡張タイプマネージャは、具体的な変換を実施する機能を言明するコードを登録させる。次いで、タイプ A のオブジェクトが入力されるとともに `cmdlet` がタイプ B のオブジェクトを指定すると、拡張タイプマネージャは、登録された変換の 1 つを用いて変換を実施し得る。拡張タイプマネージャは、タイプ A をタイプ B に強制的に変換するために、一連の変換を実施し得る。上述されたプロパティパス (「`ws.kb`」) は、変換機構を示す。 40

【0187】

別のクエリ機構 1824 は、グローバル機構を含む。グローバルは、文字列中のワイルドカード文字を指す。グローバル機構は、ワイルドカード文字を有する文字列を入力し、1 組のオブジェクトを生じる。拡張タイプマネージャは、ワイルドカード処理を指定するコードを登録させる。上述されたプロパティパス (「`exe*.ver*.description.tolower.trunc(30)`」) は、グローバル機構を示す。登録されたプロセスは、ファイル名、ファイルオブジェクト、受信プロパティなどに対してグローピングを提供し得る。 40

【0188】

別のクエリ機構 1824 は、プロパティセット機構を含む。プロパティセット機構は、1 組のプロパティ用の名称を定義させる。管理者は次いで、コマンド文字列中で名称を指 50

定して、プロパティの組を入手することができる。プロパティセットは、様々な方式で定義され得る。一方式では、「?」など、予め定義されたパラメータが、cmdlet用の入力パラメータとして入力され得る。動作環境は、予め定義されたパラメータを認識すると、受信オブジェクトのプロパティすべてを列挙する。リストは、管理者が、望まれるプロパティを容易にチェックする（たとえば、「クリックする」とともにプロパティセットに名前をつけることを可能にするGUIでよい。プロパティセット情報は次いで、拡張メタデータに格納される。プロパティセット機構を呼び出す例示的な文字列が、テーブル3の対応する出力とともに下に示されている。

【0189】

【数14】

10

\$ get/process | where han* -gt 500 | format/table config.

【0190】

この例示的な文字列では、「config」というプロパティセットが、名称プロパティ、プロセスidプロパティ(Pid)、および優先度プロパティを含むように定義されている。このテーブル用の出力が次に示される。

【0191】

【表8】

名称	Pid 優先度
ETClient	3528 Normal
csrss	528 Normal
svchost	848 Normal
OUTLOOK	2,772 Normal
msmsgs	2,584 Normal

20

テーブル 3

30

【0192】

別のクエリ機構1824は、関係機構を含む。1つの関係（すなわち、継承）をサポートする従来型システムとは対照的に、関係機構は、タイプの間の複数の関係の表現をサポートする。やはり、こうした関係も登録される。関係は、オブジェクトが使用する項目を見つけることも、オブジェクトを使用する項目を見つけることも含み得る。拡張タイプマネージャは、様々な関係を記述する存在論（オントロジー）にアクセスし得る。拡張メタデータおよびコードを用いて、どのオントロジーサービスにも、たとえばOWL、DAWLなどにアクセスする仕様が記述され得る。OWL: "string" が、関係機構を使用する例示的な文字列の一部である。

40

【0193】

「OWL」識別子は、オントロジーサービスを識別し、「string」は、オントロジーサービスにおける具体的な文字列を指定する。したがって、拡張タイプマネージャは、オントロジーサービスによって供給されるタイプにアクセスし得る。

【0194】

コマンドラインデータを表示する例示的なプロセス

本機構は、データ駆動コマンドライン出力を提供する。データのフォーマット化および出力は、cmdletからなるパイプライン中の1つまたは複数のcmdletによって実現される。通常、こうしたcmdletは、上で図2とともに説明された非管理cmdletに含まれる。cmdletは、フォーマットcmdlet、マークアップcmdlet

50

e t、変換cmdlet、変形cmdlet、およびアウトcmdletを含み得る。

【0195】

図19は、パイプライン中のこうしたcmdletの例示的なシーケンス1901~1907を図表により示す。第1のシーケンス1901は、パイプライン中の最後のcmdletとしてアウトcmdlet1910を示す。他のcmdletに対して上述されたのと同じやり方で、アウトcmdlet1910は、パイプライン中の他のcmdletによって生成され処理されたパイプラインオブジェクトのストリームを受諾する。ただし、ほとんどのcmdletとは対照的に、アウトcmdlet1910は、他のcmdlet用にパイプラインオブジェクトを送出しない。そうではなく、アウトcmdlet1910は、パイプラインによって生成された結果のレンダリング／表示を担当する。各アウトcmdlet1910は、デバイス、プログラムなど、出力の宛先に関連づけられる。たとえば、コンソールデバイスに対して、アウトcmdlet1910は、out/consoleとして指定されることができ、インターネットブラウザに対して、アウトcmdlet1910は、out/browserとして指定されることができ、ウィンドウに対して、アウトcmdlet1910は、out/windowとして指定されることができ、具体的な各アウトcmdletは、それに関連づけられた宛先の機能を熟知している。変換cmdletがパイプライン中でアウトcmdletに先行しない限り、ロケール情報（たとえば、日付および通貨フォーマット）がアウトcmdlet1910によって処理される。この状況において、変換cmdletは、ロケール情報を処理した。

【0196】

各ホストは、out/consoleなど、特定のアウトcmdletのサポートを担当する。ホストは、どの宛先固有ホストcmdlet（たとえば、表計算アプリケーションによって提供されるチャートに出力を向けるout/chart）もサポートする。さらに、ホストは、結果のデフォルト操作の提供を担当する。このシーケンス中のアウトcmdletは、他の出力処理cmdlet（形式／マークアップ／変換／変形など）をコールすることによって、その行動の実装を決定し得る。したがって、アウトcmdletは、シーケンス1901を他のシーケンスのいずれにも暗黙的に変更することもでき、それ自体の追加形式／出力cmdletを追加することもできる。

【0197】

第2のシーケンス1902は、アウトcmdlet1910の前にフォーマットcmdlet1920を示す。このシーケンスに対して、フォーマットcmdlet1920は、パイプライン中の他のcmdletによって生成され処理されたパイプラインオブジェクトのストリームを受諾する。概して、フォーマットcmdlet1920は、表示プロパティを選択するための方式と、ページレイアウト、たとえば形状、列の幅、ヘッダー、フッターなどを指定するための方式とを提供する。形状は、テーブル、ワイドリスト、縦書きリストなどを含み得る。さらに、フォーマットcmdlet1920は、総計または合計の計算を含み得る。フォーマットcmdlet1920によって実施される例示的な処理が、後で図20とともに説明される。簡潔には、フォーマットcmdletは、パイプラインオブジェクトの送に加えて、フォーマットオブジェクトを送出する。フォーマットオブジェクトは、拡張タイプマネージャまたは他の機構を介してアウトcmdlet（たとえば、シーケンス1902中のアウトcmdlet1920）によって、ダウンストリームと認識され得る。アウトcmdlet1920は、発せられるフォーマットオブジェクトの使用を選ぶこともでき、こうしたオブジェクトの無視を選ぶこともできる。アウトcmdletは、表示情報において指定されたページレイアウトデータに基づいて、ページレイアウトを決定する。特定の例において、ページレイアウトに対する修正は、アウトcmdletによって指定され得る。例示的な一プロセスにおいて、アウトcmdletは、所定のいくつかのオブジェクトの各プロパティに対する最大長（たとえば50）を見つけることによって未指定の列幅を判定し、列幅を最大長に設定することができる。フォーマットオブジェクトは、フォーマット情報、ヘッダー／フッター情報などを含む。

【0198】

10

20

30

40

50

第3のシーケンス1903は、アウトcmdlet 1910の前にフォーマットcmdlet 1920を示す。ただし、第3のシーケンス1903において、マークアップcmdlet 1930が、フォーマットcmdlet 1920とアウトcmdlet 1910の間にパイプライン処理される。マークアップcmdlet 1930は、選択されたパラメータにプロパティ注釈（たとえば、フォント、色）を追加する機構を提供する。したがって、マークアップcmdlet 1930は、出力cmdlet 1910の前に現れる。プロパティ注釈は、「プロパティバッグに影をつける」を用いて、またはプロパティバッグ中のカスタム名前空間中のプロパティ注釈を追加することによって実装され得る。マークアップcmdlet 1930は、フォーマットcmdlet 1920の処理の間、マークアップ注釈が維持され得る限り、フォーマットcmdlet 1920の前に現れ得る。 10

【0199】

第4のシーケンス1904もやはり、アウトcmdlet 1910の前にフォーマットcmdlet 1920を示す。ただし、第4のシーケンス1904において、変換cmdlet 1940が、フォーマットcmdlet 1920とアウトcmdlet 1910の間にパイプライン処理される。変換cmdlet 1940も、フォーマットcmdlet 1920によって発せられるフォーマットオブジェクトを処理するように構成される。変換cmdlet 1940は、フォーマットオブジェクトに基づいて、パイプラインオブジェクトを固有エンコーディングに変換する。変換cmdlet 1940は、固有エンコーディングに関連づけられる。たとえば、パイプラインオブジェクトをアクティブディレクトリオブジェクト（ADO）に変換する変換cmdlet 1940は、コマンドライン上で「convert/ADO」として宣言され得る。同様に、パイプラインオブジェクトを、コンマで区切られた値（comma separate value、csv）に変換する変換cmdlet 1940は、コマンドライン上で「convert/csv」として宣言され得る。変換cmdlet 1940のいくつか（たとえば、convert/XMLおよびconvert/html）はブロッキングコマンドでよく、すなわち、変換を実行する前にパイプラインオブジェクトすべてが受け取られる。通常、アウトcmdlet 1920は、フォーマットオブジェクトによって提供されるフォーマット情報を使うかどうか、判定することができる。ただし、変換cmdlet 1920がアウトcmdlet 1920の前に現れる場合、アウトcmdletがオブジェクトを受け取る前に実際のデータ変換が既に起こっている。したがって、この状況において、アウトcmdletは、変換を無視することができない。 20 30

【0200】

第5のシーケンス1905は、フォーマットcmdlet 1920、マークアップcmdlet 1930、変換cmdlet 1940、およびアウトcmdlet 1910をこの順序で示す。したがって、この順序は、マークアップcmdlet 1930が変換cmdlet 1940の前に現れ得ることを示す。

【0201】

第6のシーケンス1906は、フォーマットcmdlet 1920、固有の変換cmdlet（たとえば、convert/xml cmdlet 1940'）、固有の変形cmdlet（たとえば、transform/xslt cmdlet 1950）、およびアウトcmdlet 1910を示す。convert/XML cmdlet 1940'は、パイプラインオブジェクトを、拡張マークアップ言語（XML）に変換する。transform/xslt cmdlet 1950は、拡張スタイル言語（XSL）スタイルシートを使って、XMLドキュメントを別のXMLドキュメントに変形する。変形プロセスは一般に、拡張スタイル言語変換（XSLT）と呼ばれ、XSLTにおいて、XSLプロセッサは、XMLドキュメントを読み、XSLスタイルシート中の命令に従って、新規XMLドキュメントを作成する。 40

【0202】

第7のシーケンス1907は、フォーマットcmdlet 1920、マークアップcmdlet 1930、固有の変換cmdlet（たとえば、convert/xml cm 50

d l e t 1 9 4 0 ')、固有の変形 c m d l e t (たとえば、t r a n s f o r m / x s l t c m d l e t 1 9 5 0)、およびアウト c m d l e t 1 9 1 0を示す。したがって、第7のシーケンス 1 9 0 7は、変換 c m d l e t および変形 c m d l e t の上流にあるマークアップ c m d l e t 1 9 3 0を有することを示す。

【0203】

図20は、フォーマット c m d l e t によって実施される例示的な処理 2 0 0 0を示す。フォーマットプロセスは、フォーマット c m d l e t が、上述されたやり方でパーサおよびパイプラインプロセッサによって解析され呼び出された後、ブロック 2 0 0 1で始まる。処理は、ブロック 2 0 0 2で継続する。

【0204】

ブロック 2 0 0 2で、パイプラインオブジェクトが、フォーマット c m d l e t への入力として受け取られる。処理は、ブロック 2 0 0 4で継続する。

【0205】

ブロック 2 0 0 4で、パイプラインオブジェクトのタイプを識別するためのクエリが開始される。このクエリは、図18とともに上述されたように、拡張タイプマネージャによって実施される。拡張タイプマネージャがオブジェクトのタイプを識別すると、処理は、ブロック 2 0 0 6で継続する。

【0206】

ブロック 2 0 0 6で、識別されたタイプが表示情報中でルックアップされる。表示情報の例示的な形式は、図21に示され、後で説明される。処理は、判断ブロック 2 0 0 8で継続する。

【0207】

判断ブロック 2 0 0 8で、識別されたタイプが、表示情報中で指定されているかどうか、判定が行われる。表示情報中に識別されたタイプのエントリがない場合、処理は完了する。エントリがある場合、処理は、ブロック 2 0 1 0で継続する。

【0208】

ブロック 2 0 1 0で、識別されたタイプに関連づけられたフォーマット情報が、表示情報から入手される。処理は、ブロック 2 0 1 2で継続する。

【0209】

ブロック 2 0 1 2で、パイプライン上で情報が発せられる。情報が発せられると、処理は完了する。

【0210】

発せられ得る例示的な情報が、次にさらに詳しく説明される。情報は、フォーマット情報、ヘッダー／フッター情報、およびグループ終了／開始信号オブジェクトを含み得る。フォーマット情報は、形状、ラベル、番号づけ／点打ち、列の幅、キャラクタエンコードタイプ、内容フォントプロパティ、ページの長さ、プロパティ名によるグループ化などを含み得る。こうした情報はそれぞれ、関連づけられた追加仕様を有し得る。たとえば、形状は、形状が、テーブル、リストなどであるかを指定し得る。ラベルは、列ヘッダー、リストラベルなどを使うかどうかを指定し得る。キャラクタエンコードは、アスキー、UTF-8、Unicodeなどを指定し得る。内容フォントプロパティは、表示されるプロパティ値に適用されるフォントを指定し得る。デフォルトフォントプロパティ(たとえば、Courier New、10ポイント)は、内容フォントプロパティが指定されていない場合に使われ得る。

【0211】

ヘッダー／フッター情報は、ヘッダー／フッタースコープ、フォントプロパティ、タイトル、サブタイトル、日付、時刻、ページ番号、分離記号などを含み得る。たとえば、スコープは、ドキュメント、ページ、グループなどを指定し得る。追加プロパティは、ヘッダーまたはフッターのどちらかに対して指定され得る。たとえば、グループおよびドキュメントのフッター用に、追加プロパティは、合計／総計を計算するためのプロパティまたは列、オブジェクトカウント、総計およびカウント用ラベル文字列などを含み得る。

10

20

30

40

50

【0212】

グループ終了／開始信号オブジェクトは、フォーマット `cmdlet` が、グループ化プロパティ (`group-by property`) が変わったことを検出したときに発せられる。これが起こると、フォーマット `cmdlet` は、パイプラインオブジェクトのストリームを、予めソートされたものとして扱い、ソートし直さない。グループ終了／開始信号オブジェクトは、パイプラインオブジェクトとともに点在され得る。ネスト式ソート用に複数のグループ化プロパティが指定され得る。フォーマット `cmdlet` は、最終合計および総計を含む形式終了オブジェクトを送出することもできる。

【0213】

手短に図 21 に移ると、例示的な表示情報 2100 は、構造化形式であり、定義されている各オブジェクトに関連づけられた情報（たとえば、フォーマット情報、ヘッダー／フッター情報、グループ化プロパティまたはメソッド）を含む。たとえば、表示情報 2100 は、XML ベースでよい。上述されたプロパティはそれぞれ、次いで、表示情報中で指定され得る。表示情報 2100 中の情報は、入力されているオブジェクトタイプの所有者によって投入され得る。動作環境は、所有者が、エントリを作成し、消去し、修正することによって表示情報をアップデートすることを可能にする特定の API および `cmdlet` を提供する。

【0214】

図 22 は、特定のフォーマット `cmdlet`（たとえば、`format / table`、`format / list`、および `format / wide`）、マークアップ `cmdlet`（たとえば、`add / markup`）、変換 `cmdlet`（たとえば、`convert / text`、`convert / sv`、`convert / csv`、`convert / ADO`、`convert / XML`、`convert / html`）、変形 `cmdlet`（たとえば、`transform / XSLT`）、ならびにアウト `cmdlet`（たとえば、`out / console`、`out / file`）用の例示的なシンタクス 2201～2213 を列挙するテーブルである。図 23 は、出力処理 `cmdlet`（たとえば、フォーマット `cmdlet`、変換 `cmdlet`、およびマークアップ `cmdlet`）の様々なパイプラインシーケンスを用いて、`out / console cmdlet` によってレンダリングされる結果を示す。

【0215】

説明されたように、コマンドライン命令に拡張機能を提供する機構は、管理用ツール環境において利用され得る。しかし、この機構は、コマンドライン命令を入力する様々な環境で利用され得ることを当業者は理解するであろう。たとえば、「`what if`」機能は、「`what if`」パラメータ用のコマンドラインを解析するとともにシミュレーションモード処理を実施するのに必要な命令を挿入することによって、スタンドアロン型コマンドに組み込まれ得る。コマンドライン命令に拡張機能を提供する本機構は、従来の機能拡張機構とは大きく異なる。たとえば、従来の機構では、拡張機能を望む各コマンドが、コマンドにコードを組み込まなければならなかった。次いで、コマンド自体が、コマンド文字列を解析して、スイッチ（たとえば、`verbose`、`what if`）が提供されるかどうか判定し、それに従って拡張機能を実行しなければならなかった。対照的に本機構は、`cmdlet` が拡張機能にフックを組み込む限り、ある特定の `cmdlet` 用の拡張機能を実行するためにユーザがコマンド文字列中で引数を指定することを可能にする。したがって、本機構は、システム管理者が書く必要があるコードの量を最小限にする。さらに、本機構を用いることによって、統一されたやり方で拡張機能が実装される。

【0216】

具体的な実装形態および実施形態の詳細が上述されたが、このような詳細は、添付の特許請求の範囲を限定することではなく、法定の開示義務を満たすことを意図する。したがって、特許請求の範囲によって定義される本発明は、上述された具体的な特徴に限定されない。そうではなく、本発明は、適正な添付の特許請求の範囲内である形または修正形態のいずれにおいても権利請求され、均等論の原則に従って適切に解釈される。

10

20

30

40

50

【図面の簡単な説明】

【0217】

【図1】例示的な管理用ツール環境を使用し得る、例示的なコンピューティングデバイスを示す図である。

【図2】本管理用ツール環境のための例示的な管理用ツールフレームワークの概観を全体的に示すブロック図である。

【図3】図2に示される管理用ツールフレームワークのホスト固有コンポーネント内部のコンポーネントを示すブロック図である。

【図4】図2に示される管理用ツールフレームワークのコアエンジンコンポーネント内部のコンポーネントを示すブロック図である。

【図5】図2に示される管理用ツールフレームワークでの使用に適した `cmdlet` を指定する例示的なデータ構造を示す図である。

【図6】図5に示される `cmdlet` が派生されるコマンド基本タイプを指定する例示的なデータ構造を示す図である。

【図7】図2に示される管理用ツールフレームワークでの使用に適した `cmdlet` を指定する別の例示的なデータ構造を示す図である。

【図8】図2に示される管理用ツールフレームワーク内部で実施されるホスト処理のための例示的なプロセスを示す論理フロー図である。

【図9】図2に示される管理用ツールフレームワーク内部で実施される入力操作のための例示的なプロセスを示す論理フロー図である。

【図10】図9に示される入力操作プロセスでの使用に適したスクリプトを処理するプロセスを示す論理フロー図である。

【図11】図10に示されるスクリプト処理プロセスでの使用に適したスクリプト前処理プロセスを示す論理フロー図である。

【図12】図10に示されるスクリプト処理プロセスでの使用に適した、制約を課すプロセスを示す論理フロー図である。

【図13】図2に示される管理用ツールフレームワークにおけるコマンド文字列の処理を示す機能フロー図である。

【図14】図9に示される入力を操作するプロセスでの使用に適した、コマンド文字列を処理するプロセスを示す論理フロー図である。

【図15】図14に示されるコマンド文字列処理での使用に適した、`cmdlet` のインスタンスを作成する例示的なプロセスを示す論理フロー図である。

【図16】図14に示されるコマンド処理での使用に適した、`cmdlet` のプロパティに投入を行う例示的なプロセスを示す論理フロー図である。

【図17】図14に示されるコマンド処理での使用に適した、`cmdlet` を実行する例示的なプロセスを示す論理フロー図である。

【図18】図2に示される管理用ツールフレームワークでの使用に適した例示的な拡張タイプマネージャを示す機能ブロック図である。

【図19】パイプライン中の出力処理 `cmdlet` の例示的なシーケンスを示す図である。

【図20】図19に示される出力処理 `cmdlet` の1つによって実施される例示的な処理を示す図である。

【図21】図20の処理中にアクセスされる表示情報の例示的な構造を示す図である。

【図22】例示的な出力処理 `cmdlet` 用の例示的なシンタックスを列挙するテーブルを示す図である。

【図23】出力処理 `cmdlet` の様々なパイプラインシーケンスを用いて、`out/console cmdlet` によってレンダリングされる結果を示す図である。

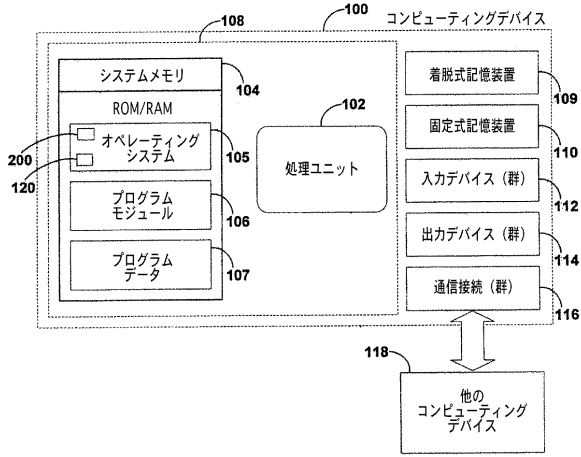
10

20

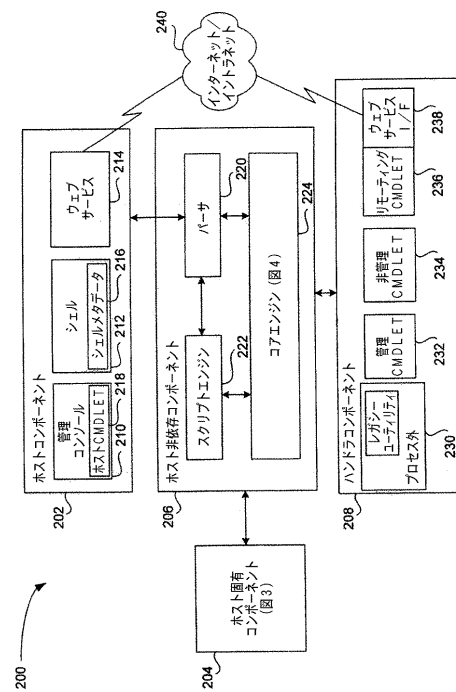
30

40

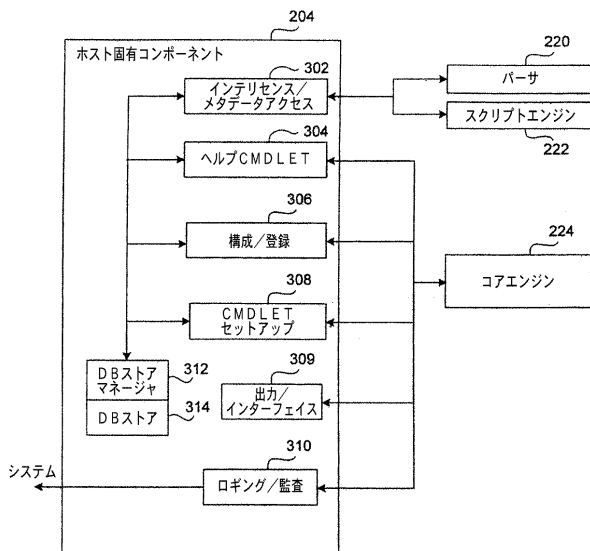
【図 1】



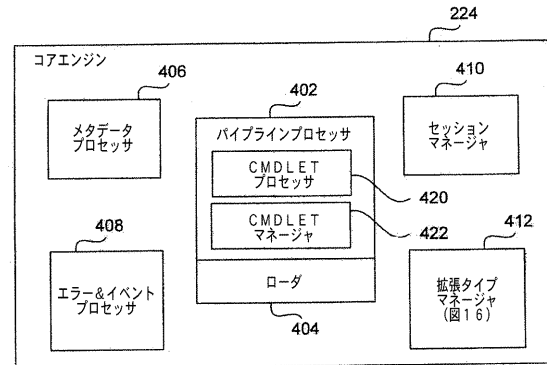
【図 2】



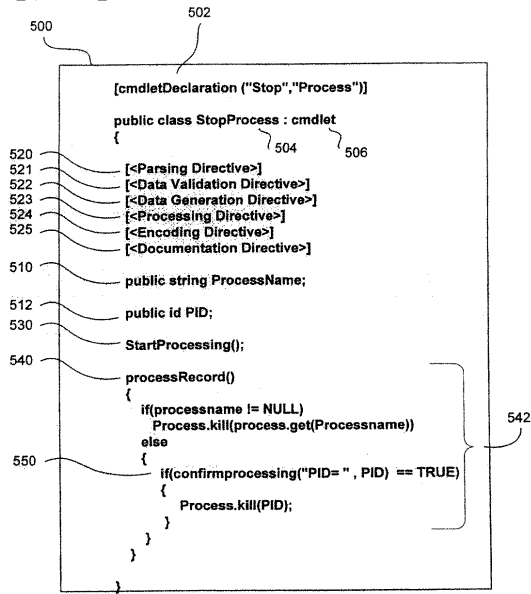
【図 3】



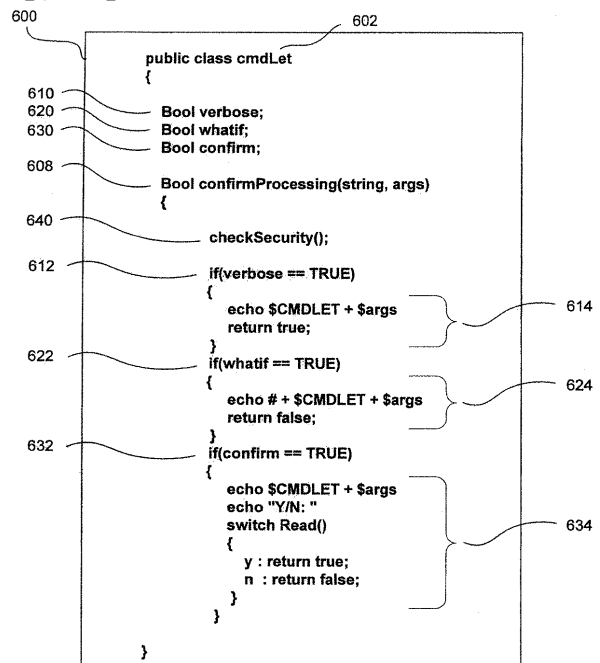
【図 4】



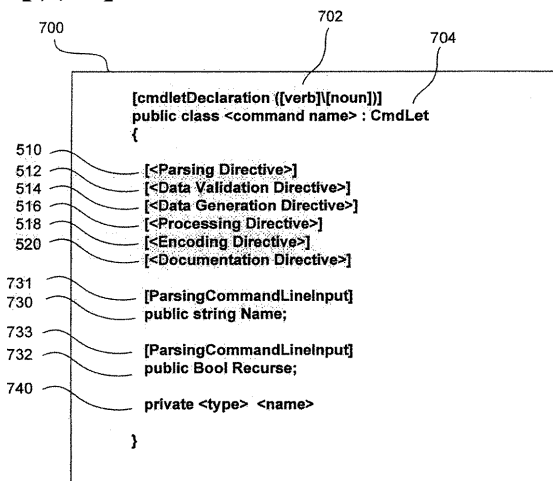
【図 5】



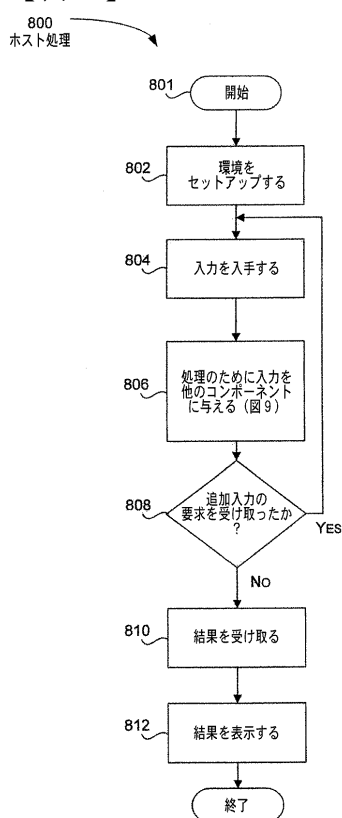
【図 6】



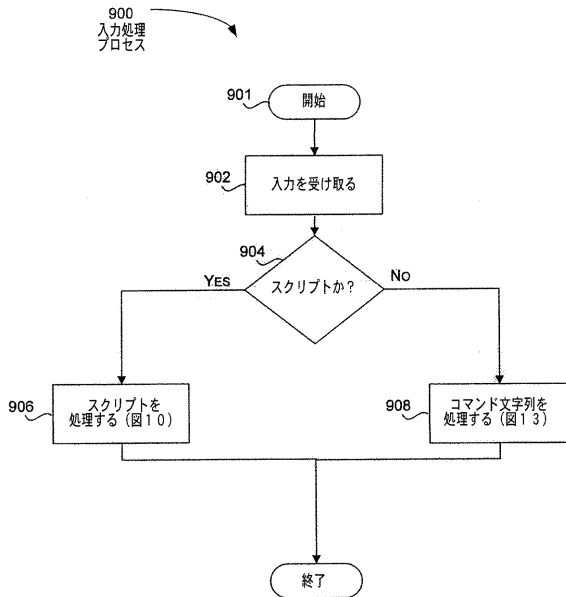
【図 7】



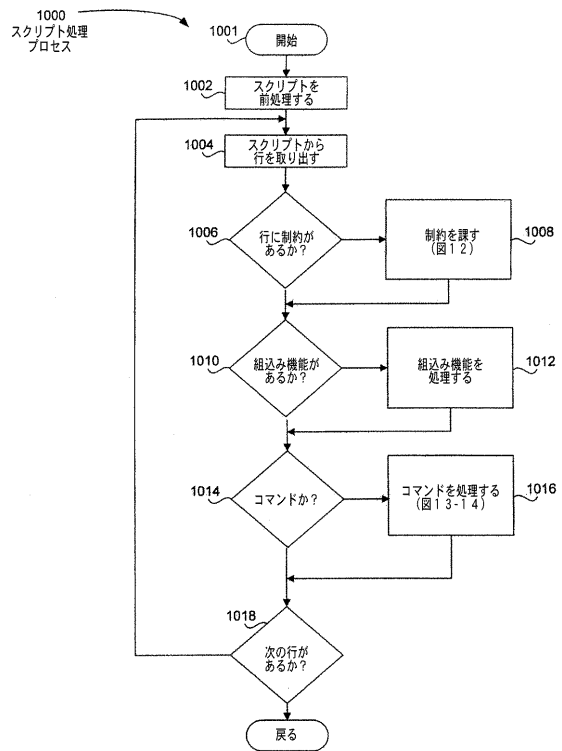
【図 8】



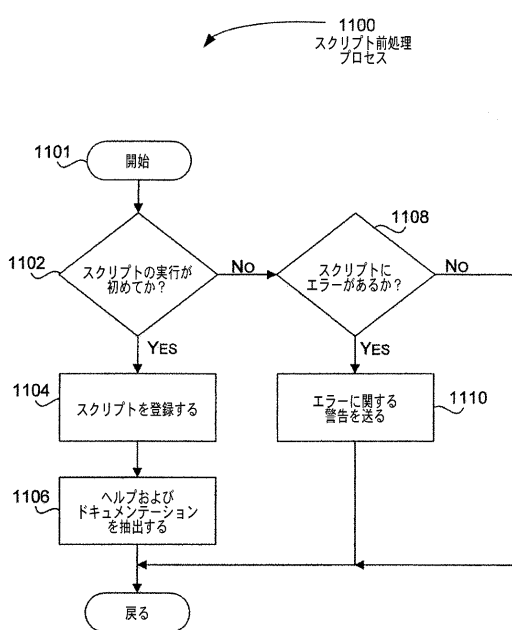
【図 9】



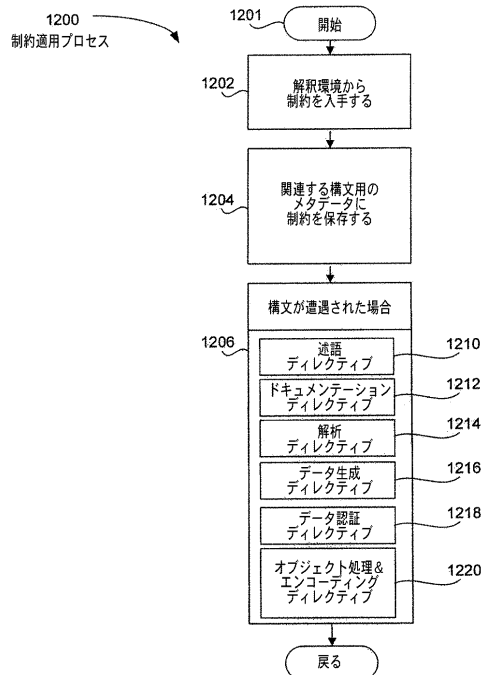
【図 10】



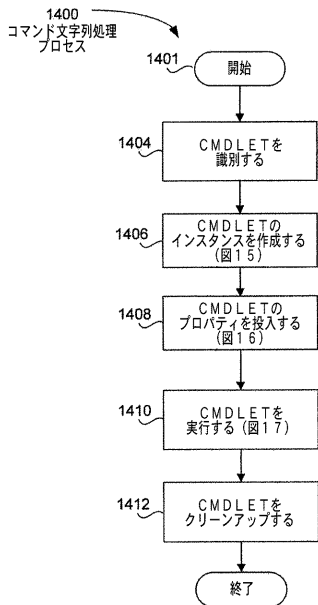
【図 11】



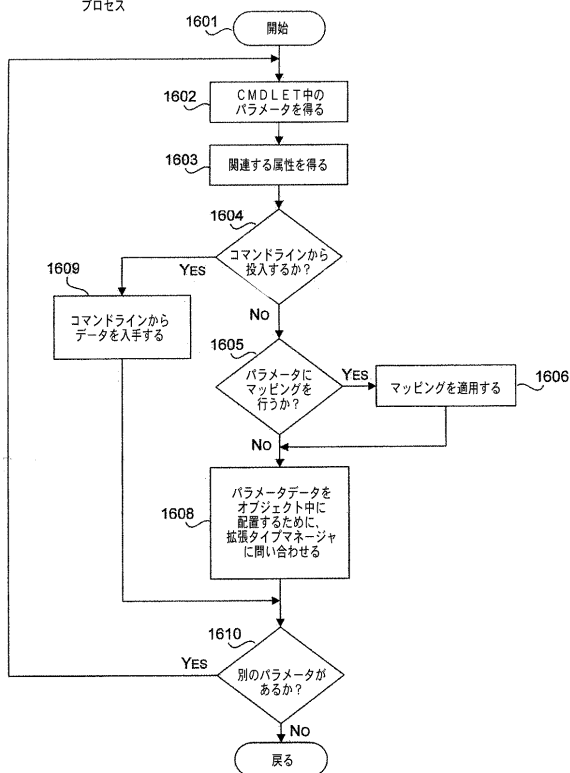
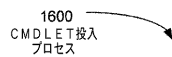
【図 12】



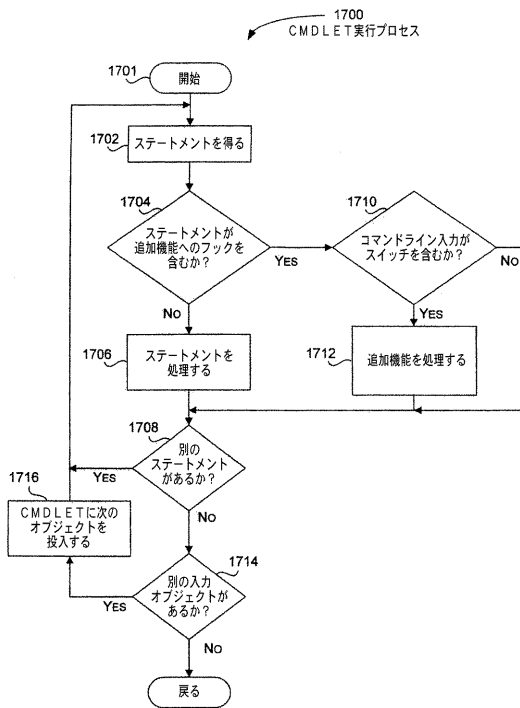
【図 14】



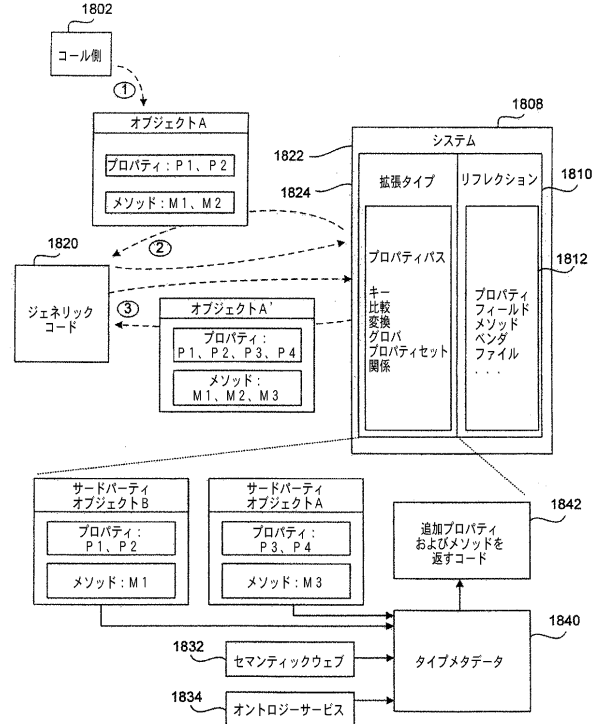
【图 16】



【図 17】

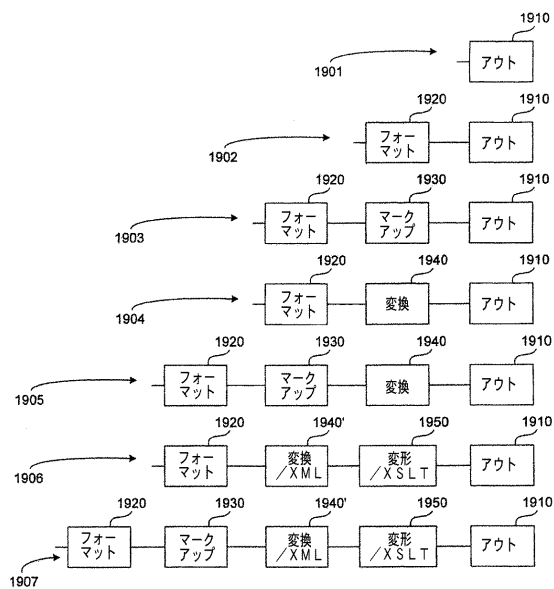


【図 18】



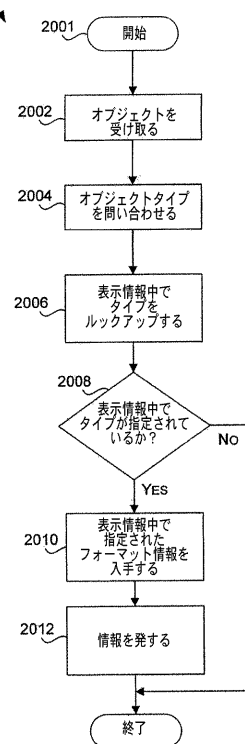
【図 19】

データ駆動出力のパイプラインシーケンス



【図 20】

2000 フォーマットプロセス



【図 2 1】

2100
例示的な
表示情報

```
<DISPLAYINFO>
<TYPE> DIRECTORYINFO </TYPE>
<SHAPE> TABLE </SHAPE>
<COLUMNS>
  <COLUMN>
    <PROPERTYNAME> LASTWRITE TIME </PROPERTYNAME>
  </COLUMN>
  <COLUMN>
    <PROPERTYNAME> SIZE </PROPERTYNAME>
    <ALIGN> LEFT </ALIGN>
    <WIDTH> 10 </WIDTH>
    <FORMATSTRING> D </FORMATSTRING>
    <LABEL> SIZE </LABEL>
  </COLUMN>
</COLUMNS>
<HEADER>
  ...
</HEADER>
<GROUPBY> PARENTPATH </GROUPBY>
...
</DISPLAYINFO>
```

【図 2 2】

2200

2201 **format/table** [-properties *propList*] [-exclude *propList*] [-groupby *propName*] [-encoding *charEncodingType*] [-header *headerSpec*] [-footer *footerSpec*] [-pagelayout *pageLayoutSpec*]

2202 **format/list** [-properties *propList*] [-exclude *propList*] [-groupby *propName*] [-encoding *charEncodingType*] [-header *headerSpec*] [-footer *footerSpec*] [-pagelayout *pageLayoutSpec*]

2203 **format/wide** [-property *propName*] [-encoding *charEncodingType*] [-header *headerSpec*] [-footer *footerSpec*] [-pagelayout *pageLayoutSpec*]

2204 **add/markup** [-markupSpec] {whereClause => {markupProp => value, [markupProp => value,...]} [{whereClause => {markupProp => value, [markupProp => value,...]}]}
| \$markupSpecVar } markupSpecFile

2205 **convert/text** [-encoding *charEncodingType*]

2206 **convert/sv** -separator *separatorType* [-encoding *charEncodingType*]

2207 **convert/csv** [-encoding *charEncodingType*]

2208 **convert/ADO**

2209 **convert/XML** [-encoding *textEncodingType*]

2210 **convert/html** [-encoding *textEncodingType*]

2211 **transform/XSLT** -XslFile *xslFilename*

2212 **out/console** [-p(age) [lines]]

2213 **out/file** -filename *filename* [-overwrite | -append]

【図 2 3】

2300

```
[1]$ dir | format/table | out/console

Written          Length.kb Name
6/28/2002 12:52:10 PM 189 sd.exe
11/7/2002 10:59:18 AM 0 sd.ini
6/28/2002 12:52:10 PM 2,633 sd.pdb
6/28/2002 12:52:18 PM 223 sdscc.dll

[2]$ dir | format/list brief.set | convert/text | out/console

Written : 6/28/2002 12:52:10 PM
Length.kb : 189
Name : sd.exe

Written : 11/7/2002 10:59:18 AM
Length.kb : 0
Name : sd.ini

Written : 6/28/2002 12:52:10 PM
Length.kb : 2633
Name : sd.pdb

[27]$ %mymarkupspec = {where length.kb -gt 500 => {applyto => all,
font.name => arial, font.style => bold, font.color => blue}
[28]$ dir | %mymarkupspec | format/table | markup %mymarkupspec | out/console

Written          Length.kb      Name
=====
9/11/2002 3:11:10 PM 1,020 Interop.Excel.dll
9/11/2002 3:11:10 PM 192 Interop.Microsoft.Office.Core.dll
9/11/2002 3:11:10 PM 120 Interop.SHDocVw.dll
9/11/2002 3:11:09 PM 12 Interop.TASKSCHEDULERLib.dll
9/11/2002 3:11:10 PM 56 Interop.VBIDE.dll

[1]$ dir | format/table | convert/csv | out/console

"Written","Length.kb","Name"
"6/28/2002 12:52:10 PM","189","sd.exe"
"11/7/2002 10:59:18 AM","0","sd.ini"
"6/28/2002 12:52:10 PM","2633","sd.pdb"
"6/28/2002 12:52:18 PM","223","sdscc.dll"
```

【国際調査報告】

60601330042



INTERNATIONAL SEARCH REPORT

International application No.

PCT/US04/23608

A. CLASSIFICATION OF SUBJECT MATTER IPC(7) : G06F 9/45 US CL : 717/143 According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) U.S. : 717/143 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) YAHOO Online keyword search		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Altiris RapidInstall, version 3.0, Release Notes, Altiris Product Marketing, July 17, 2001, pp. 1-14.	1-20
☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.		
* Special categories of cited documents:		
"A"	document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E"	earlier application or patent published on or after the international filing date	"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O"	document referring to an oral disclosure, use, exhibition or other means	"&" document member of the same patent family
"P"	document published prior to the international filing date but later than the priority date claimed	
Date of the actual completion of the international search 23 January 2006 (23.01.2006)		Date of mailing of the international search report 04 APR 2006
Name and mailing address of the ISA/US Mail Stop PCT, Attn: ISA/US Commissioner for Patents P.O. Box 1450 Alexandria, Virginia 22313-1450 Facsimile No. (571) 273-3201		Authorized officer William H. Wood <i>James R. Matthews</i> Telephone No. (703) 305-3900 06.11.2006

Form PCT/ISA/210 (second sheet) (April 2005)

フロントページの続き

(81)指定国 AP(BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), EA(AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), EP(AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR), OA(BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG), AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW

(72)発明者 ジェームス ダブリュ. トゥールサー ザ サード
アメリカ合衆国 98052 ワシントン州 レッドモンド ワン マイクロソフト ウェイ
マイクロソフト コーポレーション内