OPC技術概要書

Version 1.0

1997年2月

日本OPC協議会技術部会

はじめに

本書はOPC（OLE for Process Control）仕様の理解促進を目的に、プロセス制御分野にかかわる管理者、一般エンジニアを主な対象として、” OPC仕様書1．0版[1]”に基づき、概略機能を入門編としてまとめたものです。

OPCはひとことでいうと、マイクロソフトの技術であるOLEを利用した産業システムの標準化仕様のことであり、FA/PAに関わるものにとって重要な意味を持つ技術―正確にはインタフェース仕様―となります。しかしながら、現状、OPCに関してはOPC Foundationが策定したOPC仕様書（英語）はありますが、それは関連技術をも理解した開発者向けのインタフェース仕様書ともいうべきもので、OPC仕様で実現される機能や構造･動作、あるいは関係する基盤技術を知りたい管理者や一般エンジニアにとっては理解しづらいものでした。

日本OPC協議会(略称 OPC-J)ではこのような状況に鑑みOPCの入門書というべき“OPC技術概要書”を作成することとし、実際の作成をOPC-Jの作業部会のひとつである技術部会が行いました。

本書はOPCの唯一のドキュメントであるOPC仕様書をもとにしています。現仕様ではインタフェース仕様は明確に規定されていますが、それ以外のところ、特に実装に関わる部分はまだ不明確なところがあるような状況で本書は作成されています。実装の詳細については別途、OPC-Jから実装ガイドが発行される予定です。また、あわせて、OPCは今後改訂や機能拡充が予定されていることも付け加えておきます。

本書の構成、読み方について
1章概要でOPCの概要について説明し、OPC仕様で実現される機能は2章で、実現上の構造と動作を3章で説明しています。OPC仕様に準拠したソフトウェアの開発（主としてベンダ向け）については4章で、OPCアプリケーション例を5章で説明しています。また、ここにはOPCが規定するインタフェース仕様概略を載せてあります。OPCの基盤技術であるマイクロソフトのOLE/COMについては6章に、また、用語集と関連文献を7章付録に載せています。

本文中アンダーラインを引いた語（各章の初出のみ）は6章あるいは7章で簡単に説明されています。

本書は各章を技術部会メンバー会社で分担して執筆しました。担当は次のとおりです。

1章 概要	株式会社 東芝
2章 機能	富士電機株式会社/三菱電機株式会社
3章 構造と動作	株式会社 日立製作所
4章 OPC対応ソフトウェア開発	横河電機株式会社
5章 OPCアプリケーション	インテルーション株式会社
6章 技術基盤説明	マイクロソフト株式会社
7章 付録	マイクロソフト株式会社

[1] OLE for Process Control Standard FINAL-RELEASE Version1.0 1996/8/29
以降、単に“OPC仕様書”とし、ここで規定される仕様を“OPC仕様”と記します。

目次

1 概要

1.1 背景

以前のコンピュータシステムではハードウェアやソフトウェアなどプラットホームが異なるコンピュータ間のデータ共有、通信をおこなうために独自にプログラムを作成する必要があり、多大な労力を費やしてきましたが、現在は確立した規格化のおかげで、たとえばインターネットのように、異なるコンピュータ間を接続する広大なネットワークが実現されました。このため、例えば企業情報システムの開発においては規格に基づいたデータベースとクライアント／サーバツールを使って、本来の目的であるアプリケーション開発に注力できるようになりました。製造システムにおいても同様な進化が必要です。すなわち、異なるベンダの機器を特別なソフトウェアを開発することなく相互接続できること、また、図 1-1のようなシステムを構築する上で、プラント機器レベルのデータを扱うフィールド管理層、プロセス処理を行う分散制御システム層、さらに上位の生産管理層に至るまでシームレスにデータ共有化がはかれる規格作りと普及が急務です。このような状況のもとで製造システムにおいても今後の主流とみられるWindowsの基幹技術であるOLE/COMをプロセス制御用途に拡張するOPCが策定されました。

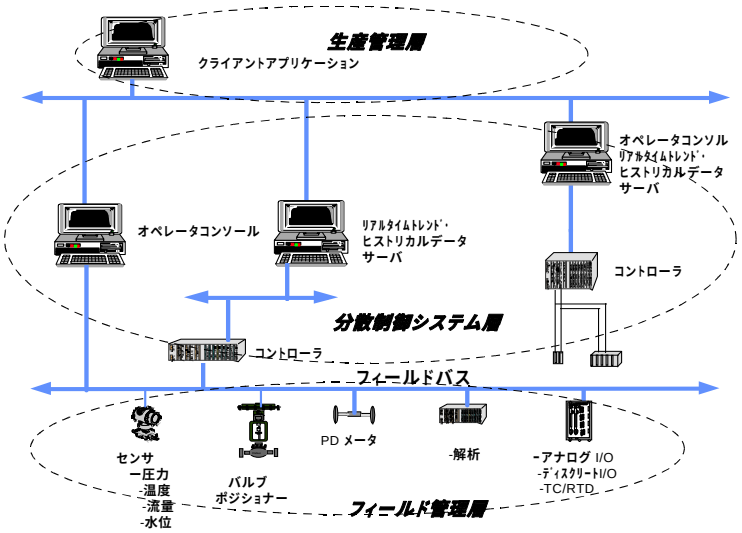


図 1-1 プロセス制御情報アーキテクチャ

1.2 OPCの特長

1.2.1 目的とスコープ

OPCは異なるベンダの機器で実行される異なるアプリケーション間のインタフェースを標準化してデータ交換を容易にすることが目的です。その結果として、開発言語や開発環境などにかかわらず統合的に利用できるプロセス制御用ソフトウェアコンポーネントをユーザへ提供できるようになります。また、現状ハードウェアのドライバインタフェースとこれらを利用するアプリケーション間のインタフェースには統一性がないため、装置ごとあるいはベンダごとに異なり、たとえば図 1-2の様に3種類のサーバ（装置）とそれを利用するアプリケーションが2種類ある時、合計6種類のインタフェースソフトを多大な時間をかけて作成する必要があります。OPCを利用することによりインタフェースを1種類に統一することができ、図1-3のようなシステムを構築することができます。アプリケーションX（Y）からは装置A、B、Cの内部仕様、ベンダなどの変更に無関係に装置を利用することができます。

OPCを利用するシステムは、大きく、アプリケーション（クライアント）の要求に応じてデータ収集ほかサービスを提供するOPCサーバ、OPCサーバを利用するためのOPCインタフェース、およびサービスをうけるOPCクライアントを用いたアプリケーションで構築することができます。OPCサーバは各ベンダのハードウェアに実装することができますが、OPCサーバにはベンダごとのハードウェア、システムの違いをベンダ固有機能対応として処理する機能、およびデータのフォーマットもクライアントの要求に応じて対応できるVARIANT型とよばれるデータタイプを処理する機能があるからです。

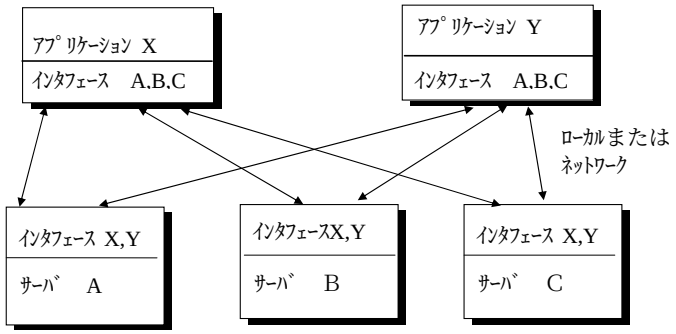


図 1-2 サーバとアプリケーション

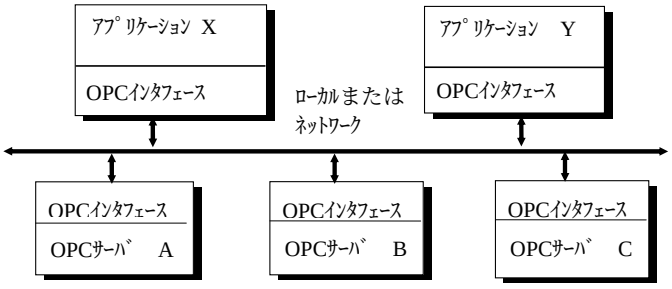


図 1-3 OPCサーバとアプリケーション

また、OPCは(1)移植しやすく、(2)多くのベンダのニーズに応えられるようフレキシブルで、(3)ハイレベルな機能性を提供し、(4)十分なオペレーションが可能であることを目標に策定されており、メーカやユーザにとってそれぞれ次のようなメリットをもたらします。

(1)装置開発者:デバイスドライバ開発の一元化が可能になります。

(2)アプリケーションソフトウェア開発者:汎用開発ツールを利用できます。独自インタフェースの開発が不要になり、装置とのインタフェースが容易になります。

(3)ユーザ:各種業務パッケージソフトとの連携が可能になり、システム開発コストを低減できます。また、フィールドデバイスやI/O機器を統合し、表示、操作性の統一されたシステムを構築できます。

OPC仕様準拠のコンポーネントが普及するにつれシステムの拡張やコンポーネントの置換が容易になるほか、データアクセスも容易になり、たとえばプロセス制御アプリケーションでのデータをデータベースソフトや表計算ソフトに直接送信し自動編集するなど高度なデータ統合が可能になります。

なお、OPC仕様を実装したサーバの実行環境はWindows NT 4.0以降を必要とします。OPCクライアントの実行環境はWindows 95(クライアントのみ実行する場合)またはサーバ同様Windows NT 4.0以降を必要とします。 機能的には装置間の効率的データ読み込み／書き込みに限定されています。アラーム、セキュリティ、履歴管理などは次期仕様で追加される予定です。

1.2.2 システムにおける位置づけ

OPCはデータの供給元とデータの利用者間を接続するためのインタフェース仕様です。データの供給元にはたとえばPLC、DCS、バーコードリーダ、計測解析機などが考えられます。図 1-4の例のように、最も低いレベルとしては物理デバイスからの生データをDCS／SCADA、自動化制御システムへの供給のために、また、さらにそこからのデータを上位アプリケーションに送る用途や、直接アプリケーションと物理デバイスとを接続する場合などにも適用できます。このようにOPCインタフェースはシステム上多くの場所に数多く実装することができる柔軟な仕様になっています。

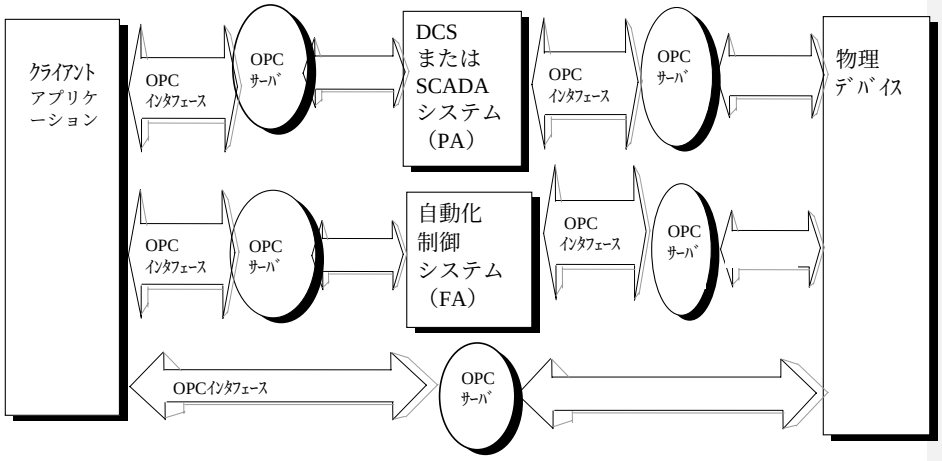


図 1-4 OPCとシステムにおける位置づけ

1.2.3 開発言語とOPCの実装形態

OPCはマイクロソフトのOLE/COM技術に基づき、クライアント／サーバモデルを継承しています。OPCサーバの実装形態には、クライアントと同一マシン上でクライアントと同じプロセスで実行するモデル（In-Proc Server）、別のプロセスで実行するモデル(Local Server)、さらにネットワーク上の他のマシン上で実行するモデル(Remote Server)などがありますが、ネットワーク構成の場合を例に図1-5に実装形態を示します。

OPCサーバはDCSやI/Oドライバなどを供給するハードウェアベンダが提供するソフトウェアで、図1-5では右側のサーバマシン上で実行されます。そのクライアントは左側のクライアントマシン上で実行されるIn-Proc Handlerです。ユーザインタフェースなどを提供するアプリケーションベンダは左側のクライアントアプリケーションプログラムを開発します。開発言語にはVB(Visual Basic)やC++などを利用することができます。一般には、VBや、VBA（VB for Applications）のようなスクリプト言語で開発されるアプリケーションはOPCオートメーションインタフェースを利用し、アプリケーションがC++で作成され、また、最大性能を引き出したい場合はOPCカスタムインタフェースを利用します。

OPCクライアントアプリケーションから、たとえばプロセス制御装置などへの要求はIn-Proc Handlerを経由してネットワーク上にあるサーバマシンのOPCオブジェクトのインタフェースを介して伝わります。具体的にはメソッド、プロパティをアクセスしてデバイスへのデータ設定、デバイスからの読みとりを行います。OPCサーバはこれらの呼びかけで要求に応じた適切な対応（たとえば、装置のﾃﾞｰﾀなど）をOPC仕様に準拠して用意しなければなりません。この装置固有の対応付けなどはOPCサーバを供給するベンダの責任で行います。In-Proc Handlerはサーバ／クライアント間のデータをバッファリングしておく一種のデータキャッシュで、OPC仕様上必須ではありませんが、ネットワーク間の通信時間を節約して高い性能を実現するために、OPCサーバとともにハードウェアベンダが提供することを推奨されているプログラムです。

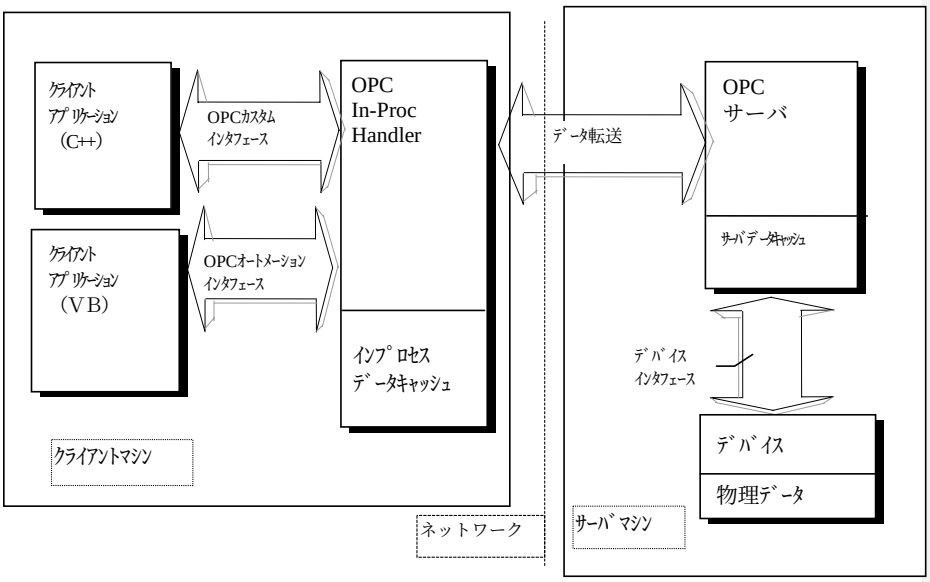


図 1-5 開発言語とOPCの実装形態の例

1.3 OPC構成要素概要

OPCが定めたオブジェクトモデルでは標準コンポーネントとして、OPCサーバオブジェクト、OPCグループオブジェクト、OPCアイテムオブジェクトと呼ぶ3種類の論理オブジェクトを提供しています。
OPCサーバは図 1-6のようにベンダごとの物理デバイスへの接続を行うもので、ベンダ固有情報を理解するI/Oドライバともいえます。OPCサーバがアクセスできる物理デバイスのデータアイテム（OPCアイテム）にはベンダが名前をつけてリストのように登録します。このアイテムの名前はフラット型でも、また、階層型（例：Tank1.Pump.PV）にすることもできます。また、OPCアイテムの名前にはサーバが実際にデータを検出するためのアクセスパス名と、識別するためのアイテム定義をベンダが指定することができます。そのほか、後述のOPCグループの管理や2種類のOPCインタフェースの提供などを行っています。OPCサーバは複数のOPCクライアントと接続することができます。

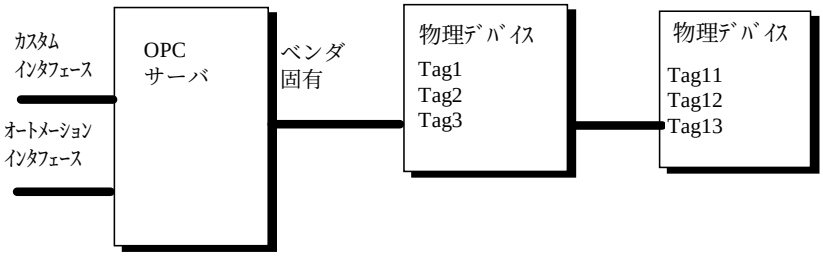


図 1-6 OPCサーバ

OPCアイテムは上述のようにサーバが認識できるように定義され、通常タグに相当するような単一の変数（セットポイントやプロセスデータ）のデータ供給元にリンクするものです。OPCクライアントはOPCアイテムのレベルでメソッドをアクセスすることによりデータの読み込み／書き込みを1点ずつ行うこともできますが（VBの場合）、さらに効率よく、また、目的別にデータをアクセスできるようOPCサーバ内にOPCグループを1つ以上設けることができます。OPCクライアントはOPCグループ内に任意の個数のOPCアイテムを追加することができ、図 1-7 に示すようOPCグループに対して処理を要求することで一括してデータ（例：Tag1－3）を読み込み／書き込みすることができます。

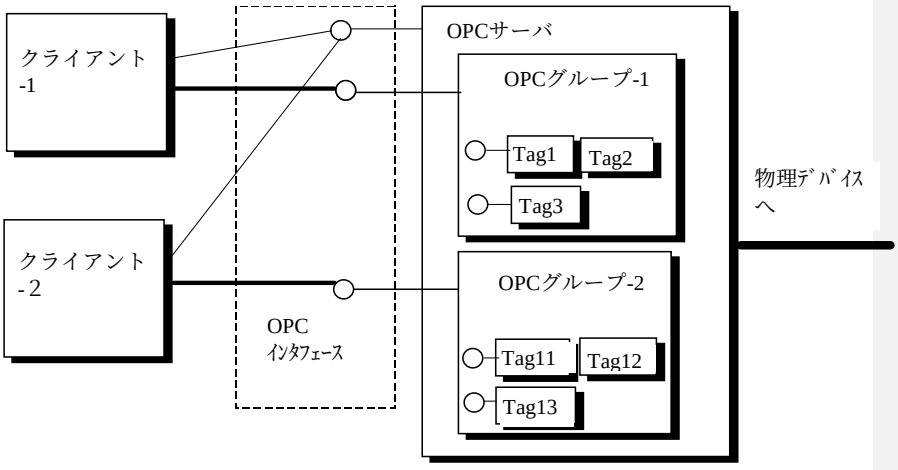


図 1-7 OPCコンポーネント構成の例

2 機能

2.1 概要

OPCはリアルタイムにプロセスデータを転送する機能を実現するための拡張性を考慮したフレームワークを提供します。アプリケーションはプロセスデータがどのようなフォーマットで、どこにあり、どのように取得しなければならないかを意識することなくプロセスデータアクセス（読み込み/書き込み）ができます。

OPCはクライアント/サーバ構造をとっています。1つのOPCクライアントは1つ以上のベンダが提供するOPCサーバ（複数）に接続できます。逆に、1つのOPCサーバは複数のOPCクライアントとの接続をサポートします。

OPCサーバは物理デバイスやそれに対するアクセス方法を知っており、OPCクライアントはそれらについて知っている必要はありません。

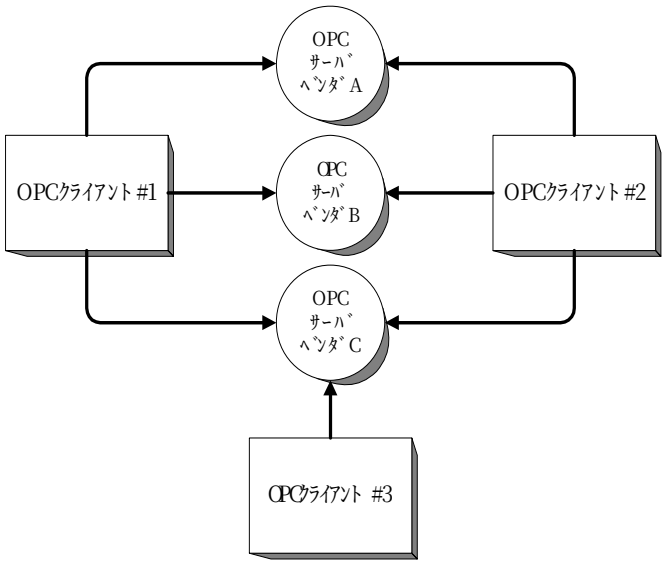


図2-1 OPCのクライアント／サーバ（例）

OPCはデータ交換に関するインタフェース規約で、それ以上のものでも、それ以下のものでもありません。この章ではOPCにより実現される機能について、FA／PAのシステムをモデルとして説明します。

一般に、監視･制御システムではフィールド機器に直結した現場コンピュータが機器からのデータ収集、保管、監視、制御、診断の各機能を担当します。

従来、データの収集、保管、監視、制御、診断、伝送、アラーム表示、その他の業務処理のための機能モジュールはメーカーなどが独自に設計し、そのインタフェースを規定しています。

これらの機能モジュールを標準の論理オブジェクトとして整理し、その呼び出しインタフェースを標準化することにより、FA/PAシステムのソフトウェア構築を次のように進めることがOPC適用のおもなねらいです。

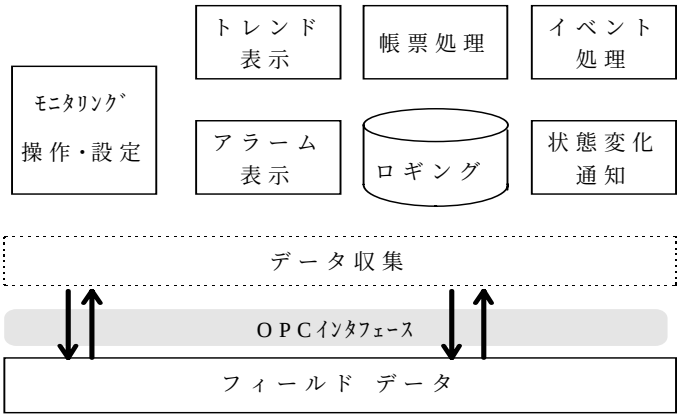


図2-2 監視･制御システムの機能構成例

監視･制御システムでは、プラント、瞬時データ、トレンド、アラームをはじめ、各種業務処理用の画面が存在します。このような表示画面をオブジェクト部品として整備していけば、それらを利用することによってシステムの生産性を大きく向上できます。さらに、ユーザ仕様に不確定部分の残る初期段階でのプロトタイプ作成や仕様変更に容易に対応できます。

機能部品を実現するためにクライアントとしては、①フィールド、②ヒストリ、③レポート、④アラーム／イベントなどの各データを必要とします。

現在のOPC仕様書ではフィールドデータのアクセス方法についてのみ規定されています。ヒストリ、アラーム／イベントなどについては現在検討が行われています。現状ではデータ収集を受け持つアプリケーションがデータのロギングを行い、また、データの整合性等をチェックし異常を検出した場合にはアラームの通知を行う等の処理を実装します。したがって、アラームの表示方法等についてはデータ収集などを行うアプリケーションの処理に依存します。

2.2 OPCの機能概要

OPC仕様を実装することにより実現される機能を以下に示します。OPCはこれらをすべて必須として要求しているのではなく、オプションと定めているものがあります。その場合、これらが実際にサポートされるかどうかはベンダ依存、サーバ依存となります。オプションか否かは機能単位よりもむしろインタフェース単位で定められており、この章でも概略が示されていますが、詳細については5章またはOPC仕様書4章、5章を参照してください。

表 2-1 OPCの機能仕様

機能	方式	説明
プロセスデータ読み込み ※ 4つの方式は、互いに独立していて干渉はありません。	同期読み込み	アイテムに対応するプロセスデータを読み込みます。 読み込み完了まで待ち合わせます。
	非同期読み込み	アイテムに対応するプロセスデータを読み込みます。 要求後直ちにリターンし、読み込み動作が完了した時点でOPCクライアント側のメソッドが呼び出されます。 要求時に受け取りデータのフォーマットを指定できます。 ・タイムスタンプ付きデータ ・タイムスタンプなしデータ
	リフレッシュ	すべてのアクティブなアイテムからプロセスデータを読み込みます。 (非同期入力の一種)
	データ変化通知 (subscription)	一定周期でチェックし、データにある幅以上の変化があった場合、OPCクライアントに通知します。
プロセスデータ書き込み	同期書き込み	アイテムにデータを出力します。 書き込み動作完了まで待ち合わせます。
	非同期書き込み	アイテムにデータを出力します。 要求後直ちにリターンし、書き込み動作が完了した時点でOPCクライアント側のメソッドが呼び出されます。
構成情報のセーブ／ロード	—	OPCサーバ全体の構成情報のセーブ／ロードをサポートします。
ブラウズ	—	グループ、アイテム、属性などの各種情報を閲覧します。

OPCはその機能仕様よりは実現の仕方、実装の仕方に特長があります。

実際には、これらの機能がOPCカスタムインタフェースとOPCオートメーションインタフェースという用途別の2つのインタフェースで提供されるよう規定されています。

これらの使い分けについては3.2節を参照してください。

2.3 プロセスデータアクセスの基本

基本的なアイデアはOPCクライアントがOPCサーバを指定して接続し、接続したOPCサーバでアクセスできるアイテムの集合（サーバアドレススペースという）中の個々のアイテムを識別する名前（アイテムID）を指定して、それに対応するプロセスデータを読み込み/書き込みするというものです。実際には、効率向上のためグループという概念を設け、グループのなかに1つあるいは複数のアイテムを対応させます。

2.3.1 アイテムID

アイテムを識別する任意サイズのUNICODE文字列です。いわゆるタグと呼ばれるものです。シンタックスはベンダ依存です。

DCSの例：TIC101.PV

PLCの例：COM1.STATION:42.REG:40001;0,4095,-100.0,+1234.0

ただし、OPCではアイテムIDについてフラットモデルと階層構造モデルが規定されており、次のような名前の付け方ができるようになっています。（フラット/階層構造いずれをサポートするかはサーバ依存）

フラット ：TIC101

階層構造：AREA1.REACTOR3.TIC101

2.3.2 プロセスデータ

プロセスデータは表2-2に示す3種類のデータで構成されます。通常、この3つのデータは組みで扱われます。

表2-2 プロセスデータ

データ	説明	備考
データ値(Value)	値そのもの	VARIANT型
品質フラグ(Quality Flag)	データの有効性などを表わす	FieldBus Foundation仕様と同様
タイムスタンプ	データの取得時刻	UTC表現

2.3.2.1 データ値（Value）

OPCにおいてはあらゆるデータ型を安全に扱えるようにプロセスデータはVARIANT型と呼ばれるデータ型を使用しています。

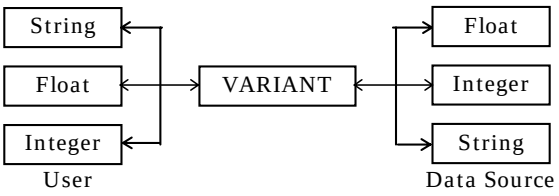


図2-3 OPCのデータ型(例)

表2-3の範囲のデータ型を自由に扱えます。

OPCサーバの返す値のデータタイプはOPCクライアントから要求することができます。OPCサーバが、要求されたデータタイプをサポートしていない場合は警告が返され、返される値はOPCサーバがそのアイテムに対して標準で規定するデータタイプ(正規化データタイプ)となります。

表 2-3 VARIANT型データ

データ型	バイト数	説明
VT_I2	2	単精度（16ﾋﾞｯﾄ）整数
VT_I4	4	倍精度（32ﾋﾞｯﾄ）整数
VT_R4	4	単精度（32ﾋﾞｯﾄ）浮動小数
VT_R8	8	倍精度（64ﾋﾞｯﾄ）浮動小数
VT_CY	8	VT_UI8と同じ 通貨（10000倍）
VT_DATE	8	VT_R8と同じ 1899/12/30からの通算日時
VT_BSTR	可変	文字列 VT_UI4で示された文字数の後にUNICODE文字とNULLターミネータを付加
VT_BOOL	1	VT_UI1と同じ 0：FALSE、1:TRUE
VT_UI1	1	符号なし文字
VT_ARRAY	可変	上記データタイプの一次元配列

2.3.2.2 品質フラグ(Quality Flag)

データ値（Value）が正しく取得できたかどうかを示すフラグです。異常時にはその原因を通知します。たとえば、デバイスやセンサの故障を検出した、あるいはアイテムが削除されている等のステータスが通知されます。この詳細についてはOPC仕様書7.8を参照ください。

2.3.2.3 タイムスタンプ (Time Stamp)

データが取得されたUTC時刻を表すFILETIME構造体（8バイト）データです。デバイス上の時刻、またはOPCサーバ上の時刻を表します。このデータ型で表現できる時刻精度は100nsです。なお、非同期読み込みの場合はこの項目の取得を省略できます。

2.3.3 アイテムの属性

アイテムの属性を表2-4に示します。これらの属性の取得/設定はOPCクライアントから任意の時点で行えます。ただし、工業単位についてはOPCサーバが提供する機能であり、取得のみサポートされています。

表 2-4 アイテムの属性

属性	値	説明
Active	TRUE/FALSE	FALSEにするとデータ読み込みは行われない。
アクセス権	読み込み可能/書き込み可能	そのアイテムが読み込み/書き込みできるかどうかを示す。セキュリティとは別個の、そのアイテム本来の性質。
要求データ型	データ型	OPC仕様では読み込みデータのタイプは読み込み要求時にOPCクライアントが指定できる。
正規化データ型	データ型	OPCサーバに保持するデータのタイプ。
工業単位情報タイプ	なし、アナログ、列挙	次項目の工業単位情報のタイプを示す。(オプション)
工業単位情報	アナログ：レンジLOW/HI指定 列挙：値の0,1,2..に対応して文字列を指定(例：OPEN, CLOSE, IN TRANSIT,...)	工業単位情報タイプにより内容が異なる。(オプション) 列挙の場合、OPCサーバがローカライゼーションをサポート可能。
アクセスパス	文字列(サーバ依存)	プロセスデータアクセスに複数の方法がある場合、そのいずれを使用するか要求時に指定できる。(オプション)

2.3.4グループの概念

OPCクライアントが効率よくプロセスデータをアクセスする仕組みとして“グループ”が用意されています。グループにはOPCクライアントが任意の個数のアイテムを登録することができ、プロセスデータアクセスは通常、これを単位として行います。グループにはActive属性や更新周期の属性などがあります。

2.3.5 データのキャッシュ

同期型読み込み、非同期型読み込み要求では（要求時Call単位で）データソースとしてCACHEあるいはDEVICEのいずれかを指定します。

パフォーマンスの問題から通常はCACHEを使用します。DEVICEは診断などの限られた場合での使用となります。DEVICEのときはグループおよびアイテムのActive属性(アクティブ／非アクティブ)に関わらずデバイスからの読み込みを行います。

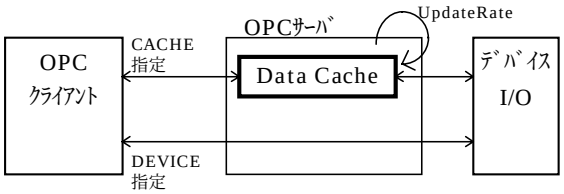


図2-4 CACHEとDEVICE

キャッシュの更新には次の2つの場合があります。

OPCクライアントからのリフレッシュ要求ではグループ内のすべてのアクティブなアイテムについてキャッシュ値を更新します。

また、OPCクライアントからの要求とは無関係に一定の更新周期(UpdateRate)でキャッシュが更新されます。

UpdateRateはグループ毎に設定される属性で、OPCクライアントからの要求により、指定された更新周期にできるだけ近い値がOPCサーバで設定され、実際に設定された更新周期はOPCクライアントに通知されます。

なお、書き込み要求に対してはOPCサーバは実際のデバイスへの書き込み完了（または失敗確認）まで実行するためキャッシュ動作は無関係です。

2.3.6データ変化通知(subscription)

表2-1におけるプロセスデータ読み込みの4方式のうち、同期読み込み／非同期読み込み／リフレッシュはOPCクライアントからの要求によるものですが、OPCクライアントの要求によらずOPCサーバから通知を受ける方式がsubscriptionです。更新周期(UpdateRate)によるキャッシュ値更新の際、値に変化があればOPCクライアントに通知されます。ただし、OPCサーバがﾃﾞｯﾄﾞﾊﾞﾝﾄﾞ(PercentDeadBand)をサポートしていて、アイテムの工業単位情報タイプがアナログの場合は、現在値と前回値の差の絶対値がこれで決まる値を超えた時のみキャッシュデータを更新し、OPCクライアント側のメソッドに通知されます。これによりアナログ値の小さなじょう乱を無視することができ、サーバおよびクライアントの負荷軽減となります。

2.4 OPCのオブジェクトモデル

OPCの特長のひとつとして、OPCサーバ、OPCグループ、OPCアイテムのオブジェクトモデルを採用していることが挙げられます。

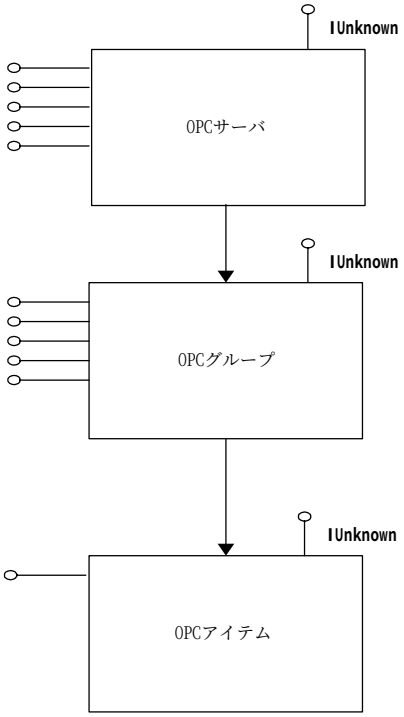


図2-5 OPCのオブジェクトモデル（オートメーション）

OPCサーバオブジェクトはOPCサーバ内の詳細をOPCクライアントから隠蔽化します。これにより、OPCサーバを利用するアプリケーション(OPCクライアント)は異なるベンダから提供されるOPCサーバを同じ手順で使用することが可能となります。OPCアイテムオブジェクトは制御する物理デバイスなどのハードウェア情報を隠蔽化します。これによりベンダ毎の物理デバイスの仕様の違いに対応できます。
OPCグループオブジェクトはこれらのオブジェクトの中で最も優れたメリットがあります。上記の2つのオブジェクトとは性格が異なり、OPCサーバ側で各OPCクライアントの情報を保持しています。たとえば、データの変更通知レートであるUpdateRate値などを各OPCクライアント毎に保持しサーバ側で管理できるので、クライアント／サーバ間での通信負荷を減らすことができ、パフォーマンスの向上を計れます。

2.4.1 OPCサーバオブジェクト

OPC サーバが最初に生成するオブジェクトです。OPC サーバオブジェクトは、OPC サーバ全体(共通)の管理情報および OPC クライアントとのコネクションにおける情報を保持しています。OPC グループオブジェクト、OPC アイテムオブジェクトは、この OPC サーバオブジェクトと関連してつくられます。

表2-5 OPCサーバの属性

項目	説明	備考
スタート時刻	OPCサーバ起動時刻	UTC
現在時刻	OPCサーバの現在時刻	UTC
バンド幅	使用率など	サーバ依存
メジャーバージョン	OPCサーバソフトウェアのメジャーバージョン番号	
マイナーバージョン	OPC サーバソフトウェアのマイナーバージョン番号	
ビルド番号	OPC サーバソフトウェアのビルド番号	
ベンダ情報	サーバベンダの文字列情報	ベンダ名とサポートするデバイスなど
ステータス	OPCサーバの現在の状態	正常、エラー、構成情報なし、テストモードなど(OPC仕様書7.7.5参照)
グループ数	OPC サーバが管理しているグループ数(プライベートも含む)	主として診断用
前回更新時刻	この OPC クライアントに対して値を更新した前回時刻	UTC

1つのOPCサーバオブジェクト内には1つ以上のOPCグループを定義できます。1つのグループの中には、1つ以上のOPCアイテムを定義できます。図に示すと以下のようになります。

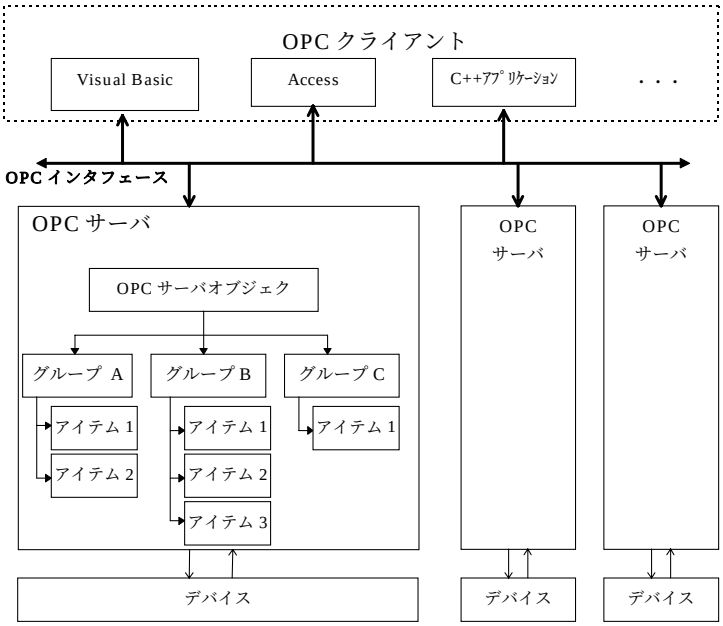


図2-6 OPCサーバオブジェクトの内部

2.4.2 OPCグループオブジェクト

OPCサーバオブジェクトがアイテムの集合を管理するために提供するオブジェクトです。データの入出力は基本的にOPCグループ単位で行います。このため、OPCクライアントはOPCグループ単位でアクティブか非アクティブの設定など、グループ単位でのデータ入出力の制御を行うことができます。表2-6にOPCグループの属性を示します。OPCクライアントはこれらの属性の取得／設定ができます。

表2-6 OPCグループの属性

属性	説明	備考
グループ名	このOPCクライアントグループの中でユニークな名前	パブリックグループ(後述)の場合はすべてのパブリックグループの中でユニークな名前。
Active	アクティブ／非アクティブ	
UpdateRate	データ変更通知のレート	単位はミリ秒。
Time Zone(Time Bias)	デバイスの時刻との差分調整用	データのタイムスタンプをデバイスのLocal timeに戻す時に使用される。単位は分。
デッドバンド	不感帯（百分率）	工業単位情報タイプがアナログの時
LCID	OPC サーバが値を文字列として返す時に用いられる言語識別子	工業単位列挙を含むアラームやステータスなど

プライベートグループとパブリックグループ

通常、グループはプライベートグループとして作成されます。プライベートグループはそれを作成したOPCクライアントのみがアクセスできますが、複数のOPCクライアントからアクセスできるパブリックグループもオプションとして用意されています。この場合、最初、プライベートグループとして作成されたグループをMoveToPublicメソッド(5.3節および5.4節参照)によってパブリックグループに変更します。[2]

OPCクライアントは、一度パブリックグループとして接続すると、プライベートグループと同じように簡単に操作することができます。グループやグループ内のアイテムを実行/停止させたり、クライアントハンドルをセットしたり、アイテムに要求データタイプを設定することができます。これらのすべての操作はそのOPCクライアントからのみ影響を及ぼし、そのパブリックグループへ接続されたほかのOPCクライアントの動作には影響しません。例外はアイテムの追加および削除ができないことです。

クライアントごとに保持される属性

グループ属性のうち、Active属性、UpdateRate、TimeZone
アイテム属性のうち、Active属性、要求データタイプ

[2] OPC仕様では、クライアントからの要求によらずOPCサーバも作成できる。

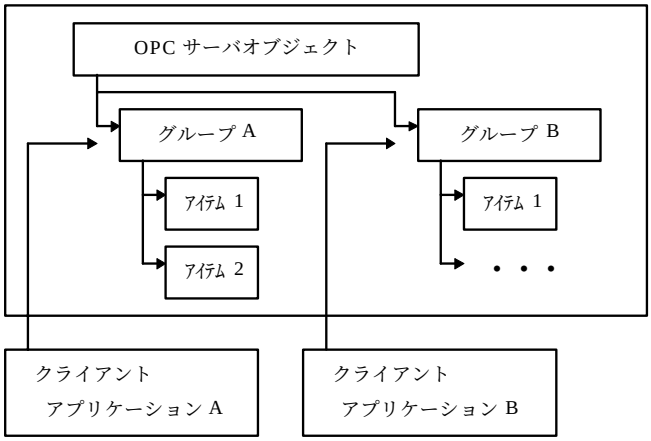


図2-7 通常のOPCグループのアクセス例

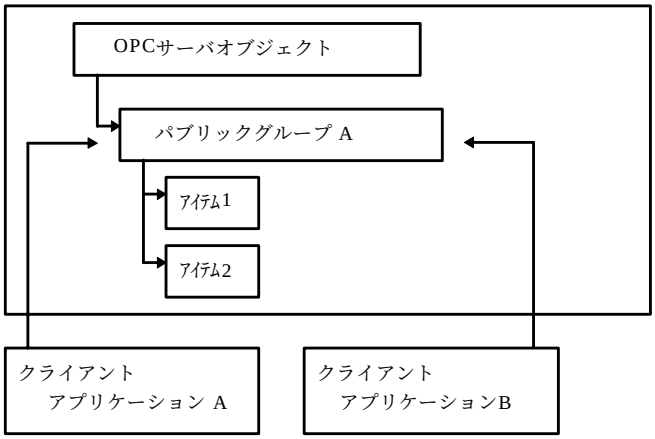


図2-8 OPCパブリックグループのアクセス例

2.4.3 OPCアイテムオブジェクト

OPC オートメーションインタフェースではアイテム単位でのアクセスが可能なため、OPC オートメーションインタフェースに限りOPCアイテムオブジェクトがあります。

2.5 ブラウズ機能

ブラウズ機能とはその内容を閲覧する機能です。これによってOPCクライアントはたとえば、そのOPCサーバでどのようなアイテムが利用可能か、OPCサーバに現在どのようなグループがあるのか、あるいはそのグループにどのようなアイテムが登録されているのか、などを知ることができます。

2.5.1 サーバアドレススペースのブラウズ（オプション）

そのOPCサーバで利用可能なアイテムをブラウズすることができます。このとき、アイテムID、データ型、アクセス権それぞれにフィルタをかけられます（このフィルタ仕様はベンダ依存）。階層構造モデルの場合はさらにブラウズ位置の変更ができ、その時のブラウズ位置から下でアイテムをブラウズできます。

2.5.2 グループのブラウズ

グループに関してはブラウズのために下記の列挙(Enumeration)機能が規定されています。

そのOPCクライアントが接続しているプライベートグループの列挙
そのOPCクライアントが接続しているパブリックグループの列挙
そのOPCクライアントが接続している全グループの列挙
そのOPCクライアントが作成したすべてのプライベートグループの列挙
OPCサーバで利用可能なすべてのパブリックグループの列挙
すべてのプライベートグループとすべてのパブリックグループの列挙

2.5.3 アイテムのブラウズ

アイテムに関してはブラウズのために下記の列挙(Enumeration)機能が規定されています。

グループ中のアイテムのブラウズ
指定アイテムのアクセスパスのブラウズ

2.6 構成情報のセーブ･ロード機能

2.6.1 サーバアドレススペースと構成情報

サーバアドレススペースの定義方法についてはOPCでは規定していませんが、例として次のような方法を挙げています。
･完全に固定
･OPC環境の外で完全に決める
･”intelligent”なOPCサーバの立ち上げ時に自動的に定義（ポーリングして）
･OPCクライアントアプリケーションが要求しているアイテムの名前に基づき、”intelligent”なOPCサーバがその場で定義する

これとは別にOPCサーバ全体の構成情報については、それらの情報のセーブ/ロードのインタフェースが用意されており、これを利用してOPCサーバ立ち上げ/シャットダウン時での構成情報の保持ができる仕組みが作られています。これは、グループやアイテムの定義などクライアント特有の情報を対象にしてはいません。

2.6.2 構成情報のセーブとロード

OPCサーバが保持する構成情報はIPersistFileインタフェースを実装することによりファイルセーブ／ロードができます。（IPersistFileインタフェースはOLE標準インタフェースのひとつで、これを実装するかどうかはベンダに依ります）

2.7 その他

2.7.1 文字コード

OPC インタフェースでの文字列パラメータはすべて UNICODE と規定されています。(Windows NT の文字コード体系をそのまま利用している)。

2.7.2 NLS(National Language Support)

OPC クライアントはグループの属性として LCID を設定できます。OPC サーバが文字列データを返すとき、この LCID を参照して処理を行うことができます。(サーバ依存)

3 構造と動作

本章ではOPC仕様に従ったサーバおよびクライアントの構造と動作について説明します。

(注意) OPCサーバの実行環境はWindows NT 4.0 以上が必要となります。また、OPCクライアントの実行環境はWindows NT 4.0以上またはWindows 95 (Remote Serverへのアクセス不可)が必要となります。以下の説明はWindows NT 4.0の使用を前提にします。

3.1 OPCサーバの構造

OPCサーバの構造にはIn-Proc Server (DLLサーバ)、Local / Remote Server (EXEサーバ)の実装形態があり、さらにパフォーマンス向上のためにLocal / Remote ServerにHandlerを併用した構造とすることが考えられます。

3.1.1 In-Proc Server

In-Proc ServerとはOPCサーバを利用するクライアントアプリケーションのプロセス空間内で動作するサーバです。すなわち、DLLとして実装されたサーバで、OPCクライアントからのOPCサーバオブジェクト生成要求があると、自動的にOPCクライアントプロセス内にサーバプログラムがロードされて実行されます。In-Proc Serverの構造を図 3-1に示します。

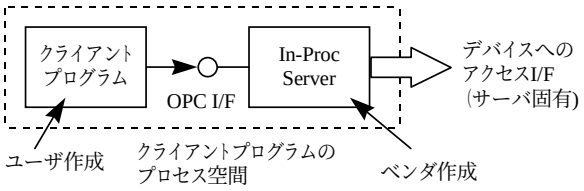


図3-1 In-Proc Serverの構造

3.1.2 Local / Remote Server

Local / Remote ServerとはOPCサーバを利用するOPCクライアントアプリケーションとは別なプロセスとして動作するサーバです。すなわち、EXEプログラムとして実装されたサーバで、OPCクライアントからのOPCサーバオブジェクト生成要求があると、自動的に該当プログラムが実行され、サーバプロセスが生成されます。Local / Remote Serverの構造の概略を図 3-2 (a)に示します。

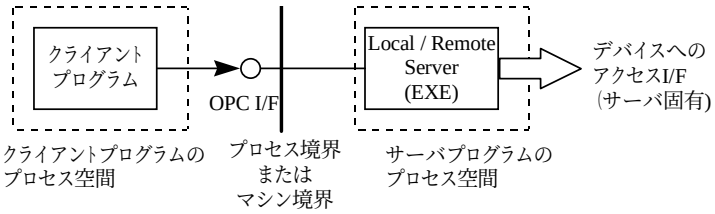
Local / Remote Serverでは複数クライアントから接続要求があった場合、新たなサーバプロセスを生成せず単一のプロセス内で全てのOPCサーバオブジェクトを作成することが可能です。この構成ではサーバ内で物理デバイスに対する複数アクセスのロックを実現できます。

サーバの動作するマシンはクライアントと同じマシン(Local Server)でも異なるマシン(Remote Server)でも可能です。異なるマシン間の接続にはDCOM (分散COM)によるRPC通信が使用されます。サーバから見た場合、同一マシン内での接続もリモートマシンからの接続も全く同様に行われるため、サーバプログラム内で接続形態を意識する必要はありません。

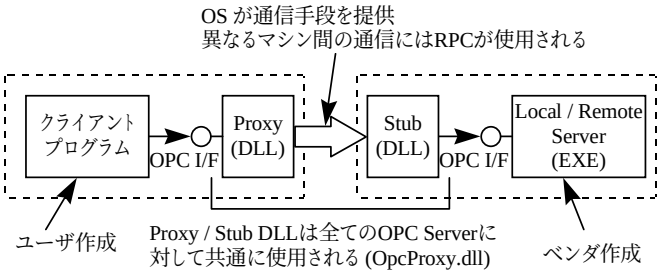
Local / Remote Serverとクライアントは異なるプロセス空間で動いているため直接、メソッドの呼び出しやデータ交換を行うことはできません。実際には、図 3-2 (b) に示すように各々のプ

ロセスにProxy / Stubと呼ばれるDLLがロードされ、OSの提供する通信機能を用いてメソッドの呼び出しやデータ交換を行います。

OPCカスタムインタフェースの場合、Proxy / Stub DLLは全てのサーバに共通に使用されます。このDLLはOPC仕様書付録のOPC.IDLから、MIDLと呼ばれるツールにより生成されます(OpcProxy.dll)。また、OPCオートメーションインタフェースの場合、OS標準のProxy / Stub DLL (oleaut32.dll)が使用されます。



(a) Local / Remote Serverの概念図



(b) Local / Remote Serverの実際の構造

図3-2 Local/Remote Serverの構造

3.1.3 Handlerの併用

Local / Remote Server (EXEサーバ)の場合、プロセス間通信が行われるため、In-Proc Server (DLLサーバ)に比べると低速です。

そこでパフォーマンス改善のため、Handlerと呼ばれる一種のサーバを併用することがOPC仕様では推奨されています。Handlerはクライアントと Local / Remote Serverの中間に位置し、サーバからのデータをキャッシュするなどの機能を持ちます。Handlerはクライアントプロセス空間内で動作し、Local / Remote Serverへのプロセスの切り替えやプロセス間通信、ネットワーク通信によるオーバヘッドを軽減することができます。

一般的に、Handlerは各Local / Remote Serverに対応してデータアクセスを最適化したものをベンダが作成して提供します。構造としてはIn-Proc Serverと同様で、Handlerの中から実際のLocal / Remote Serverに接続します。

なお、一般的なOLEの Handlerはクライアントからの特定のEXEサーバに対するアクセス時にシステムによって自動的にロードされます。(レジストリの該当サーバのInProcHandler32エントリに登録される。) しかし、OPC仕様書で規定しているHandlerは通常のDLLサーバと等価で、クライアントはHandlerを明示的に接続する必要があります。

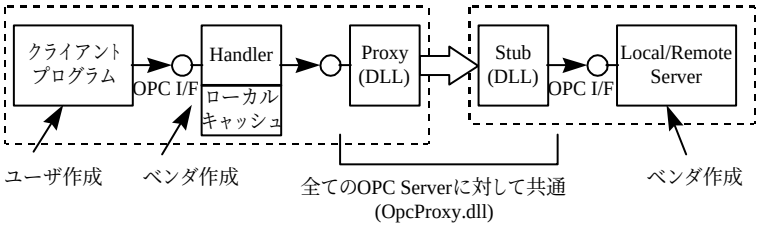


図3-3 Handlerを併用した構造

サーバの構造によるパフォーマンスの違いについては次の記事を参照してください。この記事のデータによれば、Local / Remote Serverに対して、Handlerを併用することによりかなりの性能改善が期待できることがわかります。

Microsoft System Journal 1995 December “OLE Q&A” (日本語版 1996年2月号)

3.2 OPCカスタムインタフェースとOPCオートメーションインタフェース

OPC仕様はOPCカスタムインタフェースとOPCオートメーションインタフェースと呼ばれる2種類のインタフェースを規定しています。データアクセスの機能としては2つのインタフェースはほぼ同等の機能を備えていますが、想定しているクライアントプログラムが異なります。

OPCサーバは、通常、両方のインタフェースを実装し、いずれの種類のクライアントプログラムにも対応できるようにします。

2つのインタフェースの比較を表3-1に示します。

表3-1OPCが規定するカスタムインタフェースとオートメーションインタフェースの特長

	カスタムインタフェース	オートメーションインタフェース
用途	SCADA / MES / 解析ソフトなどの専用アプリケーション向け	スクリプト言語からの簡便なアクセス向け
クライアントの使用言語	C / C++	Visual Basic, Delphiなど (C / C++からのアクセスも可能)
パフォーマンス	○	△
単一アイテムデータアクセス	△	○

OPCカスタムインタフェースはOLE / COMの基本的な機能をそのまま用いたインタフェースで、高速に動作します。

OPCオートメーションインタフェースはVisual Basic等からのアクセスを実現するOLEオートメーションインタフェースに従っています。スクリプト言語からのアクセスを容易にするための処理がオーバヘッドとなり、OPCカスタムインタフェースよりもパフォーマンスが劣ります。

オートメーションインタフェースの実装方法は複数ありますが、OPC仕様では"Dual"と呼ばれる実装方法を使用することが決められています。Dualの実装方法に従ったサーバは、Visual BasicなどのDual対応のオートメーションコントローラから呼び出された場合、IDispatch (通常のOLEオートメーション呼び出しで使用されるインタフェース)呼び出しではなく、VTABLE参照と呼ばれる呼び出し方法が使われます。これにより、OPCカスタムインタフェースに近い、高いパフォーマンスが得られます。

2つのインタフェースのパフォーマンスの違い、およびDual実装による高速化の効果は、前節で参照したMicrosoft System Journalの記事を参照してください。

3.3 OPCクライアントの構造

OPCクライアントは複数のベンダから提供されたOPCサーバを任意の個数アクセスすることが可能です。

OLE / COMの仕様により、サーバがIn-Proc Serverなのか、Local Serverなのか、Remote Serverなのかを、OPCクライアントでは全く意識する必要がありません。

また、Handlerに対しては通常のIn-Proc Serverと同様にクライアントからアクセスします。実際のデバイスに対する処理を行うLocal / Remote Serverへの接続はHandlerが行います。

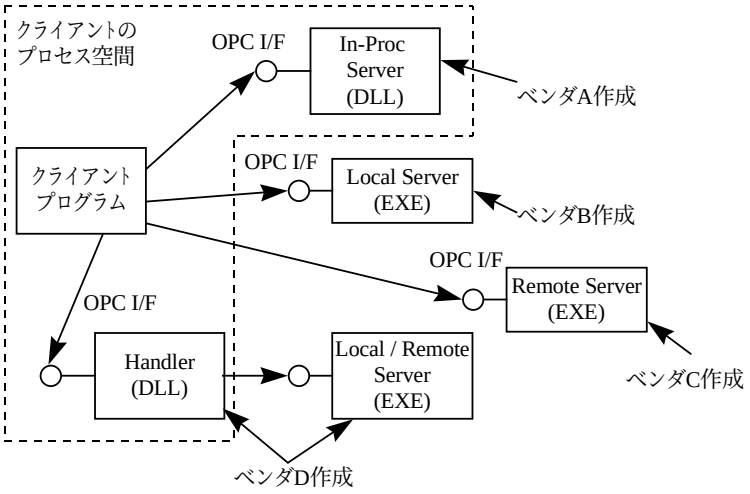


図3-4 OPCクライアントの構造

OPCサーバはそのインストール時にサーバに関するレジストリエントリ内にOPCという名前のキーを作成することになっています。OPCクライアントはレジストリからこのキーを検索することで利用可能なOPCサーバをリストアップすることができます。このリストアップ機能を実現するサンプルコードがOPC仕様書の6章に掲載されています。

3.4 データの読み込み

3.4.1 データのキャッシュ

OPC仕様ではデバイスの実データを一旦OPCサーバ内のバッファに格納する構造を想定しています。図 3-5にこの構造を示します。OPCサーバはグループ単位で設定された更新周期(UpdateRate)ごとに実際のデバイスからデータを取得しバッファに保存します。

更新周期はグループごとに設定される値です。OPCクライアントはOPCサーバに対して更新周期の希望設定値を申告します。OPCサーバは希望値にできるだけ近い更新周期を実際に設定します。実際に設定された更新周期をOPCクライアントに対して通知します。

グループやアイテムはActive属性を持ち、OPCクライアントが設定できます。この属性がFALSEに設定されるとそのグループ/アイテムのキャッシュデータ更新を行いません。各時点でOPCクライアントが使用しないグループ/アイテムに対してFALSEを設定することにより、OPCサーバの負荷を軽減することができます。

オプションの仕様である”Percent Deadband”パラメータをOPCサーバがサポートする場合、OPCサーバは更新周期ごとにキャッシュ保存値とその時点での実データを調べ、変化幅の割合が指定した値よりも小さかったときにはキャッシュデータを更新しないという動作になります。

OPCインタフェースの読み込みメソッドのパラメータには”CACHE”と”DEVICE”という属性があります。”CACHE”が指定されたとき、OPCサーバはバッファにあるデータをOPCクライアントに返します。このため処理速度は高速です。一方、”DEVICE”が指定された場合は、その都度デバイスに直接アクセスしてデータを取得します。

なお、以上のキャッシュの構造と動作は“強く推奨”されているものですが、必ずしも守らなければならない仕様ではありません。

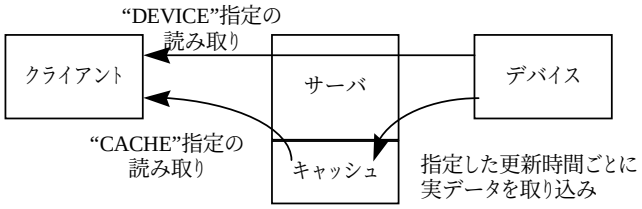


図3-5 OPCサーバの読み込みとキャッシュ

3.4.2 読み込みデータの値

OPCサーバからOPCクライアントに返されるデータ値 (Value)はVARIANT型の変数で扱われます。VARIANT型とはデータ実体とそのデータタイプを示すフラグを収めた自己記述型のデータ型です。

OPCサーバの返す値のデータタイプはOPCクライアントから要求することができます。OPCサーバが要求されたデータタイプをサポートしていない場合は警告が返され、返される値は

OPCサーバがそのアイテムに対して標準で規定するデータタイプ(正規化データ型)となります。

3.4.3 読み込み機能

OPCインタフェースではデータの読み込みについて同期読み込み、非同期読み込み、リフレッシュ、Subscriptionの4種類の方法を定めています。OPCクライアントはこれら4種類の読み込みをどのように組み合わせて使用してもかまいません。OPCサーバ側ではこの4種類の呼び出しが相互に影響を与えないように設計する必要があります。

同期読み込みを除く読み込み方法ではデータ転送のためにOPCサーバとOPCクライアント間に通信路が必要となります。これにはOLE / COMが標準で規定しているIDataObject / IAdviseSinkインタフェースによる接続が使用されます。OPCクライアント側はIAdviseSinkインタフェースを実装し、OPCサーバのIDataObject::DAdviseメソッドでこれを登録し、あらかじめ接続を確立しておく必要があります。

IAdviseSink経由のデータ転送路は各読み込み要求の応答で共通に使用されます。このため、転送されるデータの先頭にはトランザクション IDが付され、どの呼び出し要求に対応したデータなのかをOPCクライアント側で判断することができるようになっています。

3.4.3.1 同期読み込み (IOPCSyncIO::Read)[3]

OPCサーバはOPCクライアントからの要求に対してメソッドの引数にデータを格納して返します。OPCクライアントはそのメソッドがリターンするまで待ち状態となります。

OPCクライアントは読み込み方法としてCACHEまたはDEVICEを指定します。CACHE指定の場合、グループおよび各アイテムのActive属性がTRUEになっているものに対してのみ有効なデータを返します。FALSEの場合は品質フラグが“データ無効”を示します。DEVICE指定の場合、Active属性とは関係なく実データを返します。

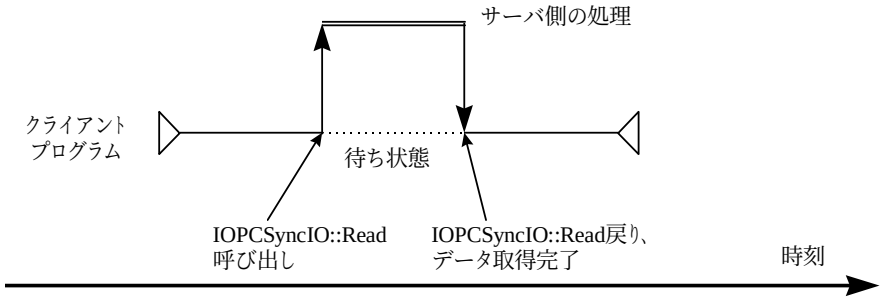


図3-6. 同期読み込みのタイムチャート

3.4.3.2 非同期読み込み (IOPCAsyncIO::Read)

OPCサーバは要求を受付けた後すぐにメソッドをリターンします。OPCクライアントはこの時点で他の処理を継続して実行できます。OPCサーバ側は要求された全アイテムのデータを返す準備ができた時点でOPCクライアント側のIAdviseSink::OnDataChangeメソッドを呼び出し、データを転送します。

[3] IOPCSyncIOインタフェースのReadメソッドを意味します。以下同様。

OPCクライアントは読み込み方法としてCACHEまたはDEVICEを指定します。CACHE指定の場合、グループおよび各アイテムのActive属性がTRUEになっているものに対してのみ有効なデータを返します。FALSEの場合は品質フラグが“データ無効”を示します。DEVICE指定の場合、Active属性とは関係なく実データを返します。

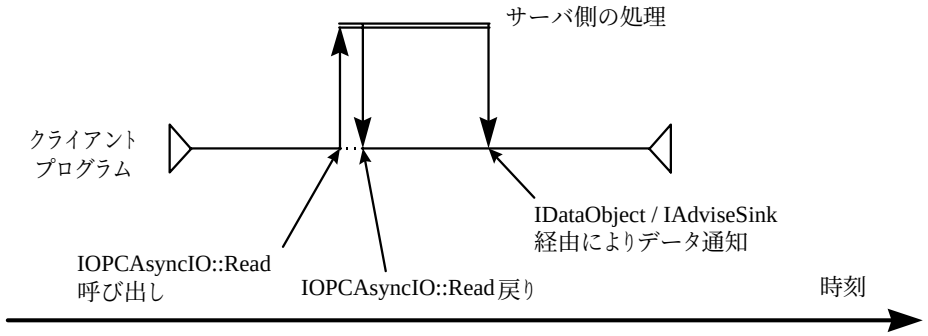


図3-7. 非同期読み込みのタイムチャート

3.4.3.3 リフレッシュ (IOPCAsynIO::Refresh)

OPCサーバはこのメソッドが呼び出された場合、グループ内のすべてのActiveなアイテムについてそのキャッシュ値を更新し、変化のあったアイテムのデータをOPCクライアント側のIAdviseSink::OnDataChangeメソッドを呼び出して転送します。

インタフェース規定上、読み込み方法としてCACHEまたはDEVICEが指定できますが、意味があるのはDEVICE指定の場合のみです。DEVICE指定でこのメソッドが呼び出されると、グループのActive属性には関係なく、グループ内のActiveなアイテムについてデバイスからデータを取得してキャッシュ値を更新し、データ転送を行います。

本メソッドの実行はグループのActive属性に無関係なため、グループをInactiveとした上でデータが必要なときに本メソッドを呼び出すことが可能です。このようにすると、メソッド呼び出し時にのみデータが更新･転送されるのでサーバの負荷が軽減されます。次節で説明するSubscription動作は不要だが非同期の読み込み動作が必要というクライアントの要求に対してはこの使用方法が適しています。

本メソッドによるキャッシュ値更新はグループの更新周期(UpdateRate)に基づくキャッシュ更新の自動実行タイミングには影響を与えません。たとえば更新周期が1時間に設定されていて、自動更新から45分後にリフレッシュを実行したとすると、さらにその15分後にキャッシュの更新が自動的に実行されます。

(注) OPC仕様書の原文では本仕様の規定に不明瞭な部分がありますが、次回改訂版で修正される予定です。

3.4.3.4 Subscription

OPCサーバはグループの更新周期(UpdateRate)ごとにOPCサーバ内のキャッシュ値を更新します。OPCクライアント/サーバ間にIAdviseSinkによる接続が確立されている場合、OPCサーバはキャッシュ更新時にデータ値の変化をチェックし、変化があったアイテムについてそのデータをOPCクライアントのIAdviseSinkへ自動的に送信します。このデータ送信はトランザクション IDとして0という特別な値を設定するため、OPCクライアントは他の非同期呼び出しやリフレッシュによるデータと区別することができます。

OPCサーバが“Percent Deadband”パラメータをサポートしている場合、アナログデータに関する上記のキャッシュ更新および通知を、指定した割合を超える変化が生じたアイテムのみに限定することができます。

3.4.3.5 まとめ

表3-2読み込み動作のまとめ

読み込み方法	ソース	グループ	アイテム	サーバの動作
同期読み込み	Cache	Active	Active	指定されたアイテムについて有効なキャッシュデータを返す。
	Cache	Active	Inactive	指定されたアイテムについて“無効”品質フラグを持つデータを返す。
	Cache	Inactive	×	指定されたアイテムについて“無効”品質フラグを持つデータを返す。
	Device	×	×	指定されたアイテムの現在のデバイス値を返す。
非同期読み込み	Cache	Active	Active	指定されたアイテムについて有効なキャッシュデータを送る。
	Cache	Active	Inactive	指定されたアイテムについて“無効”品質フラグを持つデータを送る。
	Cache	Inactive	×	指定されたアイテムについて“無効”品質フラグを持つデータを送る。
	Device	×	×	指定されたアイテムの現在のデバイス値を送る。
リフレッシュ	Cache	×	×	（無意味）
	Device	×	Active	キャッシュを更新し、値が変化したアイテムすべてのデータを送る。
	Device	×	Inactive	このアイテムのデータは送らない。
subscription	×	Active	Active	キャッシュ値が変化したアイテムすべてのデータを送る。
	×	Active	Inactive	このアイテムのデータは送らない。
	×	Inactive	×	このアイテムのデータは送らない。

×＝いずれの設定値でも同じことを示す。

同期読み込みではメソッドの戻り値としてデータを返す。

そのほかの読み込みではOPCサーバがIAdviseSinkのOnDataChangeメソッドを呼び出してデータを送る。

3.5 データの書き込み

3.5.1 書き込みデータの値

OPCクライアントは、デバイスに対して書き込みたい値をVARIANT型変数でOPCサーバに渡します。

OPCクライアントから渡されたデータのデータタイプが対象アイテムに対して適切でない（OPCサーバがサポートしていない）場合にはエラーとなります。

3.5.2 データの書き込み機能

OPC仕様ではデータの書き込みについて同期書き込み、非同期書き込みの2種類の方法を定めています。OPCクライアントはこれら2種類の書き込みを組み合わせて使用してもかまいません。OPCサーバ側ではこの2種類の呼び出しが相互に影響を与えないように設計する必要があります。

なお、書き込み動作に関してはグループやアイテムのActive属性の値は影響を与えません。また、書き込み要求に対してはOPCサーバは実際のデバイスへの書き込み完了（または失敗確認）まで実行する仕様になっており、キャッシュ動作は無関係です。

3.5.2.1 同期書き込み (IOPCSyncIO::Write)

OPCクライアントは書き込みたいアイテム（複数可）の値を引数にこのメソッドを呼び出します。OPCサーバは実際にデバイスへの書き込みを実行し、全ての要求された書き込みが完了（または実行失敗を確認）するまでリターンしません。OPCクライアントはメソッドが完了するまで待ち状態となります。

3.5.2.2 非同期書き込み (IOPCAsyncIO::Write)

OPCクライアントは書き込みたいアイテム（複数可）の値を引数にこのメソッドを呼び出します。OPCサーバは要求を受付けた後すぐにリターンします。OPCクライアントはこの時点で他の処理を継続して実行できます。

OPCサーバは実際にデバイスへの書き込みを実行し、全ての要求された書き込みが完了（または実行失敗を確認）した時点でOPCクライアントに処理完了を通知します。

OPCクライアントへの処理完了通知には非同期読み込みと同じくIAdviseSink::OnDataChangeメソッドの呼び出しが使われます。OPCクライアントはOPCサーバとの間に3．4節にて説明したIDataObject / IAdviseSinkによる通信路をあらかじめ設定しておく必要があります。

3.6 データの同期 (Synchronization)

ここで同期と呼んでいるのは1回の処理で読み込みあるいは書き込みされる複数データの同時性のことをさしています。例として次のような問題があります。

- すべてのアプリケーションはOPCサーバから返されるあるアイテムに対するデータの値、品質フラグ、タイムスタンプの同期がとれていることが必須と考えられます。
- バッチ処理ではレポート作成に必要な複数の読み込みデータの同期が取れていること、またバッチ開始前に必要な指示データが全てダウンロードされていることが必要と考えられます。

これらのデータの読み書きの同期はOPCインタフェース単独では保証していません。

サーバをマルチスレッドとした場合、処理スレッドが待ち状態になる場合があり、その再開時にはデータの処理時刻がずれてしまいます。通常、処理の同期が必要なデータにはロックをかけて対応しますが、その分だけ他の処理待ちが長くなる可能性が高くなります。処理待ちが増えると、サーバとしての処理性能は低下します。

したがって、サーバの対象とする実際のデバイスでのデータ同期の必要性を性能とのトレードオフで十分検討し、サーバを設計することが必要となります。

3.7 データの順序保証

すべてのOPCサーバは同一デバイスに対する書き込み要求に対して要求された順序に従って処理を実行します。

たとえば、バッチ処理で考えると、書き込み順序が逆転した場合、必要なデータのダウンロード前にバッチ開始フラグが立ってしまうというようなことが生じてしまいます。このため、データ書き込みの順序保証が必要となります。

OPCサーバがデータの順序保証を行うことはOPCのインタフェース規定として必須ではありませんが、“強く推奨”されています。

3.8 サーバのマルチスレッド化

サーバのパフォーマンスを高めるためにはマルチスレッド化するのが有効です。OPC仕様ではサーバがどのようなスレッディングモデルを用いるかは定めていませんが、その処理内容から考えてフリースレッディングモデルを適用するのが妥当です。

このモデルによるマルチスレッド化ではスレッドセーフの保証はすべてサーバの責任となります。前述のデータの同時性や書き込み順序保証などの設計に十分注意が必要です。

フリースレッディングモデルの詳しい説明は下記を参照してください。

Microsoft System Journal 1996 May “Introducing Distributed COM and the New OLE Features in Windows NT 4.0” (日本語版 1996年6月号)

4 OPC対応ソフトウェア開発

本章ではOPC仕様に準拠したOPCサーバ／OPCクライアントソフトウェア開発の概要を説明します。OPC仕様書ではあまり触れられていない、開発に必要な資格、知識、項目、環境についての概要を記述します。さらに詳しい実装方法や開発手順については、“OPC実装ガイド”に解説される予定です。

まず、OPCサーバの開発概要を説明し、OPCクライアントの開発については“4.6 OPCクライアントの開発”でまとめて説明します。

4.1 開発資格

OPCのロゴを使用したOPCサーバを製品化するにはOPC-J会員となるべきです。OPC-J会員になることにより次のような特典が得られます。

- ・OPCのロゴ使用と製品カタログ掲載
- ・サンプルコード取得
- ・各種ドキュメント取得
- ・最新情報入手
- ・技術サポート

なお、OPCサーバを開発するために必ずしもOPC-Jの会員となる必要はありません。公開されている仕様書に基づき開発し、販売することができます。しかし、この場合はOPCのロゴを使用して製品化することはできません。

4.2 必要な知識

OPCサーバを作成するには以下の知識を必要とします（詳細はOPC-Jから発行予定のOPC実装ガイドを参照してください）。また、各項目に関しては6章の技術基盤説明を参考にしてください。

表4-1 OPCサーバ作成に必要な知識

分 類	項 目
OLE/COMの一般知識	インタフェースの定義
	OLEコンポーネントの実装形態
	マーシャリングの概要
OLE標準インタフェースの知識	OLEオートメーションの実装形態
	統一データ転送の実装形態
OPCの知識	OPCで規定されたインタフェースの知識

また、上記の前提とする知識以外に、OPC-Jで開催が予定されている技術セミナーを受講することを推奨します。

4.3 開発項目

3章で説明したように、OPCサーバの構造には、In-Proc Server、Local/Remote Server(In-Proc Handlerなし)、Local/Remote Server(In-Proc Handlerあり)が存在します。すべての形式のサーバを開発する必要はなく、サーバベンダにとって適切なタイプのサーバを用意することになります。これらの特徴については、“3章 構造と動作”を参照してください。それぞれの開発には次の項目が必要となります。

4.3.1 In-Proc Server

(1)DLL形式のサーバ
OPC-Fから提供されるサンプルコードを利用して作成することができます。

(2)タイプライブラリ
OPC-Fから提供されるODL(オブジェクト記述言語)ファイルをMIDLコンパイラによりコンパイルして作成します。

(3)インストーラ
サーバ、タイプライブラリのインストールを行うとともに、レジストリにサーバオブジェクトの情報を登録するものです。

4.3.2 Local/Remote Server(In-Proc Handlerなし)

(1) EXE形式のサーバ
OPC-Fから提供されるサンプルコードを利用して作成することができます。

(2) Proxy/Stub DLL
OPC-Fから提供されるIDL(インタフェース定義言語)ファイルをMIDLコンパイラによりコンパイルして作成します。

(3) タイプライブラリ
OPC-Fから提供されるODL(オブジェクト記述言語)ファイルをMIDLコンパイラによりコンパイルして作成します。

(4) インストーラ
サーバ、タイプライブラリ、Proxy/Stub DLLのインストールを行うとともに、レジストリにサーバオブジェクトの情報を登録するものです。

4.3.3 Local/Remote Server(In-Proc Handlerあり)

(1) EXE形式のサーバ
OPC-Fから提供されるサンプルコードを利用して作成することができます。

(2) DLL形式のIn-Proc Handler
OPC-Fから提供されるサンプルコードを利用して作成することができます。

(3) Proxy/Stub DLL
OPC-Fから提供されるIDL(インタフェース定義言語)ファイルをMIDLコンパイラによりコンパイルして作成します。

(4) タイプライブラリ
OPC-Fから提供されるODL(オブジェクト記述言語)ファイルをMIDLコンパイラによりコンパイルして作成します。

(5) インストーラ
サーバ、In-Proc Handler、タイプライブラリ、Proxy/Stub DLLのインストールを行うとともに、レジストリにサーバオブジェクトの情報を登録するものです。

なお、サンプルコードはOPC-Fより提供が予定されていますが、現在詳細な内容は不明です。C++で記述されたコードで基本的な機能が盛り込まれたスケルトンとなる模様です。
また、インストーラはOPC仕様で規定されているものではありませんが、製品としては必須のものです。これについてのサンプルコードはありません。

4.4 開発プラットホームと開発ツール

(1) 開発プラットホーム - - - Windows NT 4.0以降

OPCサーバを開発するプラットホームとしては、Windows NT4.0以降が必要です。今後、他のプラットホームでも開発できる環境が整備されるかもしれませんが、現状はWindows NT4.0のみしか動作を確認していません。

(2) 開発ツール - - - Microsoft Visual C++ 4.2以降

OPCサーバを開発するツールとしては、Microsoft Visual C++ 4.2以降が必要です。他の開発ツールでもMicrosoft Visual C++4.2と同様の機能を持つ開発ツールであれば、開発することは可能ですが、OPC-Fが提供するサンプルコードや今後、作成が計画されているOPC実装ガイドではMicrosoft Visual C++を前提に記述されることが予定されており、他の開発ツールはサポートされません。また、Microsoft Visual C++4.1以前のバージョンでは、一部、開発に支障をきたします。

4.5 認証

OPCのロゴを使用するにはOPC-Fが規定する認証テストを受けて認証を取得する必要があります。

4.6 OPCクライアントの開発

製品としてOPCクライアントを開発する場合はOPCサーバを開発するのと同等の開発資格と認証が必要になります。製品としてではなく、各ベンダから購入したOPCサーバを単に利用するためのOPCクライアント開発には特に開発資格、認証の規定は適用されません。いずれの場合も必要とする知識はOPCサーバ開発の場合と比較して非常にわずかです。

OPCクライアントの開発ではOPCカスタムインタフェースで開発する場合とOPCオートメーションインタフェースで開発を行う場合で必要な知識、開発ツールが異なります。

OPCカスタムインタフェースでのOPCクライアント開発

CおよびC++言語での開発。

OPCオートメーションインタフェースでのOPCクライアント開発

Visual BasicやVBA (Visual Basic for Applications) での開発。

以下にOPCクライアントを開発する場合の必要な知識、開発プラットホームと開発ツールについて記述します。なお、各項目に関しては“6章 技術基盤説明”を参考にしてください。

4.6.1 必要な知識

表4-2 OPCカスタムインタフェースで開発する場合の必要な知識

分 類	項 目
OLE/COMの一般知識	インタフェースの定義
OLE標準インタフェースの知識	統一データ転送の概要
OPCの知識	OPCで規定されたインタフェースの知識

表4-3 OPCオートメーションインタフェースで開発する場合の必要な知識

分 類	項 目
OLE標準インタフェースの知識	OLEオートメーションの概要
OPCの知識	OPCで規定されたインタフェースの知識

4.6.2 開発プラットホームと開発ツール

4.6.2.1 OPCカスタムインタフェースで開発する場合

開発プラットホーム - - - Windows 95あるいはWindows NT 4.0以降

OPCカスタムインタフェースでOPCクライアントを開発するプラットホームとしてはWindows 95あるいはWindows NT 4.0以降が必要です。今後、他のプラットホームでも開発できる環境が整備されるかもしれませんが、現状はWindows 95あるいはWindows NT 4.0のみでしか動作を確認していません。

開発ツール - - - Microsoft Visual C++ 4.2以降

OPCカスタムインタフェースでOPCクライアントを開発するツールとしてはMicrosoft Visual C++ 4.2以降が必要です。他の開発ツールでもMicrosoft Visual C++4.2と同様の機能を持つ開発ツールであれば開発することは可能ですが、OPC-Fとして提供するサンプルコードや今後、作成が計画されているOPC実装ガイドではMicrosoft Visual C++を前提に記述されることが予定されており、他の開発ツールはサポートされません。また、Microsoft Visual C++4.1以前のバージョンでは、一部、開発に支障をきたす場合があります。

4.6.2.2 OPCオートメーションインタフェースで開発する場合

開発プラットホーム - - - Windows 95あるいはWindows NT 4.0以降

OPCクライアントを開発するプラットホームとしてはWindows 95あるいはWindows NT 4.0以降が必要です。今後、他のプラットホームでも開発できる環境が整備されるかもしれませんが、現状はWindows 95あるいはWindows NT 4.0のみでしか動作を確認していません。

開発ツール - - - Visual Basic 4.0 Professional Edition以降およびVBA（Visual Basic for Applications）の機能を持つ開発ツール

他の開発ツールでもOLEオートメーションのクライアント機能を持つ開発ツールであれば開発することは可能ですが、OPC-Fとして提供される予定のサンプルコードやOPC仕様書の資料ではVisual Basic以外の開発ツールはサポートされません。

5 OPCアプリケーションとOPCインタフェース

5.1 OPCアプリケーション

OPCはOLE/COMの技術を基盤としたデータ交換インタフェースです。OLE/COMの技術を基盤とするOCXやActiveXコントロールへの応用などさまざまな部分で実装することが可能です。この節ではOPCアプリケーションの適用例やOPCインタフェースを使用したサンプルプログラムについて記述します。

5.1.1 アプリケーションへの適用

アプリケーションとOPCインタフェースの結合および実装についてはさまざまな方法が考えられます。プロセス制御への適用例としては、ネットワークモニタ、デバイスステータス、履歴管理、デバイス制御、ファイル、データ、プログラムのアップロードおよびダウンロードなどで実装可能です。OPC仕様に基づいたインタフェースに準拠していれば、クライアント／サーバ間のデータ交換やコンフィグレーションなどのあらゆる通信を簡単に実行することができます。次の例ではアプリケーションとハードウェアおよび外部アプリケーションとのOPCインタフェースでの適用例を示しています。

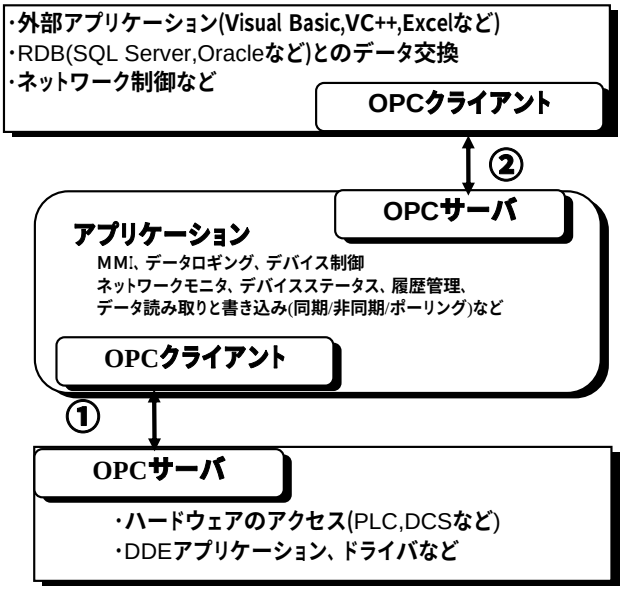


図5-1 OPCの適用例

(1)ハードウェアとのインタフェース (①)

アプリケーションとハードウェア間のインタフェースに使用する例です。この場合、OPCサーバはハードウェアとのインタフェースを制御するI/Oドライバプログラムになります。アプリケーション（OPCクライアント）はOPCサーバへの要求により、ハードウェアのデータや設定情報などを取得、設定することができます。

(2)外部アプリケーションとのインタフェース (②)

アプリケーションと外部アプリケーションとのデータ交換を行うインタフェースの例です。この場合の外部アプリケーションとは、Visual BasicやVisual C++などで作成したカスタムアプリケーションやRDBとのデータ交換を実行する制御アプリケーションなどです。アプリケーションにOPCのインタフェース(OPCサーバ機能)を実装することにより、外部アプリケーションからのアクセスが簡単に実行できます。

5.1.2 アプリケーションからオブジェクトへのアクセス

OPCオートメーションインタフェースはOLEオートメーションインタフェースに従っており、あるアプリケーションが別のアプリケーションで実装されたコンポーネントを操作したり、別のアプリケーションが操作できるようにコンポーネントをエクスポーズ(公開)することを可能にします。OLEオートメーションオブジェクトにアクセスするアプリケーションのことを、OLEオートメーションコントローラと呼びます。OLEオートメーションコントローラは、別のアプリケーションに存在するOLEオートメーションオブジェクトを作成し、OLEオートメーションオブジェクトがサポートするプロパティとメソッドを使用してこれらのオブジェクトを簡単に操作することができます。このしくみは、オブジェクトが管理するデータ領域や機能を、あたかも自分自身のアプリケーションのものであるかのように扱えるようにしています。

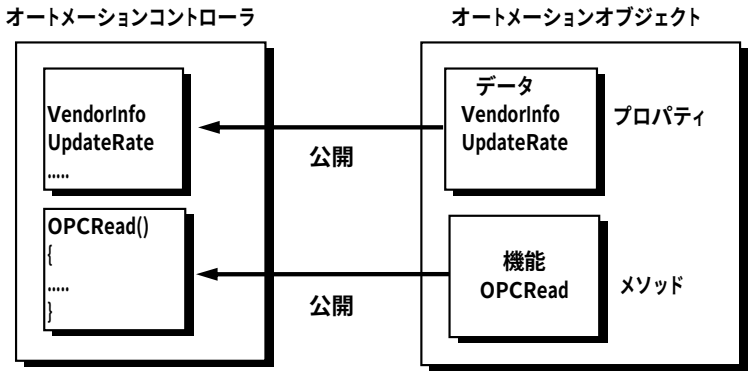


図5-2 オートメーションコントローラとオートメーションオブジェクト

Visual Basic を使用して、オートメーションオブジェクトのプロパティに値をセットするには、

<オブジェクト変数>.<プロパティ名> = <値>

と記述します。また、オブジェクトからプロパティ値を取得するには、

<格納する変数> = <オブジェクト変数>.<プロパティ名>

と記述します。これらの構文を使用することにより、アプリケーションからオートメーションオブジェクトに簡単にアクセスすることが可能です。

```
Sub Sample

Dim X1 As Object
Dim J As String
Dim Itm as IOPCItemDisp
Dim mValue as variant
Set X1 = CreateObject("OPC.Sample.1")
J = X1.VendorInfo                          ' サーバ情報の取得
Text1.Text = J                             ' データの表示
Itm.OPCRead OPC_DS_CACHE, mValue           ' アイテムからデータ読み込み
mValue = 10
Itm.OPCWrite mValue                        ' アイテムへのデータ書き込み
End Sub
```

リスト5-1 OPCオートメーションインタフェースの使用例

5.1.3 OLEコントロール、ActiveXへの応用

OLEコントロール(OCX)は、さまざまな開発環境で使用することができる、OLEを介して機能している再利用可能なソフトウェアコンポーネント（汎用性のある部品）です。OLEコントロールは機能を拡張したIn-Proc Serverです。OLEコントロールには①32ビット環境で使用できる部品規格、②データ取り込み容量の制限が大幅に緩和されている、③プロパティなどで容易にカスタマイズ可能、などの優れた特徴があります。

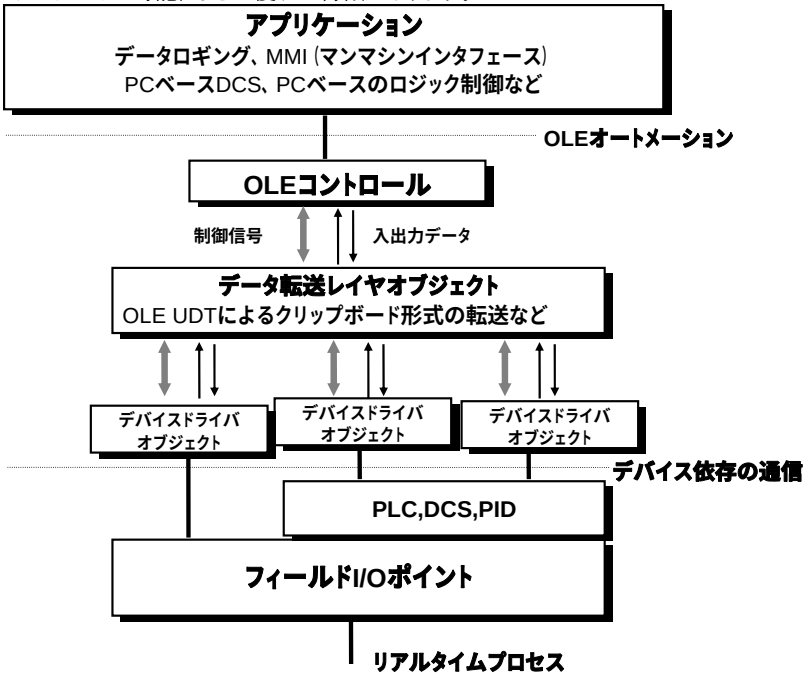


図5-3 OPCコンポーネントを連携させたシステム構成

図5-3 はOLEコントロールとOLEオートメーションを使用して、OPCコンポーネントを連携させ、プロセス制御アプリケーションを構築する例です。OLEオートメーションインタフェースを使用して、各種デバイスドライバオブジェクトを経由し、各ハードウェアにアクセスしていま

す。アプリケーションとハードウェアとのインタフェースをオブジェクトインタフェース、OLEインタフェースで標準化しています。

現在、OLEコントロールは以下のような製品が入手可能です。

・APPGALLERY Object ― フォーム部品、グラフ、レポート

・APPGALLERY Multi Media Object ― 静止画/動画/音声/CDオーディオデータ

・CCC2 ― 文字入力やボタンなどの実用的なコントロール

・Chart Builder OCX ― 3次元グラフ

・Image Gear ― 各種イメージデータに対応した、開発支援コントロール

・Spread ― 各種プロパティをダイアログで設定できる表計算コントロール

・Distinct TCP/IP OCX ― TCP/IPアプリケーション（ftp, telnet）

・PDQ Comm OCX ― 通信制御コントロール

・VBMan for RS-232C/OCX32 ― RS232Cの制御コントロール

このようなOLEコントロールとOLEオートメーションを組み合わせて開発することにより、アプリケーションで必要な機能をすべて新規に開発する必要はなくなります。すなわち、アプリケーション内で利用できるところはコンポーネントとして組み込み、それを利用できます。開発者はアプリケーションの本来の機能部分の開発に専念することができます。

ActiveXはインターネットに対応するためにいろいろな工夫が追加されています。インターネット上で遠隔地のサーバからダウンロードされることを前提に設計されており、通信負荷を下げる目的で、よりモジュールサイズが小さくなるようにシンプルに設計されています。ActiveXコントロールは他のActiveXコントロールまたは任意のCOMオブジェクトで動作します。

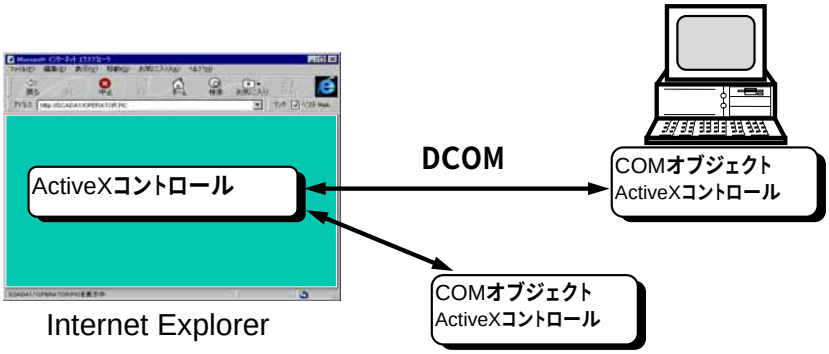


図5-4 ActiveXコントロールの例(1)

図5-5は多数のプロセス制御用のActiveXコントロールをインターネットエクスプローラ上で表示させている例です。リアルタイムチャートやメータなどの各部品のデータ表示はネットワーク上のリモートPCからのデータを表示することも可能です。

図5-5 ActiveXコントロールの例(2)

5.1.4 OPCカスタムインタフェース サンプルプログラム

リスト5-3およびリスト5-4に示すサンプルプログラムはOPCカスタムインタフェースを使用した簡単なOPCクライアントとOPCサーバ（DLLサーバ）の例です。

このサンプルプログラムはOLEの基本的な初期化処理の後、OPCインタフェースによりOPCサーバからベンダ情報を取得し、OPCグループを作成します。このプログラムを実行すると実行結果は以下のようになります。

```
Object (with IUnknown) Created for OPC.Fix.1
Status wReserved = 42
Status szVendorInfo = OPC Server for XXX V0.00
(Result GetStatus : Vendor Information)
GetState Succeeded for TestGroup
No IDispatch Found
Done...
```

リスト5-2 サンプルプログラムの実行結果

*リスト5-3 OPC **サンプルプログラム（クライアント）** - OPCTEST.CPP*

```
//

// (c) Copyright 1996, Intellution Inc
// ALL RIGHTS RESERVERD
//
//      Al Chisholm, Intellution Inc      June 1996
//                                        Nov 1996 - add missing Release()
//                                        Reference IIDs via midl generated opc_i.c file
//
#include <stdio.h>
#include <conio.h>
#include "opc.h"            // Include the GENERIC OPC header file
//-------------------------------------------------------------
// Local Functions
void LocalInit(void);
IUnknown * LocalCreateOPCServer(WCHAR* szProgID);
void LocalTryOPCServer(IUnknown * pOPC);
void LocalTryAutomation(IUnknown * pOPC);
void LocalTryGetStatus(IOPCServer * pOPC);
void LocalTryAddGroup(IOPCServer * pOPC);
void LocalCleanup(void);
// Global interface to the COM memory manager
IMalloc *pIMalloc;

//-----------------------------------------------------------------------------------------------
// main
void main(void)
{
        IUnknown *pOPC;

        LocalInit();

        pOPC = LocalCreateOPCServer(L"OPC.Fix.1");

        if(pOPC)
        {
                // Try using the custom IOPCServer interface if present
                // and also the IDispatch interface if present
                LocalTryOPCServer(pOPC);
                LocalTryAutomation(pOPC);
                pOPC->Release();
        }

        LocalCleanup();

        printf("Done...\n");
        getch();
        exit(0);
}

//-----------------------------------------------------------------------------------------------
// LocalInit
// This is generic initialization for a task using COM
void LocalInit(void)
{
        HRESULT             r1;
        // General COM initialization...
        //
        r1 = CoInitialize(NULL);

        if (FAILED(r1))
        {
                printf("Error from CoInitialize\n");
                exit(1);
        }

        // Also get access to the COM memory manager
        //
        r1 = CoGetMalloc(MEMCTX_TASK, &pIMalloc);

        if (FAILED(r1))
        {
                printf("GetMalloc failed\n");
```

```
                CoUninitialize();
                exit(1);
        }
}

//-----------------------------------------------------------------------------------------------
// LocalCreateOPCServer
// Create the requested OPC Server
IUnknown *LocalCreateOPCServer(WCHAR*szName)
{
        CLSID clsid;
        IClassFactory *pCF;
        HRESULT r1, r2, r3;
        IUnknown * pOPC;

        // Get the CLSID from the Name
        r1 = CLSIDFromProgID(szName, &clsid);
        if(FAILED(r1))
        {
                printf("CLSIDFromProgID failed for %ls\n", szName);
                return NULL;
        }

        // Create an OPC Sample Server Class Factory
        //
        r2 = CoGetClassObject(clsid, CLSCTX_INPROC_SERVER , //try inproc first
                NULL, IID_IClassFactory, (void**)&pCF);

        if (FAILED(r2))
        {
                printf("CoGetClassObject- no InProc server for (%lx)\n", r2);

                // try local if no inproc
                r2 = CoGetClassObject(clsid, CLSCTX_LOCAL_SERVER,
                        NULL, IID_IClassFactory, (void**)&pCF);

                if (FAILED(r2))
                {
                        printf("CoGetClassObject- no Local server for (%lx)\n", r2);
                        printf("**Unable to create server**\n");
                        return NULL;
                }
        }

        // And use the class factory to create the OPC Server
        // Request an IUnknown Interface to it
        // and release the class factory which is no longer needed
        //
        r3 = pCF->CreateInstance(NULL, IID_IUnknown, (void**)&pOPC);

        pCF->Release();

        if (FAILED(r3))
        {
                printf("Error from CreateInstance (%lx)\n", r3);
                return NULL;
        }
        printf("Object (with IUnknown) Created for %ls\n", szName);
        return pOPC;
}

//-----------------------------------------------------------------------------------------------
// LocalTryOPCServer
// Use the OPCServer to perform some functions
void LocalTryOPCServer(IUnknown * pUNK)
{
        HRESULT r4;
        IOPCServer *pOPC;

        // Request an IOPCServer interface from the object.
        //
        r4 = pUNK->QueryInterface(IID_IOPCServer, (void**)&pOPC);
        if (FAILED(r4))
        {
                printf("No IOPCServer Found\n");
                return;
```

```
        }

        // Try out a couple of methods on the Server
        //
        LocalTryGetStatus(pOPC);
        LocalTryAddGroup(pOPC);

        //Release the OPCServer interface now that we are done with it.
        //
        pOPC->Release();
}

//--------------------------------------------------------------------------------------------
// LocalTryGetStatus
// Use the OPCServer to perform some functions
void LocalTryGetStatus(IOPCServer * pOPC)
{
        OPCSERVERSTATUS         *pss;
        HRESULT r1;

        // Invoke a method on the OPCServer interface
        //
        r1 = pOPC->GetStatus(&pss);

        if (FAILED(r1))
        {
                printf("Error from GetStatus(%lx)\n", r1);
        }
        else
        {
                // And print some of the results of the method
                //
                printf("Status.wReserved = %d\n", pss->wReserved);
                printf("Status.szVendorInfo = %ls\n", pss->szVendorInfo);

                // Dont forget to release the memory returned by the method
                //
                pIMalloc->Free(pss->szVendorInfo);
                pIMalloc->Free(pss);
        }
}

//--------------------------------------------------------------------------------------------
// LocalTryAddGroup
// Use the OPCServer to perform some function
void LocalTryAddGroup(IOPCServer * pOPC)
{
        HRESULT r1;
        DWORD  hServerGroup;
        DWORD  RevisedRate;
        IOPCGroupStateMgt * pGRP;
        float       DeadBand = (float)0.0;

        // Try to create a group
        //
        r1 = pOPC->AddGroup(L"TestGroup", TRUE, 0, 0, 0, &DeadBand,
                0, &hServerGroup, &RevisedRate, IID_IOPCGroupStateMgt,
                (LPUNKNOWN*)&pGRP);

        if (FAILED(r1))
        {
                printf("Error from AddGroup(%lx)\n", r1);
        }
        else
        {
                DWORD     UpdateRate;
                BOOL      Active;
                LPWSTR    pName;
                LONG      TimeBias;
                FLOAT     PercentDeadband;
                DWORD     LCID;
                OPCHANDLE hClientGroup;
                OPCHANDLE hServerGroup;

                // If it created OK then get it's status
                //
```

```
                r1 = pGRP->GetState(&UpdateRate, &Active,
                        &pName, &TimeBias, &PercentDeadband,
                        &LCID, &hClientGroup, &hServerGroup);

                if (FAILED(r1))
                {
                        printf("Error from GetState(%lx)\n", r1);
                }
                else
                {
                        // Print the name (to verify it worked)
                        // And don't forget to Free the returned string!
                        //
                        printf("GetState Succeeded for %ls\n", pName);
                        pIMalloc->Free(pName);
                }


                // When done
                // Release the Group interface
                // and remove the group from the server
                //
                pGRP->Release();
                r1 = pOPC->RemoveGroup(hServerGroup, FALSE);
        }
}

//--------------------------------------------------------------------------------------------------
// LocalTryAutomation
// Use the OPCServer to perform some functions via Automation
// This is seldom done from C but is useful for test & debug
//
void LocalTryAutomation(IUnknown * pUNK)
{
        HRESULT r4;
        IDispatch *pOPCAuto;
        DISPID dispid;
        DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};
        VARIANT varResult;
        WCHAR  *szMember = L"BuildNumber";

        // Try to get an IDispatch
        //
        r4 = pUNK->QueryInterface(IID_IDispatch, (void**)&pOPCAuto);

        if (FAILED(r4))
        {
                printf("No IDispatch Found\n");
                return;
        }

        // Get the dispid for the 'BuildNumber' property
        // Note lcid=0
        //
        r4 = pOPCAuto->GetIDsOfNames(IID_NULL, &szMember, 1,
                0, &dispid) ;
        if (FAILED(r4))
        {
                printf("DispID Not Found\n");
                pOPCAuto->Release();
                return;
        }

        // and use that dispid to invoke the 'get' method
        //
        r4 = pOPCAuto->Invoke(
                dispid,
                IID_NULL,
                0,
                DISPATCH_PROPERTYGET,
                &dispparamsNoArgs, &varResult, NULL, NULL);
        if (FAILED(r4))
        {
                printf("Invoke Failed\n");
                pOPCAuto->Release();
                return;
```

```
        }

        // And print some of the results of the method
        //
        printf("Result.type = %d, value = %d\n", (int)varResult.vt,varResult.iVal);

        // Dont forget to release the interface
        //
        pOPCAuto->Release();
}

//---------------------------------------------------------------------------------------------
// LocalCleanup
// This is generic cleanup for any task using COM.
void LocalCleanup(void)
{
        // Finally, release the memory manager
        // as well as COM
        //
        pIMalloc->Release();
        CoUninitialize();
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

リスト5-4 OPCサンプルプログラム（サーバ）- I_SERVER.CPP

```
//
// I_Server.cpp
//
//  This file contains the implementation of
//  the IOPCServer interface for the XXX server.
//
//
//      (c) COPYRIGHT 1996, INTELLUTION INC.
// ALL RIGHTS RESERVED
//
// Original Author: Al Chisholm
//
// Modification Log:
//      Vers    Date   By    Notes
//      ----  -------- ---   -----
//      0.00  11/18/96 ACC
//
//
#define WIN32_LEAN_AND_MEAN
#include "windows.h"
#include "OLE2.h"

#include "OPCXXX.h"

////////////////////////////////////////////////////////////////////////////
// Constructor /Destructor functions
//

////////////////////////////////////////////////////////////////////////////
// IXXXServer()
//   Constructor for this Interface
//
////////////////////////////////////////////////////////////////////////////
IXXXServer::IXXXServer( LPUNKNOWN parent )
{
        m_Parent = (XXXServer *)parent;
}

////////////////////////////////////////////////////////////////////////////
// ~IXXXServer()
//   Destructor for this Interface
//
////////////////////////////////////////////////////////////////////////////
IXXXServer::~IXXXServer( void)
{
}

////////////////////////////////////////////////////////////////////////////
```

```
// IUnknown functions Delegate to Parent
//

STDMETHODIMP_(ULONG) IXXXServer::AddRef( void)
{
	return m_Parent->AddRef();
}

STDMETHODIMP_(ULONG) IXXXServer::Release( void)
{
	return m_Parent->Release();
}

STDMETHODIMP IXXXServer::QueryInterface( REFIID iid, LPVOID* ppInterface)
{
	return m_Parent->QueryInterface(iid, ppInterface);
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer (IOPCServer) interface functions
//

static		WCHAR	VendorInfo[] = L"OPC Server for XXX V 0.00 \n(Result GetStatus : Vendor Information)";

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::GetStatus()
//   This function fills in the OPCSERVERSTATUS structure that was passed in.
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP IXXXServer::GetStatus( OPCSERVERSTATUS** ppServerStatus)
{
	OPCSERVERSTATUS* pServerStatus;

	if ( pServerStatus == NULL)
		return E_INVALIDARG;

	// allocate some memory for the struct
	pServerStatus = (OPCSERVERSTATUS*) pIMalloc->Alloc( sizeof(OPCSERVERSTATUS) );
	if(pServerStatus)
	{
		// and for the serverinfo string
		pServerStatus->szVendorInfo = (WCHAR *) pIMalloc->Alloc( sizeof(WCHAR) *
(wcslen(VendorInfo)+1) );
		if( pServerStatus->szVendorInfo )
		{


			pServerStatus->ftStartTime = serverStartTime;
			CoFileTimeNow( &pServerStatus->ftCurrentTime);
			pServerStatus->ftLastUpdateTime = m_Parent->mLastUpdate;

			pServerStatus->dwServerState = OPC_STATUS_RUNNING;
			pServerStatus->dwGroupCount = 0;
			pServerStatus->dwBandWidth = 0;

			pServerStatus->wMajorVersion = 0;
			pServerStatus->wMinorVersion = 0;
			pServerStatus->wBuildNumber = 0;
			pServerStatus->wReserved = 42;

			wcscpy(pServerStatus->szVendorInfo, VendorInfo);
		}
		else
		{
			// else the string alloc failed so free the struct
			pIMalloc->Free(pServerStatus);
		}
	}
	else
	{
		// else the struct alloc failed
	}

	// return the result (if any) and the error
	*ppServerStatus = pServerStatus;

	if(pServerStatus) return S_OK;
```

```
        return E_OUTOFMEMORY;
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::GetErrorString()
//   For server specific error codes we need to return a user displayable string
//   in the user's language.
// The easiest way to do this is to put the strings in the RC file and use LoadString
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP IXXXServer::GetErrorString( HRESULT hr, LCID locale, LPWSTR *ppstring)
{
        return E_NOTIMPL;
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::AddGroup()
//   This function creates a new group of the specified name on this server.
//   Note TimeBias, Deadband and LCID are not implemented
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP        IXXXServer::AddGroup(
                        LPCWSTR szName,
                        BOOL bActive,
                        DWORD dwRequestedUpdateRate,
                        OPCHANDLE hClientGroup,
                        LONG *pTimeBias,
                        FLOAT *pPercentDeadband,
                        DWORD dwLCID,
                        OPCHANDLE *phServerGroup,
                        DWORD *pRevisedUpdateRate,
                        REFIID riid,
                        LPUNKNOWN *ppUnk
                        )
{
        int             j;
        XXXGroup *newgroup;
        XXXServer &s = *m_Parent;
        HRESULT                 r1;

        // find a place to put the group
        //
        for(j=0; j<N_GRPS; j++)
        {
                if(s.m_groups[j].inuse == 0) break;
        }
        if(j >= N_GRPS)
                return E_OUTOFMEMORY;

        // Create the group (returns IUnknown)
        // and do an 'AddRef' since we will hold this IUnknown
        // in the Server
        //
        newgroup = new XXXGroup(m_Parent);
        if(newgroup == NULL)
        {
                return E_OUTOFMEMORY;
        }
        newgroup->AddRef();

        // And request a 2nd interface for the caller
        //
        r1 = newgroup->QueryInterface(riid, (LPVOID*) ppUnk);
        if(FAILED(r1))
        {
                // If error - delete group and return
                delete newgroup;
                return r1;
        }

        // If OK then Record the group in the server for future use
        //
        s.m_groups[j].inuse = 1;
        s.m_groups[j].pGroup = newgroup;

        newgroup->ServerGroupHandle = j;
        newgroup->ClientGroupHandle = hClientGroup;
        newgroup->dwRevisedRate = dwRequestedUpdateRate;
```

```
        newgroup->bActive = bActive;
        newgroup->szName = BSTRFromWSTR(szName);

        // Return handle and updaterate to the caller
        //
        *phServerGroup = j;
        *pRevisedUpdateRate = newgroup->dwRevisedRate;

        return S_OK;
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::GetGroupByName()
//   This function scans the set of groups known to this OPC server and returns a pointer to the
//   IOPCGroup interface for the specified group.
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP IXXXServer::GetGroupByName( LPCWSTR szGroupName,
                REFIID riid, LPUNKNOWN *ppUnk)
{
        return E_NOTIMPL;
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::RemoveGroup()
//                 This function removes the specified group from the server.
//                 Note that the group doesn't
//                 go away until the last reference to it is removed.
//                 bForce is not currently implemented
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP IXXXServer::RemoveGroup( OPCHANDLE groupHandleID, BOOL bForce)
{
        int             j;
        XXXServer &s = *m_Parent;
        XXXGroup *group;

        // find the group with the passed ServerHandle
        //
        for(j=0; j<N_GRPS; j++)
        {
                if(s.m_groups[j].inuse &&
                        (groupHandleID == s.m_groups[j].pGroup->ServerGroupHandle))
                        break;
        }
        if(j >= N_GRPS)
                return E_FAIL;
        group = s.m_groups[j].pGroup;

        // release this reference to it
        // (which will delete it if the reference goes to 0)
        // and mark the slot unused
        //
        group->Release();
        s.m_groups[j].inuse = 0;

        return S_OK;
}

////////////////////////////////////////////////////////////////////////////////
// IXXXServer::CreateGroupEnumerator()
//
////////////////////////////////////////////////////////////////////////////////
STDMETHODIMP IXXXServer::CreateGroupEnumerator(
                        OPCENUMSCOPE dwScope,
                        REFIID riid,
                        LPUNKNOWN *ppUnk
                        )
{
        return E_NOTIMPL;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
```

5.2 OPCインタフェース概要

OPCサーバオブジェクトはデータソースの特定のクラスからデータを得る方法を提供します。利用できるソースの数とタイプはサーバインプリメンテーションの関数です。

OPCサーバには“グループ“として定義される機能があります。これらのグループによって、OPCクライアントはアクセスしたいデータを取りまとめて扱うことができ、それを一つの単位として起動させたり停止させたりすることができます。さらに、グループはOPCクライアントに、変更するとき、通知できるように項目リストへ”署名”方法を提供します。

すべてのCOMオブジェクトがインタフェースを経由してアクセスされることに注意してください。クライアントはインタフェースだけを見ます。したがって、ここで述べられているオブジェクトはサーバの実際の内部のインプリメンテーションに関係してもしなくてもいい“論理的な”表現です。

一般に、クライアントプログラムはVBAのようなスクリプト言語などを利用してオートメーションインタフェースを使用します。また、一般に、C++言語で作成される、最大のパフォーマンスを達成する目的をもつクライアントプログラムはカスタムインタフェースを使用することが最も簡単でしょう。

図5-6～図5-8はOPCオブジェクトとインタフェースの要約です。インタフェースのいくつかがオプションであることに注意してください。（オプションのインタフェースは[ ]で示されています。）

なお、インタフェースの名前には次のような取り決めがあります。

IOPCxxx	OPCカスタムインタフェース
IOPCxxxDisp	OPCオートメーションインタフェース
Ixxx	OLE標準インタフェース(IPersistFile,IDataObjectなど)

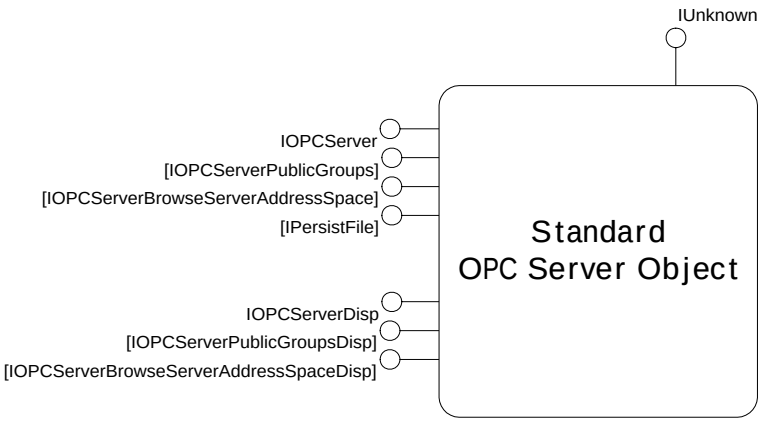


図5-6 標準OPCサーバオブジェクト

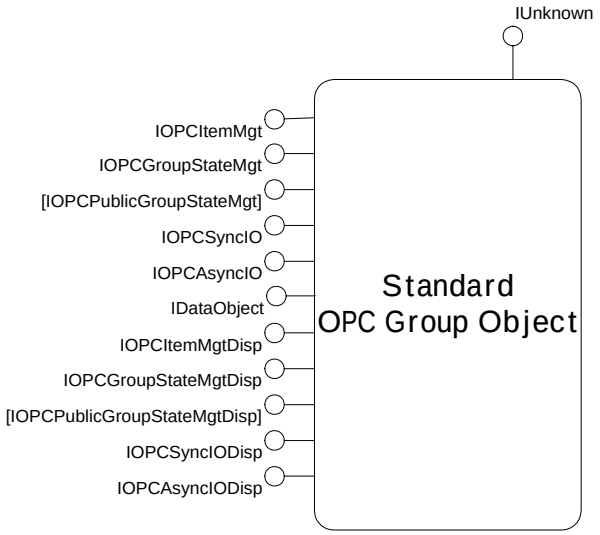


図5-7 標準OPCグループオブジェクト

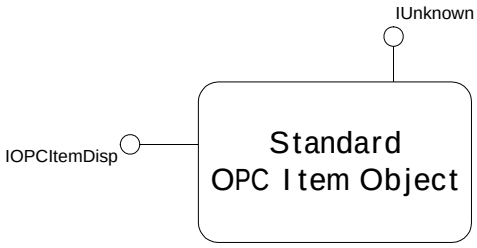


図5-8 標準(オートメーション)OPCアイテムオブジェクト

5.3 OPCオートメーションインタフェース

この節ではOLEオートメーションのIDispatchインタフェースを通してどのようにカスタムインタフェースをエクスポーズするかを示します。IDispatchインタフェースはVC++およびVisual Basic 4.0によって完全にサポートされています。IDispatchインタフェースはOPCインタフェースをアクセスするOLEオートメーションコントローラを持つアプリケーション(VB, VC++, VBA in Excel や MS Project, Visioなど)をサポートします。

OPCオートメーションインタフェースは図 5-9のようにOPCサーバ、OPCグループおよびOPCアイテムのオブジェクト階層から成り立っています。OPCグループはOPCサーバオブジェクトに含まれています。

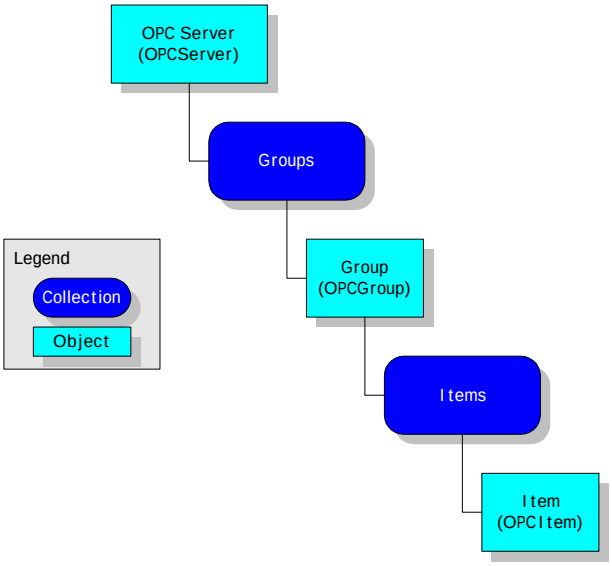


図5-9 オートメーションオブジェクトの階層構造

5.3.1 OPCサーバオブジェクト

このオブジェクトは動作中のOPCサーバへのコネクションを持つオブジェクトです。グループとアイテムのオブジェクトを作成する前にOPCサーバオブジェクトを作成します。OPCサーバは次に示すいくつかのインタフェースを用意しています。

5.3.1.1 IOPCServerDisp インタフェース

OPCサーバオブジェクトの標準インタフェースです。このインタフェースはVTABLEに直接バインドできるDualインタフェースとして定義されています。プロパティおよびメソッドは次のものがあります。

プロパティ	説明
Count	OPCサーバのアドレス空間内で定義されているグループ数。
_NewEnum	コレクション内アイテムのIEnumVariantの列挙。(読み込みのみ)
StartTime	OPCサーバが起動された時間。(UTC)
CurrentTime	OPCサーバで識別する現在時間。(UTC)
LastUpdateTime	このOPCクライアントへの最終データ更新時間。(UTC)
MajorVersion	OPCサーバソフトウェアのメジャーバージョン。
MinorVersion	OPCサーバソフトウェアのマイナーバージョン。
BuildNumber	OPCサーバソフトウェアのビルド番号。
VendorInfo	OPCサーバについての追加情報として用意されるベンダ独自の文字列。(会社名およびサポートするデバイスのタイプを含めることを推奨します。)

メソッド	説明
Item	BasicのFOR/NEXT機能を実行するオートメーションで使用される特別なメソッド。
AddGroup	OPCサーバにグループを追加します。
GetErrorString	エラーコードに対応するエラー文字列を返します。
GetGroupByName	同一OPCクライアントによって作成されたプライベートグループのポインタを返します。 (パブリックグループへのアタッチにはGetPublicGroupByNameを使用します。)
RemoveGroup	グループを削除します。
SaveConfig	シリアルポートの通信速度などのOPCサーバで設定された構成情報を保存します。
LoadConfig	シリアルポートの通信速度などのOPCサーバ用の構成情報をロードします。
SetEnumeratorType	OPCサーバで用意されたグループの様々な列挙を作成します。

5.3.1.2 IOPCServerPublicGroupsDisp インタフェース(オプション)

パブリックグループの管理をするために用意されたオプションのインタフェースです。

メソッド	説明
GetPublicGroupByName	OPCクライアントをパブリックグループへ接続します。グループへのポインタを返します。
RemovePublicGroup	パブリックグループを削除します。

5.3.1.3 IOPCServerBrowseServerAddressSpaceDisp インタフェース(オプション)

OPCクライアントからOPCサーバ内のデータアイテムをブラウズするためのインタフェースです。

プロパティ	説明
_NewEnum	コレクション内ストリングのIEnumVariantの列挙。(読み込みのみ)
Organization	OPCサーバのネームスペースのタイプ（フラットまたは階層）。

メソッド	説明
ChangeBrowsePosition	階層スペース内で'Up' または'Down'へ移動します。
SetItemIDEnumerator	_NewEnumで要求された時に使用する列挙のタイプをセットします。（ブラウスの位置はChangeBrowsePositionで設定される。）
GetItemIDString	階層スペース内のアイテムIDを取得します。
SetAccessPathEnumerator	_NewEnumで要求された時に使用する列挙のタイプをセットします。（アイテムIDのアクセスパスをブラウズするため）

5.3.2 OPCグループオブジェクト

OPCグループオブジェクトはアイテムのコレクションを管理するためのオブジェクトです。

5.3.2.1 IOPCItemMgtDisp インタフェース

OPCグループ内のアイテムの追加、削除や制御などを実行する、OPCサーバオブジェクトへのインタフェースです。

プロパティ	説明
Count	コレクション内のアイテムの数。(読み込みのみ)
_NewEnum	コレクション内アイテムのIEnumVariantの列挙。(読み込みのみ)

メソッド	説明
Item	BasicのFOR/NEXT機能を実行するオートメーションで使用される特別なメソッドです。
AddItems	グループへアイテムを追加します。
ValidateItems	アイテムが有効かどうか調べます。アイテムの情報を返します。
RemoveItems	グループからアイテムを削除します。(AddItemsの逆)
SetActiveState	グループ内のアイテムにactive/inactiveを設定します。
SetClientHandles	グループ内のアイテムのクライアントハンドルを変更します。
SetDataTypes	グループ内のアイテムの要求データタイプを変更します。

5.3.2.2 IOPCGroupStateMgtDisp インタフェース

グループを総合的に管理するOPCサーバオブジェクトへのインタフェースです。

プロパティ	説明
ActiveStatus	グループの現在のステータス(active/inactive)。
ClientGroupHandle	グループの現在のクライアントハンドル(ClientHandle)。
ServerGroupHandle	グループのサーバグループハンドル(ServerGroupHandle)。
Name	グループ名。
UpdateRate	グループの現在の更新周期。
TimeBias	グループの現在のTimeBias。
PercentDeadBand	アイテム値のデッドバンド。(百分率)
LCID	OPCサーバが値を返す時に使用する言語識別子。

メソッド	説明
CloneGroup	グループのコピーを作成します。

5.3.2.3 IOPCSyncIODisp インタフェース

OPCサーバへの同期読み込み/書き込み処理を実行するインタフェースです。

メソッド	説明
OPCRead	グループ内のアイテムのデータ値、品質フラグ、タイムスタンプの読み込み。
OPCWrite	グループ内のアイテムにデータを書きます。

5.3.2.4 IOPCASyncIODisp インタフェース

OPCサーバへの非同期読み込み/書き込み処理を実行するインタフェースです。処理は“キュー”に格納されるのでOPCクライアントは続けて次の処理を実行することができます。各処理はトランザクションとして扱われ、トランザクションIDが発行されます。処理が完了するとOPCクライアント内のAdviseSinkにコールバックがなされます。コールバック内の情報はトランザクションIDとエラー結果を示します。

メソッド	説明
AddCallbackReference	コールバックの設定をします。
DropCallbackReference	コールバック接続を切断します。
OPCRead	グループ内のアイテム読み込み。
OPCWrite	グループ内のアイテムへの書き込み。
Cancel	OPCサーバに未処理のトランザクションをキャンセル要求します。
Refresh	グループ内のアクティブなアイテムをコールバックします。

5.3.2.5 IOPCPublicGroupStateMgtDisp インタフェース(オプション)

プライベートグループをパブリックグループにコンバートするための、OPCサーバオブジェクトへのインタフェースです。OPCクライアントによって作成されるグループは初めは常にプライベートグループとして作成されます。このインタフェースはプライベートグループをパブリックグループへコンバートする目的で使用されます。

プロパティ	説明
State	グループの状態(パブリックまたはプライベート)。

メソッド	説明
MoveToPublic	プライベートグループをパブリックグループに変更します。

5.3.3 OPCアイテムオブジェクト

5.3.3.1 IOPCItemDispインタフェース

アイテムオブジェクトのプロパティとメソッドを提供するOPCアイテムオブジェクトへのインタフェースです。

プロパティ	説明
AccessPath	アイテムのアクセスパス(AccessPath)。
AccessRights	アイテムで設定されたアクセス権(AccessRights)。
ActiveStatus	アイテムの現在のステータス (active/inactive)。
Blob	アイテムで設定されたBlobのデータ。
ClientHandle	アイテムの現在のクライアントハンドル(ClientHandle)。
ItemID	現在のアイテムID。
ServerHandle	現在のサーバハンドル(ServerHandle)。
RequestedDataType	現在の要求データタイプ(RequestedDataType)。
Value	アイテムデータの読み込み/書き込み。
Quality	品質フラグ。
TimeStamp	タイムスタンプ。
ReadError	アイテム読み込み処理での直前のエラー。
EUType	エンジニアリングユニット（工業単位）。
EUInfo	エンジニアリングユニット情報（工業単位情報）。
WriteError	アイテム書き込み処理での直前のエラー。

メソッド	説明
OPCRead	個々のアイテムのデータ値、品質フラグ、タイムスタンプを読み込む。
OPCWrite	デバイスへデータを書き込む。

5.4 OPCカスタムインタフェース

OPCカスタムインタフェースには次のカスタムオブジェクトがあります。

- OPCサーバオブジェクト
- OPCグループオブジェクト

OPCカスタムインタフェースは、OPCコンポーネントとオブジェクトのインタフェースと動作についてのインタフェースです。OPCサーバの開発者はこの節で定義された機能を用意することによりOPCオブジェクトを実装することができます[4]。また、この節ではOPC準拠のコンポーネントを作成するために必要な標準のOLEインタフェースの動作についても記述しています。

列挙オブジェクトとそのインタフェースについてはOLEの仕様によって十分に定義されているので、ここでは簡単に記述されています。列挙には以下のインタフェースがあります。

- グループの列挙 - (IOPCServer::CreateGroupEnumerator)
- アイテム属性の列挙 - (IOPCItemMgt::CreateEnumerator)
- サーバアドレススペースの列挙 - (IOPCBrowseServerAddressSpace::BrowseOPCItemIDs)
- アクセスパスの列挙 - (IOPCBrowseServerAddressSpace::BrowseAccessPaths)

5.4.1 OPCサーバオブジェクト

OPCサーバオブジェクトはOPCサーバをエクスポーズするメインオブジェクトです。

[4] OLE標準の一部のインタフェースについては省略されています。OPC仕様書を参照ください。

5.4.1.1 IOPCServerインタフェース

OPCサーバのメインインタフェースです。

メンバ	説明
AddGroup	OPCサーバへグループを追加します。
GetErrorString	OPCサーバのエラーコードに対応するエラー文字列を返します。
GetGroupByName	同一OPCクライアントによって作成されたプライベートグループのポインタを返します。(パブリックグループへのアタッチにはGetPublicGroupByNameを使用します。)
GetStatus	OPCサーバの現在の状態を取得します。
RemoveGroup	グループを削除します。
CreateGroupEnumerator	OPCサーバで用意されたグループの様々な列挙を作成します。

5.4.1.2 IOPCServerPublicGroupsインタフェース(オプション)

パブリックグループの管理のためのインタフェースです。

メンバ	説明
GetPublicGroupByName	OPCクライアントをパブリックグループへ接続する。グループへのポインタを返します。
RemovePublicGroup	パブリックグループを削除します。

5.4.1.3 IOPCBrowseServerAddressSpaceインタフェース(オプション)

OPCサーバで利用可能なアイテムIDをブラウズするためのインタフェースです。

メンバ	説明
QueryOrganization	システム構造がフラットか階層構造かを調べます。
ChangeBrowsePosition	階層スペース内で'Up' または'Down'へ移動します。
BrowseOPCItemIDs	アイテムIDのリスト(IEnumString)を返します。(ブラウズの位置はChangeBrowsePositionで設定される。)
GetItemID	階層スペース内のアイテムIDを取得します。
BrowseAccessPaths	アイテムIDのアクセスパス(AccessPaths)をブラウズします。

5.4.1.4 IPersistFileインタフェース(オプション)

OPCサーバの構成情報をファイルにセーブしたり、ファイルからロードするためのインタフェースです。

メンバ	説明
IsDirty	構成情報に変更があったかどうかを返します。
Load	構成情報をロードすることをOPCサーバに要求します。
Save	構成情報を保存します。
SaveCompleted	スタブとして実装します。
GetCurFile	現在ロードされている構成情報ファイルの名前を返すようにOPCサーバに要求します。

5.4.2 OPCグループ オブジェクト

OPCグループオブジェクトはOPCサーバがアイテムコレクションの管理をするためのオブジェクトです。

5.4.2.1 IOPCGroupStateMgtインタフェース

グループの全体的な状態を管理します。

メンバ	説明
GetState	グループの現在の状態を取得します。
SetState	グループのプロパティを設定します。
SetName	プライベートグループの名前を変更します。名前はユニークです。また、パブリックグループは変更できません。
CloneGroup	グループのコピーを作成します。

5.4.2.2 IOPCPublicGroupStateMgtインタフェース(オプション)

プライベートグループをパブリックグループに変更するためのオプションのインタフェースです。

メンバ	説明
GetState	グループがパブリックグループかどうか調べます。
MoveToPublic	プライベートグループをパブリックグループに変換します。

5.4.2.3 IOPCSyncIOインタフェース

OPCクライアントからOPCサーバへの同期読み込み／書き込みを実行します。

メンバ	説明
Read	グループ内のアイテムのデータ値、品質フラグ、タイムスタンプの読み込み。
Write	グループ内のアイテムにデータを書きます。

5.4.2.4 IOPCAsyncIOインタフェース

OPCクライアントからOPCサーバへの非同期読み込み／書き込みを実行します。

メンバ	説明
Read	グループ内のアイテムを読み込む。
Write	グループ内のアイテムへ書き込む。
Refresh	グループ内のアクティブなアイテムをコールバックします。
Cancel	OPCサーバに未処理のトランザクションをキャンセル要求します。

5.4.2.5 IOPCItemMgtインタフェース

グループ内のアイテムの動作を制御します。

メンバ	説明
AddItems	グループへアイテムを追加します。
ValidateItems	アイテムが有効かどうか調べます。アイテムの情報を返します。
RemoveItems	グループからアイテムを削除します。（AddItemsの逆）
SetActiveState	グループ内のアイテムにactive/inactiveを設定します。
SetClientHandles	グループ内のアイテムのクライアントハンドルを変更します。
SetDatatypes	グループ内のアイテムの要求データタイプを変更します。
CreateEnumerator	グループ内のアイテムの列挙を作成します。

5.4.2.6 IEnumOPCItemAttributesインタフェース

グループのアイテムと属性を検索します。

メンバ	説明
Next	グループからアイテムを取得します。
Skip	アイテムをスキップします。
Reset	列挙の先頭アイテムにリセットします。
Clone	列挙の現在状態のコピーを作成します。

5.4.2.7 IDataObjectインタフェース

IDataObjectはOPCグループ上で実行されます。

メンバ	説明
DAdvise	OPCグループとOPCクライアントの接続を確立します。
DUnadvise	OPCグループとOPCクライアントの接続を切断します。

5.4.3 OPCクライアント側インタフェース

5.4.3.1 IAdviseSinkインタフェース

クライアントはOnDataChangeの実装を用意する必要があります。

メンバ	説明
OnDataChange	OPCグループからの例外ベースのデータ変化を通知させるためにOPCクライアントが用意するメソッドです。

6 技術基盤説明

現在のOPC仕様による機能はプロセスデータのアクセスに限られており、その機能自体はそれほど特長のあるものではありませんが、OPCの特長の多くは、それがOLE/COM技術の上に構築されていることに起因しています。したがって、OLE/COMを理解することはOPCを理解する早道でもあります。

ここでは、マイクロソフトのオブジェクト技術であるOLE/COMについて概略理解し、さらなる情報へのポインタを示すことを目的として、説明を行っています。

注意 ：以下で参照する情報は、複数の情報源のうち最も適切でかつ詳細な情報だけを選択しています。

- オブジェクトモデル

COMのオブジェクトモデルは、アプリケーションをCOMオブジェクトの集合として構築した場合に、そのオブジェクト構造が外部アプリケーションからどのように見えるかを示している。狭義にはCOMサーバの階層的なオブジェクト構造を指す場合が多い。オブジェクトモデルは、必ずしもアプリケーションの実装上の構造とは一致しない。たとえば、C++でアプリケーションを構築した場合、C++のクラス定義がCOMオブジェクトに1対1に対応するわけではない。オブジェクトモデルをどのように設計するかでアプリケーション間連携の性能や機能の使い勝手が決まってしまうので、設計は重要な問題である。OPCでは、すでにOPCアプリケーションのオブジェクトモデルは決められているので、開発者が設計の必要はない。よりわかりやすいオブジェクトモデルの一般的説明は、**http://www.microsoft.com/oledev/oleauto/ole2soln.htm**に書かれている。

- COMオブジェクト生成

COM オブジェクト生成は C++では CoCreateInstance、CoGetClassObject などの API、Visual Basic では、CreateObject、GetObject の API で生成される。これにより、オブジェクトのインタフェースのポインタがローカル変数へ返されるので、それを用いてプロパティやメソッドへアクセスする。オブジェクト生成には、そのクラス識別子を指定する必要があるが、一旦生成されると、そのインタフェースポインタがどのオブジェクトのものかを考えてプログラムする必要はなくなる。これらとは別に、モニカーと呼ばれるものや、HTMLの<OBJECT>タグでオブジェクトを生成することも可能である。より詳しいオブジェクト生成の説明は、 **OLE KBase Q104139** および **Microsoft Systems Journal 1996 May, Introducing Distributed COM and the New OLE Features in Windows NT 4.0** に書かれている。

- クラスファクトリ

オブジェクトの機能をクライアントが利用できるようにするためには、オブジェクトを作り、そのインタフェースをクライアントへ返す機能をサーバが提供しなければならない。サーバとはオブジェクトを提供するアプリケーション、クライアントはそのオブジェクトを利用するアプリケーションである。サーバのオブジェクトを作り出す機能、クラスファクトリをシステムに登録し、クライアントからの要求がシステムからサーバに導かれるようにする。クラスファクトリの登録方法は、DLLサーバとEXEサーバでは異なる。クラスファクトリの作成方法はサーバの開発者が知らなければいけない基本知識であり、**Component Object Model(COM) Specification 0.9 Chapter 6, 6.2 Implementing the Class Factory, 6.3 Exposing the Class Factory**に書かれている。

- コレクションオブジェクト

コレクションオブジェクトとは、オブジェクトの集合を扱うため、集合への追加、削除、要素検索、要素総数、などを容易に扱えるようなメソッドやプロパティをもつ集合オブジェクトである。配列やリスト構造のようなもので実装される。実装に関する詳しい説明は、**http://www.microsoft.com/oledev/oleauto/collect.htm (MFC), OLE KBase Q107546** に書かれており、OPC の OPCGroup オブジェクトの実装などサーバ開発で必要とされる。

● オートメーションインタフェース

クライアントがサーバのインタフェースを実行時に選択してアクセスする動的バインディングと呼ばれる機能。これに対して、開発時にサーバのインタフェースをコード上に固定的に書き込むのがVTABLEインタフェースに基づく静的バインディングと呼ばれるものである。オートメーションでは、インタフェース名の一致からサーバのインタフェースが選択され、実行される。サーバにおけるオートメーションの実装方法は、**Inside OLE 2nd Edition, Chapter 14 Five Variations on the Theme of Implementing a Simple Automation Object**に書かれている。クライアントからサーバへのオートメーションの利用を説明した例は、Visual Basicは**Visual Basic Programmer's Guide, Chapter 9 Programming Other Applications' Objects**、Visual C++は、**Microsoft Systems Journal 1995 June OLE Q&A, MSDN Technical Articles/Windows Articles/OLE Articles/Component Object Model, MFC/COM Objects 7: Creating and Using COM Objects with OLE Automation Interfaces**にある。

● カスタムインタフェース

OLEが標準で定義したインタフェース以外の、個別拡張インタフェースをカスタムインタフェースと呼ぶ。カスタムインタフェースはクライアントからVTABLE経由で静的バインディングでアクセスされるインタフェースである。カスタムインタフェースをEXEサーバが実装する場合、インタフェースごとのプロキシコード、スタブコードと呼ばれるDLLを同時に開発し、インストールする必要がある。プロキシとスタブコードの説明は、**Component Object Model(COM) Specification 0.9 Chapter 7 Interface Remoting**にあり、その開発はカスタムインタフェースの開発の例として、**http://www.microsoft.com/msdn/sdk/platforms/doc/activex/src/custintf_2.htm**以降の**ActiveX SDK**および**http://www.microsoft.com/msdn/library/mfccom.htmのTom's Handy Dandy MFC/COM/MIDL Recipe Book for Creating Custom Interfaces**に書かれている。

● Dualインタフェース

Dualインタフェースとは、オートメーションインタフェースの動的バインディングの機能にカスタムインタフェースの静的バインディングを組み合わせ、どちらの方法でもクライアントからオブジェクト機能にアクセスできるようにするための技術である。その説明と実装方法は、**Microsoft Systems Journal 1995 Volume 10 December 12 OLE Q&A, Visual C++ MFC Technical Notes TN065 (MFC)**に書かれている。OPCではカスタムインタフェースとDualインタフェースをサポートするオートメーションインタフェースを個別に定義している。

● In-Proc Handler(in-processハンドラ)

クライアントがサーバと連携するとき、連携の性能向上、セキュリティや信頼性の強化などの目的に必要とされる技術である。したがって、必ずしも連携のための必須機能ではない。in-processハンドラはクライアントマシンのクライアントプログラムと協調するDLL形式のプログラムで、そのプログラムが擬似的なサーバオブジェクトを生成して、クライアントに対して真のサーバオブジェクトの“ふり”をする形で実装される。in-processハンドラは、通常のDLL形式のCOMサーバと実装上は変わらない。したがって、その実装方法はDLLサーバの実装方法、例えば、**http://www.microsoft.com/intdev/sdk/DLLSERVE-Lesson 7**を参照するとよい。

● イベント通知

クライアントがサーバへ非同期要求を行ったとき、あるいは、サーバやネットワークエラーが発生したとき、その結果通知をイベントとしてクライアントは受信する。クライアントはあらかじめ受信するイベントの種類と、イベント通知で呼ばれる処理をサーバに登録しておく必要がある。イベント通知にはOLEの標準インタフェース、IAdviseSinkを利用することがOPCでは決められている。その開発方法は、**Inside OLE 2nd Edition, Chapter 10 Uniform Data Transfer and Notifications**などに書かれている。

- スレッディングモデル

COM/DCOMにはオブジェクトをマルチスレッド環境で実行させる方法の規定がある。これをスレッディングモデルと呼ぶ。オブジェクトの開発者は、実行に使うスレッディングモデルをシステムに登録し（DLLサーバではレジストリに、EXEサーバではCoInitializeEx()の引数に設定）、そのモデルに合った実装を行う必要がある。スレッディングモデルの種類は、**OLE KBase Q150777**に書かれている。その実装のサンプルとして、OPCのサーバオブジェクトに適切なフリースレッドモデルの例が、**http://www.microsoft.com/intdev/sdk/ FRESERVE-Lesson 17** にある。

- 管理ツール

✓ OLEView－システムにインストールされたCOMオブジェクト、そのインタフェース、各種レジストリ情報を一覧する。
✓ DCOMCNFG－DCOMオブジェクトの起動方法、セキュリティ、その他の管理をするNT4.0標準のGUIツール

- デバッグツール

✓ HookOLE－OLE、COM呼び出しのトレースを行う
✓ ROTView－Running Object Tableに登録されたCOMオブジェクトのチェック
✓ NuMega社のBoundsChecker 4.2

7 付録

7.1 用語集

アルファベット順

A

◆ ActiveX コントロール
ActiveX コントロールはコンテナと呼ばれるアプリケーションで利用してソフト製品を開発します。コンポーネント間の連携をスクリプト言語で記述することで、カスタマイズ可能です。ActiveX の技術により、インターネットから利用時にダウンロードして利用もできます。

◆ Application Programming Interface (API)
OS がアプリケーションに対して公開しているプログラムインタフェースです。アプリケーションは、基本的にすべての処理をこの API を経由して行ないます。 現在、一般的な OS の API は関数の形式をとっており、アプリケーションからは適当なパラメータ（引数）を指定して API の関数を呼び出します。

C

◆ COM (Component Object Model)
ソフトコンポーネントの持つべき基本サービスを提供するアーキテクチャです。オブジェクトの生成と消去、バージョン管理、インタフェースの実行時折衝、メモリ管理、ネットワーク透過性などを提供します。ActiveX コントロールは、OLE 複合ドキュメントなどのオブジェクト間連携技術です。DCOM 欄参照。

D

◆ DCOM (Distributed COM、分散 COM)
COM のオブジェクト間連携機能をネットワークのクライアント／サーバへ拡張したものです。Windows NT 4.0 標準装備です。COM の欄参照。

◆ Dynamic Link Library (DLL)
Windows プログラム（Windows カーネルなどのシステムプログラム、アプリケーション）は、通常の.EXE ファイルとは別にダイナミックリンクライブラリを用意し、これを実行時にリンクすることができます。 実際には Windows カーネル（KERNEL、USER、GDI）やデバイスドライバもダイナミックリンクライブラリです。複数のプログラムで 1 つの.DLL ファイルを共有できるので、ディスクサイズや実行時メモリを節約することができます。

◆ DCS
Distributed Control System の略です。

F

◆ FILETIME
FILETIME 構造は、ファイルの 64 ビット長の時間値を持ちます。この値は、1601 年 1 月 1 日からの通算時間を 100 ナノ秒の単位で表現します。

G

◆ Globally Unique Identifier (GUID)
OLE コンポーネントオブジェクトに割り振られる ID であり、特殊なアルゴリズムにより生成されます。 分散システム上に無数のオブジェクトが存在するような状況においても、名前の競合が起こらない環境を実現するユニークな ID をこのアルゴリズムは全てのコンポーネントに対して提供します。 OLE2 SDK で提供されるソフトウェアを使って、ソフトウェアデベロ

ッパーはコンポーネントオブジェクトに対する GUID を簡単に作成することができます。

I

◆ IDispatch インタフェース
OLE オートメーションインタフェースが備えるべきインタフェースで、クライアントは通常、このインタフェースを通してメソッドやプロパティにアクセスします。

◆ In-Proc Server
インプロセスサーバはコンテナのプロセス内で動作する、DLL タイプの OLE サーバのことです。 具体的には OCX のことを示します。これまでの OLE サーバがユーザやオートメーションプログラムからのトリガーによりメモリ上にプロセスを作成したのに対して、DLL である OCX は常にメモリ上に展開されています。

◆ Interface Definition Language (IDL)
オブジェクトを実装するとき、実装とインタフェースを分離することが実装のカプセル化には重要です。そのために、実装言語とは独立する中間言語でインタフェースを定義し、それをターゲットの実装言語にマップするようにします。こうすることで、オブジェクトの実装は言語非依存となります。IDL が表現する COM オブジェクトの属性は、インタフェースの引数のデータ型や並び順、戻り値、インタフェース識別子、クラス識別子、各国語情報（locale ID）などです。IDL のソースファイルはコンパイルされ、ヘッダファイル、プロキシ、スタブのソースコードへ変換され、コンパイルされて、オブジェクトやクライアントの実装コードとリンクされます。

L

◆ LCID
lcid 属性は、タイプライブラリあるいは関数 argument 用のロカールを確認します。

◆ Local Server
Local Server は同一マシン上の独立したプロセス空間で走る OLE サーバアプリケーションです。この場合、クライアントと Local Server は別プロセスとして実行されるため、信頼性は向上しますが、性能は In-Proc Server に対してかなり遅くなります。したがって、サーバの設計ではどちらの方法で実装するほうがいいかの選択をする必要があります。

M

◆ MES
Manufacturing Execution System の略です。

◆ Microsoft Foundation Classes (MFC)
マイクロソフトの Windows（Windows 95）、Windows NT アプリケーション開発用の C++クラスライブラリです。マイクロソフトが販売する C++処理系パッケージである Visual C++に付属しています。

◆ MIDL 3.0
IDL をコンパイルしてヘッダ、プロキシとスタブのソースコードを出力するためのツールです。MIDL は OpenGroup の DCE RPC と上位互換機能を持ち、また、MIDL3.0 は従来の ODL ファイルをコンパイルする MKTYPLIB ツールを代替する機能を持ちます。

O

◆ OCX(OLE Custom Control)
OCX はコンポーネントオブジェクトおよびオートメーションオブジェクトの一種です。OCX は複数のソフトウェアベンダから提供され、DB アクセスやマルチメディア、グループウェア機能などのソフトウェア部品が含まれることになります。OLE による複合ドキュメント作成のケースと同じ方式で、OCX を利用した“複合アプリケーション”の開発を行うことができます。ただし、OCX はコンテナのプロセス内で動作する In-Proc Server であるため、通常の OLE

サーバとは異なり瞬時に反応することができます。

◆ OLE (Object Linking & Embedding)
マイクロソフトが開発したオブジェクト連携技術です。 COM のベースアーキテクチャの上で提供される種々のアプリケーション機能です。複合ドキュメント、ドラッグ＆ドロップ、オブジェクト保存、OLE オートメーション、OLE コントロールなどが存在します。そのうち、ソフトウエア再利用技術と分散処理技術が ActiveX 技術として改称され、拡張されました。

◆ OLE オートメーション
外部のプログラムから既存のアプリケーション機能を利用するための技術です。ActiveX コントロールが主に GUI を持つ機能に向くのに対して、オートメーションは計算処理など負荷が大きなサーバー型のアプリケーションに適用されます。スクリプト言語（例：Visual Basic）により機能の制御が可能です。
クライアント（OLE オートメーションコントローラ）とサーバ（OLE オートメーションサーバ）の間のオブジェクト連携を可能にしています。

◆ OPC-F
OPC Foundation

P

◆ PLC
Programmable Logic Controller の略です。

◆ Proxy
Proxy はクライアントアプリケーションと同一のアドレス空間に存在するライブラリです。クライアントがリモートオブジェクトと通信するとき、メソッド呼び出しやプロパティへのアクセスを RPC メッセージの正規表現に変換するためのコードです。Proxy はクライアントプログラムの実行時に自動的にロードされます。

R

◆ Remote Procedure Call (RPC)
RPC は分散したプログラム同士が通信し合うための基本機能です。 各プログラムの入出力インタフェースをそれぞれ IDL（インタフェース定義言語）という統一した言語で記述します。

◆ Remote Server
Remote Server はクライアントアプリケーションと同一マシン上の別プロセス空間で実行されるか、または、ネットワーク上の別マシン上で実行される OLE サーバアプリケーションです。

S

◆ SCADA
Supervisory Control and Data Acquisition の略です。

◆ Software Development Kit (SDK)
Windows アプリケーションプログラマ向けのソフトウェア開発キットです。 マイクロソフトが自社の OS 向けのアプリケーション開発環境をセットにしたものをこう呼んでいます。SDK には、各種サンプルプログラムやヘッダファイル、ライブラリ（スタティックライブラリ／ダイナミックリンクライブラリ／インポートライブラリ）、ドキュメントなど、言語処理系（コンパイラ）を除く関連ファイルがひとまとめにされています。

◆ Stub
サーバ側のオブジェクトに RPC メッセージを渡すときに、正規表現からローカル表現に変

換するためのコードです。さらに、オブジェクト実行中のサーバのメモリからのアンロードを抑制するロック機能も果たします。

T

◆ TCP/IP (Transmission Control Protocol / Internet Protocol)
現在のインターネットを支えるネットワークプロトコルです。米国の研究機関や大学を接続した ARPANET 用に開発されたプロトコルです。ARPANET はのちインターネットの源流の一つになりました。

U

◆ URL (Uniform Resource Locator)
ネットワークにどのようなプロトコルでアクセスするかを明示的に指定するための書式です。http や ftp など URL の先頭に指定されたプロトコルに応じてアクセス用のアプリケーションを使い分けます。

◆ UNICODE
UNICODE は米国のコンピュータメーカ団体である UNICODE コンソーシアムによって規格化された、業界標準の統一文字コード国際規格です。

◆ UTC
Windows はシステム時間のコーディネートされた普遍的な時間（UTC）の基礎を形成します。UTC 時間はほぼグリニッジ（イングランド）の現在の日付と時間として定義されています。

V

◆ Variant
OLE オートメーション互換データ型ともいわれ、Visual Basic 変数のデフォルトデータ型としても知られる自己記述データ型です。

◆ Visual Basic(VB)
マイクロソフトの簡易版言語開発ツールです。

◆ Visual Basic for Applications(VBA)
マイクロソフト社が自社のアプリケーションのための共通マクロ言語として採用しようとしているプログラミング言語です。基本的な言語仕様は Visual Basic に準拠しています。VBA はマイクロソフト オフィスを中心とするアプリケーションプログラミング用の Visual Basic のサブセットとして位置づけられ、単体でのパッケージ販売はされません。

◆ Visual C++
マイクロソフトの C 言語開発ツールです。

◆ VTABLE
COM インタフェースを表現するメモリ上構造。これをオブジェクト間の連携のバイナリ表現として標準化したのが COM の基本仕様です。VTABLE は C++のネストクラスやインタフェースクラスの多重継承で実装されるが、C や Java などの言語でも同様の構造体を作成して実装可能です。

五十音順

イ

◆ インターネット
企業や学術施設、商用ネットワークを相互に結んだ世界最大のネットワークです。電子メール、ファイル転送、リモート接続、WWW など、TCP/IP をサポートする様々なサービスが提供されています。利用者は現在世界 150ケ国以上、1 億人と言われています。

◆ イントラネット
インターネットの仕組みをそのまま、企業や団体などの組織内に構築したものです。社内ネットワークに WWW サーバーを設置して、ユーザはブラウザで社内情報を閲覧します。

◆ インタフェース
インタフェースはシンプルなメカニズムであり、ソフトウェア内の異なる部分を接続する働きを持ちます。 具体例として、API はオペレーティングシステムとともに提供され、システムレベルのサービスに対するプログラミング言語からのアクセスを実現します。なお、COM および OLE における"インタフェース"の定義は、オブジェクト間のコミュニケーションを定義するメソッドのコレクションとなります。

エ

◆ エクスポーズ
エクスポーズとはオブジェクトの機能を開示することです。このエクスポーズを利用して OLE オートメーションは実行されます。

オ

◆ オートメーション
OLE2.0 で提供されるオートメーションは、複数のコンポーネントを結合し、アプリケーションを自動化するための機能です。 オートメーションはコンポーネントから他のコンポーネントの持つサービスを、ダイナミックに利用する環境を実現します。 OLE オートメーションに対応したコンポーネントは、オートメーションオブジェクトと呼ばれるアブストラクションを利用して、自分自身のサービスをエクスポーズします。

カ

◆ クラス
オブジェクトのタイプ指定に関するデータとメソッドの定義を行う、抽象的なデータタイプをクラスといいます。

◆ クラスライブラリ
オブジェクト指向型言語において、あらかじめ設計されたクラスモデルを提供するライブラリのことです。

キ

◆ キャッシュ
データへのアクセスを効率的にするため、よく使用するデータを一時的に格納するメモリ領域です。

サ

◆ サーバ
一般に、サーバという単語はクライアントアプリケーションにサービスを提供するコンピュータを示すために使われます。しかし、OLE2 の世界では OLE コンテナに対してオブジェクト

を提供する、OLE2 対応アプリケーションのことをサーバといいます。例として、グラフをワープロ内にエンベッドしたとき、グラフは OLE サーバでありワープロは OLE コンテナです。OLE2 対応アプリケーションはコンテナとサーバの双方の機能を持つことができます。 なお、サーバはコンテナから独立したプロセスを持ち、OCX はコンテナのプロセス内で動作します。

ス

◆ スクリプト
オペレーティングシステム(OS)やアプリケーションソフトの機能（コマンド）を使う一連の処理手順をテキスト(文字)で記述した簡易なプログラムです。

◆ スレッド（Thread）
マルチスレッド OS におけるプログラム実行の単位です。マルチスレッド OS では、プログラム（プロセス）の中で複数のスレッドを実行することができます。同じプロセスの中のスレッド同士はメモリ資源などを共有するので、スレッド間のデータ転送は比較的容易です。

タ

◆ タイプライブラリ
タイプライブラリは COM オブジェクトのクラス識別子、インタフェース識別子、メソッド名、その引数、そのデータ型、引数の並び順、戻り値のデータ型、プロパティ名とその引数の情報など、COM オブジェクトに関する情報を OLE コンパウンドドキュメントファイル（.tlb ファイル）に格納したバイナリファイルです。

ハ

◆ ハンドル
MIDL は、OLE サーバアプリケーションを含むリモートアプリケーションに接続するために、いくつかの種類のハンドルを提供します。ハンドルは、アプリケーションにネットワーク接続をするためのプロトコル情報、アドレス、その他の情報を含みカスタマイズ可能です。

フ

◆ プロトコル
データやメッセージをやりとりするために必要な手順や約束事、方式を決めたものです。

◆ プロパティ
プロパティはコンポーネントオブジェクトに組み入れられるアトリビュートです。

マ

◆ マーシャリング（Marshaling）
プロセス空間、ネットワークをまたいだ通信を行うために、CPU アーキテクチャや OS によるデータ型、ビット配列順、境界の違いを吸収するために行う正規表現への変換です。クライアントの要求はプロキシのマーシャリング機能で正規表現のパケットを送信し、サーバのスタブはこのパケットをローカル表現に逆変換してオブジェクトに渡す。戻り値はサーバからクライアントに逆向きの変換が行われれます。

メ

◆ メソッド
メソッドはオブジェクトにより提供される論理的な操作であり、オブジェクト上で機能するオ

ペレーションは、"オブジェクトのメソッド"として定義されます。 メソッドを実行するためにオブジェクトは、receiving object と特定のメソッド名で構成されるメッセージを送信します。メソッドの名前はセレクタとも呼ばれます。

レ

◆ レジストリ
Windows NT や Windows 95 において、デバイスドライバ設定からアプリケーション設定まで、コンピュータに関するあらゆる設定情報を集中管理するデータベースのことです。

◆ 列挙オブジェクト
ある項目のリストを順次処理する場合に使用されるオブジェクト。項目のリストを列挙オブジェクト（Enumerator）として作成し、このオブジェクトが持つインタフェース（Ienumxxx）を通して処理を行います。

7.2 関連文献

オンライン：

OPC1.0仕様
ftp://zilker.net/pub/opc/OPCFinal.zip

OPC Foundation ホームページ
http://www.industry.net/opc

COM自習サンプル (Win32 SDKのサンプルと同じ)
http://www.microsoft.com/intdev/activex/tutorial/com.htm

OLE/COM仕様
http://www.microsoft.com/intdev/sdk/docs/com/comintro.htm

DCOM white papers
http://www.microsoft.com/windows/common/a2399.htm

マイクロソフト インダストリー ソリューション部のホームページ (OPC技術情報)
http://www.microsoft.co.jp/Partners/Industry/opctec.htm

マイクロソフト インダストリー ソリューション部のホームページ (日本OPC協議会情報)
http://www.microsoft.co.jp/Partners/Industry/opcorg.htm

雑誌 (OPC関連)：

プラントエンジニア 1996年 6月号
プロセス制御システムの可能性を拡げるOPC

計装 1996年 9月号
32ビット対応分散制御ソフトウェアの進展とOPCへの取り組み

オーム社エレクトロニクス 1996年11月号
OPCの目指すもの

Open Network 1996年 12月号
マイクロソフトの企業情報システム戦略

日経メカニカル別冊 ディジタルファクトリ
生産プロセスのためのOLE

オートメーション 1996年11月号
OPCと汎用監視制御用パッケージソフトFIXへの適用

オートメーション 1996年11月号
FA/PAにおけるOPCの適用

計装 1997年1月号

“OLE for Process Control「ＯＰＣ」ーその技術概要と期待される効果”

オートメーション 1997年1月号
DCSとその周辺におけるオープンソフトウェア利用システム技術

オートメーション 1997年1月号
RELAY! ESSAY 計装と情報処理 ― OPC(OLE for Process Control)

書籍･雑誌 (OLE/COM/ActiveX関連):

Inside OLE, second edition (1995)
OLEのバイブルというべきもの。約1200ページの大部、OLEを駆使するためには必読の書。1995年に出た本なので、DCOMやActiveXは触れられていない。

Understanding ActiveX and OLE (1996) Microsoft Press, $22.95 ISBN 1572312165
"A guide for developers & managers"と銘打っているだけあって、320ページほどの分かりやすく、最新機能(DCOM、ActiveX、など)の解説ドキュメント。日本語版も出ている。

OLE2 Programmer's Reference(1994)
プログラマー向け。日本語版も出ている。

OLE Automation Programmer's Reference(1996)
プログラマー向け。日本語版も出ている。

マイクロソフトシステムジャーナル日本版 No.44 August 1996
･Windows NT4.0の分散COMと新しいOLE機能
･コンポーネントソフトウェアデザインに対するOLEとCOMの解答

Cマガジン 1996年 11月号
ActiveX特集の中で、ActiveXの基礎技術として11ページにわたりDCOMを解説

日経コンピュータ 1996年11月25日、12月9日
分散オブジェクト技術 DCOMの全貌(上)、(下)

Microsoft System Journal December 1995
“OLE Q&A” (日本語版 1996年2月号)