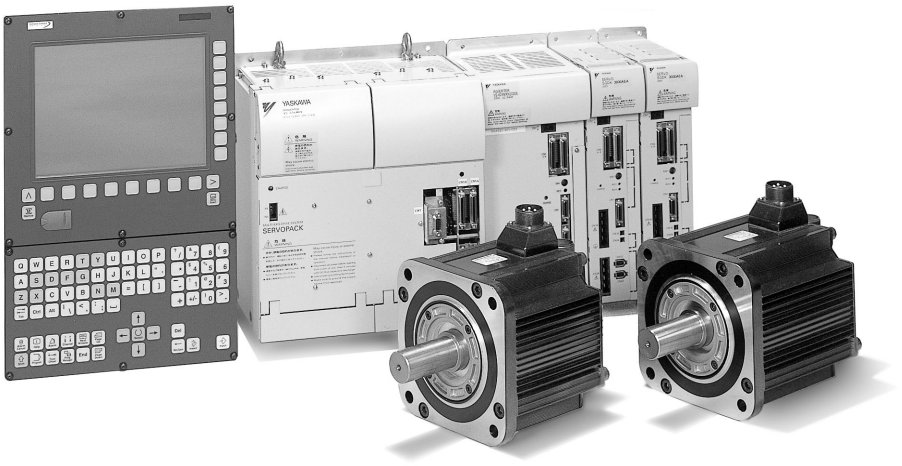
Yaskawa Siemens
numerical controls

Yaskawa Siemens CNC シリーズ

API取扱説明書
HMIプログラミングパッケージ
COM及びOPCクライアント編



資料番号　NCSI-SP02-14

安川シーメンス NC 株式会社はシーメンス株式会社に統合の後，2010 年 8 月より
シーメンス・ジャパン株式会社へ社名を変更いたしました。本書に記載の「安川
シーメンス NC 株式会社」などの社名に類する名称は「シーメンス・ジャパン株式
会社」へ読み替えをお願いします。

本マニュアルは Yaskawa Siemens 840DI，Yaskawa Siemens 830DI 両モデル用に
作成されています。本文中の記述では両モデルの機能差は区別されておりませんの
で，それぞれのモデルにどの機能が標準装備されているか，どの機能がオプション
で装備可能かについては別途，機能一覧表をご参照ください。また，本文中に
840DI と言った表現が出て来ますが，830DI も意味していることがあるとご理解
ください。

まえがき

■ 資料の編成

Yaskawa Siemens の資料は 3 部に分けられます。

- 一般資料
- ユーザ資料
- メーカー／サービス資料

本システムでの該当する資料／マニュアルの詳細については，お近くの SIEMENS 営業所へお尋ねください。

■ 対象となるグループ

この資料は，機械のメーカーと計画エンジニアのために準備されました。

■ 目的

本書「Yaskawa Siemens SINUMERIK API 取扱説明書 HMI プログラミングパッケージ COM および OPC クライアント編」に載せられている情報は，一般的な SW エンジニアリングツール（オペレーティングシステムと言語アプリケーション）を使ってヒューマンマシンインタフェースを生成／補足するときの入門情報として役立ちます。

■ 標準の有効範囲

「Yaskawa Siemens SINUMERIK API 取扱説明書 HMI プログラミングパッケージ COM および OPC クライアント編」のソフトウェア CD には，本書に加えて，以下の資料が付属しています。

- 840DI COM サーバへのリファレンス
- OEM パッケージ MMC のユーザーズマニュアル
- OPC インタフェースについての一般および特定の説明

注

これらの資料の現行バージョンについては，OPC Foundation のホームページを参照してください。

“http://www.opcfoundation.org/opc_spec_document.htm”

“Documentation” をインストールすると（“Installation Guide“ を参照），下記の画面（図 0.1 を参照）から始めてすべての資料を表示できるようになります。

［スタート］–>［プログラム］–>［SINUMERIK］–>［HMI Programming Package］–>［Documentation］–>［Start Documentation］

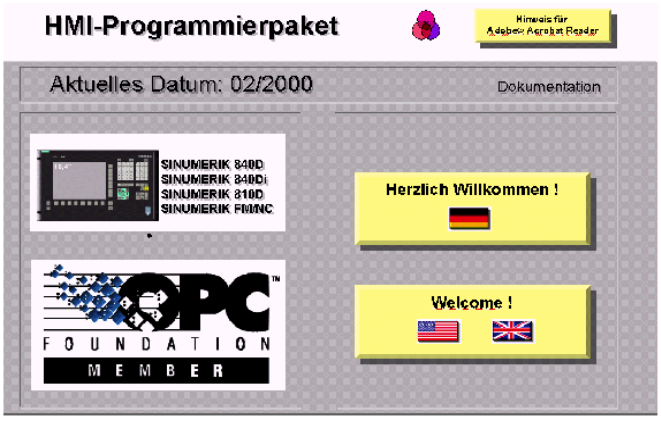


図 0.1 Start Documentation

■ 検索ツール

的確に検索するために，目次に加えて，付録に次のツールが備えられています。

1. 略称のリスト
2. 索引

■ 注

資料では，特別な意味を持つ次の注記が使われています。

注

資料では，さらに情報がある場合，常にこのシンボルが使われます。

■ 特長

現行バージョンの説明書では，章／節（たとえば 4 章「マスターコントロール」）の内容によっては，別の資料に含まれているものもあります。

このような章／節については，次の版で完成します。

■ 表記について

”No control”

引用符で囲まれた言葉は，対話式画面の入力フィールドへの入力，または入出力を表します。これらは，オペレータ装置の画面に表示されます。

File –> New

メニューオプションはイタリックになっています。レベルが異なるものは，矢印で分けられています。どの場合でも，関係のあるメニュー項目への完全なパスが示されます。

Two times OK

ボタン，オブジェクト，タブ，およびフィールドの名前はイタリックになっています。

第 1 部：基礎知識

1 概要

■ この章の目的

この章のねらいは，HMI プログラミングパッケージの機能を紹介することであり，いくつかの基本的な用語を説明しています。

1.1 HMI– プログラミングパッケージとは

本書「Yaskawa Siemens 840DI API 取扱説明書 HMI プログラミングパッケージ COM および OPC クライアント編」は，オープン性を実現するためのビジュアル化に必要な製品である，HMI– プログラミングパッケージ（HMI: ヒューマンマシンインタフェース）に適用されます。

HMI– プログラミングパッケージは，Yaskawa Siemens 840DI（以降，840DI と略します）の HMI モジュラシステムのコンポーネントです。

このモジュラシステムには，840DI でアプリケーションを作成するための，次のようなモジュールが備えられています。

表 1.1 840DI HMI モジュラシステム

モジュール	説明	資料
Ready–to–Run（実行可能）	変更したり拡張する必要のない特定タイプのマシンおよびシステム向けに，標準システムとして提供されているオペレータ制御およびモニタシステム	...
Parameterizing（パラメーター化）	内部値（事前に決定されていて制限されているもののみ）を設定することによってユーザインタフェースソフトウェアに変更を加える。	...
Configuring（構成）	特別な構成ツールを使ってユーザインタフェースを作成したり追加する。これまでのソフトウェアエンジニアリングのノウハウは必要ない。	Operator’s Guide「840DI ProTool/Pro Option Package 840DI」
Programming（プログラミング）	**一般的に使うことのできるソフトウェア構成ツール（オペレーティングシステムと高水準言語アプリケーション）によってユーザインタフェース**を作成／補足する。	お手持ちの機能説明書「Yaskawa Siemens 840DI API 取扱説明書 HMI プログラミングパッケージ COM および OPC クライアント編」

1.2 HMI– プログラミングパッケージの内容

HMI– プログラミングパッケージには，次の 2 つの機能があります。

- HMI Advanced の標準ソフトウェアを使ってアプリケーションを作成する
  – 当社の標準アプリケーションである“シーケンス制御”用の“Regie”プログラムおよびフレームワークを使ってアプリケーションを組み込みます。
  – NC/PLC データにアクセスします。

注

これらの機能については，「第 2 部 : アプリケーションの組み込み」と「第 3 部 : データアクセス」で説明されています。

- Windows アプリケーションを作成する

  これにより，データアクセスが可能なクライアントアプリケーションを作成できます。

注

この機能については，「第 3 部 : データアクセス」で説明されています。

1.3 基本用語 “ アプリケーションの組み込み ”

次の図には，“Regie” プログラム，“OEM フレーム ”，および当社標準のアプリケーション “ シーケンス制御 ” のフレームワークを使って, アプリケーションを組み込む方法が示されています。

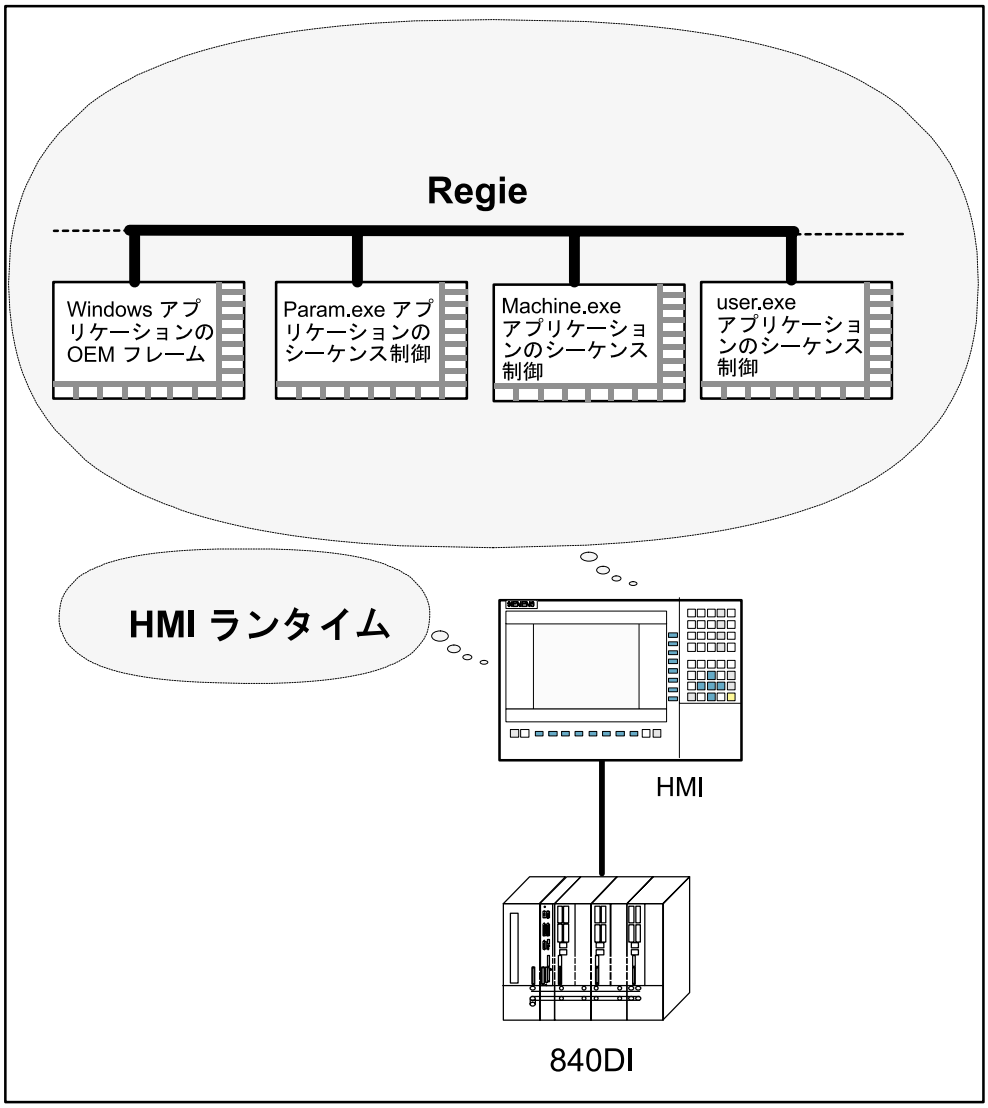


図 1.1 アプリケーションの組み込み

1.3.1 Regie とは

Regie は，ユーティリティやグループアプリケーションの管理（タスク管理）のために設計されたプログラムです。次のような役割があります。

- システムの初期設定
- システムのパワーアップ
- ダイナミックリンクライブラリと MMC のロード
- 正しい順序でのアプリケーションの始動
- システム構成
- 領域の切り替え

Regie は，初期設定ファイルからパラメータを受け取ります。この初期設定ファイルについては，付属する構成ツールを使って編集することができます。これにより，アプリケーションの組み込みが簡単になります。

1.3.2 シーケンス制御とは

シーケンス制御は，当社標準アプリケーションおよび互換性のある OEM アプリケーションのためのフレームワークです。次のような機能があります。

- シーケンスの構造の管理（状態の " メニューツリー "）
- ソフトキー（縦形と横形）の管理
- NC 特殊キーの管理
- ソフトキーのテキストの表示
- ダイアログ行の管理
- 言語のサポート

1.3.3 OEM フレームとは

OEM フレームを使うと，Windows アプリケーションをグループアプリケーションとして作成できます。

1.4 基本用語“ データアクセス ”

次の図には，新しい 840DI でのクライアントとサーバの配置が示されています。

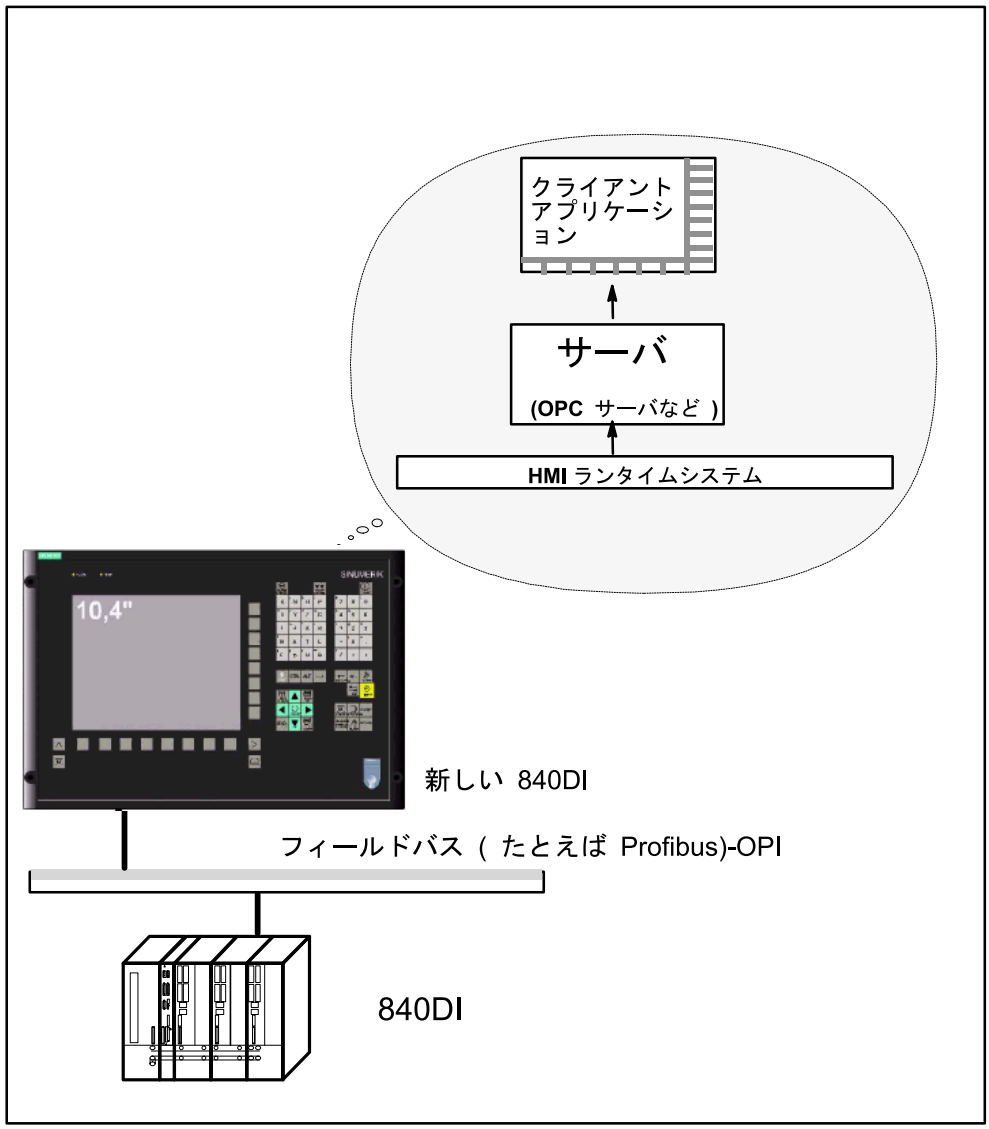


図 1.2 “ データアクセス ” の概要

次の図には，PC 上のクライアントとサーバの配置と共に，OPC/COM インタフェースが示されています。

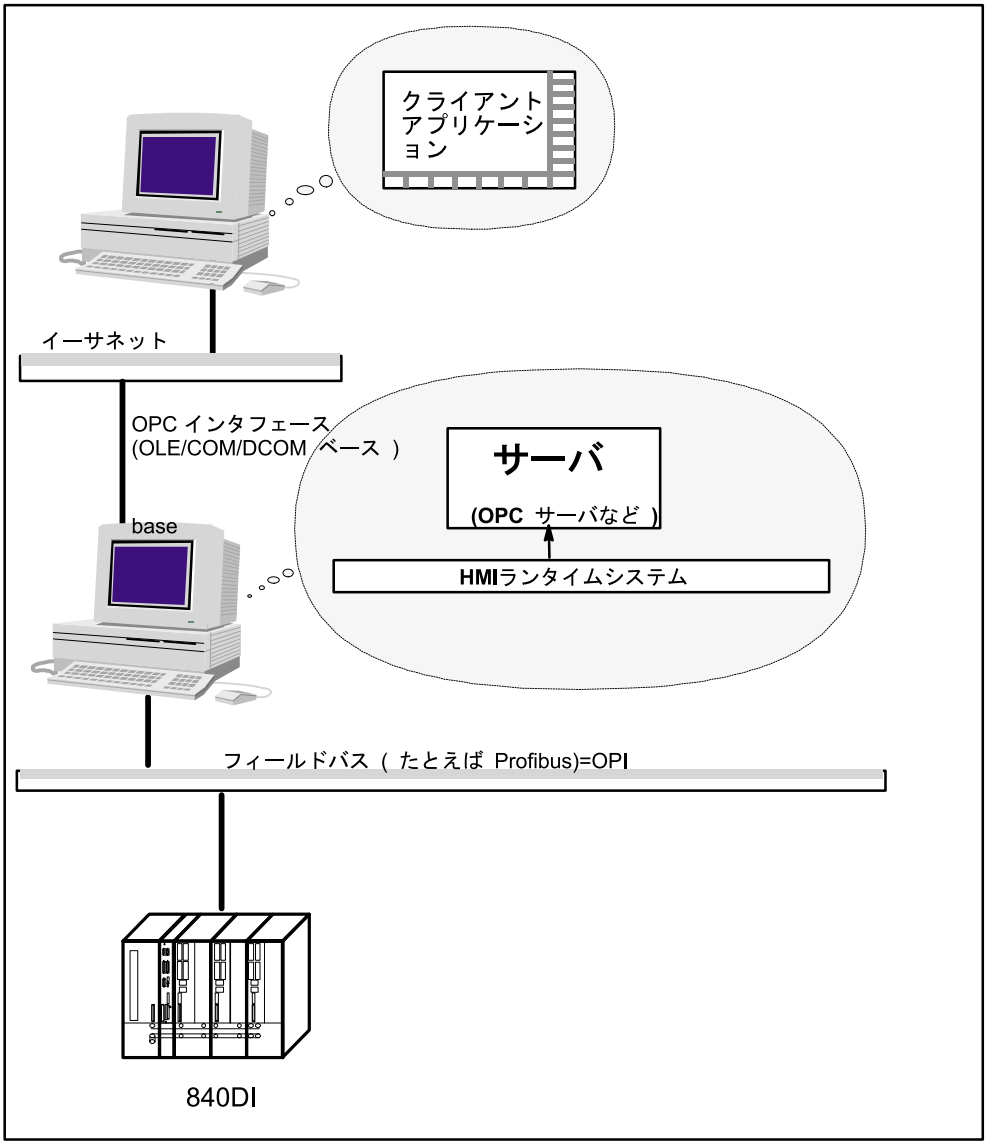


図 1.3 “データアクセス”の概要

1.4.1 クライアントとは

クライアント／サーバシステムでは，クライアントは，サーバとのデータ交換に使われるコンピュータ（または該当するプログラム）のことです。

840DI では，クライアントコンピュータは，サーバ経由で制御システムと通信するために，ユーザが独自のユーザインタフェース（アプリケーション）をプログラムしたコンピュータのことです。

1.4.2 サーバとは

サーバとは，対応するコンピュータにインタフェース経由でデータやサービスを提供する際に必要なソフトウェアです。

HMI– プログラミングパッケージには，次のようなサーバが含まれています。

- OPCServer
  - OPC アラームおよびイベント
  - OPCData（変数）
- COM サーバ
  - ADSI（エクスプローラ）
  - IMCFile（データ管理）
  - IMCDomain
  - IMCCommand（PI サービス）

次の図には，サーバのアーキテクチャが示されています。

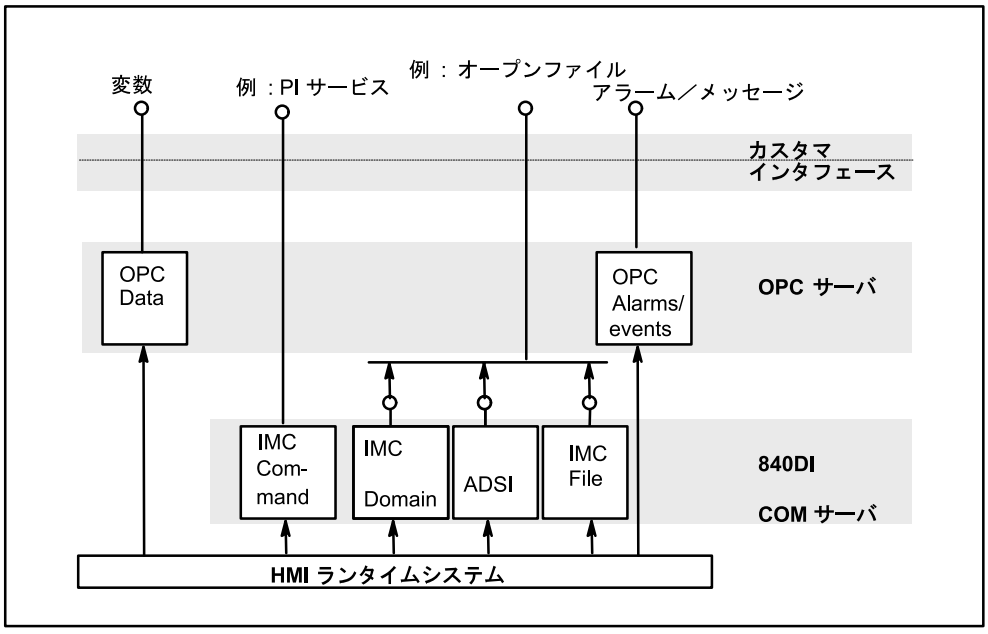


図 1.4 サーバアーキテクチャの概要

1.4.3 OPC の意味

OPC は，**O**LE for **P**rocess **C**ontrol の略語で，自動化エンジニアリングにおける通信の標準インタフェースを意味します。

Object **L**inking and **E**mbedding（OLE）は，Microsoft 社によって作成された規格であり，異なるプログラムのデータを統合して複合ドキュメントにするために発行されました。OLE バージョン 2 が発表されてからは，OLE は COM 規格の通信サービスに基づいたものとなっています（2 章も参照）。

OPC は Microsoft による Distributed Internet Application（DNA）構造および Component Object Model（COM）に基づいているので，OPC という名前が採用されました。これは，簡単に拡張できるように設計されています。OPC 仕様では，自動製造におけるプロセス制御やアプリケーションで COM を使えるようにする，業界標準のインタフェースを定義しています。

OPC の目的は，信号処理端末のために，**統一されていてメーカに依存しないソフトウェアインタフェース**を定義することです。この文脈では，信号処理端末のことを，プログラマブルコントローラ（PLC），数値制御またはその他のフィールド機器のソフトウェアインタフェースとして定義しています。これにより，制御と表示システムまたはオフィスユーザとの間のデータ交換インタフェースは，メーカに関係なく統一されます（4 章も参照）。

1.4.4 OPCServer とは

OPCServer は，標準化された OPC インタフェースをいろいろなメーカのアプリケーションへ提供するプログラムです。

単純な呼び出しを行うだけで，産業用ネットワーク経由で通信できるようになります。OPCServer は，通信ネットワークでは扱うことが難しいメーカ固有の特性をすべて“カプセル化”します。

適用される通信ネットワークおよびプロトコルに関係なく，OPC インタフェース経由でのプロセスデータへのアクセスは，常にこの方法で行われます。

したがって，OPCServer は，プロセスデータを処理するアプリケーションと，それらのデータにアクセスする種々のネットワークプロトコルやインタフェースとの間の中間層を構成します。

1.4.5 COM とは

Component **O**bject **M**odel（COM）は，プロセスとプログラム間の通信を実現するのを援助する，Microsoft 社のソフトウェア規格です。さらに，COM はオブジェクト指向インタフェースを定義します。このインタフェースを使い，プログラムまたはソフトウェア構成要素は良質なサービスを提供します。この規格では，サードパーティのコンポーネントを使うことができるため，ソフトウェアの開発を単純化することができます（2.2.1 節も参照)。

1.4.6 OPCData とは

OPCData とは，OPCServer の 1 つです。クライアントは，このサーバ経由で 840DI の変数の読み書きを行えます。

このインタフェースの詳細な説明については，HMI– プログラミングパッケージの資料に掲載されています。

最新情報については，「OPC Data Access Custom Interface Specification」および「OPC Data Access Automation Interface Specification」

が，OPC Foundation のホームページに掲載されています。

“http://www.opcfoundation.org/opc_spec_document.htm”

（6.1 節も参照）

1.4.7 OPC アラームおよびイベントとは

“OPC アラームおよびイベント ” を使って，SINUMERIK ユーザは，Windows でのそれぞれの条件に応じて，任意で 840DI のアラームを評価することができます。

このインタフェースの詳細な説明については，HMI– プログラミングパッケージの資料に掲載されています。

最新情報については，「OPC Alarms and Events Custom Interface Specification」

および「OPC Alarms and Events Automation Interface Specification」

が，OPC Foundation のホームページに掲載されています。

http://www.opcfoundation.org/opc_spec_document.htm”

（6.2 節も参照）

1.4.8 840DI データ管理とは

840DI データ管理は，ユーザおよびプログラマが，2 つの全くタイプの異なるデータを表示できるようにするために開発されました。

- NC メモリの部分
- オペレータ構成要素でのファイルシステム

両方の方向で表記／ミラーリングできるようにするため，NC で必要とされた同じ制限を，NC メモリのミラーリング先となるオペレータ構成要素のファイルシステム部分にも適用する必要があります。その際に，データ管理ツリーが作成されます。制限は，' スキーム ' と呼ばれます。スキームでは，どのファイルタイプをどこで使えるのかが定義されます。

1.4.9 ADSI とは

ADSI は，階層オブジェクトを扱うための汎用設計を実現する目的で開発されたインタフェースの集まりです。次のような条件があります。

- 設計時に複数のオーバヘッドを保管する
- クライアントとサーバの開発を分離する

840DI データ管理では，インタフェースとして IADs と IDAsContainer とが実現しました。

1.4.10 IMCFile とは

IMCFile には，ADSI を使って設計されたのではないすべての 840DI データ管理機能が含まれています。次のものが関係します。

- 840DI 固有の機能（例：activate）
- 対応する ADS 機能よりも詳細にパラメーター化できる機能（例：copyHere）

1.4.11 IMCDomain とは

IMCDomain を使うと，NC 記憶域と PLC のパーツをファイルシステムのように使うことができるようになります。パラメータ化も扱いますが，これはデータ管理の一貫性を保証するものではありません。

1.4.12 IMCCommand とは

IMCCommand を使うと，コマンドを解釈して実行するために，そのコマンドを NC へ転送できます。

1.4.13 HMI ランタイムシステムとは

ランタイムシステムには，外部プログラムから 840DI へアクセスするときに必要なすべてのコンポーネントが含まれています。これには，適切な COM サーバやフィールドバスのドライバが含まれます。

2 COM/OPC の基本

■ この章の目的

OPC インタフェースの詳しい説明は，本書には載せられていません。OPC の基本についての英語のオリジナル文書ファイルがこの製品に添付されていて，そこに詳しい説明が載せられています。ファイルをインストールしたら，次のメニューから文書にたどり着くことができます。

［スタート］–>［プログラム］–>［SINUMERIK］–>［HMI – Programing Package］–>［Documentatison］–>［Show Documentation］

2.1 OPC サーバの概要

この節では，OPC サーバの機能についての概要を紹介します。産業用通信に OPC を適用することの利点を説明します。本書では，お持ちの背景知識に応じて，資料のどの部分を参照すればよいかが示されています。

2.1.1 OPC インタフェース

OPC インタフェースは，OLE をベースとする，標準化された**メーカに依存しない**ソフトウェアインタフェースです。Microsoft の支援を受けて，オートメーション業界の先端企業が業界標準として設計しました。

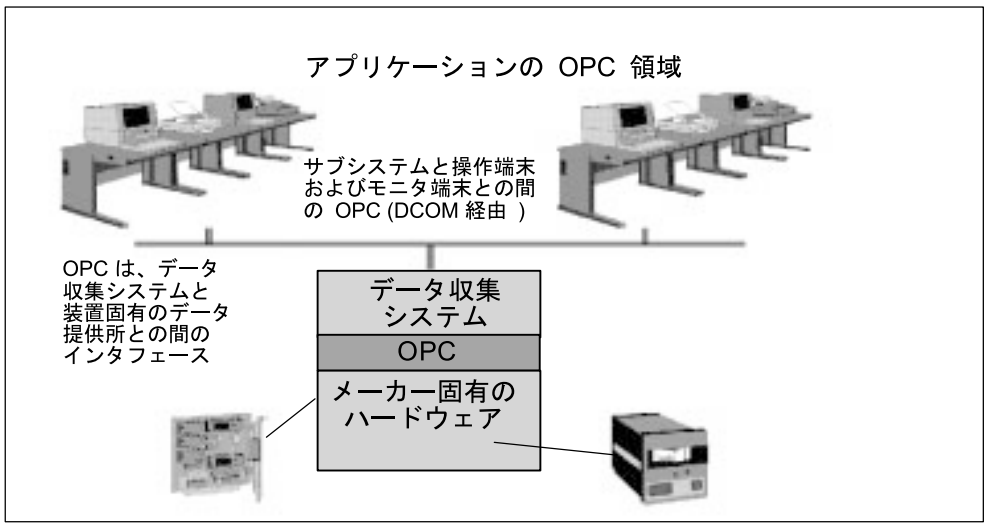


図 2.1 OPC インタフェース

これまでは，ユーザが処理するプロセスにアクセスするアプリケーションは，メーカー固有のインタフェースにリンクされていました。

しかし，標準化された OPC インタフェースが装備されたので，ユーザはプロセスデータへ一様な方法でアクセスできるようになりました。たとえば，OPC インタフェース経由でソフトウェアの操作とモニタを実行できるようになりました。

2.1.2 OPC サーバの利点

■ スタートアップエンジニアにとっての利点

“DCOM” OLE メカニズムを使うと，別のコンピュータにインストールされているアプリケーションは，OPC サーバアプリケーションへローカルネットワーク経由で，またはグローバルネットワーク（インターネット）経由でアクセスできます。

■ 開発者にとっての利点

840DI 向けに，OPC サーバの OPC インタフェース経由でプロセスデータにアクセスするアプリケーションを開発する開発者にとっては，OPC サーバには次のような利点があります。

- OPC インタフェースはメーカに依存しないインタフェースです。このインタフェースを利用すると，作成したアプリケーションの再利用を強化したり，投入したアプリケーションを保護することができ，これまでよりはるかに大規模な市場が開けてきます。
- 開発されたアプリケーションは，メーカの通信システムに依存していないので，適切にプログラミングされていれば，他のメーカーの OPC サーバと通信できます。
- OPC インタフェースでは，OPC サーバや下位レベルの通信ネットワークへ十分にアクセスするかどうかは，それぞれのユーザの自由です。
- OPC には，C/C++ プログラミング言語用の高性能なインタフェースが備えられています。
- Visual Basic（Microsoft）や Delphi（Borland）などのプログラムからプロセスデータにアクセスすることが，ユーザフレンドリでシンプルになりました。
- プロトコルやメーカ固有のインタフェースについてのトレーニングは必要ありません。
- トレース出力により，障害検出が容易になりました。

2.1.3 必要な背景知識

■ ユーザグループ

’OPC Server for 840DI’ 製品は，次のようなユーザグループによって使われます。

- クライアントアプリケーション（自動化エンジニアリング向けの製品）の開発者
- スタートアップエンジニア

OPC クライアントアプリケーションを専門に使っているユーザは，ここに掲載される情報は必要ありません。ユーザは OPC クライアントのサーバアーキテクチャを意識する必要はありません。

■ 必要な背景知識

次の表には，OPC サーバを使うときに必要な背景知識がそれぞれのユーザグループごとに示されています。

表 2.1 必要な背景知識

必要な背景知識	クライアントアプリケーションの開発者	スタートアップエンジニア
PC の知識	必要	必要
C/C++ プログラム言語の知識	必要（Custom インタフェースを使う場合）	必要なし
Visual Basic，Delphi または Power Builder の知識	必要（Automation インタフェースを使う場合）	必要なし
OLE メカニズムについてのアプリケーション固有の知識	必要	いくらかの知識が必要
低レベル通信モジュールの構成	必要	必要
バス上のパートナ端末についての知識	必要	必要

2.2 OLE の基礎

OPC インタフェースは，OLE メカニズムを使い，Windows の通信サービスにアプリケーションを組み込みます。アプリケーションを（Visual Basic や C/C++ で）開発する場合，OLE の基本構造についての基礎的な知識が必要です。

COM をベースにした OLE メカニズムの運用については，技術的な資料の中でよく取り上げられているので，この章では簡単な概要だけにとどめます。

2.2.1 OPC の基礎としての COM

■ 概要

OLE 1 では，Microsoft Windows 3.1 によって，異なる個別のアプリケーションを組み合わせて 1 つにする最初の機会が提供されました。これは，アプリケーションソフトウェアのモジュール化に向けての明確なステップになりました。OPC で使う OLE 2 規格については，Microsoft 社は，OLE 1 を拡張したものであると紹介するにとどまらず，基本的に新しいアーキテクチャであると紹介しています。すなわち，Microsoft Windows での全く新しい方式のソフトウェア開発であるということです。

■ COM

すべての OLE メカニズムの基本は，Microsoft による COM（すなわち **C**omponent **O**bject **M**odel）です。

COM では，オブジェクトを Windows の閉じられた単位で定義できるようにする規格，および処理限界を超えてオブジェクトにアクセスできるようにする規格を定義しています。

オブジェクトは，オペレーティングシステムの拡張部分とみなすことができます。これらはプログラム言語に依存しておらず，原則的に，すべてのアプリケーションで使うことができます。COM 構成要素モデルはクライアント／サーバ構造になっています。

サーバは，サービスを提供するオブジェクトです。

クライアントは，オブジェクトのサービスを使うアプリケーションです。クライアントは，コンテナアプリケーションと呼ばれることもあります。

次に示されているように，サーバからクライアントへのサービス（例：イベント処理）はすべて，COM を使って実現できます。

■ 接続可能なオブジェクトおよびイベントインタフェース

本来のサーバ／クライアントアーキテクチャでは，サーバもしくはサーバオブジェクトがサービスを提供します。

COM では，オブジェクトがインタフェースを提供し，タスクをサーバオブジェクトへ転送するためにクライアントによってメソッドが呼び出されます。

必ずクライアント側から共同作業が開始されます。クライアントは行う必要のある作業をサーバに指示し，サーバはその作業を実行してから，結果があればそれを戻します。

しかし実際には，機能はサーバで実行され，イベントは散発的に発生することがあります。このことをクライアント側が認識する必要があります。

OPC データアクセスサーバでは，たとえば，変数または変数の質を変更するときに，このことが生じます。

クライアントがサーバに対して，たとえば 10 ミリ秒ごとに項目の値が変更されたかどうかを確認するようにすると，非常に煩雑になります。

しかし，サーバがクライアントへ呼び出しを送信できるようにする，より簡単な方法があります。

クライアントはそれ自体のコールバックオブジェクトを実装することができ，そのようにするならば，サーバには，すでに知られている特定のインタフェースが提供されます。

このインタフェースのことを，コールバックインタフェースまたは基底インタフェースといいます。クライアントがインタフェースをサーバへ転送すると，サーバは基底インタフェースのメソッドを呼び出し，クライアントに対してイベントを報告します。

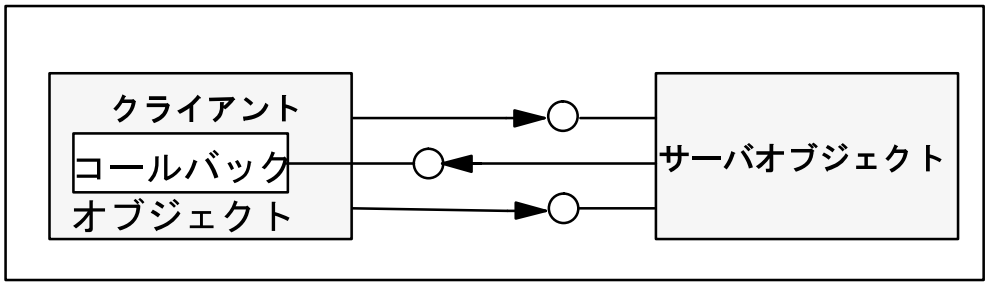


図 2.2 サーバとクライアントとの間のコールバックメカニズム

この理由のため，OPC 向けに IOPCDataCallback インタフェースが定義されています。

OPC クライアントはこのインタフェースを実装する必要があります。どのようにそうするのでしょうか。

このインタフェースを実装する場合，クライアントは，まさにこのインタフェースとそのメソッドを含むオブジェクトを作成する必要があります。

C++ では，定義済みインタフェースのクラスが派生されて開始されるので，仕組みは簡単です。

このインタフェースは，特定のメソッドを宣言します。これらのメソッドは，派生クラスで実装されています。イベントが発生すると，サーバはクライアントインタフェースに実装されているいずれかのメソッドを呼び出します。

たとえば，IOPCCallback OPC インタフェースでは，“OnDataChange()” メソッドを使います。

クライアントがこのインタフェースを OPC サーバ内のグループに転送した場合，そのグループがインタフェースの OnDataChange() メソッドを呼び出します。

これで，このメソッドがクライアントに明確に実装されました。たとえば，この値が変更されるとログに記録されるか，値の表示が更新されます。

このイベントメカニズムは非常に重要なため，Microsoft 社は，いわゆるイベントインタフェース経由で任意のサーバとクライアントとを接続するときの規格を開発しました。

そのような接続は，この規格に基づいて確立され，サーバオブジェクトにより，いわゆる接続点が提供されます。

この目的のため，サーバオブジェクトは，接続点コンテナを提供しています。クライアントは，そのコンテナから，そしてそのコールバックインタフェースのインタフェース ID を転送することにより，接続点を要求します。

サーバがこのコールバックインタフェースをサポートしていれば，その接続点（”IConnectionPoint”）がクライアントに戻されます。

このインタフェースでは，“Advise()” メソッドを使います。このメソッドを使うことにより，サーバの接続点とクライアントの基底インタフェースとの間の接続が確立され，クライアントはそのコールバックオブジェクトの基底インタフェースへのポインタを戻します。これで，サーバはこのインタフェースのメソッドを呼び出し，イベントをクライアントへ報告できるようになります。

このメカニズムはやや高価であるため，Visual Basic のランタイムシステムに組み込まれています。ランタイムシステムは，Visual Basic のプログラマが行う数多くのジョブを実行します。

1. コールバックインタフェースを提供する
2. そのコールバックインタフェースをサーバオブジェクトへ接続し，再び切断する
3. イベントを独自の処理ルーチンに中継する

これにより，サーバのイベントに応答するクライアントのプログラミングを，相対的にユーザフレンドリかつシンプルに保てます。

■ DCOM

Windows NT のバージョン 4 では，COM インタフェース機能は，コンピュータの境界を越えてオブジェクトへアクセスできるように拡張されています。アプリケーションで使われているオブジェクトは，ネットワーク経由で配布されます。この COM の拡張機能のことを DCOM（**D**istributed **COM**）といいます。

2.2.2 オブジェクトとインタフェース

■ OLE オブジェクト

OLE オブジェクトは，他のオブジェクトに対し，それぞれのインタフェースを使って定義済みの機能を提供するときの，Windows における単位のことです。OLE オブジェクトは，定義されたインタフェースを使い，それぞれのサービスを提供します。

オブジェクトの“内容”（データとコード）は，オブジェクトのユーザ側からは見えません。OLE オブジェクトはそれぞれのインタフェースによって定義されます。

OLE では，“オブジェクト”という用語は，オブジェクト指向プログラム言語でのオブジェクトの定義には対応していません。たとえば，OLE オブジェクトでは継承をサポートしていません。

■ OLE オブジェクトの構造

次の図には，例として 4 つのインタフェースを備えた OLE オブジェクトが示されています。このオブジェクトへは，いずれか 1 つのインタフェースを使ってアクセスされます。実際のオブジェクト（そこに含まれるデータまたはコード）には，アクセスされません。インタフェースは，割り当てられたメソッドを隠しています。

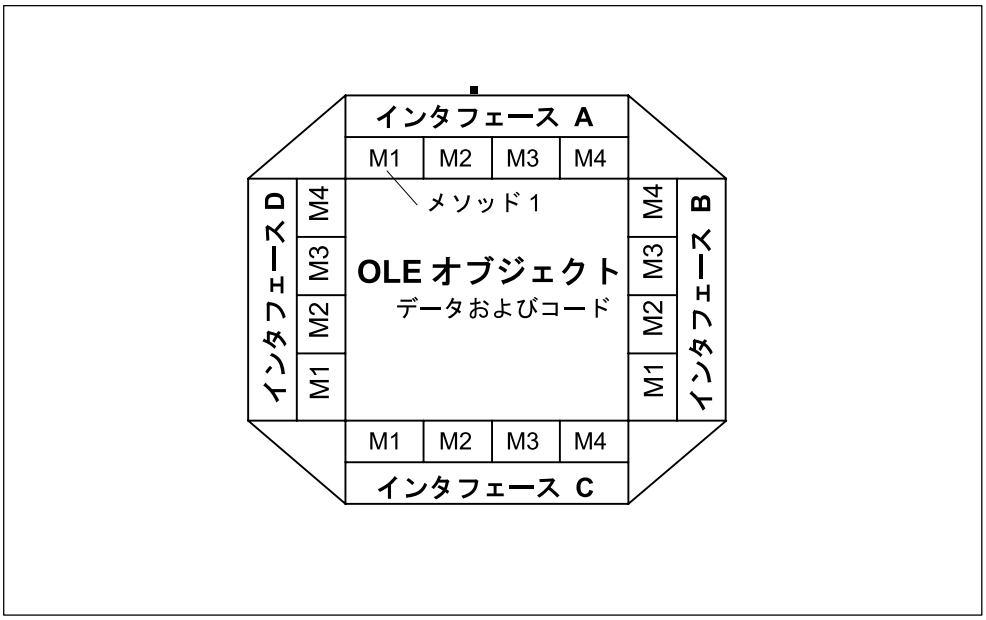


図 2.3 OLE オブジェクトの構造

■ 用語 ' インタフェース '

オブジェクトは複数のインタフェースを持つことができ，それぞれのインタフェースはオブジェクトを完全に指定します。メソッド（オブジェクト機能）はインタフェース経由でのみ実行することができ，オブジェクトデータも同様にインタフェース経由でのみアクセスできます。

オブジェクトを複数のアプリケーションで使うことができます。1 つのインタフェースは，機能へのポインタのテーブルで構成されています。

■ インタフェースの構造

下記の図には，インタフェースの基本設計が示されています。OLE 2.0 では，インタフェースへのポインタは，機能ポインタを備えたブランチテーブルへのポインタです。

呼び出し元は，望みのインタフェースへのポインタを使います。このときに，このインタフェースには，OLE オブジェクトメソッドを実装する機能ポインタのリストへのポインタが含まれます。

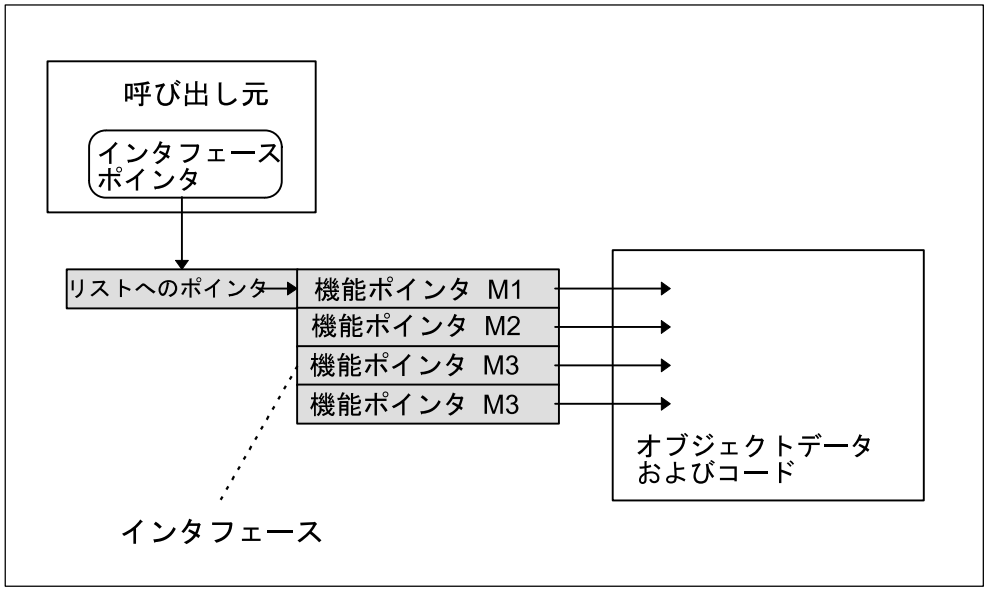


図 2.4 インタフェースの構造

2.2.3 OLE でのオブジェクトの標準表記

■ 説明

OLE の資料では，OLE オブジェクトを表すときに，一般にグラフィカルな方法で表記します。本書でもそのように表記します。特定のインタフェースをサポートする機能のリストがまとめられ，インタフェースそのものは円で表されます。この円からは一本の線が伸ばされます。インタフェースは，そのインタフェースがサポートするオブジェクトと共に表記されます。オブジェクト固有のインタフェースは，オブジェクトを横から指します。すべてのオブジェクトに提供されている IUnknown インタフェースは，オブジェクトの上面にリンクされます。

■ OLE オブジェクトの図

次の図には，オブジェクトのユーザが，オブジェクトの個々のインタフェースだけにアクセスできることが示されています。オブジェクトそのものへのポインタはありません。

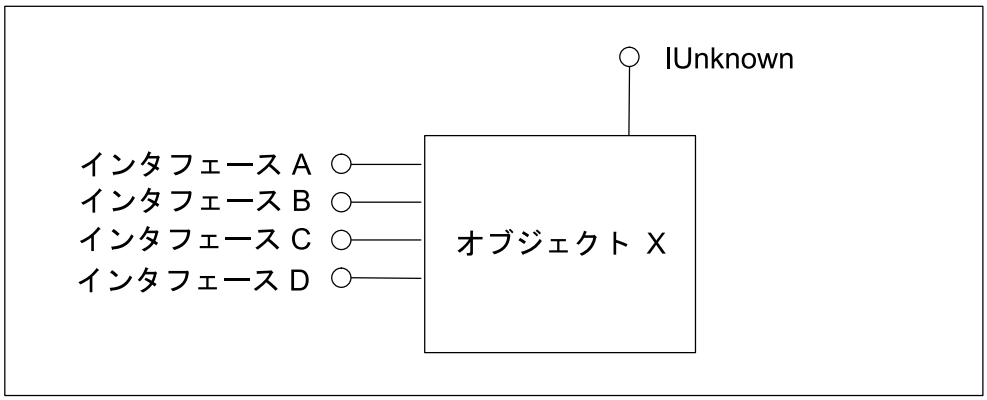


図 2.5 OLE オブジェクトの図

2.2.4 Automation インタフェースによるオブジェクト制御

■ Automation

Automation は，COM コンポーネントモデルに基づく OLE 2 のコンポーネントです。Automation は，オブジェクトが配布インタフェースで認識するコマンドを明らかにします。元々は，Automation インタフェースは，マクロ言語を使ってオブジェクトを制御する（たとえば，テキストプログラムにグラフィックを組み込む）ために設計されたものです。

■ スクリプト言語の使用

ここまでで説明されてきた COM インタフェースは，スクリプト言語を使った開発ツールの分野にはあまり適していません。分散インタフェースを使って OLE オブジェクトを拡張することにより，オブジェクトメソッドに対して，簡単なスクリプト言語でもアクセスできるようになります。これにより，例えば Visual Basic，Delphi または Power Builder などの開発環境でも，OLE オブジェクトを組み込むことが可能になります。

開発ツールでは，オブジェクトについての呼び出しの詳細がすべて隠されます。OLE Automation サーバへのプログラムによるアクセスは，プロパティとメソッドによって定義されます。

2.3 OPC の基礎

この節では，OPC の概念を説明します。OPC における OLE の役割について学習します。OPC インタフェースの概念について学習し，基本として使われているクラスモデルに精通することを目指します。

OPC のクライアント／サーバアーキテクチャについての詳細は，英語版の OPC Foundation の資料を参照してください。

本書は，OPC Foundation の資料とは別に，文書資料の一部として提供されています。

2.3.1 OPC における OLE の役割

■ オペレーティングシステムの一部としての OPC サーバ

OPC サーバには，メーカーに依存しないインタフェースが備えられています。COM コンポーネントモデルを使うと，OPC サーバは Windows オペレーティングシステムの一部のようになるので，ファイル名，記憶域，およびバージョンに拘束されなくなります。

■ OLE を使う理由

このような特性に加え，一番大きなこととして，Microsoft 社の Windows が広範囲に使われていることが，COM そして OLE に基づいてメーカーに依存しないインタフェースを設計した理由です。

■ 強力な開発環境

OLE を使う利点をさらに挙げるならば，強力な開発環境が備えられていて，OLE オブジェクトによって実装される機能に簡単に対応できるということです。（Visual Basic，Delphi など。）

2.3.2 2 種類の OPC インタフェース

■ OLE インタフェース

OLE コンポーネントは，他のコンポーネントまたはアプリケーションに，オブジェクトとそのメソッドを提供します。OLE インタフェースは，これらのオブジェクト経由でアクセスされます。OLE におけるインタフェースは，相互に所属する一群の論理機能です。

■ 2 種類のインタフェース

OPC Server for SIMATIC NET は，2 つの異なる種類の OLE インタフェースをサポートしています。

- Automation インタフェース
- Custom インタフェース

どちらのインタフェースも，オブジェクト間のやり取りに使われます。

Automation インタフェースと Custom インタフェースとの違いは，インタフェースのメソッドを呼び出す方法です。OPC サーバには，さらに別のインタフェース仕様もあります。

- Automation インタフェースのインタフェース仕様
- Custom インタフェースのインタフェース仕様

■ OPC サーバの図

次の図には，どの OPC サーバインタフェースを使って，どの種類のアプリケーションを実行できるかについての一例が示されています。

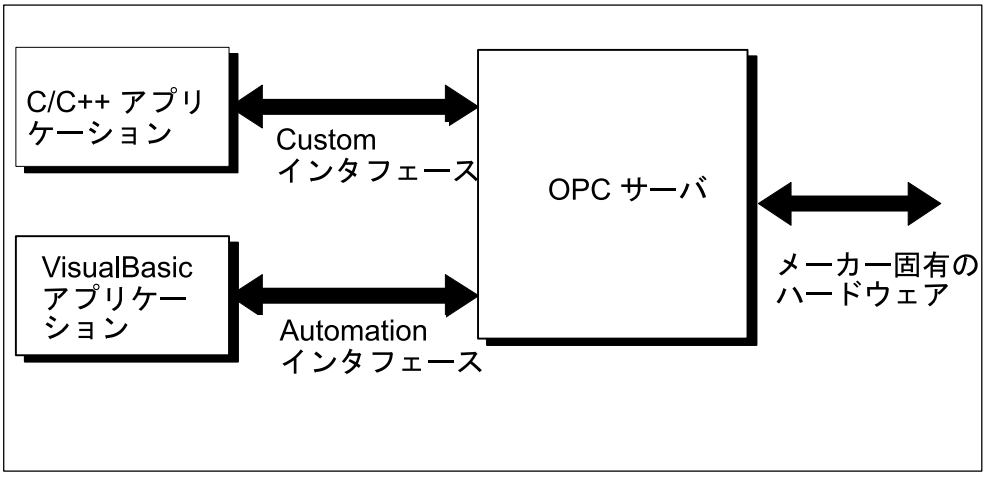


図 2.6 OPC サーバのインタフェース

■ インタフェースの決定方法

Visual Basic スクリプトまたは VBA などのスクリプト言語に基づくクライアントアプリケーションを使う場合，Automation インタフェースを使うことをお勧めします。

一般的な規則として，C/C++ でプログラムされたアプリケーションでは，最大のパフォーマンスを得るためには，Custom インタフェースを使うべきです。

しかし，C/C++ では，Automation インタフェースを使うことも可能です。

2.3.3 OPC のクライアント／サーバアーキテクチャ

■ 説明

すべての OLE のインプリメンテーションと同じように，OPC はクライアント／サーバアーキテクチャを有しています。OPC サーバには，定義された機能を持つ OPC クライアントインタフェースが備えられています。クライアントはそれらのインタフェースを使い，サーバでオブジェクトを作成，使用，および削除することができます。

■ OLE サーバの種類

サーバの組み込み方に応じて，OLE サーバは次の 3 つの種類に区分されます。

- プロセス内サーバ
- ローカルサーバ
- リモートサーバ

呼び出し側のアプリケーションは，どの種類のサーバを使っているのかを区別する必要はありません。呼び出しメソッドの構文はすべて同じです。

■ プロセス内サーバ

プロセス内サーバは，クライアントと同じ処理領域にあります。使えるのは，このプロセスだけです。

プロセス内サーバは DLL です。

OPC サーバの OPC Automation インタフェースでは，Automation インタフェースを実装するプロセス内サーバが必要です。

プロセス内サーバは，OPC Custom インタフェース経由でローカルサーバにアクセスします。

■ ローカルサーバ

ローカルサーバは，クライアントと同じコンピュータ上で稼働します。ローカルサーバは，独立したそれぞれのアプリケーションの場合と同じように，独自のプロセスおよびアドレス領域を持っています。OPC Server for SINUMERIK はローカルサーバであり，EXE ファイルとして実装されます。

■ リモートサーバ

リモートサーバは別のコンピュータ上にありますが，ネットワーク経由でアクセスできます。

OPC サーバの提供側は，このサーバがプロセス内サーバなのかローカルサーバなのかを判別します。ユーザは，リモートサーバとして運用を構成する必要があります。

■ コンポーネントへのアクセス

次の図には，さまざまな種類のクライアントから，OPC Server for 840DI のコンポーネントへアクセスする方法が示されています。

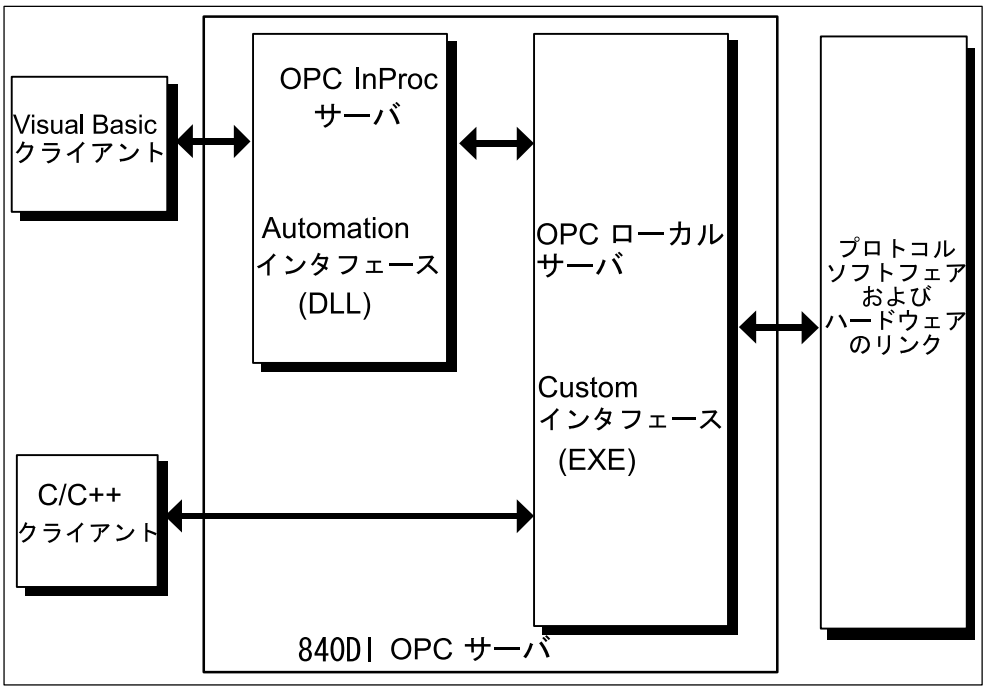


図 2.7 コンポーネントへのアクセス

3 入門ーアプリケーションを作成する方法

■ この章の目的

この節では，アプリケーションを作成する手順の原則について説明します。

以下の事柄について解説します。

- アプリケーションとは
- カスタマ特定のユーザインタフェースの構成
- 必要とされる前提知識
- アプリケーションを実行するために必要な前提事項
- ユーザがどのように進めるか
- アプリケーションに応じて読む必要のある資料のセクション

3.1 840DI におけるアプリケーションとは

840DI においては，アプリケーションは PC または MMC 上で実行する HMI 固有のユーザインタフェースです。

HMI ユーザプログラムは，既存のインタフェースを介して 840DI の NC/PLC と通信します。

これらのユーザプログラムは，C++ または Visual Basic などのプログラム言語を使って作成されます。

これらのプログラム言語は COM および DDE をサポートします。

ユーザプログラムは，ユーザインターフェースを視覚化します。

NC/PLC からの（NC/PLC への）データ／イベントは，これらのユーザインタフェースを使って読み取る（書き込む）ことができます。

こうしたアプリケーションは，HMI–Advanced 標準ソフトウェアに統合するか，HMI–Advanced 標準ソフトウェアを使わずに完全に実行することができます。

3.2 カスタマ特定のユーザインタフェースの構成

カスタマ特定のユーザインタフェースは，たとえば以下のもので構成できます。

- 次のようなデータ
  - NC/PLC 変数にアクセスする
  - データ管理でファイルを開く／閉じる
  - アラームとメッセージを受信する
  - NC/PLC への送信コマンド（PI サービス）
- ボタン
- テキスト
- グラフィックス
- 会社のロゴ

3.3 必要とされる前提知識

HMI アプリケーションプログラマは，一般の Windows プログラムと NC 固有の要件との間にあるインタフェースを使って作業します。そのため，両方の内容に精通し，組織的なソフトウェアプロジェクトでの経験を積んでいる必要があります。

プログラマには，以下のものが必要です。

- PC レベルでソフトウェアプロジェクトを実装した経験
- オブジェクト指向の問題解決およびプログラミングの知識
- COM および DDE をサポートしているプログラム言語の基礎知識，たとえば，
  - ー Visual Basis Professional での豊富なプログラミングの経験
  - ー Visual C++ などのリソースエディタでの経験
- NC の分野に関する知識

  この分野の要求は，主にタスク定義の技術的な“深さ”に依存しています。

  - ー 制御システムの構造に関する知識
  - ー 必要なデータについての精通度
  - ーテクノロジ固有のユーザフレンドリなユーザインタフェースを設計した経験

3.4 補足的な条件

■ 概説

アプリケーションは，以下の場所で実行するために作成されます。

- HMI–Advanced 標準ソフトウェアを持つ標準の PCU
- たとえば，PC

■ 前提条件

カスタマアプリケーションを開発するには，次のような前提条件があります。

- プログラム開発のための標準的な PC
- Windows アプリケーション用の標準的な開発ツール
- 独自の開発を行う基礎としての HMI プログラミングパッケージ
- システム環境でカスタマアプリケーションを検査するのに使用する 840DI コントロール
- NC システムに関する幅広い知識と，最近の高水準言語での十分なプログラム技術を備えた開発者

■ ハードウェア

HMI–Advanced カスタマアプリケーションの開発には，WindowsNT® Service Pack 6 で動作する業界標準の PC が必要です。

システムの要件については，“インストールガイド“ で要約されています。

■ ソフトウェア

HMI–Advanced カスタマアプリケーションの開発では，以下に示す標準的な開発ツールを使用します。

- Microsoft Visual Studio 6.0
- DLL にテキストを実装するツール（たとえば，Visual C++）
- HMI プログラミングパッケージ

外部ツールで必要とされる有効範囲や運用に関する推奨事項については，“インストール資料 ” を参照してください。

■ テストに関する情報

HMI プログラミングパッケージのユーザが作成した Windows プログラムは,

- PC でテストされ,
- モデル環境でシミュレートされ,
- PCU に転送されます。

最終的なテストは，フィールド内に実際の条件を想定し，ターゲットのハードウェア，関連する構成を使った完全な制御システムで実行します。

■ HMI プログラミングパッケージでのサーバおよびソース

以下の表では，個々のパッケージでインストールされるサーバおよびソースを示します。

表 3.1 インストールされるサーバとソース

	DDE/COM/OPC サーバ	HMI 環境	PC 用の HMI Advanced
DDE サーバ	○	○	○
IMC サーバ	○	○	○
OPC サーバ	○	−	−
Regie OEM フレーム	○	−	○
シーケンス制御	−	○	−
ヘッダ	−	○	−
サンプル	○	○	○

3.5 アプリケーションを実行する際の前提条件

HMI プログラミングパッケージを使用すると，以下を基礎とするカスタマアプリケーションを作成できます。

- DDE または
- COM または
- OPC

OP または PC/PG（プログラミング機器）でさまざまなカスタマアプリケーションを実行するには，適切な実行時環境を設計する必要があります。

以下の表には，HMI プログラミングパッケージを使った個々のインストールが含まれています。

■ OP での HMI 実行時環境

OP で HMI 実行時環境を作成するには，以下のインストールが必要です。

表 3.2 OP HMI 実行時環境

	HMI Advanced なしの完全 OP（標準ソフトウェア）	HMI Advanced なしの OP	HMI Advanced 付きの OP
DDE（HMI 環境なし）でカスタマアプリケーションのみ	DDE/COM/OPC サーバインストール	OP 用の HMI Advanced インストール	これ以上のインストールは必要ありません
COM（HMI 環境なし）でカスタマアプリケーションのみ	IDDE/COM/OPC サーバインストール	OP 用の IHMI Advanced インストール	これ以上のインストールは必要ありません
OPC（HMI 環境なし）でカスタマアプリケーションのみ	IDDE/COM/OPC サーバインストール	OP 用の HMIAdvanced インストール，および DDE/COM/OPC サーバインストール	DDE/COM/OPC サーバインストール

■ PC プログラミング機器（PG）での HMI 実行時環境（HMI Advanced なし）

PC/PG で HMI Runtime を操作するには MPI カードが必要です。

PC/PG で HMI 実行時環境を作成するには，以下のインストールが必要です。

表 3.3 PC/PG HMI 実行時環境

	HMI Advanced なしの完全 PC/PG	HMI Advanced なしの PC/PG
DDE（HMI 環境なし）でカスタマアプリケーションのみ	DDE/COM/OPC サーバインストール	PC 用の HMI Advanced インストール
COM（HMI 環境なし）でカスタマアプリケーションのみ	DDE/COM/OPC サーバインストール	PC 用の HMI Advanced インストール
OPC（HMI 環境なし）でカスタマアプリケーションのみ	DDE/COM/OPC サーバインストール	PC 用の HMI Advanced インストール，および DDE/COM/OPC サーバインストール

3.6 主なプロシージャ

ユーザ用のサンプルが HMI プログラミングパッケージに入っています。

注 :“インストールガイド”の「提供されているユーザサンプル」という節を参照してください。

通常はこれらのサンプルを使用し，アプリケーションの作成方法についての理解が深まるに従ってこれらのサンプルを拡張してください。

3.7 参照する必要のある章

「まえがき」と 1.1 ～ 3.6 の章／節を読んだという前提で，以下の表では，特定のアプリケーションを作成する場合に読む必要のある章／節を示します。

表 3.4 参照する必要のある章

実行したい事柄	参照する章
Regie 用のアプリケーション	4 章「Regie] 5 章「シーケンス制御」
OEM 出口ルーチン（HMI 環境）	4 章「Regie」 5 章「シーケンス制御」
シーケンス制御付きアプリケーション	5 章「シーケンス制御」
HMI Advanced 標準なしの HMI アプリケーション	必要なサーバについて説明した適切な章
Windows アプリケーション（OEM フレーム）	4 章「Regie」
NC/PLC からの個々のデータの読み取り／書き込み	6.1「OPCData」 8.2.1「DCTL32.OCX」
NC/PLC からのアラーム／メッセージの転送	6.2 「OPC アラームおよびイベント」
NC/PLC からのファイルの転送	7.1 「データ管理インタフェース」
NC/PLC からのコマンドの転送	7.3「コマンドインタフェース」

第２部：アプリケーションの組み込み

第２部：アプリケーションの組み込み

4 Regie

■ この章の目的

この章の内容は，“UserManual_HMI–Environment.pdf” ファイルに含まれています。

これらの章は，次回の版で補足される予定です。詳細については，「Yaskawa Siemens 840DI API 取扱説明書 HMI プログラミングパッケージ基礎編」の 6 章「Regie」を参照してください。

5 シーケンス制御

■ この章の目的

この章の内容は，“UserManual_HMI–Environment.pdf” ファイルに含まれています。

これらの章は，次回の版で補足される予定です。

詳細については，「Yaskawa Siemens 840DI API 取扱説明書 HMI プログラミングパッケージ基礎編」の 7 章「シーケンス制御」を参照してください。

第 3 部 : データアクセス

6 OPCServer インタフェース

■ この章の目的

この章では，以下のインタフェースについて説明します。

- OPCData
- OPC アラームおよびイベント

ここでは，プログラミングの簡単な紹介にとどめます。さらに詳しい情報については，OPC Foundation の仕様一覧を参照してください。

6.1 OPCData

OPCData とは，OPCServer の 1 つです。クライアントは，このサーバを介して SINUMERIK 変数の読み書きを行うことができます。

このインタフェースに関する詳細な説明は，HMI– プログラミングパッケージの資料に掲載されています。

現在詳しく説明されているインタフェースとしては,

“OPC Data Access Custom Interface” および

“OPC Data Access Automation Interface” があり,

これらは，以下の OPC Foundation のホームページでご覧いただけます。

“http://www.opcfoundation.org/opc_spec_document.htm”

6.1.1 OPCData クラスモデル

■ 構造

OPCData の仕様では，インタフェースとそのメソッドが 3 層のクラスに分類されています。この構造をクラスモデルと呼びます。

OPCData の Automation および Custom インタフェースは両方とも, このクラスモデルに基づいています。このクラスは OLE オブジェクトによって実装され，オブジェクトとしての参照がさらに簡単になります。

ただし，“OPCItem” クラスのオブジェクトは，Automation インタフェースにのみ存在します。

■ クラスモデル

クラスモデルは，3 つのオブジェクトクラスで構成されています。

- OPCServer
- OPCGroup
- OPCItem

以下の図には，OPC クラスモデルが示されています。

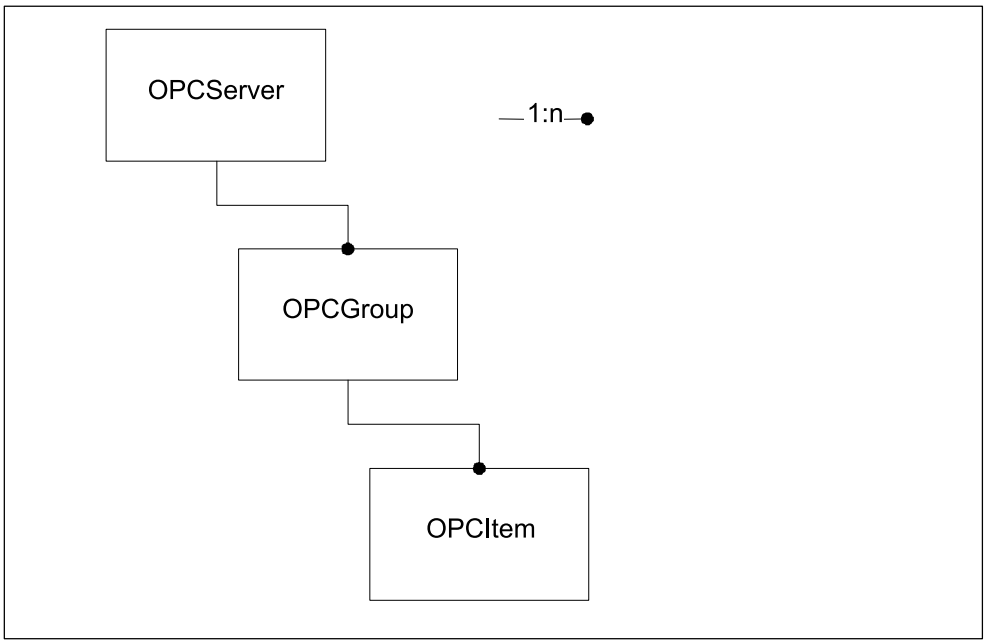


図 6.1 クラスモデル

■ OPCServer クラス

OPCServer は，最上位にあります。OPCServer クラスには，OPCServer オブジェクトの状況，バージョンなどの情報を含むさまざまな属性が入ります。さらに，OPCServer には，クライアントが OPCGroup クラスでオブジェクトを管理するために適用するメソッドがあります。クライアントアプリケーションが，このクラスで行えるのは，COM メカニズムを通じて直接オブジェクトを使用することだけです。その他のオブジェクトは，該当する OPCMethod によって作成されます。

■ OPCGroup クラス

OPCGroup クラスは，個々のプロセス変数（OPCItem）を管理します。

これらのグループオブジェクトを使用して，クライアントは意味のある OPCItem を作成して，その操作を実行できます。

たとえば，マシン A のプロセス変数をすべて 1 つのグループに入れ，マシン B のプロセス変数をすべて 2 番目のグループに入れることができます。

しかし，オペレータ制御およびモニタシステムの画面ページにあるすべてのプロセス変数を，グループに要約することもできます。このグループは，ページが表示されるとアクティブになります。

サーバクラスのオブジェクトは，OPCGroup クラスにあるオブジェクトのコンテナとみなすことができます。

■ OPCItem クラス

OPCItem クラスにあるオブジェクトは，たとえば，プログラマブル制御装置の入力モジュールを使って，プロセス変数とのリンクを構成します。プロセス変数は，プロセス入出力の書き込み可能／読み取り可能（あるいはその両方）のデータです（たとえば，タンクの温度）。

OPCGroup クラスのオブジェクトは，OPCItem クラスにあるオブジェクトのコンテナとみなすことができます。

6.1.2 数量操作

■ 定義

OPC 仕様は，さまざまな方法で，OPCGroup クラスのオブジェクトのいわゆる数量操作をサポートします。メソッドを呼び出して複数のオブジェクトを操作できる場合に，数量操作について考慮できます。

■ OPC ユーザの利点

たとえば，“Read” OPC メソッドを呼び出すことにより，1 つのコマンドでグループのすべての OPCItem オブジェクトを読み取ることができます。数量操作の別の利点は，ソフトウェアコンポーネントのメソッドを呼び出した結果生じる変更処理の数を少なくできるという点です。これにより，メソッド呼び出しの実行速度が向上するため，ソフトウェアコンポーネントが，さまざまなコンピュータ，さまざまなプロセスで検出される場合に特に便利です。

さらに大きな利点は，数量操作の最中に，SIMATIC NET OPC サーバはしばしば OPCItem オブジェクトへのアクセスを最適化するため，ネットワーク上の負荷が減少するということです。

6.1.3 OPC インタフェースの記述のために使用する用語

インタフェースを記述する際に使用されるいくつかの用語について，以下に説明します。OPC 用語に関するさらに詳しい説明は，840DI サーバの DOC ディレクトリに保管されている OPC 仕様書を参照してください。

840DI サーバの特殊性については，この章で言及します。

■ 用語の限定

用語はアルファベット順にリストされています。各定義では，この用語が OLE，OPC，または Windows 環境のどれに属するかを示します。

■ ActiveStatus （OPCData）

OPCData サーバは，プロセス変数の読み取り値や書き込み値に加え，プロセス変数をモニタするための代替手段を提供します。

一定の間隔を開けて実行した後（モニタの間隔を参照），サーバがプロセス変数の値が変化したかどうか検査することをモニタといいます。値が変化すると，新しい値は自動的にクライアントに送信されます。クライアントは，ActiveStatus を実装することにより，プロセス変数のモニタを制御します。グループや OPCItem オブジェクトにはそれぞれ，ActiveStatus 属性があります。

クライアントがプロセス変数をモニタする場合，このプロセス変数への接続を実装している OPCItem の ActiveStatus を **true** に設定します。

さらに，グループの ActiveStatus には **true** 値がなければなりません。

グループがアクティブでない場合，このグループでのプロセス変数のモニタは行われません。

■ キャッシュ（OPCData）

キャッシュは，OPCItem を介して参照されるすべての変数用のバッファです。キャッシュはすべてのユーザがグローバルに使用します。それゆえ，複数のユーザが同じ変数を参照する場合には，その変数は 1 度だけキャッシュに格納されます。

読み取り操作（読み取りおよび最新表示など）では，OPC_DS_CACHE オプションでサーバのキャッシュからデータを読み取ることができます。

注 :840DI サーバは，キャッシュの管理は行いません。

■ 列挙型定数（Windows）

列挙型定数を使うと，ひとまとまりのオブジェクトを繰り返すことができます。

■ ItemID （OPCData）

ItemID は，プロセス変数を個別に識別する文字列です。

ID の構文はサーバによって異なります。ItemID によって，項目に割り当てられるプロセス変数が決まります。

■ LCID （Windows）

ローカル ID の略語。LCID は，ユーザが Windows オペレーティングシステムで選択した地域の設定を示します。

この情報は，国に固有の形式でデータを表示するために使用します。LCID は，Windows の API 呼び出しで判別できます。

注 :840DI OPC サーバは，希望に応じて，ドイツ語または英語のいずれかの言語のエラーメッセージを送信します。

■ OPCHANDLE （OPC）

OPCHANDLE は，グループと OPCItem に使用します。これを使うと，クライアントとサーバ両方の個々のオブジェクトに高速でアクセスできるようになります。

ハンドルの値は，選択したインプリメンテーションによって異なります。クライアントは，値の範囲を前もって想定することはできません。同様に，サーバは，クライアントハンドルについて想定することはできません。

サーバハンドルは，OPCServer オブジェクトが存在している間だけ有効です。そのため，保存しておいて再始動した後，再使用することはできません。グループのサーバハンドルは，サーバ内で固有です。項目のサーバハンドルは，グループ内で固有です。

クライアントハンドルは，OPCItem ID としてコールバックを使って返信されるため，非同期操作に不可欠です。

■ REFIID（Windows）

REFIID は，インタフェース ID（短縮形 : IID）に対する参照またはポインタです。

インターフェース ID は，あいまいでないインタフェースキーです（たとえば，OPC では “IOPCItemMgt” インタフェース）。特定のインタフェースへのポインタを取得するには，そのインタフェースの IID をパラメータとして含める必要があります。

■ 要求データ型（OPCData）

ユーザは，要求データ型を指定することにより，データ型を決定することができます。データ型を明示的に選択しない場合，変数には正規データ型が与えられます。OPC 仕様では，スカラー型の変換だけが規定されています。これに加えて，SINUMERIK の OPCServer は，VT_BSTR のすべてのデータ型（フィールドを含む）を変換することができます。

■ 時差（OPC）

時差（Time Bias）は，UTC 時間と地方時間帯との差を指定します。この点を考慮することにより，OPCServer や OPCClient を世界中のネットワークに簡単に配布することができます。

■ UpdateRate （OPCData）

UpdateRate モニタ間隔では，OPC 値が変化したかどうかをサーバが検査する時間の間隔を決定します（“ActiveStatus” を参照）。モニタ間隔は，グループオブジェクトの属性です。

840DI の OPCData サーバは，100 ミリ秒の UpdateRate だけをサポートします。

希望するレートがサポートされていない場合，このサーバは，次善の値で構成される代替レート（”RevisedUpdateRate”）を報告します。

■ VARIANT （Windows）

VARIANT は，OLE オートメーションで主に使われる特殊データ型です。

VARIANT 型の変数には，さまざまな変数タイプの値に加えて，Empty 値，エラー値，および Null 値を含めることができます。

■ 正規データ型（OPCData）

正規データ型は，OPC サーバが内部で使用する変数のオリジナルのデータ型です。

■ AccessPath （OPCData）

“AccessPath” は，OPCItem クラスにあるオブジェクトの任意指定情報です。アクセスパスの要件および解釈は，それぞれのサーバによって異なります。必要であれば，アクセスパスは，プロセス変数のデータをサーバが読み書きするパスを指定します。

たとえば，アクセスパスは，パスまたは接続名にすることができます。

6.1.4 OPCData の Custom インタフェース

この節では，OPCData の Custom インタフェースの使用方法を示します。

■ OPC インタフェースのバージョン

OPC Custom インタフェースの拡張バージョン（バージョン 2.0）が作成されました。このバージョンでは，既存の OPC Custom インタフェースに，非同期通信の取り扱いを簡単にするためにいくつかの点が追加されています。

バージョン 1.0 の OPCData のインタフェースとバージョン 2.0 との間には，完全な下位互換性があります。

■ C/C++ での OLE オブジェクトの作成と使用

以下の節では，OLE クラスのインスタンスのメソッドを C++ で呼び出す方法について，段階的に示します。OLE のクラス用語と C++ でのクラス用語の違いに注意してください。

■ OLE クラスと C++ クラス

Windows オブジェクトは，OLE クラスに属するインスタンスです。OLE クラスの用語は，C++ のクラスで使用する用語とは異なっています。C++ では，クラスとは型定義のことです。OLE クラスは，オブジェクト記述のことで，型は含まれていません。

■ クラスの識別コード

各 OLE クラスは，128 ビット長の識別コード（CSLID）によって明確に識別できます。オペレーティングシステムは，このクラスによって実装されている DLL または EXE ファイルを明確に割り当てるために，この CLSID を使用します。クライアントがクラスのオブジェクトを使用するために必要なのは，CLSID だけです。

■ ProgID

サーバの ID を単純化するため，通常 CLSID には読み取り可能な名前（ProgID）が割り当てられます。生成用のアルゴリズムが原因で CLSID は常に固有に確定されますが，あいまいな ProgID が生じる場合があります。ProgID と CLSID は両方とも，サーバのメーカによって定義されます。

OPCServer を介してデフォルトの NC にアクセスするための ProgID は, 以下のとおりです。

OPC.SINUMERIK.Machineswitch

■ 手順

COM オブジェクトは，5 段階で作成されます。

表 6.1 OLE オブジェクトを作成する手順

ステップ	説明
1	COM の初期化
2	CLSID の確認
3	オブジェクトの作成
4	OPC 関数の呼び出し
5	使用するインタフェースを利用できるようにする

■ ステップ 1: COM の初期化

COM 関数を使用する前に，COM ライブラリを初期化する必要があります。これは，以下のように呼び出して行います。

```
HRESULT r1;
r1 = CoInitialize (NULL);
```

図 6.2 例

■ ステップ 2: CLSID の確認

オブジェクトの名前が分かれば，“CLSID-FromProgID”（OLE 関数）を使って CLSID を判別することができます。

以下のコード断片は，840DI OPCServer の CLSID を判別する方法を示します。

```
CLSID clsid;

// Get the CLSID from the Name
r1 = CLSIDFromProgID(_
(L”OPC.840DI.Machineswitch”),&clsid);
```

図 6.3 例

■ ステップ 3: オブジェクトの作成

クライアントがオブジェクトを使用する場合，オペレーティングシステムに CLSID が転送され，オブジェクトのインスタンスが要求されます。サーバの場所にかかわりなく，オブジェクト要求は常に COM に送信されます。

CoCreateInstance 関数が，希望するクラスのオブジェクトを作成します。この関数の処理中に，IClassFactory インタフェースから中間的なステップがいくつか生じます。あるクラスの複数のオブジェクトを作成する場合は，IClassFactory を使ってオブジェクトを作成した方が効率的です。

以下の行は，IUknown インタフェースに関連する OPCServer クラスオブジェクトを作成する方法を示します。

```
IUnknown * pOPCUnknown;
r1 = CoCreateInstance (clsid, NULL, CLSCTX_LOCAL_SERVER,
IID_IUnknown, (void**)&pOPCUnknown );
```

図 6.4 例

■ ステップ 4: OPC 関数の呼び出し

このステップでは，作成したオブジェクトの IOPCServer メソッドを使用して，サーバの状況を調べます。最初に，IUknown を使って IOPCServer でポインタが初期化されます。これは，theGetStatus メソッドを呼び出すことにより行われます。

以下に示すのは，サーバの状況とメーカの情報を表示するコードです。GetStatus メソッドが正常に呼び出されると，OPCServer は IMalloc インタフェースを介して戻りデータにメモリを割り振ります。ユーザはメモリを解放する必要があります。

```
IOPCServer *pOPCServer;
OPCSERVERSTATUS *pss;

r1 = pOPCUnknown->QueryInterface(IID_IOPCServer, _
(void**)&pOPCServer);

r1 = pOPCServer ->GetStatus(&pss);

printf("Status.szVendorInfo = %ls¥n", _
pss->szVendorInfo);

// Don't forget to release the memory returned by themethod

pIMalloc->Free(pss->szVendorInfo);
pIMalloc->Free(pss);
```

図 6.5 例

■ ステップ 5: 使用したインタフェースの解放

オブジェクトには参照カウンタが含まれています。これは，オブジェクトをこれ以上使用せず，解放できる時を判別するためのものです。QueryInterface 関数が呼び出されるたびに，参照カウンタが増加します。オブジェクトは，カウンタがゼロに減少するまでは解放されません。

以下に示すのは，IUnknown および IOPCServer インタフェースの参照カウンタをリセットするコマンドです。

```
pOPCServer->Release();
pOPCUnknown->Release();
```

図 6.6 例

6.1.5 OPCData の Automation インタフェース

■ 概要

OPCData の Automation インタフェースは, OPC Foundation によりバージョン 1.0 で指定されました。しかし，この仕様にはいくぶん不明瞭な点があります。最大の弱点については，Visual Basic 5.0 の導入の際に明らかになりました。

■ 新しいバージョン

Automation インタフェースのバージョン 2.0 仕様は，1998 年の半ばから使用可能です。

OPC Foundation では, 定義済みインタフェース用の Automation Wrapper を今後は提供しないことを決定しました。

それで，SIEMENS は “Siemens OPC DA Automation 2.0” Wrapper を開発しました。

■ Automation インタフェースのオブジェクトモデル

OPC Automation インタフェース（2.0 仕様）のオブジェクトモデルは，6.1.1 節で説明されているモデルとは異なります。

OPCGroup および OPCItem オブジェクトを管理するため，別々のコレクションオブジェクトを使用することができます。コレクションオブジェクトには，割り当てられるオブジェクトをリストする関数が用意されています。さらにブラウズ関数が，別のオブジェクトに要約されています。

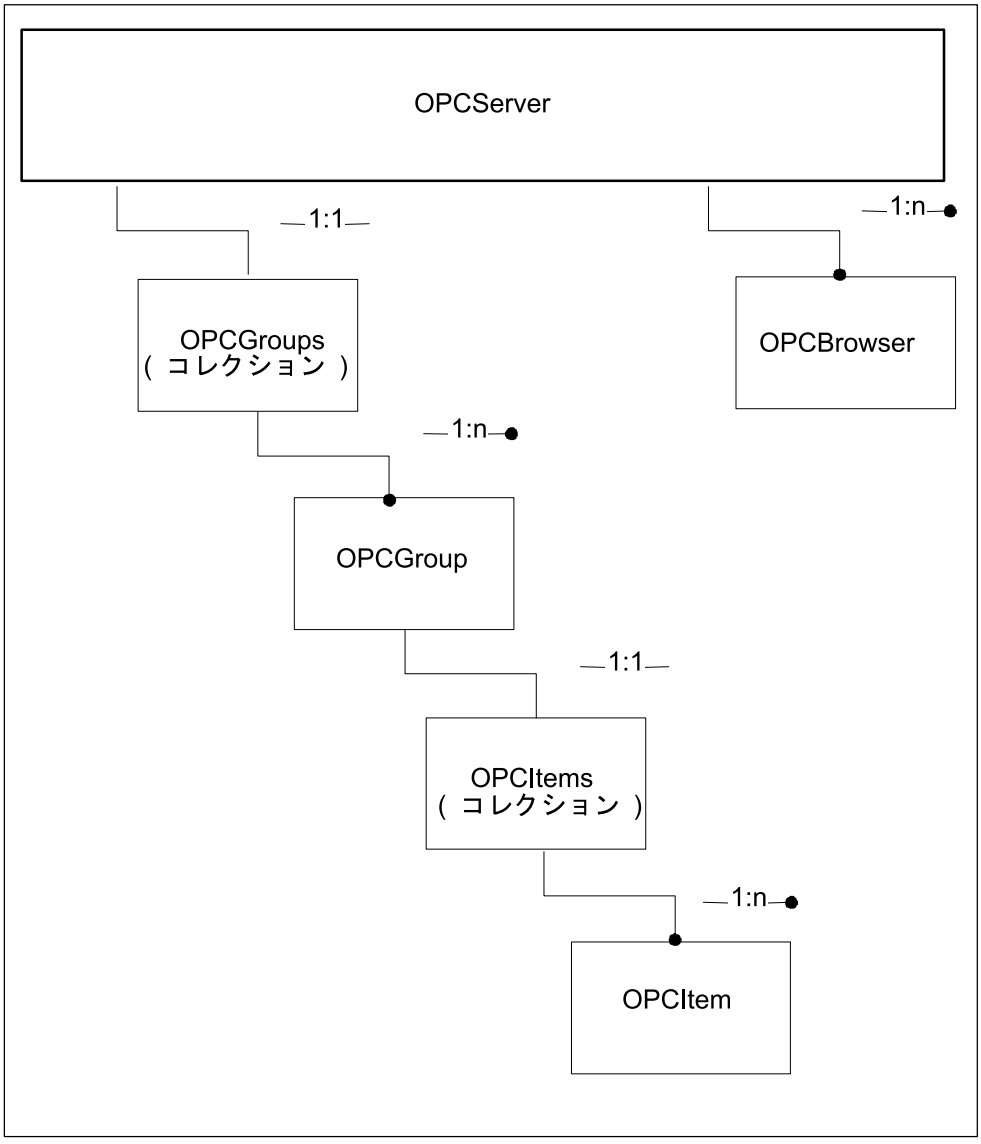


図 6.7 Automation インタフェースのオブジェクトモデル

次に，OPCData Automation インタフェースのバージョン 2.0 について説明します。

■ Visual Basic での OLE オブジェクトの作成と使用

Microsoft Visual Basic は，OLE オブジェクトを簡単に組み込むための自動化インタフェースをサポートする開発環境です。

Visual BasicでAutomationインタフェース2.0を介してOPCServerを使用する方法について，以下に説明します。バージョン 4.0 より前の Visual Basic は使用できません。

■ 手順

原則として，Visual Basic で OLE オブジェクトを作成する手順は以下のとおりです。

表 6.2 Visual Basic で OLE オブジェクトを作成する原則的な手順

ステップ	説明
1	変数の宣言
2	OPCServer への接続
3	OPCGroup の作成
4	OPCItem の追加
5	同期読み取り

■ 準備としてのステップ

新規の Visual Basic プロジェクトを作成し，［プロジェクト］–>［参照設定］メニューで，OPC サーバの Automation インタフェースの参照をアクティブにします。

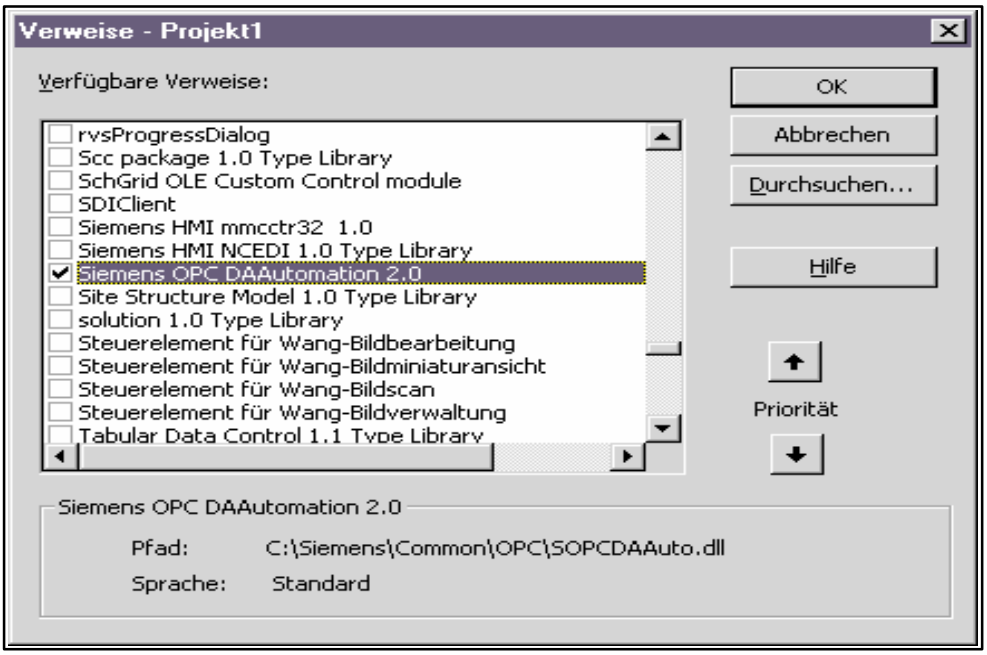


図 6.8 ［プロジェクト］->［参照設定］メニュー

■ ステップ 1: 変数の宣言

Visual Basic および Visual Basic for Applications では, 変数は OLE オブジェクトを参照します。DIM ステートメントがオブジェクトのインスタンスを宣言します。

Visual Basic では，フォームの宣言部分で変数を作成する必要があります。

注 : 一般的な “Object” タイプではなく，該当するタイプ（ここでは OPCServer）を指定すると，Visual Basic は構文検査および型検査を実行できます。

```
Dim ObjServer As OPCServer
```

図 6.9 例

■ ステップ 2: オブジェクトの作成および割り当て

Visual Basic プログラムは，OPC クライアントでもあります。プロセス変数にアクセスするには，最初にクライアントはサーバオブジェクトを作成し，それから OPC サーバに接続する必要があります。

以下に示すコード断片は，Visual Basic クライアントが OPC サーバオブジェクト（たとえば，SINUMERIK サーバ）に接続する方法を示しています。

```
Set ObjServer = New OPCServer
Server Connect( "OPC.SINUMERIK.Machineswitch" )
```

図 6.10 例

注 : アドレス可能なサーバを判別するには，［スタート］->［プログラム］->［SINUMERIK HMI–Programming Package］->［TOOLS］->［OPCScout］メニューにある，“OPC Scout” ヘルプツールを参照してください。

■ ステップ 3: OPCGroup の作成

次のステップでは，プロセス変数を記録するためのグループオブジェクトを作成します。

グループオブジェクト，次いでコレクションオブジェクトを宣言します。その後，OPCGroup（サーバオブジェクトのプロパティ）を，宣言したコレクションオブジェクトに割り当てます。最後に，コレクションオブジェクトメソッド “Add” を呼び出して，グループを作成します。

以下に示すのは，Visual Basic でグループを作成する方法を示した例です。

```
Declaration
Dim GroupObj As OPCGroup
Dim GroupCollection As OPCGroups

'following statement into program e.g. Form_Load
Set GroupCollection = ObjServer.OPCGroups
Set GroupObj = GroupCollection.Add("FirstGroup")
```

図 6.11 例

■ ステップ 4: OPCItem の結合

ここで，作成したグループオブジェクトに OPCtem を挿入します。この項目は，プロセス変数への接続を表しています。この項目の ItemID パラメータで，アドレス指定する変数を定義します。このパラメータの構文は，NC 変数の説明（BTSS.HLP）から取ることができます。

AddItem メソッドを使用すると，1 度の呼び出しで複数の項目をグループに追加することができます。

したがって，転送パラメータとその戻り値は，同じサイズの 1 次元配列です。
NumItem 変数には，挿入項目の数が入っています。

メソッドが呼び出されるたびに，サーバは “ServerHandles” および “Errors” 配列に値を書き込みます。Errors 配列には，追加した項目に関する状況情報が入り，項目が正常に追加されたかどうかに関する情報が示されます。

以下の例では，それ以前に生成した GrpObj OPCGroup に 2 つの項目を作成します。

```
'Declaration
Dim ItemCollection As OPCItems
Dim ItemServerHandle() As Long
Const MAX_INDEX = 2

Dim lNumItems As Long
Dim lClientHandles(MAX_INDEX) As Long
Dim perror() As Long

Dim szItemIDs(MAX_INDEX) As String
Dim AccPath(MAX_INDEX) As String
Dim ReqDataTypes(MAX_INDEX) As Integer

'Definition of ItemIDs
szItemIDs(1) = "/bag/state/opmode"\par
szItemIDs(2) = "/channel/spindle/speedovr"\par

AccPath(1) = ""
AccPath(2) = ""

ReqDataTypes(1) = vbVLong
ReqDataTypes(2) = vbVString

lClientHandles(1) = 1
lClientHandles(2) = 2

'Add Items to Group
Set ItemCollection = GroupObj.OPCItems

ItemCollection.AddItems MAX_INDEX, szItemIDs(),_
lClient-Handles(), ItemServerHandle(), perror(), _
ReqDataTypes(), AccPath()
```

図 6.12 例

■ ステップ 5: 同期読み取り

最後のステップでは，グループのプロセス変数に対する同期読み取り操作を実行します。同期とは，通信システムを通して要求した結果が戻された後にのみ，サーバが Visual Basic プログラムに制御を戻すということです。そのため，プログラムの流れに，通信時間分の遅れが生じます。

OPCRead と AddItems コマンドは両方とも，数量操作です。

OPCRead は，1 度の呼び出しでグループ内の複数のプロセス変数にアクセスできます。NumItems パラメータは，読み取る変数の数を指定します。個々のパラメータそのものについては，サーバが選択したハンドルにより，ServerHandles 配列に記述されます。

```
'Variables definition for OPCRead
'Out Parameter

Dim vValues() As Variant
Dim pErrors() as Long
Dim Zeiten() as Variant
Dim Quals() as Variant

GroupObj.SyncRead (OPCDevice,2,ItemServerHandle(),_
vValues(),pErrors(), Quals(), Zeiten())
```

図 6.13 例

注 : サンプルは HMI プログラミングパッケージに入っています。
OPCData のサンプルは，［スタート］->［プログラム］->［840DI］->[HMI–Programming Package］->［OPC］-［Samples］->［DataAccess］にあります。

6.2 OPC アラームおよびイベント

840DI ユーザは，OPC アラームおよびイベントサーバを使用し，Windows を使用する個々の要件にしたがって，840DI アラームそのものを評価することができるようになりました。

OPCEvent サーバは，“OPC アラームおよびイベント ” を指していますが，
OPCEventServer（ハイフンなし）がクラス名です。

このインタフェースに関する詳細な説明は，HMI– プログラミングパッケージの資料に掲載されています。

現在詳しく説明されているインタフェース仕様としては，
“OPC アラームおよびイベントの Custom インタフェース仕様 ”，および
“OPC アラームおよびイベントの Automation インタフェース仕様 ” が，
OPC Foundation のホームページに掲載されています。
“http://www.opcfoundation.org/opc_spec_document.htm”\par

6.2.1 OPCEvent サーバの用語

（840DI 対 OPC のアラームおよびイベントという）2 つの分野の用語は，常に同じ意味があるわけではないので，OPCEvent サーバの用語について，以下で簡単に説明します。

この節では，“ アラーム ” という語には 840DI の用語と同じ意味があります。

■ 状態（OPCEvent）

状態（Condition）は，固定した名前を持つプロセス変数の状況を区別します。840DI では，NC および PLC アラームはそれぞれの状態に対応しています。

アラーム番号は名前としての役割を果たします。つまり，アラーム番号 4002 は，4002 というエラー状態が発生したことを示します。

840DI には，プロセスイメージを直接表すアラームがあります。こうしたアラームには，一つ一つ応答する必要はありません。その他のアラームは欠陥プロセスのイメージを保管し，明示的に応答した後でなければ消えません。

840DI の応答メカニズムが原因で，アラーム 4002 はこの状態が依然として存在するかどうかには言及しません。

■ イベント（OPCEvent）

イベントは，OPCEvent サーバによって監視されている要素の状態変換を識別し，そのようにしてアラームの発着をそれぞれ識別します。

■ サブスクリプション（OPCEvent）

サブスクリプションとは，クライアントが通知を望むイベントについて，クライアントとサーバの間を調整することです。

■ フィルタ（OPCEvent）

フィルタは，サブスクリプションに含まれるイベントを指定します。

■ 通知（OPCEvent）

サーバは，希望する領域におけるイベントについてクライアントに通知します。

■ 接続ポイント（COM）

接続ポイントとは，同期フィードバックを待たずにクライアントに通知するためのサーバの標準オプションです。

■ 使用されないオプション :

OPCEvent サーバ仕様の以下のオプションは使用されません。

- 領域（Area）
- 副状態（Subcondition）
- カテゴリ（Category）

6.2.2 OPCEvent サーバのクラスモデル

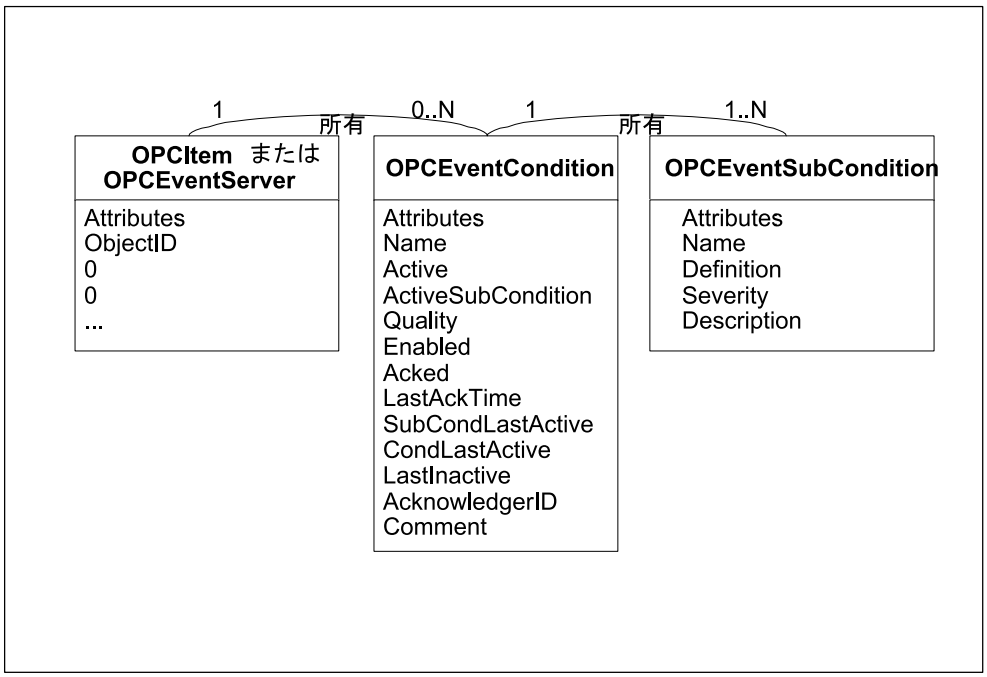


図 6.14 OPCEvent イベントサーバのクラスモデル

■ OPCEventServer クラス

OPCData と同様, OPCEventServer は，クライアントが COM 呼び出しで作成できる唯一のオブジェクトであり，それゆえ，クライアントにとっての入り口点です。このオブジェクトは，OPCEventServer オブジェクトを使用して，さらに多くのオブジェクトを作成および管理できます。

さらに，このオブジェクトは，OPCEventServer オブジェクトに関する情報（たとえば，状態，バージョンなど）を提供します。

■ OPCEventCondition クラス

840DI ユーザの観点では，OPCEventCondition クラスの各オブジェクトは，コントロールでの欠陥状態（アラーム）を表します。このオブジェクトには，アラームがアクティブかどうかに関する情報に加え，名前，アラームの説明，重大度が含まれています。また，最後に変更した時のタイムスタンプや，使用可能であれば応答情報も含まれます。

■ OPCEventSubscription クラス

状態の変化に関する通知を受けるには，ユーザがサーバで OPCEventSubscription を調整する必要があります。

6.2.3 840DI でのイベントソース

840DI では，以下のイベントソース（たとえば，フィルタの実装）がソースパラメータとして受け入れられます。

表 6.3 840DI でのイベントソース

ソースパラメータ	内容
/MMC	MMC アラーム
/MMC/<NCU–DisplayName	接続に固有の MMC アラーム，たとえば，接続の取り消し
/NCU	NCK アラーム
/NCU/Channel#<Nr>/Partprogram	パートプログラムからのメッセージ
/PLC	alarm_S/SQ 以外のすべての PLC アラーム
/PLC/PCM	PLC からの Alarm_S/SQ

複数の NCU を，HMI Advanced の初期化ファイル，“netna-mes.ini” に構成することができます。それから，“/NCU” を指定すると，採用したデフォルト NC とその PLC を参照します。

他の NC および PLC は，NCU DisplayName を入力することにより選択できます。

■ netnames.ini ファイルの例

Section [param <NCU–ID>]
name = otto

ソース “NCU” から，該当するマシンの NC アラームを取得します。

“/MMC/otto” を使用して，接続に固有の MMC アラーム（たとえば，接続の取り消し）を取得します。

注 : さらに，アラームの生成中にユーザがソースとして指定したイベントソースはすべて可能です。

6.2.4 OPCEvent サーバの Automation インタフェース

■ Automation インタフェースのオブジェクトモデル

OPCData の場合と同じように，Automation インタフェースのオブジェクトモデルは下記のように拡張されます。

OPCSubscription 型のオブジェクトを管理するため，別のコレクションオブジェクトが準備されています。これは，従属オブジェクトをリストする機能を提供します。

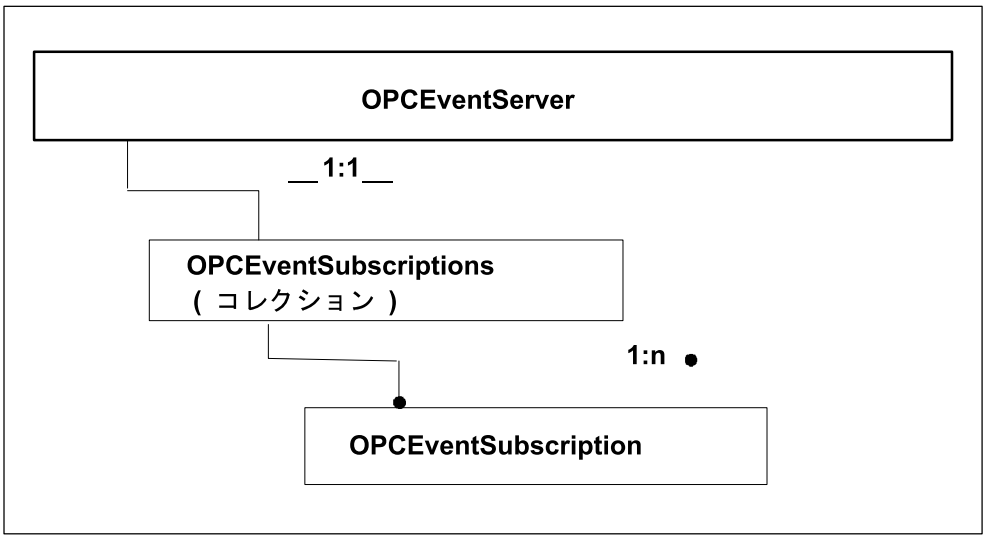


図 6.15 オブジェクトモデル

■ Visual Basic での OPCEventServer オブジェクトの作成と使用

次に，Automation インタフェースを使用する方法を示します。バージョン 4.0 より前の Visual Basic は使用できません。

■ 手順

OLE オブジェクトを Visual Basic で作成する一般的な手順は以下のとおりです。

表 6.4 OLE オブジェクトを作成する一般的な手順

ステップ	説明
1	変数の宣言
2	OPCEvent サーバへの接続
3	OPCEventSubscription の作成
4	非同期イベントの評価
5	サーバのクローズ

■ 準備としてのステップ

新規の Visual Basic プロジェクトを作成し，［プロジェクト］–>［参照設定］で，OPCAlarmEvent サーバの Automation インタフェースの参照をアクティブにします。

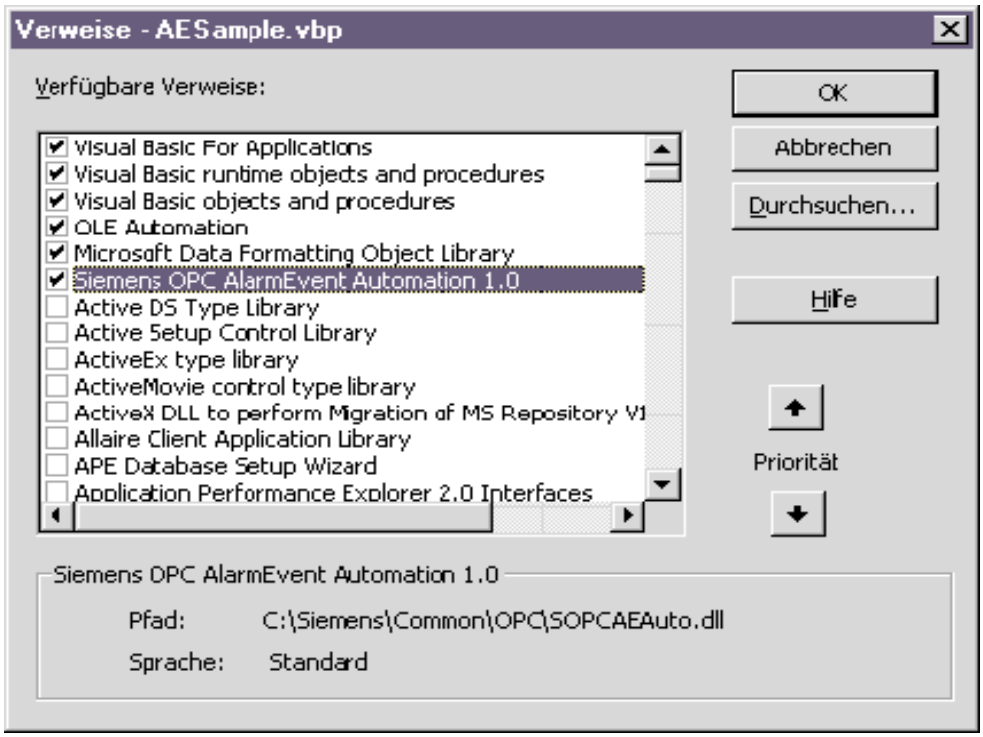


図 6.16 参照

■ ステップ 1: 必要な変数の宣言

使用する変数は，フォームの宣言部分で定義します。Public および WithEvents を指定して，サーバがクライアントを非同期的に起動できるようにする必要があります。

```
Public WithEvents James As OPCEventServer
Public AllSubs As OPCEventSubscriptions
Public WithEvents sub1 As OPCEventSubscription
```

図 6.17 ステップ 1

■ ステップ 2: オブジェクトの作成と割り当て

Visual Basic プログラムは OPC クライアントでもあります。プロセスアラームにアクセスするには，最初にクライアントはサーバオブジェクトを作成し，それから OPC サーバに接続する必要があります。

```
Set James = New OPCEventServer
James.Connect "OPC. ", "" SINUMERIKEvents
```

図 6.18 ステップ 2

■ ステップ 3: Eventsubscription の作成

次のステップで，サーバはイベントが到着するとすぐ応答するよう指示されます。

コレクションオブジェクトに Add メソッドを呼び出すことにより，サブスクリプションを作成します。クライアントは，サブスクリプションがアクティブである場合にのみ通知を受け取ります。サブスクリプションをアクティブにするには，特性 IsActive を設定します。

```
Set AllSubs = James.OPCEventSubscriptions
Set sub1 = AllSubs.Add("Sub1")
sub1.IsActive = True
```

図 6.19 ステップ 3

■ ステップ 4: 非同期イベントの評価

イベントを受け取る場合，Sub1_ConditionEvent プロシージャが自動的に開始されます。これは，サブスクリプションがアクティブになるとすぐ，このプロシージャが処理される時について制御がなくなることを意味します。

このプロシージャは，新しい OPCEvent にパラメータとしてコレクションを転送します。

```
Dim OneEvent As OPCEvent
Dim Output As String
For Each OneEvent In Events
Output = sub1.Name & ", "
Output = Output & OneEvent.Source & ", "
Output = Output & OneEvent.Message & ", "
Output = Output & OneEvent.ConditionName & ", "
Output = Output & OneEvent.SubConditionName
lstEventList1.AddItem Output
Next
```

図 6.20 ステップ 4

■ ステップ 5: サーバのクローズ

VB アプリケーションを閉じると，VB ランタイムは割り当てられたすべてのリソースを解放しようとしますが，これはユーザ自身がプログラムする必要があります。そうすれば，順序を識別して，例外に応答することができます。

```
AllSubs.RemoveAll
James.Disconnect
Set AllSubs = Nothing
Set James = Nothing
```

図 6.21 ステップ 5

注 : サンプルは HMI プログラミングパッケージに入っています。
OPC アラームおよびイベントのサンプルは，［スタート］->［プログラム］->［Sinumerik840D］->［HMI–ProgrammingPackage］->［OPC］-［Samples］->［AlarmEvent］に収録されています。

7 840DI COM サーバインタフェース

■ この章の目的

OPCData や OPCAlarm および Event（6 章を参照）のインタフェースに加えて，840DI は以下のものも提供しています。

- ファイルシステムなどの NC メモリ内の領域にアクセスするためのデータ管理用インタフェース
- コマンドインタフェース

HMI SINUMERIK のバージョン 6.0 以降では, COM をベースにしたプログラミングインタフェースが提供されています。

Microsoft Active Directory Services（ADS）は，階層構造の管理とビジュアル化に関する一般的なコンセプトを提供します。この一般的なコンセプトは，840DI の要件に適うものです。このコンセプトを使用すると，以前は 840DI 固有のソリューションを必要とした場合でも，現在ではこうした規格を使用することができます。

この章では，ファイルやコマンドに関連したプログラミングについて紹介します。この章はマニュアルではありません。これらのインタフェースの構造だけを説明します。

さらに，この章には最低限のサンプルが含まれています。それらは，レシピのような構造になっており，意味のあるアプリケーションを作成できるわけではありませんが，コントロールからデータを受け取るために必要な最小限のコードを示します。

840DI によって定義されるインタフェースはすべて，DUAL（二重）インタフェースです。それで，Custom インタフェースと Automation インタフェースを区別することはありません。

注 **:“Interfaces for File Management“，“Command Interfaces“，および MSDN TMLibrary in Visual Studio_ 6.0 の各マニュアルは，インタフェースの詳細を掘り下げて説明しています。**

注 **:840DI-COM サーバを使用して，変数にアクセスすることができます。この処理については，8.2.1 節 “DCTL32.OCX” で説明されています。**

7.1 データ管理インタフェース

SINUMERIK-COM サーバには，以下のようなファイル編集用のインタフェースがあります。

- ADSI
- IMCFile

続くセクションで，SINUMERIK のデータ管理のコンセプトについて説明します。

サーバは SINUMERIK データ管理を使用するので，以下のセクションでデータ管理のコンセプトについて説明します。

7.1.1 データ管理のコンセプト

SINUMERIK データ管理は, ユーザやプログラマが 2 つの完全に異なるデータ型を 1 つの規格で表示できるようにするために開発されてきました。

- NC メモリの部分
- オペレータ構成要素でのファイルシステム

統一的なビューを作成するため，NC に適用される制限と同じ制限が，NC メモリがイメージされるオペレータ構成要素のファイルシステムの部分にも適用される必要があります。この方法で，データ管理ツリーが作成されます。

この制限をスキームと呼びます。スキームは，ここで許されるデータ型を判別します。

ユーザが使用するオブジェクトには，FileNode と DirNode という 2 つのオブジェクトがあります。COM のインタフェース要件に加え，IAD と IMCFile は両方とも DirNode および IADsContainer を提供します（MSDN TMLibrary in Visual Studio_ 6.0 を参照してください)。

NC がファイルシステムをサポートする必要はないことを思い出してください。SINUMERIK HMI 基本システムのジョブは，ユーザの呼び出しを解釈して，NC から必要なデータを作成します。そのため，NC メモリを直接アドレス指定することはできません。

7.1.2 どのプログラムでどのインタフェースを使用すればよいか

■ ADSI

Active Directory Services（ADS）は，以下の事柄を行うための一連のインタフェースを，ツリー構造で定義します。

- オブジェクトの検出
- オブジェクトのインスタンス化と格納
- 属性の処理

ADS は，オブジェクトのプロパティを照会および格納するためのインタフェースを定義します。これらのプロパティは，フィルタを定義するためにも使用します。

ADS インタフェースを使用すると，オブジェクトをツリー構造でリストすることができます。オブジェクトのリスト表示の結果は，フィルタを使うことにより影響を受けます。

ADS インタフェースを介して，個々のオブジェクトは以下の事柄を行うことができます。

- コピー
- 削除，または
- 移動

これらのオブジェクトにはそれぞれ，オブジェクトのプロパティを簡単に記述するいくつかの属性があります。

ADSI オブジェクトにはそれぞれ，いくつかの標準の属性（名前，パスなど）と，必要であればその他のオブジェクト固有の属性があります。

注 **: それぞれのブラウズ機能では，ADS インタフェースを使用するようお勧めします。**

■ IMCFile

ファイルインタフェースに対する要求は，使用可能な構造を表示することだけにはとどまりません。

840DI は，各ファイルシステム（たとえば，ActivateFile）では操作不能であることも認識しています。さらに，オープンおよび書き込みなどのアクセスは，オペレータ構成要素のファイルとそれに対応する NC 構造ごとに，一様に実行できます。

この目的で使用するインタフェースを IMCFile と呼びます。

注 **:IMCFile は，840DI 固有のオペレーションで使用します。**

7.1.3 ADS インタフェース

ADSI の説明については，たとえば，MSDN TM ライブラリ（すべての Microsoft 開発製品のパーツ）を参照してください。これは，いくつかのインタフェースの説明で構成されています。

ADSI プロバイダは，ADSI メカニズムを介してアプリケーションを使ってアドレス指定するため，こうしたインタフェースのサブセットをサポートしなければなりません。プロバイダを通して作成されるオブジェクトは，ADSI インタフェースのサブセットを提供します。

データ管理を実装するため，最初のステップでは，使用可能な機能の範囲で必要とされる ADSI インタフェースのサブセットだけが提供されます。

ADS の Automation インタフェースには, Custom インタフェースと同じ機能がありますが，より一層簡単です。

Automation インタフェースは，個々の属性にのみアクセスします。Custom インタフェースは，1 度に呼び出したい属性をいくつでも読み書きできます。

以下の ADSI インタフェースが実装されています。

- IADS

ADSI オブジェクトへのパスを通じたアクセス

その属性へのアクセス

- IADsContainer

コンテナの従属オブジェクトのリスト

- IADsClass（使用可能になる予定）

このオブジェクトクラスのスキーム記述

7.1.4 IMCFile

データ管理内のファイルとディレクトリは，IMCFile を介してアドレス指定できます。

最初に，サーバは該当するオブジェクトを作成して，関連する機能を実行できるようにする必要があります。オブジェクトの作成は，エクスプローラでのファイルの選択に対応し，コンテキストメニューを使って使用可能な機能を実行できます。

ファイルインタフェースは，ADSI インタフェースの構文とセマンティクスを使用します。これは，たとえば，現行のコンテナオブジェクトに新しいオブジェクトを作成する Copy，Move ／ Rename，および Create メソッドに該当します。

IMCFile インタフェースの CopyHere および MoveHere メソッドは，ユーザ用の標準メソッドです。

対応する IMCFile メソッドには，デフォルトの処理により IADSContainer に割り当てられる追加のパラメータがあります。

IMCFile インタフェースには，ADS インタフェースで網羅されないデータ管理メソッドがすべて含まれます。これには，以下のメソッドがあります。

- ADS に存在していないメソッド（たとえば，Activate）
- および，ADS インタフェースの標準メソッドによって拡張されたメソッド（たとえば，CopyHere）

オブジェクトの名前，拡張子，およびデータ管理内のデータ型相互の密接な関連により，このインタフェースが要求されます。

データ管理にあるオブジェクトのタイプまたはクラスは，オブジェクトの拡張子やそれが特別な命名スキームに対応するかどうかを定義します。

IMCFile で行うことができるのは，整合性のある変更だけです。

7.1.5 Visual Basic におけるデータ管理インタフェースの使用手順

この手順は，ADS インタフェースと IMCFile の両方に当てはまります。

表 7.1 IMC オブジェクトで作業する一般的な手順

ステップ	説明
1	変数の宣言
2	必要なオブジェクトの要求
3	オブジェクトでの操作

■ 例

以下の例では，“Otto.wpd” が作成されます。

■ 準備のためのステップ

新規の Visual Basic プロジェクトを作成し，IMCFile の参照をアクティブにします。こうするには，［プロジェクト］->［参照設定］メニューを参照します。

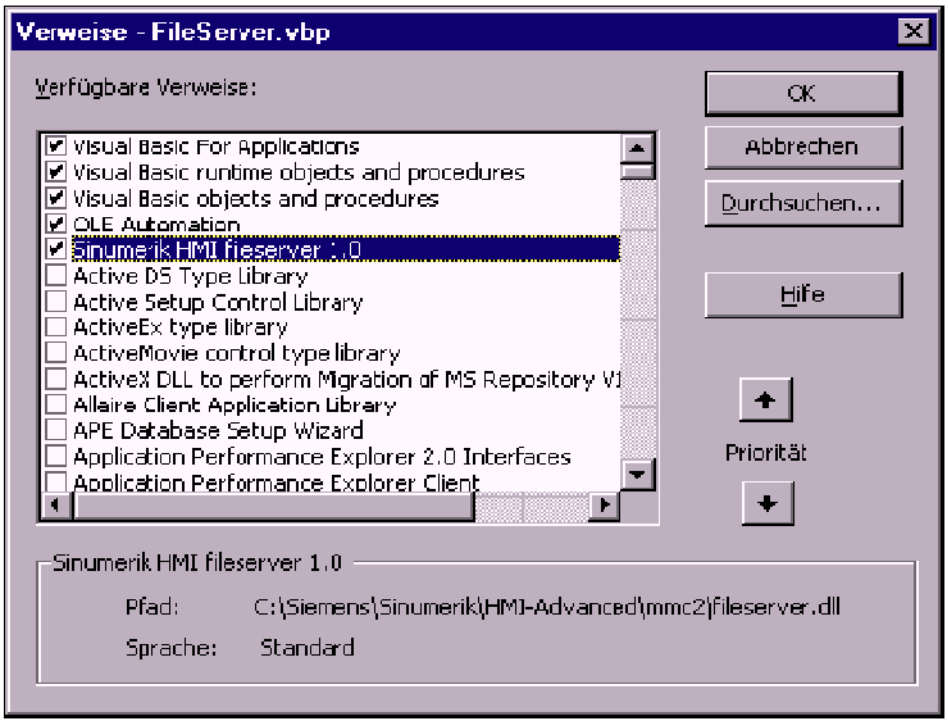


図 7.1 ［プロジェクト］->［参照設定］メニュー

■ ステップ 1: 必要なタイプの変数の宣言（たとえば，IMCFile）

```
Dim WksDir as IMCFile, Teil as IMCFile
```

図 7.2 例

■ ステップ 2: 必要なオブジェクトの要求

```
Set WksDir = GetObjekt ("MC://domain/machinswitch/wks.dir")
```

図 7.3 例

セクションを変更する予定のオブジェクトを要求します。

たとえば，ファイルを削除する場合，そのファイルが入っているディレクトリを要求する必要があります。ファイルをコピーする場合には，宛先のディレクトリを要求する必要があります。

■ ステップ 3: オブジェクトでの操作

```
Set Teil= WksDir.Create("*.wpd", "Otto.wpd", MC_CREATE_NORMAL,"")
```

図 7.4 例

注 : **サンプルは HMI プログラミングパッケージに入っています。**
データ管理のインタフェースの例は，［スタート］->［プログラム］->［SINUMERIK］->［HMI-ProgrammingPackage］->［IMC-Samples］->［IMCFile］にあります。

7.2 IMCDomain

■ IMCDomain

わたしたちは，現在のユーザから，使い慣れた機能性を差し控えたいとは思いません。将来は，データ管理との整合性を気にせずに，NC とのデータのやり取りを行うこともできるようになります。

この目的で使用するインタフェースを，IMCDomain と呼びます。これは，ファイルオブジェクトやディレクトリオブジェクトではなく，サーバオブジェクトだけを区別します。

データ管理の ADSI オブジェクトは，このインタフェースをベースに構成されていないため，ADSI インタフェースではなく，この低水準インタフェースを提供する独立した MC://domain/session オブジェクトが存在します。

IMCDomain インタフェースの非同期メソッドは，コールバックインタフェースの Progress および Complete イベントを介して結果を戻します。低水準インタフェースが提供する各オブジェクトは，セッションオブジェクトです。セッションでは，1 度に 1 つのジョブだけを実行できます。これは，非同期ジョブにも適用されます。

注 **:IMCDomain は十分慎重に使用してください。**

■ 手順

表 7.2 IMCDomain で作業を行う手順

ステップ	説明
1	変数とコールバック項目の宣言
2	ドメインオブジェクトの作成
3	コールバックルーチンの実装
4	ジョブのディスパッチング
5	サーバのクローズ

■ 準備としてのステップ

新規の Visual Basic プロジェクトを作成し，IMCDomain の参照をアクティブにします。こうするには，［プロジェクト］-［参照設定］メニューを参照します。

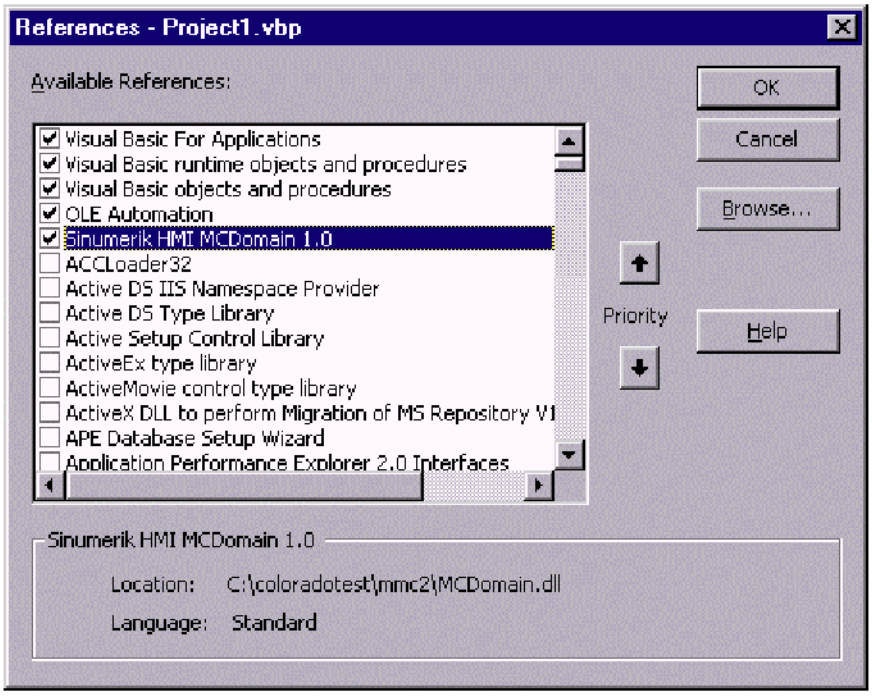


図 7.5 参照

■ ステップ 1: 変数とコールバック項目の宣言

使用する変数は，フォームの宣言部分で定義します。
キーワード Implements を使用して，クライアントがサーバに同期を取らずにアドレス指定されるよう宣言します。

```
Dim session As IMCDomain
Implements IMCDomainAsyncCallback
```

図 7.6 ステップ 1

■ ステップ 2: ドメインオブジェクトの作成

“GetObject” を使用して，対応する名前でオブジェクトを作成するよう COM に指示します。@ 文字は，COM が以下の名前を解釈する方法を指定します。

```
Set session =GetObject(”@SinHMIMCDomain.MCDomain”)
```

図 7.7 ステップ 2

■ ステップ 3: コールバックルーチンの実装

Implements で作成したインタフェースの**すべての**メソッドを実装する必要があります。

```
Private Sub IMCDomainAsyncCallback_OnCommandComplete_
(ByVallCookie As Long,ByVal lResult As Long, ByVal_
lQuality As Long)
Beep
End Sub
```

図 7.8 ステップ 3

■ ステップ 4: ジョブのディスパッチング

同期ジョブは順番に実行することができます。非同期ジョブは，コールバックルーチンを使って調整する必要があります。

```
session.Create ”/NC/WKS.DIR/ Test.wpd”, MCDOMAIN_CREATE_NORMAL
session.Create ”/NC/WKS.DIR/ Test.wpd/Test.ini”, _
MCDOMAIN_CREATE_NORMAL
```

図 7.9 ステップ 4

■ ステップ 5: サーバのクローズ

VB アプリケーションを閉じると，VB ランタイムは割り当てられたすべてのリソースを解放しようとしますが，これはユーザ自身がプログラムする必要があります。これにより，順序を判別して，例外に応答することができます。

```
Set session =Nothing
```

図 7.10 ステップ 5

注 **: サンプルは HMI プログラミングパッケージに入っています。**
IMCDomain のサンプルは，［スタート］–>［プログラム］–>［SINUMERIK］–>［HMI-ProgrammingPackage］–>［IMC–Samples］–>［IMCFile］にあります。

7.3 コマンドインタフェース

840DI-COM サーバは，NC に転送可能なコマンド用のインタフェースを提供します。それらは以下のとおりです。

- IMCCommand
- IMCCommandAsyncCallback

7.4 IMCCommand Automation インタフェース

■ Visual Basic での IMCCommand の作成と使用

続く部分で，Automation インタフェースを介して IMCCommand を使用する方法を示します。Visual Basic のバージョン 4 以降が必要です。

■ 手順

表 7.3

ステップ	説明
1	変数とコールバックインタフェースの宣言
2	望みのサーバへの接続
3	ジョブのディスパッチング
4	非同期イベントの評価
5	サーバのクローズ

■ 準備としてのステップ

新規の Visual Basic プロジェクトを作成し，IMCCommand インタフェースの参照をアクティブにします。こうするには，［プロジェクト］->［参照設定］メニューを参照します。

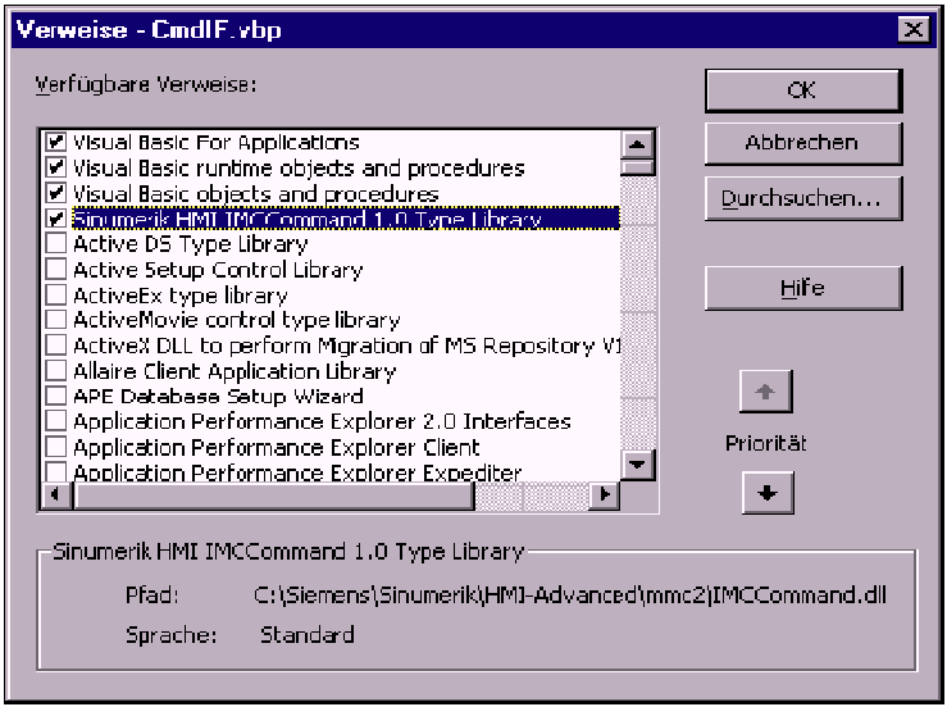


図 7.11 参照

■ ステップ 1: 変数とコールバックインタフェースの宣言

使用する変数は，フォームの宣言部分で定義します。Implements を使ってコールバックを実装すると，サーバがそのプログラムの実行に対する通知の同期を取ることができるようになります。

```
Implements IMCCommandAsyncCallback
Dim CmdObj As IMCCommand
```

図 7.12 ステップ 1

■ ステップ 2: 望みのサーバへの接続

```
Set CmdObj =GetObject("@MCVar.MCCommand:machineswitch")
```

図 7.13 ステップ 2

■ ステップ 3: ジョブのディスパッチング

非同期ジョブはコマンドと情報を受け取り，それによってクライアントとサーバは影響を受けたジョブを後で判別できます。しかし，同期ジョブはコマンドだけを受け取ります。

```
CmdObj.ExecuteAsync CmdText,ClHandle, Me, Me,Shandle
```

図 7.14 ステップ 3

■ ステップ 4: 非同期イベントの評価

サーバは，非同期ジョブの処理が終了すると，クライアントにそのことを通知します。サーバは，最終的な状態を通知し，ジョブのタイプによって計画されていれば，要求された情報を渡します。

```
Private Sub IMCCommandAsyncCallback_OnCommandCompleted(_
ByVal Cookie As Long, ByVal Feedback As Long, _
ByVal OutArguments AsVariant)
If Feedback <> 0 Then
'Error
Else
'Command executed, continue
End If
```

図 7.15 ステップ 4

■ ステップ 5: サーバのクローズ

必要なくなったオブジェクトは解放されます。

```
Set CmdObj =Nothing
```

図 7.16 ステップ 5

注 **: サンプルは HMI プログラミングインタフェースに入っています。IMCDomain のサンプルは，［スタート］-> ［プログラム］-> ［SINUMERIK］-> ［HMI-Programming Package］-> ［IMC-Samples］-> ［IMC-Command］にあります。**

8 ActiveX コントロール

■ この章の目的

この章では，HMI プログラミングパッケージの ActiveX コントロールについて説明します。

8.1 テクノロジオブジェクト（使用可能になる予定）

8.2 コントロール

8.2.1 DCTL32.OCX

DCTL32.OCX コントロールは，ユーザインタフェースの入出力フィールドを作成します。

このコントロールを使用すると，32 ビットアプリケーションから，PI，変数，およびドメインサービスにアクセスすることができます。

この章の内容は，「Yaskawa Siemens 840DI API 取扱説明書 HMI プログラミングパッケージ 基礎編」に含まれています。

これらの章は，次回の版で補足される予定です。

8.2.2 Fileviewer コントロール（使用可能になる予定）

8.2.3 エディタコントロール（使用可能になる予定）

8.2.4 MMC コントロール（使用可能になる予定）

第 4 部：Regie COM サーバインタフェース

Regie COM Server は，関数とイベントをサポートします。続く章では，使用可能な関数とイベントを，パラメータと共に説明します。

オペレータ用ガイドの ”HMI 環境 ” では，「Regie サーバ関数」および「Regie サーバイベント」の章で，これらの取り扱いについて説明しています。

9 Regie API 関数

重要：

他のいずれかのメソッドを呼び出す前に，必ず **InitSvr** を呼び出してください。

9.1 AsyncCompleted

説明

この関数は, IRegieSvrEvents からの ” 非同期 ” コールバックを確認する場合に必ず使用してください。ここで言う ” 非同期 ” とは，コールバックの結果が，クライアント側のコールバック関数から戻される時ではなく，後で提供されることを意味します。

この関数には状況が提供される必要があります。以下に可能な状況を示します。
1 –>TRANSACTION WAS OK 2 –> NOT_ACCEPTED_DO_MESSAGE（FAILED: REGIE は関連 DialogAlarm をポップアップ表示する）3 –> NOT_ACCEPTED_NO_MESSAGE (FAILED) 4 –> 最後のコールバック FAILED，REGIE は制限時間中にトランザクションを再試行する必要がある 5 –> 最後のコールバック FAILED，REGIE は永続的にトランザクションを再試行する必要がある

構文

Sub AsyncCompleted (IStatus As Long)

パラメータ

IStatusl

トランザクションの状況

C++ の構文

STDMETHODIMP AsyncCompleted (long IStatus);

戻り値

HESULT

戻りコード

常に S_OK

9.2 ContrastDown

説明

この関数は，STN またはモノクロ LCD ディスプレイでのコントラストを下げます。ご使用のハードウェアのディスプレイが異なる場合，この関数には効果がありません。

構文

Sub ContrastDown (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP ContrastDown (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.3 ContrastUp

説明

この関数は，STN またはモノクロ LCD ディスプレイでのコントラストを上げます。ご使用のハードウェアのディスプレイが異なる場合，この関数には効果がありません。

構文

Sub ContrastUp (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP ContrastUp (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.4 GetCurrentTaskIndex

説明

この関数は，現行タスクの索引を戻します。CurrentTaskIndex は，REGIE.INI に従って REGIE を介して活動化された最新のタスクです。

構文

Sub GetCurrentTaskIndex (pnRetVal As Long)

パラメータ

pnRetVal

戻り値
CurrentTaskIndex

C++ の構文

STDMETHODIMP GetCurrentTaskIndex (int* pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.5 GetMMCDir

説明

この関数は，MMC のパスを戻します。このパスは，mmc2 ディレクトリを指し示します。

構文

Sub GetMMCDir (bstrModuleExePath As String,
pnRetVal as Long)

パラメータ

bstrModuleExePath

ここにパスが戻される

pnRetVal

戻り値

0

失敗

その他

成功

C++ の構文

STDMETHODIMP GetMMCDir (BSTR * bstrModuleExePath,
int *pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.6 GetMMCLanguagePath

説明

この関数は，必要な言語 DLL の言語パスを戻します。たとえば，現在の言語がドイツ語で，bstrPrefix 内で OE を渡す場合，関数は ...¥mmc2¥language¥oe_gr.dll などを戻します。

構文

Sub GetMMCLanguagePath(bstrPrefix As String,
bstrModuleExePath as String,
pnRetVal as Long)

パラメータ

bstrPrefix

App–prefix（2 文字）

bstrModuleExePath

ここに DLL 名が戻される

pnRetVal

戻り値

0

失敗

その他

成功

C++ の構文

STDMETHODIMP GetMMCLanguagePath(BSTR *bstrPrefix,
BSTR *bstrModuleExePath,
int *pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.7 InitComplete

説明

この関数は，InitComplete を REGIE に通知します。

この関数は今後は呼び出さないで，代わりに InitCompleteEx() を使用してください。

構文

Sub InitComplete (dwThreadId As Long,
hOwnerWnd as Long,
hControlWnd as Long)

パラメータ

dwThreadId

呼び出し側のスレッド ID

hOwnerWnd

呼び出し側の TopLevelWindow–Handle (HWND)

hControlWnd

呼び出し側の Callback–Wnd (HWND)

C++ の構文

STDMETHODIMP InitComplete (long dwThreadId,
long hOwnerWnd,
long hControlWnd);

戻り値

HESULT

戻りコード

常に S_OK

9.8 InitCompleteEx

説明

InitCompleteEx() を呼び出して，初期設定フェーズが終了し，REGIE が処理を続行できることを REGIE に通知する必要があります。処理開始後，REGIE は指定された TIMEOUT（–>REGIE.INI）を待機します。この時間枠内に InitComplete を呼び出さないと，REGIE は ”xxx didn’t post InitComplete” テキストボックスを表示します。その後，REGIE が処理を開始したのではないかのように扱います。

ご使用のアプリケーションが REGIE.INI の [StartupConfiguration] セクションで参照されている場合，0 の TIMEOUT も提供できます。この場合，REGIE はアプリケーションが InitComplete を通知するのを予期しません。

ご使用のアプリケーションが REGIE.INI の [TaskConfiguration] セクションで参照されている場合，REGIE はアプリケーションが InitComplete を通知するのを常に予期します。

詳細について，InitSvr() も参照してください。

構文

Sub InitCompleteEx ()

C++ の構文

STDMETHODIMP InitCompleteEx (void);

戻り値

HESULT

戻りコード

常に S_OK

9.9 InitSvr

説明

重要 : これはメインの初期関数です。

他のいずれかのメソッドを呼び出す前に，必ずこの関数を呼び出してください。

次に，InitCompleteEx() を呼び出して，初期設定フェーズが終了し，REGIE が処理を続行できることを通知してください。

詳細について，InitCompleteEx() も参照してください。

構文

Sub InitSvr (vt1 as Variant,
vt2 as Variant)

パラメータ

vt1

TopLevelWindow への HWND

vt2

Visual Basic では空，Visual C++ では VT_EMPTY でなければならない

C++ の構文

STDMETHODIMP InitSvr (const VARIANT &vt1,
const VARIANT &vt2);

戻り値

HESULT

戻りコード

常に S_OK

9.10 IsChanMenuLocked

説明

この関数は，chanmenu ロックカウントをチェックします。

ChanMenu がロックされていない場合，-1 を戻します。そうでない場合は，chanmenu にロックを設定した最初のタスクの Taskindex を戻します。

詳細については，LockChanMenu() および UnlockChanMenu() も参照してください。

構文

Sub IsChanMenuLocked (pfRetVal As Long)

パラメータ

pfRetVal

ロック ID

-1

ChanMenu がロックされていない

その他

最初のロックタスクの TaskIdx

C++ の構文

STDMETHODIMP IsChanMenuLocked (int* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.11 IsCurrentNCULocked

説明

この関数は，NCU ロックカウントをチェックします。

CurrentNCU がロックされていない場合は –1 を，その他の場合は，ロックを Current-NCU 上に設定した最初のタスクの TaskIndex を戻します。

詳細について，LockCurrentNCU() および UnlockCurrentNCU() も参照してください。

構文

Sub IsCurrentNCULocked (pfRetVal as Long)

パラメータ

pfRetVal

ロック ID

–1

ロックされていない CurrentNCU

その他

最初のロックタスクの TaskIdx

C++ の構文

STDMETHODIMP IsCurrentNCULocked (long *pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.12 LockChanMenu

説明

この関数は，chanmenu をロックします。この呼び出しの後は，異なるチャネルまたは異なる NC に切り替えることはできません。

LockChanMenu() を呼び出す場合，後で UnlockChanMenu() も呼び出さなければなりません。そうしないと，チャネル切り替えが非活動状態のままです。

LockChanMenu() は，タスクが REGIE によって開始されていない場合 FALSE を戻します。

詳細については，UnlockChanMenu() および IsChanMenuLocked() も参照してください。

構文

Sub LockChanMenu (pfRetVal As Long)

パラメータ

pfRetVal
戻り値
TRUE
正常
FALSE
失敗

C++ の構文

STDMETHODIMP LockChanMenu (BOOL *pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.13 LockCurrentNCU

説明

この関数は，NCU スイッチをロックします。この呼び出しの後は，別の NC に切り替えることはできません。もちろん，現行の NCU では別のチャネルへの切り替えが可能です。

LockCurrentNCU() を呼び出す場合，後で UnlockCurrentNCU() も必ず呼び出してください。これを怠ると，NCU スイッチが非活動状態のままになります。

LockCurrentNCU() は，タスクが REGIE によってまだ開始されていない場合，または M2N トランザクション中に REGIE によってロックが一時的に使用不可になっている場合，FALSE を戻します。

詳細について，UnlockCurrentNCU() および IsCurrentNCULocked() も参照してください。

構文

Sub LockCurrentNCU (pnRetVal As Long)

パラメータ

pnRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP LockCurrentNCU (long* pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.14 MMCScreenOff

説明

この関数は，バックライトをオフにし（該当する HW 上で実行している場合），状態を PLC に通知します。

構文

Sub MMCScreenOff (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP MMCScreenOff (long* pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.15 MMCScreenOn

説明

この関数は，バックライトをオンにし（該当する HW 上で実行している場合），状態を PLC に通知します。

構文

Sub MMCScreenOn (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP MMCScreenOn (long* pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.16 ReadCmdLine

説明

指定されたタスクに応じて，REGIE からコマンド行を読み取ります。各タスクにはコマンド行が与えられ，それは REGIE による開始時にプロセスに渡されます。

関数は，読み取られる ANSI 文字の数を戻します。NULL でない BSTR を渡すと，この関数によって再配置されます。コマンド行に入る文字数の制限は，256 個の ANSI 文字です。

詳細について，WriteCmdLine()，WriteCmdLineEx()，ReadCmdLineMe() も参照してください。

構文

Sub ReadCmdLine (nTaskIdx As Long,
bstrCmdLine As String,
pnRetVal AS Long)

パラメータ

nTaskIdx

TaskIndex

bstrCmdLine

コマンド行を戻す BSTR

pnRetVal

ASCII 文字数で戻される長さ

C++ の構文

STDMETHODIMP ReadCmdLine (long nTaskIdx,
BSTR *bstrCmdLine,
long *pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.17 ReadCmdLineMe

説明

呼び出しタスクに応じて，REGIE からコマンド行を読み取ります。各タスクにはコマンド行が与えられ，それは REGIE による開始時にプロセスに渡されます。

関数は，読み取られる ANSI 文字の数を戻します。NULL でない BSTR を渡すと，この関数によって再配置されます。コマンド行に入る文字数の制限は，256 個の ANSI 文字です。

詳細について，WriteCmdLine()，WriteCmdLineEx()，ReadCmdLine() も参照してください。

構文

Sub ReadCmdLineMe (bstrCmdLine As String,
pnRetVal As Long)

パラメータ

bstrCmdLine

コマンド行を戻す BSTR

pnRetVal

ASCII 文字数で戻される長さ

C++ の構文

STDMETHODIMP ReadCmdLineMe (BSTR *bstrCmdLine,
long *pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.18 ResumeRegieEvents

説明

この関数は，REGIE 内のイベント処理を再開します。StopRegieEvents または TestAndStopRegieEvents を呼び出した場合，REGIE を実行し続けるためにこの関数を必ず呼び出す必要があります。

重要 : TestAndStopRegieEvents() を正常に呼び出す前には，この関数を呼び出さないでください（詳細については，TestAndStopRegieEvents を参照）。

構文

Sub ResumeRegieEvents (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

REGIE が再開された

FALSE

失敗

C++ の構文

STDMETHODIMP ResumeRegieEvents (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.19 SetModeChannelSwitchKey

説明

この関数は，ChannelSwitch キーにモードを設定します。使用可能なモードは次のとおりです。nMode = 0 は ChannelSwitchKey をロックします。チャネル切り替えはこのモードの場合不可能になります。nMode = 1 は ChannelSwitchKey のロックを解除します。チャネル切り替えがこれで可能になります。

SetModeChannelSwitchKey(0) を呼び出す場合，後で SetModeChannelSwitchKey(1) も必ず呼び出してください。これを怠ると，ChanSwitchKey が非活動状態のままになります。

構文

Sub SetModeChannelSwitchKey(nMode as Long)

パラメータ

nMode

モード

0

ChannelSwitchKey のロック

1

ChannelSwitchKey のロック解除

C++ の構文

STDMETHODIMP SetModeChannelSwitchKey (int nMode);

戻り値

HESULT

戻りコード

常に S_OK

9.20 Subsystem

説明

このメソッドは，サブシステムを開始または停止するのに使用できます。サブシステムはサーバアプリケーションの集合です。サブシステムは，REGIE.INI [StartupConfiguration] - Style INI ファイルを使って記述する必要があります。この INI ファイル中のすべてのサーバが開始または停止されます。サブシステムは名前によって識別されます。そのため，始動しているサブシステムを停止したい場合，サブシステムを開始するのに使用した INI ファイル名と同じファイル名を使用します。サブシステムは，同期または非同期呼び出しを使って開始できます。

サブシステムは，all-or-nothing 式に開始されることが保証されています。サブシステムのサーバがすでに実行中である場合，REGIE がそれを再度開始しようとすることはありませんが，このサーバの内部使用量カウントは増加します。

この関数のほとんどは，nMode パラメータを使って制御されます。次のパラメータが現在使用可能です。

REGSVR_SUBS_START: このビットをセットすると，サブシステムが開始します。このビットが 0 である場合，サブシステムは停止します。

REGSVR_SUBS_ASYNC: このビットによって，順序が REGIE にエンキューされた直後に関数が戻されます。サブシステムが正常に開始している場合には通知は受け取りません。

重要 : 特別な目的のために使用される予約済みの Ini ファイル名が 2 つあります。

4. bstrIniFile を "hmi_adv" に設定すると，関数は MMC を開始または停止します。
5. bstrIniFile を "all" に設定すると，すべてのサブシステムをシャットダウンできます。この特殊な Ini ファイル名は，nModeREGSVR_SUBST_START が設定されていない場合にしか使用できません。

どんな場合でも，最後のサブシステムがシャットダウンされると，REGIE は終了します。

構文

```
Sub Subsystem ( bstrIniFile As String,
                nMode As Long,
                nParam As Long,
                pfRetVal as Long)
```

パラメータ

bstrIniFile
INI ファイルの名前
nMode
モード
nParam
必ず 0
pfRetVal
戻り値
TRUE
REGIE が停止した
その他
失敗

C++ の構文

```
STDMETHODIMP Subsystem ( BSTR bstrIniFile,
                         int nMode,
                         int nParam,
                         BOOL *pfRetVal);
```

戻り値

HESULT

戻りコード

常に S_OK

9.21 SwitchToChild

説明

指定された子タスクに切り替えるため，REGIE をチャージします。子タスクは，活動化の点で呼び出しタスクを置き換えます。つまり，この呼び出しが正常に戻ると，呼び出しタスクを活動化する各試み（REGIE–API を介する）の結果，子タスクが活動化されます。この動作は，子アプリケーションが SwitchToParentxxx() を呼び出すまでそのままです。いずれにしても，別の子タスクによってそれ自体を置き換えるため（および親になるため）に SwitchToChild を呼び出すこともできます。

構文

Sub SwitchToChild (nTaskIdx As Long,
pfRetVal As Long)

パラメータ

nTaskIdx

RegieTaskIndex（REGIE.INI で使用されているもの）

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToChild (long nTaskIdx,
long *pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.22 SwitchToChildEx

説明

この関数を使用して，タスク番号ではなく論理名を使用して子タスクに切り替えることができます。論理名を解決するために，REGIE は呼び出しタスクの INI ファイルの中を探します。次の項目を判別します。

[CHILDS] MYEDIT = 24

アプリケーションが SwitchToChildEx(”MYEDIT”) を呼び出すと，REGIE が SwitchToChild(24) を実行します。

構文

Sub SwitchToChildEx (bstrChildName As String,
pfRetVal As Long)

パラメータ

bstrChildName

論理子名

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToChildEx (LPCTSTR bstrChildName,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.23 SwitchToChildImmediate

説明

この関数は，SwitchToChild と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToChildImmediate (nTaskIdx As Long,
pfRetVal As Long)

パラメータ

nTaskIdx

RegieTaskIndex（REGIE.INI で使用されているもの）

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToChildImmediate (long nTaskIdx,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.24 SwitchToChildImmediateEx

説明

この関数は，SwitchToChildEx と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToChildImmediateEx (bstrChildName As String,
pfRetVal As Long)

パラメータ

bstrChildName

論理子名

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToChildImmediateEx (LPCTSTR bstrChildName,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.25 SwitchToHelpTask

説明

ヘルプタスク（REGIE.INI / [Miscellaneous]/ HelpTaskIndex で構成）に切り替えるために REGIE をチャージします。ヘルプタスクは，現行のタスクでは子として開始しません。

構文

Sub SwitchTohelpTask (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

REGIE が停止した

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToHelpTask (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.26 SwitchToHelpTaskImmediate

説明

この関数は，SwitchToHelpTask と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchTohelpTaskImmediate (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

REGIE が停止した

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToHelptaskImmediate (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.27 SwitchToParent

説明

SwitchToChild() を介してこの子を開始した，親タスクに切り替えるために REGIE をチャージします。呼び出しタスクが SwitchToChild() を介して開始されていない場合，この呼び出しは失敗します。子タスクは，REGIE によって終了されません。

構文

Sub SwitchToParent (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToParent (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.28 SwitchToParentAndKillMe

説明

SwitchToChild() を介してこの子を開始した，親タスクに切り替えるために REGIE をチャージします。呼び出しタスクが SwitchToChild() を介して開始されていない場合，この呼び出しは失敗します。子タスクは，REGIE によって終了されます。

構文

Sub SwitchToParentAndKillMe (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToParentAndKillMe (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.29 SwitchToParentAndKillMeImmediate

説明

この関数は，SwitchToParentAndKillMe と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToParentAndKillMeImmediate (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToParentAndKillMeImmediate (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.30 SwitchToParentImmediate

説明

この関数は，SwitchToParent と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToParentImmediate ()

C++ の構文

STDMETHODIMP SwitchToParentImmediate (void);

戻り値

HESULT

戻りコード

常に S_OK

9.31 SwitchToPreviousTask

説明

前に選択されたタスクに切り替えるため，REGIE をチャージします。前のタスクが存在しない場合（スタートアップ直後），現在のタスクが活動状態のままになります。

構文

Sub SwitchToPreviaousTask (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

REGIE が停止した

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToPreviaousTask (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.32 SwitchToPreviousTaskImmediate

説明

この関数は，SwitchToPreviousTask と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToPreviaousTaskImmediate (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

REGIE が停止した

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToPreviaousTaskImmediate (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.33 SwitchToTask

説明

SwitchToChild() を介してこの子を開始した，親タスクに切り替えるために REGIE をチャージします。呼び出しタスクが SwitchToChild() を介して開始されていない場合，この呼び出しは失敗します。子タスクは，REGIE によって終了されます。

構文

Sub SwitchToTask (nTaskIdx As Long,
pfRetVal As Long)

パラメータ

nTaskIdx

RegieTaskIndex（REGIE.INI で使用されているもの）

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToTask (long ntaskIdx,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.34 SwitchToTaskImmediate

説明

この関数は，SwitchToTask と同じ働きをします。ただし，例外として，REGIE への要求は REGIE イベントキューの中の最上位で待機状態になります。つまり，イベントは，その時点で REGIE で待機している他のすべてのイベントの前に処理されます。

構文

Sub SwitchToTaskImmediate (nTaskIdx As Long,
pfRetVal As Long)

パラメータ

nTaskIdx

RegieTaskIndex（REGIE.INI で使用されているもの）

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP SwitchToTaskImmediate (long ntaskIdx,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.35 TestAndStopRegieEvents

説明

この関数は，REGIE 内のイベント処理を停止します。つまり，TestAndStopRegieEvents への呼び出し後，REGIE はどのイベントも処理せず，それらを内部的にキューに入れます。これは，アプリケーションがどの REGIE アクティビティ（DEACTIVATE，FORM_UNLOAD，QUERY_FOR_SHUTDOWN など）によっても混乱されないようにしたい状況で使用してください。イベントインタフェース IRegieSvrEvents は，ResumeRegieEvents への呼び出しまで開始しません（詳細について，ResumeRegieEvents も参照）。

構文

Sub TestAndStopRegieEvents (pfRetVal As Long)

パラメータ

pnRetVal

戻り値

–1

失敗

0

REGIE はすでに別のアプリケーションによりロック済み

1

成功。REGIE がロックされたので，後で必ず ResumeRegieEvents を呼び出す必要がある。

C++ の構文

STDMETHODIMP TestAndStopRegieEvents (int* pnRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.36 UnlockChanMenu

説明

この関数は，ChanMenu のロックを解除します。この呼び出しの後，場合によっては異なるチャネルまたは異なる NC に切り替えることができます。

LockChanMenu() および UnlockChanMenu() はロックカウントを増加または減少させています。そのため，ロックを解放しても，常にチャネルを切り替える可能性が再度活動化されるとは限りません。

チャネル切り替えがその時点で可能かどうかを知る必要がある場合，IsChannelMenuLocked() を呼び出してこの情報を判別してください。

LockChanMenu() を呼び出していない場合は，UnlockChanMenu() を呼び出してはなりません。

UnlockChanMenu() の呼び出し回数は，LockChanMenu() の呼び出し回数と同じでなければなりません。

UnlockChanMenu() は，REGIE によってタスクが開始されていない場合，FALSE を戻します。

詳細については，LockChanMenu() および UnlockChanMenu() も参照してください。

構文

Sub UnlockChanMenu(pfRetVal as Long)

パラメータ

pfRetVal
戻り値
TRUE
正常
FALSE
失敗

C++ の構文

STDMETHODIMP UnlockChanMenu (BOOL *pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.37 UnlockCurrentNCU

説明

この関数は，NCU スイッチのロックを解除します。この呼び出しの後，別の NC への切り替えが可能になります。

LockCurrentNCU() および UnlockCurrentNCU() は，ロックカウントを増加または減少させています。したがって，ロックを解除しても，必ず NCU を切り替える可能性を再度活動化できるとは限りません。

現時点で NCU 切り替えが可能かどうかを知る必要がある場合，IsCurrentNCULocked() を呼び出してこの情報を判別してください。

LockCurrentNCU() を呼び出していない場合，UnlockCurrentNCU() を呼び出すことはできません。

UnlockCurrentNCU() は，LockCurrentNCU() を呼び出した回数と同じだけ呼び出すことが必要です。

UnlockCurrentNCU() は，タスクが REGIE によってまだ開始されていない場合，または M2N トランザクション中に REGIE によってロックが一時的に使用不可になっている場合，FALSE を戻します。

詳細について，LockCurrentNCU() および IsCurrentNCULocked() も参照してください。

構文

Sub UnlockCurrentNCU (pfRetVal As Long)

パラメータ

pfRetVal

戻り値

TRUE

正常

FALSE

失敗

C++ の構文

STDMETHODIMP UnlockCurrentNCU (long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.38 WriteCmdLine

説明

指定されたタスクに応じて，コマンド行から REGIE に書き込みます。各タスクにはコマンド行が与えられ，それは REGIE による開始時にプロセスに渡されます。

関数は，書き込まれる ANSI 文字の数を戻します。エラーが発生した場合，またはコマンド行が空の場合，長さは 0 です。コマンド行に入る最大文字数は，256 個の ANSI 文字です。

詳細について，WriteCmdLineEx()，ReadCmdLineMe()，ReadCmdLine() も参照してください。

構文

Sub WriteCmdLine (nTaskIdx As Long,
bstrCmdLine As String,
pfRetVal As Long)

パラメータ

nTaskIdx

TaskIndex

bstrCmdLine

コマンド行を含む BSTR

pnRetVal

ASCII 文字数で戻される長さ

C++ の構文

STDMETHODIMP WriteCmdLine (long nTaskIdx,
LPCTSTR bstrCmdLine,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

9.39 WriteCmdLineEx

説明

指定されたタスクに応じて，コマンド行から REGIE に書き込みます。各タスクにはコマンド行が与えられ，それは REGIE による開始時にプロセスに渡されます。

関数は，書き込まれる ANSI 文字の数を戻します。エラーが発生した場合，またはコマンド行が空の場合，長さは 0 です。コマンド行に入る最大文字数は，256 個の ANSI 文字です。

詳細について，WriteCmdLine()，ReadCmdLineMe()，ReadCmdLine() も参照してください。

構文

Sub WriteCmdLineEx (bstrTaskName As String,
bstrCmdLine As String,
pfRetVal As Long)

パラメータ

bstrTaskName

タスク名

bstrCmdLine

コマンド行を含む BSTR

pnRetVal

ASCII 文字数で戻される長さ

C++ の構文

STDMETHODIMP WriteCmdLineEx (LPCTSTR bstrTaskName,
LPCTSTR bstrCmdLine,
long* pfRetVal);

戻り値

HESULT

戻りコード

常に S_OK

10 Regie イベントインタフェース

■ 一般的な注意事項

Regie がアプリケーションに何かを伝える必要がある場合, **Notify()** メソッドを ' 呼び出し ' ます。

アプリケーションは，このイベントに応答する必要があります。これは，次の異なる 2 つの方法で実行されます。

- lRetVal 戻りパラメータで同期的に応答する
- **AsyncCompleted()** メソッドで非同期的に応答する

  アプリケーションを非同期的に応答する意図を示すには，lRetVal 戻りパラメータに –1 を割り当てる必要があります。

この節では，すべての可能な応答状態（同期的および非同期的な応答の両方）をリストします。

10.1 応答状態

REGEVT__NOTIFY_CONF__ASYNC

後で AsyncCompleted() を介してこのイベントに応答することを通知します。

REGEVT__NOTIFY_CONF__OK

このイベントが正常に処理されたことを通知します。

REGEVT__NOTIFY_CONF__NOT_ACCEPTED_DO_MESSAGE

このイベントがまだ処理されていないことを通知します。Regie はダイアログアラームを表示します。

REGEVT__NOTIFY_CONF__NOT_ACCEPTED_NO_MESSAGE

このイベントがまだ処理されていないことを通知します。Regie はダイアログアラームを表示しません。

REGEVT__NOTIFY_CONF__FAILED_RETRY_LIMITED

アプリケーションが現在このイベントを処理できないことを通知します。その後 Regie はこのイベントを繰り返します（最大 5 回）。

REGEVT__NOTIFY_CONF__FAILED_RETRY_FOREVER

アプリケーションが現在このイベントを処理できないことを通知します。その後，完全に応答されるまで，Regie はこのイベントを繰り返します。

次の点に注意してください。

REGEVT__NOTIFY_CONF__OK

REGEVT__NOTIFY_CONF__NOT_ACCEPTED_DO_MESSAGE

REGEVT__NOTIFY_CONF__NOT_ACCEPTED_NO_MESSAGE

上記のイベントは，Regie の観点からすると肯定応答です。

10.2 イベント記述

Event FormLoad

lEvent = EVT__FORMLOAD

vtParam = ＜空＞

説明:

アプリケーションが，ウィンドウを表示するように要求されます。アプリケーションは，まだここではアクティブデータを表示しません。

Event FormUnload

lEvent = EVT__FORMUNLOAD

vtParam = ＜空＞

説明:

アプリケーションが，ウィンドウを非表示にするよう要求されます。

Event Activate

lEvent = EVT__ACTIVATE

vtParam = ＜空＞

説明:

アプリケーションが，ウィンドウの内容を活動状態で表示するよう要求されます。

Event Deactivat

lEvent = EVT__DEACTIVATE

vtParam = ＜空＞

説明:

アプリケーションが，ウィンドウの内容を非活動にするよう要求されます。

Event QueryForShutdown

lEvent = EVT__QUERYFORSHUTDOWN

vtParam = VT_I4（タイプ）

lVal = 1 ：唯一のタスクが終了です。

lVal = 2 ：Regie が終了します。

説明:

アプリケーションが，終了の準備をするよう要求されます。このイベントがいったん肯定応答されると，Regie はアプリケーションを終了する許可を得ます (WM_CLOSE を介して)。

Event GetSpecialKey

lEvent = EVT__GETSPECIALKEY

vtParam = VT_I4（タイプ）

lVal = 提供される RegieEvent

説明 :

アプリケーションが Regie イベントに転送されます。

注意 : キーイベントの場合, 現在存在するのはキーが ' 押されている ' イベントだけで, ' 押されていない ' イベントはありません。

10.3 現在定義されているイベント（RegieUsr.h）

以下のイベントが，現在定義されています（RegieUsr.h）。

REGEVT__HSK1

水平ソフトキー 1 が押されました。

REGEVT__HSK2

水平ソフトキー 2 が押されました。

REGEVT__HSK3

水平ソフトキー 3 が押されました。

REGEVT__HSK4

水平ソフトキー 4 が押されました。

REGEVT__HSK5

水平ソフトキー 5 が押されました。

REGEVT__HSK6

水平ソフトキー 6 が押されました。

REGEVT__HSK7

水平ソフトキー 7 が押されました。

REGEVT__HSK8

水平ソフトキー 8 が押されました。

REGEVT__RECALL

再呼び出しキーが押されました（^）。

REGEVT__INFO

INFO キーが押されました。

REGEVT__VSK1

垂直ソフトキー 1 が押されました。

REGEVT__VSK2

垂直ソフトキー 2 が押されました。

REGEVT__VSK3

垂直ソフトキー 3 が押されました。

REGEVT__VSK4

垂直ソフトキー 4 が押されました。

REGEVT__VSK5

垂直ソフトキー 5 が押されました。

REGEVT__VSK6

垂直ソフトキー 6 が押されました。

REGEVT__VSK7

垂直ソフトキー 7 が押されました。

REGEVT__VSK8

垂直ソフトキー 8 が押されました。

REGEVT__ETC

ETC キーが押されました（>)。

REGEVT__WNDTOGGLE

ウィンドウ切り替えキーが押されました。

第 5 部 : ファイルアクセス用インタフェース

840DI ファイルサーバは，以下のものとのインタフェースがあります。

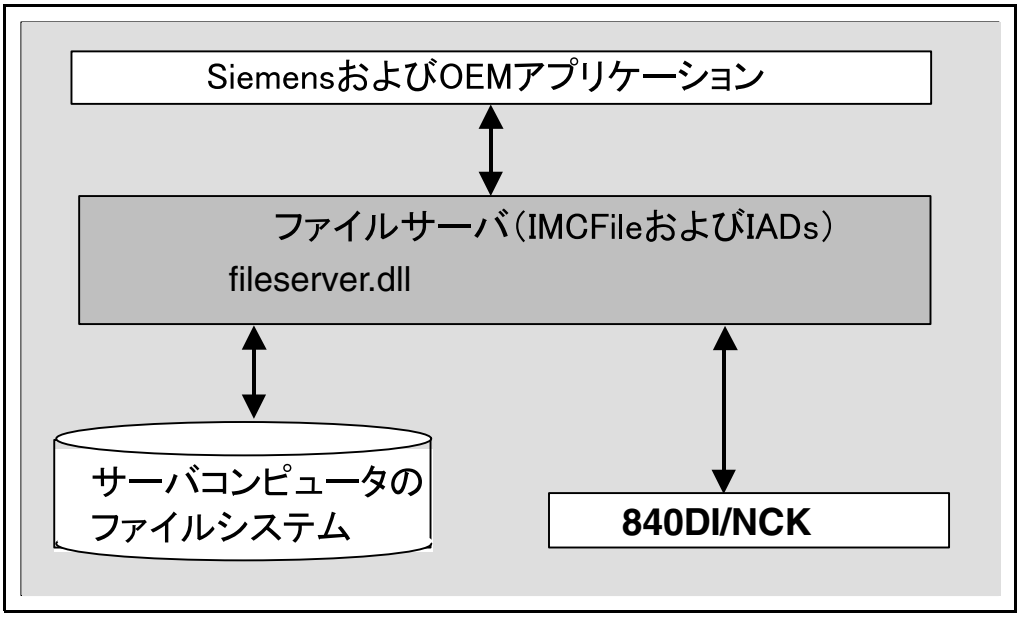
- IMCFile
- IMCFileEvent
- IADs
- IADsContainer
- IstaticCollection

840DI ファイルサーバは，NC の場合と同じ設定を使って，オペレータコンピュータ上のファイル構造を管理します。これによって，NC 上およびオペレータコンピュータ上のファイルについて，一貫性のあるビューを提供します。

このファイル構造は，’ データ管理 ’ と呼ばれ，オペレータコンピュータのファイルシステムのどの位置にでもインストールできます。

データ管理は，データスキーム，つまりハードディスク上のデータ構造の記述に基づいています。それは実行時に，読み取り可能な形式でシステムに保管されます。ファイルサーバを介して何らかのアクセスが実行される場合，スキームに関する一貫性は継続的に保証されます。

データ管理へのアクセスは，次のような図で表すことができます。



11 サーバコンピュータへのデータ管理の組み込み

データスキーム

HMI Base がインストールされているコンピュータを, 本書では ' サーバコンピュータ ' と呼びます。これはたとえば，開発コンピュータおよび PCUxx ファミリのすべてのターゲットシステムです。

データスキームは，ファイルおよびディレクトリの編集および管理時にファイルサーバが従う必要のあるディレクトリ構造の定義です。

これには次のものが含まれます。

- 個々の要素（ファイル／ディレクトリ）の記述
- ファイル構造の記述

データスキームは，上記の機能を使用して，データ管理ツリーにあってデータ管理によって保管できる要素の構造を記述します。

したがってスキームは，操作が許されるものかどうかを検査するテンプレートとしての役割を果たします。

データ管理ツリー

データ管理は，NC の 1 つの受動ファイルシステムと，サーバコンピュータのファイルシステムにある，対応するファイル構造から構成されています。ファイル構造はデータスキームで定義されます。

ファイルシステム内のデータ管理スキームのルートパスは，サーバコンピュータの DH.INI ファイルに入力する必要があります。

例 : C:¥siemens¥SINUMERIK¥DH

また，データ管理サーバを使って，データ管理ツリーの外側のデータにアドレス指定することもできます。これらのオブジェクトは，標準データタイプ “Dosfile” のものです。

初期設定ファイル

MMC2 ディレクトリにある DH.INI ファイルには，特に以下の項目が含まれています。

表 11.1 初期設定ファイル

セクション	項目	意味
DHSTART	mmchome	サーバコンピュータのファイルシステムにあるデータ管理のルートディレクトリ
DHSTART	Accesslevel	アクセス権限
SCEME	sceme	データスキームのバイナリファイル名

例 :

[DHSTART]

mmchome= c:¥siemens¥SINUMERIK¥dh

accesslevel=4

[SCEME]

sceme=schema.bin

ファイルシステム MMC / NCK

データスキームで記述される要素は，サーバコンピュータのファイルシステムと，NCK のファイルシステムの両方に置くことができます。

データスキームで記述されるディレクトリ構造には，NCK システムのファイル構造のイメージが含まれます。

データ管理によって，ユーザは一度にすべてのデータを概観することができ，さらに両方のファイルシステムのファイルに同じようにアクセスできます。

システムの実行時，データ管理機能はデータスキームの設定を使用し，指示されているファイルおよびディレクトリの構造を監視します。

注

ただし，指示される構造の監視は，ファイルサーバ操作中のみモニタされます。ハードディスク上のファイルは無許可アクセスに対してロックされていません。

ADSI パス

ADSI パスは，データ管理ツリーを明確に識別します。

12 データ管理ツリーの要素の特性

データ管理ツリーでは，データスキームで定義されるディレクトリ構造は，個々の要素から構成されます。データ管理の観点からの要素は，ファイルまたはディレクトリです。

データ管理ツリーの各要素ごとに，ファイルサーバから要求できる以下の情報が保管されています。

ファイルタイプ記述は，パートプログラムの特定の特性を記述します。

各要素は，以下の表でリストされるパラメータによって記述されます。

表 12.1 要素のパラメータ

パラメータ	意味
名前	ファイルまたはディレクトリの名前
拡張子	DOS ファイルシステム用の名前の拡張子
プレーンテキスト	読み取り可能な形式で要素を表示するための ID
データフォーマット	ディレクトリまたはファイル

名前

これは，ファイルまたはディレクトリがデータツリーに保管されるときに使用される名前です。

要素が NCK ファイルシステムのファイルまたはディレクトリのイメージで，このシステムが固定名を必要とする場合，この名前がここで使用されます。

作成されるファイルまたはディレクトリでどんな名前でも許可される場合，”*”が名前として使用されます。

名前の最大長は，25 文字です。

拡張子

ファイルシステムによって管理され，データ管理で編集できる要素の一部であるファイルおよびディレクトリには，すでに 3 文字の拡張子が付いています。

このため，NCK と MMC ファイルシステムの両方で保管できる要素の拡張子は，NCK システムによって指定されます。NCK によって処理されない要素の場合，MMC のハードディスク専用の拡張子が定義されます。

プレーンテキスト

言語に依存する ID が，各要素ごとに定義されます。

例としては，名前が NCK によって定義される場合，ファイルまたはディレクトリ名の言語に依存する表示を明確にするのに使用されます。たとえば，ドイツ語の `*.wpd` のプレーンテキストは "Werkstück" で，英語の場合 "part" です。選択される言語は，OPCData の項目 `Long prefix` に対応します。

データフォーマット

ファイルツリー中のオブジェクトが，ファイルかディレクトリかを特定します。

アーカイブ位置

システム全体でファイルおよびディレクトリをアーカイブする場合，3 つの可能性があります。

- ハードディスク上の MMC ファイルツリーで
- NCK ファイルシステムのデータ構造でのみ
- MMC ハードディスク上と NCK ファイルシステムの両方で

アクセス許可 :

表 12.2 アクセス許可の 8 レベル

アクセスレベル	アクセスに必要なもの	ユーザグループ
S0	システムパスワード	当社
S1	マシンメーカのパスワード	マシンメーカ
S2	保守パスワード	スタートアップエンジニア，サービス（マシンメーカ）
S3	エンドユーザのパスワード	特権を持つエンドユーザ（独自のサービス）
S4	キースイッチ位置 3	プログラマ
S5	キースイッチ位置 2	限定オペレータ
S6	キースイッチ位置 1	オペレータ
S7	キースイッチ位置 0	訓練オペレータ（NC 開始／ NC 停止，MCP）

アクセス許可レベル 0 が最高レベルで，アクセス許可レベル 7 が最低レベルの権限です。

このレベルは，現在のキースイッチ位置によって，またはパスワードの入力によって設定されます。

アクセス権限

次の権限が割り当てられます。

READ：読み取り

WRITE：書き込み

EXECUTE：実行

SHOW：`List` で要素を表示できる

DELETE：削除

アクセスマスク

ファイルツリーの各要素には，どのアクセスレベルからファイルを編集できるかを定義する，アクセスマスクを割り当てます。

これらのアクセスマスクは，5 桁の数字で表されます。

表 12.3 アクセスマスク

1 桁目	2 桁目	3 桁目	4 桁目	5 桁目
READ	WRITE	EXECUTE	SHOW	DELETE

アクセスマスクの例 : 7 4 7 7 5

アクセスレベル 0 ～ 4 だけが，書き込みを許可されます。

アクセスレベル 0 ～ 5 だけが，削除を許可されます。

すべてのアクセスレベルで，読み取り，実行，および表示が許可されます。

要素のアクセスマスクは，データ管理関数 *Get_Attributes* を使用して決定され，*SetAccess* によって変更されます。

アクセス権限の変更

特定のアクセスレベルに対応する個々のアクセス権限は，より高位のレベルからしか変更できません。この変更を行うために，データ管理関数 *SetAccess* が用意されています。

グローバルに有効な，データスキームのデフォルトのアクセスマスクは変更できません。

データ管理関数を使って管理できる，最も頻繁に使用されるファイルとディレクトリについては，15 章「データ管理定義」を参照してください。

13 データ管理クラス

データ管理クラスのモデル

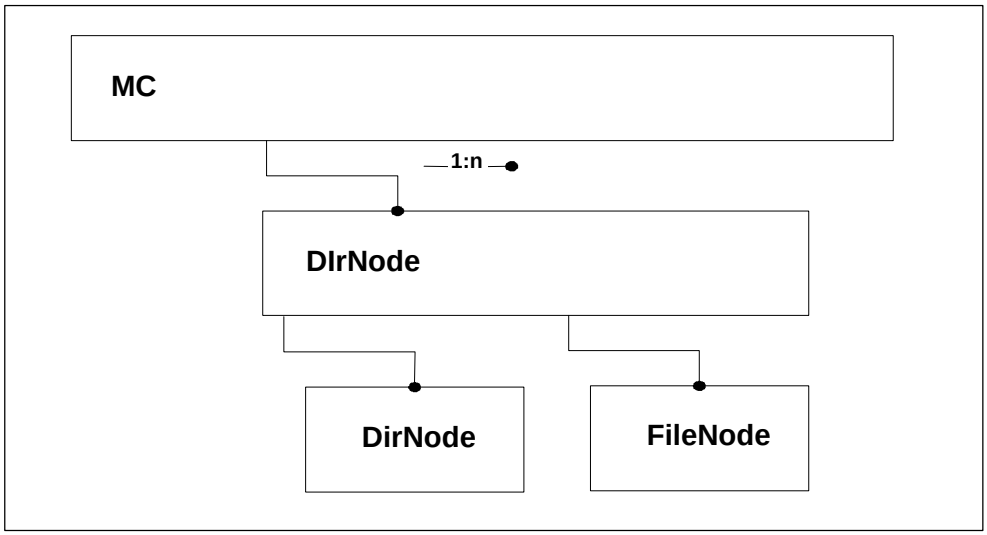


図 13.1 データ管理クラスのモデル

MC クラス

MC クラスは，最上位のクラスです。このクラスには，ADS ツリーのルートを構成するオブジェクトが 1 つだけ入っています。個々の ADS ドメイン，たとえば NC，HD... などがここにあります。このオブジェクトは，直接のユーザインタフェースは提供しません。

DirNode クラス

DirNode クラスは，インタフェース IADs と IADsContainer を使用して，ここに入っているディレクトリと要素に関する情報を提供します。また，IStaticCollection を使用して非同期にディレクトリをリストすることもできます。

さらに，MC 関連の機能も，インタフェース IMCFile および IMCFileEvent を介して可能です。ADSI と IMCFile によって適切な機能性が提供されていれば，ほとんどの場合，IMCFile がより包括的なパラメータ化機能を提供します。

FileNode クラス

FileNode クラスは DirNode のサブセットです。インタフェース IADsContainer および IStaticCollection は含まれていません。

14 データ管理インタフェース

以下のインタフェースは，データ管理へアクセスするためのものです。

- IMCFile
- IMCFileEvent
- IstaticCollection

14.1 IMCFile

IMCFile インタフェースは，可能な限り，一般的なデータインタフェースの構文とセマンティクスを提供します。これは，たとえば Copy，Move/Rename，および現行のコンテナオブジェクトで新しいオブジェクトを作成する Create メソッドに当てはまります。

IMCFile メソッドの Create，CopyHere，および MoveHere は，ユーザーが使用する標準メソッドです。IADSContainer インタフェースの該当するメソッドで提供する機能性は，いくつかのパラメータが欠落しているため，限定されています。これらのインタフェースについては，デフォルト処理だけが実行されます。

IMCFile インタフェースは，標準の Active Directory Service Interface （ADSI）で扱われていないデータ管理のすべてのメソッドをまとめたものです。以下のものが含まれます。

- ADSI に存在しないメソッド
- ADS インタフェースの標準メソッドを拡張したメソッド

このインタフェースをインプリメントする理由は，データ管理におけるオブジェクトの名前，拡張子，およびデータタイプ間の関係が緊密であることです。

データ管理のオブジェクトのタイプまたはクラスは，その拡張子と，特定の名前スキームに対応する必要があるかどうかを定義します。

標準 ADS インタフェースでは，オブジェクトの名前，拡張子，およびクラスを一度に変更することはできません。たとえば，ワークに含まれるパートプログラムが同じワークに含まれるサブルーチンにコピーされると，オブジェクトの名前，拡張子，およびクラスも変更されます。前述の項目すべてが一度の呼び出しで変更される場合のみ，オブジェクトは一貫性を保ちます。そうでない場合，拡張子がデータタイプと一致せず，呼び出しは取り消され，エラーが出されます。

14.1.1 管理メソッド

ファイルの作成

このメソッドは，割り当てられたデータ管理オブジェクトで ADSI オブジェクトを作成するのに使用されます。任意の Windows ファイルをソースとして指定できます。ソースファイルを移動するか，または同じディレクトリにソースファイルのコピーを作成するか，どちらかが可能です。「上書き」を使用することができます。

ターゲットコンテナのパスは，新しいファイルまたはディレクトリの保管先を指定します。

Create メソッドは，同期的に実行されます。クライアントアプリケーションは，新しいオブジェクトが作成された場合のみ続行します。NewObject にはこのオブジェクトへのポインタが含まれます。CreateAsync メソッドは，非同期的に実行するジョブのみを送ります。結果は，IMCFileEvent インタフェースを介して戻されます。

表 14.1 ファイルの作成

IMCFile::Create/IMCFile::CraeteAsync		
パラメータ	**タイプ**	**意味**
Class	BSTR	このオブジェクトのスキームクラスの名前
RelativeName	BSTR	name.ext を作成するオブジェクトの相対名
mode	MC_CREATE_MODES	作成用のモード（15.3 節を参照）
SourceFile	BSTR	使用する Windows ソースファイル
NewObject	[out] Idispatch **	新しいオブジェクトのインタフェース

ファイルの削除

このメソッドは，指定した ANSI オブジェクトを削除します。

表 14.2 ファイルの削除

IMCFile::Delete		
パラメータ	**タイプ**	**意味**
Class	BSTR	このオブジェクトのスキームクラスの名前
RelativeName	BSTR	name.ext を作成するオブジェクトの相対名
Mode	MC_DELETE_MODES	定義される唯一の値は 0（15.3 節を参照）

ファイルの移動

このメソッドは，ADSI オブジェクトを移動するか，またはオブジェクトを改名し，必要であればデータ管理タイプを変更します。同期メソッドは即座に結果を出しますが，非同期変形は結果を IMCFileEvent を介して戻します。

表 14.3 ファイルの移動

IMCFile::MoveHere/IMCFile::MoveHereAsync		
パラメータ	**タイプ**	**意味**
SourceObject	BSTR	ソースオブジェクトの ADSI パス
NewName	BSTR	新規名（name.ext）
MCClass	BSTR	このオブジェクトの MCscheme クラスの名前（*.mpf）
mode	MC_COPY_MODES	コピー用のモード（15.3 節を参照）
NewObject	[out] Idispatch **	新しいオブジェクトのインタフェース

ファイルのコピー

このメソッドは，データ管理オブジェクトを現行のコンテナオブジェクトにコピーします。ソースオブジェクトは，そのフル ADS パスで指定されます。この名前が変更されると，既存のデータ管理タイプまたは新しく指定されたデータタイプと一致する名前だけが許可されます。同期メソッドは即座に結果を出しますが，非同期変形は結果を IMCFileEvent を介して戻します。

表 14.4 ファイルのコピー

IMCFile::CopyHere/IMCFile::CopyHereAsync		
パラメータ	**タイプ**	**意味**
SourceObject	BSTR	ソースオブジェクトの ADSI パス
NewName	BSTR	新規名（name.ext）
MCClass	BSTR	このオブジェクトの MCscheme クラスの名前（*.mpf）
Mode	MC_COPY_MODES	コピー用のモード（15.3 節を参照）
NewObject	[out] Idispatch **	新しいオブジェクトのインタフェース

ファイルの活動化

活動化は，混合パス（MC://domain/HD–...）のオブジェクトにのみ定義されます。これらのパスのみが，新しい ADS パスを割り当てなくても，保管される位置を変更できます。他のエリアからのオブジェクトは，このメソッドに対するエラーを受け取ります。

同期メソッドは即座に結果を出しますが，非同期変形は結果を IMCFileEvent を介して戻します。

表 14.5 ファイルの活動化

IMCFile::Activate/IMCFile::ActivateAsync		
パラメータ	**タイプ**	**意味**
Mode	MC_ACTIVATE_MODES	活動化用のモード（15.3 節を参照）

ファイルの受動化

受動化は，混合パス（MC://domain/HD–...）のオブジェクトにのみ定義されます。これらのパスのみが，新しい ADS パスを割り当てなくても，保管される位置を変更できます。他のエリアからのオブジェクトは，このメソッドに対するエラーを受け取ります。

同期メソッドは即座に結果を出しますが，非同期変形は結果を IMCFileEvent を介して戻します。

表 14.6 ファイルの受動化

IMCFile::Passivate/IMCFile::PassivateAsync		
パラメータ	**タイプ**	**意味**
Mode	MC_ACTIVATE_MODES	受動化用のモード（15.3 節を参照）

パートプログラムの選択

このメソッドは，チャネル内の実行用の NC でパートプログラムとワークを選択するのに使用できます。

表 14.7 パートプログラムの選択

IMCFile::Select		
パラメータ	**タイプ**	**意味**
Channel	BSTR	パートプログラムを実行するチャネル

外部パートプログラムの選択

このメソッドは，NC に含まれないパートプログラムを選択するのに使用します。

表 14.8 外部パートプログラムの選択

Extern		
パラメータ	**タイプ**	**意味**
Channel	BSTR	パートプログラムを実行するチャネル

可能なファイルタイプの入手

2 次元の安全配列，つまり 2 つの列と n 行を持つ安全配列のコンテナに含まれるデータタイプのリストを提供します。各行の最初の列には，データタイプの言語に依存する ID が含まれており，2 番目の列には ADS インタフェースのクラス名が含まれます。

表 14.9 可能なファイルタイプの要求

IMCFile::PossibleDatatypes		
パラメータ	**タイプ**	**意味**
Mode	MC_FILTER_MODE	パートプログラムを実行するチャネル（15.3 節を参照）
DataTypes	[out] VARIANT *	安全配列として可能なファイルタイプ

データ管理の優先タイプの入手

拡張子とデータスキームに基づいて，このメソッドは DOS ファイルが最適に変換できるデータタイプを判別するのに使用できます。

表 14.10 データ管理の優先タイプの要求

IMCFile::BestDatatype		
パラメータ	**タイプ**	**意味**
DataType	[out] BSTR *	最適なデータ管理のデータタイプを提供する

データ管理操作の選択項目の入手

このメソッドは，2 次元の安全配列，つまり 2 つの列と n 行を含む安全配列でオブジェクトを変換できるデータタイプのリストを提供します。最初の行の列には，データタイプの言語に依存する ID が含まれており，2 番目の列には ADS インタフェースのクラス名が含まれます。

表 14.11 データ管理操作の選択項目の要求

IMCFile::ConvertPossibleDatatypes		
パラメータ	**タイプ**	**意味**
DataTypes	[out] VARIANT *	安全配列として可能なファイルタイプ

データ管理特性の入手

データ管理特性（たとえば編集可能性など）のリストを提供します。1 ビットにつき 1 つの特性がコード化されます。

表 14.12 データ管理特性の入手

IMCFile::Get_PropertyNames		
パラメータ	**タイプ**	**意味**
名前	[out] VARIANT *	特性ビットの名前

14.1.2 NC またはハードディスクを使ったストリーム操作のメソッド

以下のメソッドは，NC とハードディスクの両方に適用できます。これらのメソッドは，Windows ファイルシステムの標準メソッドであるオープン，シーク，読み取り，書き込み，およびクローズに対応します。

Open

Windows または NC にあるファイルを，ファイル内のサブバッファの編集，または読み取り／書き込みのためにオープンします。オープンは排他的です。

表 14.13 Open

IMCFile::Open		
パラメータ	**タイプ**	**意味**
Mode	MC_OPEN_MODE	（15.3 節を参照）

Seek

シークはファイル内の現行編集位置を設定し, オフセットは ' モード ' を介して定義された初期位置のオフセットを定義します。

表 14.14 Seek

IMCFile::Seek		
パラメータ	**タイプ**	**意味**
Offset	Long	オフセット
Mode	MC_SEEK_MODE	オフセットの開始点を定義する（15.3 節を参照）

NCRead

必要なデータ域を出力バッファに転送します。出力データはタイプ VT_UI1 の安全配列に入力されます。実際に読み取られるバイト数は，安全配列の値から判別され，必要な数とは異なる場合があります。

表 14.15 NCRead

IMCFile::NCRead		
パラメータ	**タイプ**	**意味**
VarBuf	[out] VARIANT *	ターゲット域へのポインタ
BytesToRead	Long	必要な長さ

NCWrite

転送されたデータバッファをファイルに書き込みます。バッファはタイプ VT_UI1 の安全配列として転送されます。データバッファの長さは，安全配列のパラメータから導出されます。実際に書き込まれるバイト数は，出力パラメータ lNumberOfBytesWritten に戻され，必要なバイト数とは異なる場合があります。

表 14.16 NCWrite

IMCFile::NCWrite		
パラメータ	**タイプ**	**意味**
VarBuf	VARIANT *	バッファを転送するポインタ
BytesWritten	[out] long *	書き込まれる長さ

Close

オープンしているファイルをクローズします。現在実行している転送があれば，取り消されます。

表 14.17 Close

IMCFile::Close		
パラメータ	**タイプ**	**意味**
なし		

14.1.3 NC の受動ファイルシステムのメソッド

このセクションで説明されるメソッドは，パイプメカニズムを使用して，NC への，および NC からのファイル転送を行います。

データは，コピーの場合と同じように，ファイルを介してではなく，メモリ域を介して転送されます。

読み取り時，転送が始まると，内部転送バッファは継続的に NC ファイルのデータで満たされます。

転送バッファがいっぱいになると，転送はデータがバッファから読み取られるまで中断します。ReadPipe メソッドが呼び出されると，転送バッファからデータの必要なボリュームを読み取ろうとする試みがなされます。転送バッファ内のデータが不十分である場合，データの必要なボリュームがメモリ域に転送されるか，または転送が取り消されるまで，データが転送バッファにさらに再ロードされます。

書き込みでは，データは転送されるメモリ域から内部転送バッファへ，さらにここから NC ファイルへ転送されます。転送バッファ内の空きスペースがデータの書き込みには小さすぎる場合，転送バッファ内のデータが完全に転送されるまで，書き込み呼び出しは使用不可になります。データが占めるメモリが再び解放され，メモリ域の欠落データが転送バッファに再ロードされます。WritePipe メソッドは，すべてのデータの書き込みが終了するか，または転送が取り消された場合のみ完了します。

Open_Memory_Pipe

このメソッドは PipeTransfer を開始します。オブジェクトごとに同時に実行できるパイプ転送操作は，1 つだけです。転送は，読み取りつまり NC からの転送，または NC への書き込みのどちらかです。

表 14.18 Open_Memory_Pipe

IMCFile::Open_Memory_Pipe		
パラメータ	**タイプ**	**意味**
mode	MC_PIPE_MODE	読み取り／書き込み（15.3 節を参照）

ReadPipe

転送バッファがパイプから満たされます。

必要なデータサイズは，パラメータ lNumberOfBytesToRead を介して指定されます。

メソッドは，すべてのデータが転送されるか，または必要なメモリが満たされた場合に完了します。

新しいデータがまだ存在しない場合，呼び出しは，データが提供されるか転送が取り

消されるまで使用不可です。

実際に送達されるデータは，安全配列の特性を介して要求できます。

これらはタイプ VT_UI1 の安全配列として転送されます。

表 14.19 ReadPipe

IMCFile::ReadPipe		
パラメータ	**タイプ**	**意味**
VarBuf	[out] Variant *	転送バッファ
NumerBytes	long	バイト単位のバッファ長
Percent	[out] byte *	すでに転送されたデータのパーセンテージ

WritePipe

バッファをファイルに書き込みます。このメソッドは，バッファ全体が書き込まれた場合，または転送処理がクローズされた場合に完了します。転送は，タイプ VT_UI1 の安全配列で実行されます。ReadPipe および WritePipe 呼び出しは，転送処理で混合することができません。

表 14.20 WritePipe

IMCFile::WritePipe		
パラメータ	**タイプ**	**意味**
StrBuf	Variant *	転送バッファ

ClosePipe

転送処理をクローズします。

表 14.21 ClosePipe

IMCFile::ClosePipe		
パラメータ	**タイプ**	**意味**
なし		

Cancel

現在実行中の非同期ジョブを取り消します。最終の CompleteEvent が生成されます。

表 14.22 Cancel

IMCFile::Cancel		
パラメータ	**タイプ**	**意味**
なし		

14.2 MCFileEvent

IMCFileEvent インタフェースは，二重インタフェースとしてインプリメントされます。

ProcessStatus

ADSI オブジェクトは，このメソッドを使ってジョブの現在の状況をパーセンテージとして通知します。

エラー，取り消し，または正常終了の場合，メソッド CompleteStatus が呼び出されます。

表 14.23 ProcessStatus

IMCFileEvent::ProcessStatus		
パラメータ	**タイプ**	**意味**
ValStatus	Variant *	現在のジョブ状況
ValueRet	Long	追加のエラー情報

CompleteStatus

ADSI オブジェクトは，このメソッドを使って，実行後にジョブの状況を通知します。

S_OK またはエラーコードが valRet 値で転送されます。

エラーが発生しなかった場合，新しく作成されたオブジェクトの Idispatch インタフェースポインタが，IMC ファイルインタフェースの非同期メソッド（Create，CopyHere，MoveHere，Activate，Passivate）に，valDisp を介して転送されます。

他のすべての非同期メソッドは，このパラメータを無視します。

追加のエラー情報の可能な値については，15 章「データ管理定義」を参照してください。

表 14.24 CompleteStatus

IMCFile::CompleteStatus		
パラメータ	**タイプ**	**意味**
ValRet	Long	エラー値
ValueEx	Long	追加のエラー情報
NewObject	Idispatch *	何らかのメソッドの作成済みオブジェクト

14.3 IstaticCollection

コンテナの内容，つまりディレクトリの内容は，コレクションに内部的に保管されます。

コレクションの充てんは，コンテナオブジェクトの列挙型定数が最初に作成されたとき，または IStaticCollection の Reflesh メソッドが明示的に呼び出されたときに開始されます。

NC ディレクトリの内容のリスト作成にはしばらく時間がかかるので，Reflesh メソッドは非同期メソッドとしてインプリメントされます。したがってアプリケーションは，同期メソッド get_NewEnum の場合と同様，このプロセス中にブロック化されることなく，簡単にコレクションを充てんできます。

IMCFileEvent インタフェースのメソッドを使って，フィードバックが提供されます。

Refresh

表 14.25 Refresh

IStaticCollection::Refresh		
パラメータ	**タイプ**	**意味**
mode	long	予約済み，必ず 0

15 データ管理定義

15.1 データタイプ

ここでは，データ管理機能を使用して管理できる，最も頻繁に出てくるファイルおよびディレクトリのタイプを示します。

すべてのオプションの完全なリストは，IMCFile::PossibleDatatypes 関数を使って各ディレクトリで入手できます。

表 15.1 データタイプ

要素	拡張子	名前	フォーマット	内容／意味／使用法
アーカイブ	ARC	*	ファイル	アーカイブ用に MMC ハードディスクに書き込まれるデータ
ブートデータ	BOT	*	ファイル	NCK/611D の起動に必要なバイナリ形式のマシンデータ
クリップボード	CLP	CLIP	ディレクトリ	データ管理のクリップボードディレクトリ
定義	DEF	*	ファイル	定義としてシステム機能が必要とするデータ
設定可能なゼロオフセット／FRAME	UFR	*	ファイル	設定可能なゼロオフセットおよびフレーム（UFR ユーザフレーム）
グローバルユーザデータ	GUD	*	ファイル	マシン固有のユーザデータ（GUD：Global User Data）
初期設定プログラム	INI	*	ファイル	特定のタイプに固定されない NCK または MMC 変数への値割り当て
コメント	COM	*	ファイル	任意の ASCII テキストの入ったファイル
補正データ	IKA	*	ファイル	補間された補正データ
マシンデータ	TEA	*	ファイル	NCK マシンデータ（TEA：Testing Data Active）

表 15.1 データタイプ（続き）

要素	拡張子	名前	フォーマット	内容／意味／使用法
NC データ	MDN	*	ディレクトリ	NC マシンデータディレクトリ
OEM 固有のデータ	USR	*	ファイル	OEM ユーザが定義する任意のデータ
設定データ	SEA	*	ファイル	設定データ（SEA：Setting Data Active）
システムデータ	SYF	*	ファイル	システム機能の実行に必要なデータ
パートプログラム	MPF	*	ファイル	タイプが ' メインタイプ ' または 'サブルーチン'のNCパートプログラム
パートプログラム固有またはローカルのユーザデータ	LUD	*	ファイル	パートプログラムで定義されているローカル変数（LUD：Local User Data）
サブルーチン	SPF	*	ファイル	NC サブルーチン : サイクルプログラムおよび他のプログラムから呼び出されるすべてのプログラム
ディレクトリ	DIR	*	ディレクトリ	データツリー内の任意のディレクトリ
ワークのディレクトリ	WPD	*	ディレクトリ	完全な加工のためのワークを記述するのに必要なすべてのファイル
工具補正	TOA	*	ファイル	工具補正（TOA：Tool Offset Active）
アクセス記述	ACC	*	ファイル	現在設定されているアクセス許可に基づき，特定の NCK 変数へのアクセスを記述する

アーカイブ

MMC ハードディスク上でアーカイブ用に格納されたデータ。

ブートデータ

NCK/611D のインストールに必要なバイナリ形式のマシンデータ。

ブートデータは，メインの主軸と，送り駆動機構（複数提供可能）をインストールするのに必要なデータです。各駆動機構には独自の起動データがあるので，各駆動機構ごとに，対応するデータが入っている，固定名を持つファイルが定義されます。

クリップボード

このディレクトリは，ファイルを一時的に格納するためにデータ管理によって使用されます。

このディレクトリでは，スキームで定義されたすべてのデータタイプを，グローバルデータ構造で定義されるのと同じ形式で定義できます。つまり，データツリー全体をこのノードの下でもう一度作成することができます。

定義

システム機能が定義として必要とするデータ。

ここでも，いくつかのデータタイプが，NCK で一度だけ現れる固定名で定義されます。

設定可能なゼロオフセット /FRAME

設定可能なゼロオフセットおよびフレーム（UFR：User Frame）。

グローバルユーザデータ

マシン固有のユーザデータ（GUD：Global User Data）。

このタイプのファイルでは，ユーザは自分のマシンに固有のデータを定義できます。これは，グローバルに有効な R パラメータにも関係します。

データは，パートプログラムおよびサイクル内でアドレス指定するか，または画面上での表示用に構成することによってアドレス指定できます。

その構造は，データタイプマシンデータに対応します。

初期設定プログラム

特定のタイプに固定されない，NCK または MMC 変数への値割り当て。

‘ 初期設定プログラム ’ には，特定のタイプに固定されない，NCK または MMC 変数への値割り当てが含まれます。

このブロックは，ワークディレクトリ内でマシンの初期設定ファイルとして格納できます（「ワークディレクトリ」も参照）。

コメント

ASCII テキストの入ったファイル。

ここでは，固定された名前でデータタイプとして定義されるいくつかのコメントファイルが提供されます。これは，これらのファイルの内容が特定であるためです。これらのファイルは，名前と拡張子の組み合わせによってデータ管理に識別されます。

補正データ

補間された補正に必要なデータ。

マシンデータ

NCK マシンデータ（TEA：Testing Data Active）。

NC データ

NC マシンデータのディレクトリ。

このディレクトリには，活動中の NCK データの状態をシミュレートできるデータを格納できますが，そのデータは MMC ファイルシステムに格納されることになります。

OEM 固有のデータ

OEM ユーザが定義する任意のデータ。

ここでも，さまざまなデータタイプを固定名で定義できます。

設定データ

設定データ（SEA：Setting Data Active）。

システムデータ

システム機能を実行するのに必要なデータ。

パートプログラム

タイプが ' メインプログラム ' または ' サブルーチン ' の NC パートプログラム。

NCK は，メインプログラムとサブルーチンを区別しません。

注

プログラムが，コンピュータリンクを介して中央実動コンピュータから制御システムにロードされると，提供される拡張子があればそれを使用して 2 つのタイプの間に区別が付けられ，プログラムはプログラムタイプに従って適切なディレクトリに組み込まれます。

パートプログラム固有／ローカルユーザデータ

パートプログラムで定義されるローカル変数（LUD：Local User Data）。

この種のファイルには，パートプログラムで定義されるローカル変数が含まれます（LUD：Local UserData）。

注

これらの変数は，関連するパートプログラムが実行中である間だけ存在します。しかし，データ管理からアクセスされるファイルとして保管することができます。

サブルーチン

このデータタイプには，サイクルプログラムおよび他のプログラムから呼び出されるすべてのプログラムが含まれます。NCK システムはメインプログラムとの間に区別を設けません。

ディレクトリ

データツリー内のすべてのディレクトリ。

特殊ディレクトリの一例は，ワークディレクトリです。

プレーンテキスト： ワーク
拡張子： DIR
名前： WCS
フォーマット： ディレクトリ
属性： –

このワークディレクトリでは，タイプが ’ ワーク ’/WPD’ のディレクトリのみが作成されます。

ワークディレクトリ

完全に加工できるようワークを記述するのに必要なすべてのデータタイプ。

ワークディレクトリでは，さまざまなデータタイプを作成できます。これは，ワークを完全に加工するのに必要なすべてのデータに関連します。これらは，パートプログラム，設定データ，工具補正，初期設定ブロックなどです。

データタイプ ’ ワーク ’/WPD’ には，以下の特長があります。

NCK でのパートプログラムとして，ワークを加工用に選択できます。

データスキームのアクセスマスクでは，この特性はアクセス権限 EXECUTE を使用して表現されます。

加工用のワークの選択時には，ワークと同じ名前の INI ファイルが NCK にロードされ，ワークと同じ名前のメインプログラムが自動的に選択されます。

同じ名前のパートプログラム／ MPF が使用できない場合，エラーメッセージが出力され，前に選択されたパートプログラムが選択されたままになります。

同じ名前の INI ブロックが存在しない場合，他の初期設定ブロックが適切なオペレータ処置を介して実行されます。

工具補正

工具補正（TOA：Tool Offset Active）。

アクセス記述

現在設定されているアクセス許可に基づき，特定の NCK 変数へのアクセスを記述します。

このタイプのファイルは常に特定の NCK 変数を参照するため，定義済みデータタイプの一連の固定名が提供され，値のこの特定の部分へのアクセスを記述します。

データ管理は，名前と拡張子の組み合わせを使用して，NCK ファイルのアクセス記述を見つけることができます。

例 :

C:¥MMC_OEM.V42¥MMC2¥IB¥BT_TEA.ACC

15.2 許可できる ADSI パス

表 15.2 許可できる ADSI パス

ADSI パス	意味
MC://domain/HD–NC	現在設定されている NCU とデータ管理ディレクトリの混合表示
MC://domain/HD	データ管理ディレクトリ
MC://domain/NC	現在設定されている NCU
MC://domain/NCU_1	特定の NCU（名前は netnames.ini にあるとおり）
MC://domain/HD–NCU_1	特定の NCU とデータ管理ディレクトリの混合表示
MC://domain/C:/temp	PC の視点からのパス（データ管理内ではない）

15.3 列挙型タイプの定義済み値

MC_CREATE_MODES の値

MC_CREATE_NORMAL = 0

MC_CREATE_OVERWRITE = 1

MC_CREATE_MOVE = 16

MC_COPY_MODES の値

MC_COPY_NORMAL = 0

MC_COPY_OVERWRITE = 1

MC_ACTIVATE_MODES の値

MC_ACTIVATE_NORMAL = 0

MC_ACTIVATE_OVERWRITE = 1

MC_ACTIVATE_NO_SOURCE_DELETE = 4

MC_DELETE_MODES の値

MC_DELETE_NORMAL = 0

MC_FILTER の値

MC_FILTER_FILE = 256

MC_FILTER_DIR = 512

MC_FILTER_READABLE = 1024

MC_FILTER_WRITEABLE = 2048

MC_OPEN_MODES の値

MC_OPEN_EXCLUSIVE = 1

MC_SEEK_MODES の値

MC_SEEK_BEGIN = 0

MC_SEEK_CURRENT = 1

MC_SEEK_END = 2

MC_PIPE_MODES の値

MC_PIPE_READ = 0

MC_PIPE_WRITE = 1

15.4 エラーメッセージ

機能呼び出しの非同期変形は，コールバックルーチン CompleteStatus の 2 番目のパラメータで以下のエラーメッセージを出す場合があります。

表 15.3 エラーメッセージ

エラーコード	エラーメッセージテキスト	意味
101	DOS ファイル作成もしくはオープン時エラー	読み取り専用ディレクトリにファイルを作成しようとしました。
102	ファイルが存在しません	存在しないファイルをオープンしようとしました。
103	ファイルがリードオンリーです	書き込み保護されているファイルに書き込もうとしました。または空でないディレクトリを削除しようとしました。
104	No matching node found	パスがデータスキームと一致しません。
105	No matching data type found	データタイプはこの位置，たとえば SPF ディレクトリの MPF では使用できません。
106	File/Dir already exists	すでに存在するデータを上書きしようとしました。
107	No valid DHpath	無効なデータパスに移動しようとしました。
108	パラメータエラー	完全なセットがないか，または間違ったパラメータのセットで呼び出しを実行しようとしました。
109	Error copying data blocks	ハードディスク上で書き込み／読み取りエラーが検出されました。
110	Invalid source	データ管理パスが間違っています。
111	Illegal target data type	たとえば SPF を MPF にコピーしようとしました。
112	Transmission error	NCU へのリンクに関するグループエラーメッセージです。
113	No read access Leserecht	十分な読み取り権限がないのにアクセスを実行しようとしました。
114	No write access	十分な書き込み権限がないのにアクセスを実行しようとしました。
115	実行しません	実行の十分な権限がないのにアクセスを実行しようとしました。
116	No display access	十分な表示権限がないのにアクセスを実行しようとしました。
117	No deletion access	十分な削除権限がないのにアクセスを実行しようとしました。

表 15.3 エラーメッセージ（続き）

エラーコード	エラーメッセージテキスト	意味
118	Access rights, general	NCU へのアクセス権限の違反についてのグループメッセージ。
119	Resource error	NCU でのリソースエラー。
120	メモリ (Malloc) エラー	MMC メモリマッピングのエラー。解決策 : すべてのアプリケーションをクローズします。
121	情報ファイルエラー	追加の情報を持つファイルでのエラー（25 文字のファイル名，アクセス権限など）。解決策 : ファイルを削除し，DOS ファイルから回復します。
122	リストバッファオーバーフローエラー	固定バッファ長 64KB を超えました。
123	ハードディスクがいっぱいです	ハードディスクの残りの容量が，実行中のジョブを完了するのに不十分です。
124	Command canceled	CANCEL によってコマンドが取り消されました。
125	Floppy disk drive not locked	フロッピディスクドライブが，ドライブへアクセスを試みたときにロックされていませんでした。
126	No further file type possible	
200	DDE エラー	MMC 領域での DDE 通信のエラー。

15.5 ADSI スキームおよびデータ管理スキーム

ADSI 標準属性

ADSI オブジェクトの以下の標準属性は，直接的に変更できません。

表 15.4 ADSI オブジェクトの標準属性

属性	説明
クラス	ADSI オブジェクトの ADSI クラス。
GUID	このスキームクラスのオブジェクト用の，プロバイダ特定の明確な識別（オプション）。MC プロバイダは E_ADS_PROPERTY_NOT_SUPPORTED を提供します。
名前	オブジェクトの相対名，つまり ADSI パスの最後の部分。
ADsPath	完全な ADSI パス。
親	高水準 ADSI オブジェクトのパス。ADSI では，“親”に“名前”をつなげた結果がオブジェクトのパスになるという保証はありません。この手順がすべてのプロパイダで通用するわけではありません。
スキーマ	ADSI オブジェクトのスキームクラスにインタフェース IADsClass を提供する ADSI オブジェクトのパス

プロバイダがスキームを提供しなくても，各 ADSI オブジェクトにこれらの標準属性が備わっています。

属性は ADSI オブジェクトを明確に定義するので，直接的に変更することはできません。

これらの属性のいずれかを変更する場合，新しいオブジェクトを作成する必要があります。

以下のメソッドを適用する場合にこれを守ってください。

表 15.5 メソッド

メソッド	説明
名前変更	名前とタイプの変更。
コピー	コピーが異なる名前を持つ場合の名前の変更。
活動化， 受動化	ADSI パスにアーカイブ位置が含まれる場合，新しいオブジェクトを作成するか，このメソッドをエラーで取り消すことが必要です。

クラス

データ管理スキームは，ADSI スキームに完全にマップします。

スキームの各データタイプは，ファイル中の ADSI オブジェクトのクラスに対応します。

データタイプは単純なクラスにマップし，ディレクトリデータタイプはコンテナクラスにマップします。

クラスインタフェース ADsClass を使用して，コンテナは，含まれるオブジェクトに許可されるクラスのための列挙型定数も提供します。データスキームに従った ADSI クラスのみがここで提供されます。

ADSI オブジェクトの作成時，データ管理のオブジェクト，つまり指定されるデータタイプのファイルまたはディレクトリが作成されます。

オブジェクトのデータタイプは変更できないことに注意してください。

属性

ディレクトリリストの項目を構成するすべての特性は，属性として ADSI スキームに入力されます。

しかし，個々の属性もいくつかの値を持つことができます。

たとえば，ワークディレクトリは NC とハードディスクの両方に存在しており，どちらにもタイムスタンプがあります。

規則として，データ管理が準備したタイムスタンプ（NC ファイルシステムの日付）が表示されます。

しかし，タイムスタンプは，ハードディスクでも内部的に必要です。

このため，異なる値がハードディスクと NC ファイルシステムに存在できるすべての属性について，値の配列が転送されます。

- 最初の値は，データ管理により決定され，表示される値です。
- 2 番目の値は，NC で現在使用中のオブジェクトの値です。
- 3 番目の値は，ハードディスク上のオブジェクトで必要な値です。

次の表は，スキームの属性を説明しています。

表 15.6 スキームの属性

名前	説明	データタイプ	読み取り／設定（R/W）
WinPath	ハードディスク上のパス	BSTR	R
NCPath	現在の NCU の NC パス	BSTR	R

表 15.6 スキームの属性（続き）

名前	説明	データタイプ	読み取り／設定（R/W）
MCAccessrights	アクセス権限 アクセス権限は常に設定できます。アクセス権限は検査されません。（もしあれば）制限は，ユーザインタフェースを介してインプリメントされる必要があります。	安全配列，BSTR BSTR 変形の順序 : データ管理，ハードディスク，NC	R/W
WinAccessrights	ファイル用の Windows アクセス権限	LONG READONLY：1 HIDDEN：2 SYSTEM：4 ARCHIVE：8	R/W
Typedescriptor	データタイプのプレーンテキスト記述	BSTR 言語依存	R
Datatype	データタイプ（名前 スキーム . 拡張子）	BSTR	R
Extension	拡張子は，データタイプを介して設定されるため，読み取り専用です。	BSTR	R
Date	日付	変形の安全配列（VT_I4）データ管理，ハードディスク，NC フォーマット（データ管理サーバの場合）	R
Persist	格納位置	変形，格納位置ごとに 1 ビット PERSIST_MC = 1 PERSIST_NC = 2	R
Name	ファイルまたはディレクトリの名前	BSTR	R
WinName	Windows 名。データ管理サーバが提供します。	BSTR	R

表 15.6 スキームの属性（続き）

名前	説明	データタイプ	読み取り／設定（R/W）
Size	ファイルサイズ	変形の安全配列（VT_I4）データ管理，ハードディスク，NC	R
MCProperty	特性ビット（例：編集可能）	VT_I4	R
External	外部ソースからの処理。getinfo に関係なく，トリガを介して更新されます。	VT_BOOL	R
Edit	編集されます。getinfo に関係なく，トリガを介して更新されます。	VT_BOOL	R

16 低水準のファイル処理

次の低水準インタフェースは，ファイルおよびディレクトリを処理する機能を提供します。このインタフェースは，既存のクライアントの機能性に影響を与えることなく，低水準をカプセル化して現在の必要に適応させます。このインタフェースは，ADSI メカニズムの一部には**できません。**

IMCDomain インタフェースの非同期メソッドは，結果をコールバックインタフェース IMCDomainEvent の Progress（進行中）および Complete（完了）イベントに戻します。インタフェースのポインタは，非同期ジョブが伝送される時に含められます。低水準インタフェースを提供する各オブジェクトは，セッションオブジェクトです。セッションで一度に実行できるジョブは 1 つだけです。これは，非同期ジョブにも適用されます。

16.1 IMCDomain

CopyNC

ファイルを NC または PLC に，またはその逆にコピーするのに使用します。NC からファイルにディレクトリをコピーすることも可能です。この場合，ファイルの内容は NC 上のディレクトリの内容に対応します。構造は，これまで NCDDE から提供された構造と同じです。

表 16.1 CopyNC

パラメータ	タイプ	意味
Source	BSTR	ソースファイルのファイル名
Destination	BSTR	ターゲットファイルのファイル名
Mode	MCDOMAIN_COPY_MODE	

CopyNCAsync

ファイルを NC または PLC に，またはその逆にコピーするのに使用します。NC からファイルにディレクトリをコピーすることも可能です。この場合，ファイルの内容は NC 上のディレクトリの内容に対応します。構造は，これまで NCDDE から提供された構造と同じです。

表 16.2 CopyNCAsync

パラメータ	タイプ	意味
Source	BSTR	ソースファイルのファイル名
Destination	BSTR	ターゲットファイルのファイル名
Mode	MCDOMAIN_COPY_MODE	
Cookie	LONG	ユーザ定義の識別
pCB	IMCDomainAsyncCallback *	コールバックインタフェース
pDisp	Idispatch *	Idispatch コールバックインタフェース
PreqID	[out] long *	サーバが割り当てる ID

MapACC_NC

グローバルユーザデータと，NCK のマシンデータを通知します。

表 16.3 MapACC_NC

パラメータ	タイプ	意味
Source	BSTR	ソースファイルのファイル名
Destination	BSTR	ターゲットファイルのファイル名
wArea	BYTE	NC の範囲識別
wDatablock	BYTE	NC のブロック識別
wTimeOut	long	時間制限
Prefix	BSTR	読み取られるデータ記述の接頭部

MapACC_NCAsync

グローバルユーザデータと，NCK のマシンデータを通知します。

表 16.4 MapACC_NCAsync

パラメータ	タイプ	意味
Source	BSTR	ソースファイルのファイル名
Destination	BSTR	ターゲットファイルのファイル名
wArea	BYTE	NC の範囲識別
wDatablock	BYTE	NC のブロック識別
wTimeOut	long	時間制限
Prefix	BSTR	読み取られるデータ記述の接頭部
Cookie	LONG	ユーザ定義の識別
pCB	IMCDomainAsyncCallback *	コールバックインタフェース
pDisp	Idispatch *	Idispatch コールバックインタフェース
PreqID	[out] long *	サーバが割り当てる ID

Create

NC でファイルまたはディレクトリを作成するのに使用します。NC デフォルト権限を持つ空のファイルまたはディレクトリが作成されます。

中身のある NC ファイルを直接作成するには，適切な CopyNC 呼び出しを使用する必要があります。

表 16.5 Create

パラメータ	タイプ	意味
NewFile	BSTR	新規ファイルのファイル名
Mode	MCDOMAIN_CREATE_MODE	

Open_Pipe

パイプを使用したファイルまたはディレクトリの伝送を開始するのに使用します。転送範囲は，転送されるデータによって作成されます。読み取り時，IMCDomainobject はデータを範囲に書き込み，アプリケーションが範囲からデータを読み取ります。この実行中，IMCDomainobject は，データが再び読み取り用に使用可能になっている限り使用不可です。書き込み中，アプリケーションは範囲を満たします。最後のデータがまだ読み取られていない場合，必要なスペースが使用可能になるまでアプリケーションは使用不可です。一度に許可されるパイプは，IMCDomain インスタンスごとに 1 つだけです。転送は，読み取りまたは書き込みのどちらかです。

表 16.6 Open_Pipe

パラメータ	タイプ	意味
NewFile	BSTR	NC ファイルの名前
Mode	MCDOMAIN_PIPE_MODE	

Read_Pipe

転送バッファからデータを読み取るのに使用します。データは，タイプ VT_ARRAY|VT_UI1 の VARIANT で転送されます。転送されるデータのボリュームは，安全配列パラメータから取得できます。読み取り呼び出しは，十分なデータが読み取られた場合，または転送が取り消された場合にのみ戻されます。転送バッファのサイズは，内部的に定義されます。データの終了時に，長さ 0 のデータバッファが伝送されます。その後，転送を Close_Pipe で完了することが必要です。転送バッファは，呼び出し側によって使用可能にする必要があります。

表 16.7 Read_Pipe

パラメータ	タイプ	意味
pvBuffer	[out] VARIANT *	バッファ

Write_Pipe

転送バッファにデータを書き込むのに使用します。書き込み呼び出しは，データをバッファから転送し，データが転送されたか，または転送が終了した場合にのみ完了します。データの一部だけが書き込まれた場合，書き込まれたバイト数の入ったパラメータ ByteCount を使ってそれを判別できます。

表 16.8 Write_Pipe

パラメータ	タイプ	意味
Buffer	VARIANT	バッファ
ByteCount	[out] long *	書き込まれたバイト数の入った Long 値へのポインタ

Close_Pipe

パイプを介して転送を終了するのに使用します。

この関数にはパラメータはありません。

Delete

NC 上のファイルを削除するのに使用します。

表 16.9 Delete

パラメータ	タイプ	意味
NCFile	BSTR	NC ファイルの名前

Select

NC 上のパートプログラムファイルを選択するのに使用します。

この呼び出しは，NC パスしか受け入れません。

表 16.10 Select

パラメータ	タイプ	意味
NCFile	BSTR	NC ファイルの名前
Channel	long	パートプログラムが選択されるチャネルの番号

External

外部ソースからの処理用のファイルを選択するのに使用します。

表 16.11 External

パラメータ	タイプ	意味
WinFile	BSTR	Windows ファイルシステムのファイル名
Channel	long	パートプログラムが選択されるチャネルの番号

Prot

ファイルのアクセス権限を設定し，現行のアクセス権限を監視するのに使用します。

この呼び出しは，NC パスしか受け入れません。

表 16.12 Prot

パラメータ	タイプ	意味
NCFile	BSTR	NC ファイルの名前
Access	BSTR	アクセス権限

Rename

NC 内のファイルの名前変更または移動のために使用します。ここで許可されるのは NC パスだけです。

表 16.13 Rename

パラメータ	タイプ	意味
NCSource	BSTR	NC ファイルの古い名前
NCDest	BSTR	NC ファイルの新しい名前

Cancel

現在実行中の非同期ジョブを取り消すのに使用します。完了の CompleteEvent はやはり作成されます。ジョブは，RequestID または Cookie のどちらかによって識別されます。転送されない値には，値 0 が割り当てられます。

表 16.14 Cancel

パラメータ	タイプ	意味
Cookie	long	取り消される非同期ジョブの ID
RequID	BSTR	

OpenEditorFile

NC 上のファイルを編集するためオープンする場合に使用します。

表 16.15 OpenEditorFile

パラメータ	タイプ	意味
Source	BSTR	ファイル名
EdiFile	BSTR	

CloseEditorFile

NC 上のエディタファイルをクローズするのに使用します。

表 16.16 CloseEditorFile

パラメータ	タイプ	意味
EdiFile	BSTR	

SeekEditorFile

NC 上でエディタファイルを位置決めし，編集される範囲のサイズを定義するのに使用します。

表 16.17 SeekEditorFile

パラメータ	タイプ	意味
EdiFile	long	
Mode	MCDOMAIN_SEEK_MODE	
Pos	long	
Len	long	
SearchString	BSTR	
SkipCount	long	

16.2 IMCDomainAsyncCallback

最初のコールバックが，開始呼び出しの完了前に行われる場合があるので，RequestID は提供されません。

OnCommandProgress

ADSI オブジェクトは，このメソッドを使って，ジョブの現在の状況をパーセンテージとしてクライアントに通知します。エラー，取り消し，または正常終了の場合，メソッド CompleteStatus が呼び出されます。

表 16.18 OnCommandProgress

パラメータ	タイプ	意味
Cookie	long	取り消される非同期ジョブの ID
Progress	long	

OnCommandComplete

非同期ジョブは完全に実行されます。エラー値は Result パラメータによって戻されます。エラーがない場合，S_OK が通知されます。補足エラー値は，OPC 品質フラグに適用できる定義に従って，Quality を介して戻されます。“Result” に基づいて，呼び出しが成功するかどうかが決定され，エラーの原因に関する補足情報が “Quality” を介して提供されます。6 部「コマンドインタフェース」を参照してください。

表 16.19 OnCommandComplete

パラメータ	タイプ	意味
Cookie	long	取り消される非同期ジョブの ID
Result	long	
Quality	long	

16.3 ドメインインタフェース定義

ドメインインタフェースは，次のものを介して定義されます。

- パス
- 列挙（Enums）

16.3.1 パス

許可されるパスの仕様は，Windows パス，NC パス，および PLC パスです。

パス内では，大文字（上シフト文字）と小型の大文字（Small Capital）との間に区別はありません。

Windows パス

Windows パスは，次のように，一般的な規則に対応します。

C:¥tmp¥test.txt

NC パス

NC パスは次のようなスキームを使用します。

/NC/WKS.DIR/Test.wpd/test.mpf

/NC は，当該ファイルが NC 上にあることをマークするための ID として使用されます。それに続くパスは，NC 上のパスに対応します。表記は，データ管理に使用される表記向けになっています。

PLC パス

PLC パスは次のようなスキームによって定義されます。

/PLC/xxx

/PLC は，当該オブジェクトが PLC 上にあることを定義します。他のすべてのパスの構成要素は，NCDDE に使用される表記に類似しています。

16.3.2 列挙（Enums）

MCDOMAIN_CREATE_MODE

MCDOMAIN_CREATE_NORMAL	0x01

MCDOMAIN_SEEK_MODE

MCDOMAIN_SEEK_BYTE	0x01
MCDOMAIN_SEEK_RECORD	0x02

MCDOMAIN_PIPE_MODE

MCDOMAIN_PIPE_READ	0x01
MCDOMAIN_PIPE_WRITE	0x02
MCDOMAIN_PIPE_BINARY	0x04
MCDOMAIN_PIPE_NC	0x08

MCDOMAIN_COPY_MODE

MCDOMAIN_COPY_BINARY	0x01
MCDOMAIN_COPY_NC	0x02

第 6 部：コマンドインタフェース

NC および PLC のコマンドは，840DI コマンドインタフェースを使って呼び出すことができます。また，別々の 840DI コンポーネント同士で相互に通信するときに，サポートされているネームスペースを操作することができます。

上記のメソッドはすべて，HRESULT タイプの戻り値を戻します。

17 IMCCommand メソッド

■ コマンドの同期実行

表 17.1 Execute

Execute		
パラメータ	タイプ	意味
Command	VARIANT	安全配列 : コマンドとそのパラメータによって異なる
pOutArguments	[out optional] variant *	戻り値（あれば）

■ コマンドの非同期実行

表 17.2 ExecuteAsync

ExecuteAsync		
パラメータ	タイプ	意味
Command	VARIANT	安全配列 : コマンドとそのパラメータによって異なる
Cookie	long	クライアント側でのあいまいでない ID
pCB	IMCCommandAsync-Callback*	オブジェクトのコールバックインタフェースへのポインタ
pDisp	IDispatch *	オブジェクトのディスパッチインタフェースへのポインタ。pCB がゼロポインタの場合にだけ考慮される
pReqId	[out] long*	サーバ側でのあいまいでない ID

注意

サーバがコールバックを送るときに，ExecuteAsync メソッドが完了しているとは限りません。

■ 非同期コマンドの取り消し

表 17.3 Cancel

Cancel		
パラメータ	タイプ	意味
ReqID	long	サーバの端末でのあいまいでない ID コマンド（ExecuteAsync を呼び出すときに，ReqId パラメータに指定する）

18 IMCCommandAsync Callback メソッド

クライアントは，非同期機能呼び出しを送る場合に，このインタフェースメソッドを実装している必要があります。このコールバックインタフェースへの参照は，非同期呼び出しのパラメータに含められます。

サーバは，コマンドの実行を完了したときに，このメソッドを呼び出します。

表 18.1 OnCommandComplete

OnCommandCompleted		
パラメータ	**タイプ**	**意味**
Cookie	long	完了したコマンドの ID（ExecuteAsync の呼び出しに含まれる）
Error	long	詳細なエラーコード
OutArguments	Variant	戻り値（あれば）

19 サポートされているコマンド

ここで定義されているコマンドは，IMCCommand の Command パラメータとして送ることができます（17 章を参照）。

19.1 データサーバのネームスペース用のコマンド（OPC データ）

■ NEW

データサーバに新しい変数を作成します。この変数は，すべてのクライアントですぐに宛先指定することができます。

この新しいパラメータは最初のパラメータとして送られ，2 番目のパラメータは初期設定値になります。

■ LINK

データサーバのネームスペースを新しいデータで拡張します。アドレス情報は，ローカルに指定する必要があります。

このコマンドは，NC カーネルの拡張機能（NCK–OEM）を使えるようにするときにだけ必要になります。

■ FREE

“New” または “Link” を使って作成した変数を，データサーバのネームスペースから削除します。パラメータとしては，削除される変数の名前が送られます。

19.2 NC/PLC でのプログラム実行のためのコマンド

■ PI_START(_BINARY)

PI_START は NC 用のコマンドを生成し，PI_START_BINARY は PLC 用のコマンドを生成します。

■ PI_STOP

PI_START_BINARY で初期設定したプログラム呼び出しを取り消します。最初のパラメータとして端末アドレス（通常は ”/PLC”）が送られ，2 番目のパラメータとして PI 名が送られます。元の呼び出しのパラメータは省略されるので，コマンドに _BINARY の指定はありません。

■ PI_RESUME_BINARY

PI_STOP で取り消したコマンドを継続します。

- 最初のパラメータとして端末アドレス（通常は ”/PLC”）が送られます。
- 元の呼び出しのパラメータはすべて，2 番目のパラメータとして指定します。
- PI 名は 3 番目のパラメータとして送られます。

NC 用の PI 名は _N_ で始まりその後に 6 文字が続きます。PLC 用の名前はこの定義とはわずかに異なっています。

PI サービスを呼び出すためのコマンド配列は，次のような構成になります。

表 19.1 コマンド配列

PI START
サーバ名
範囲またはユニット
パラメータ（名）
:
パラメータ（名）
PI 名

使うことのできるサーバ名は，/NC と /PLC です。

範囲またはユニットとして使うことのできる値は，次のとおりです（xx は 2 桁を意味します)。

001: NCK 固有のサービス

1xx: モードグループ固有のサービス

2xx: チャネル固有のサービス

3xx: 軸固有のサービス

4xx: 工具固有のサービス

5xx: FDD 固有のサービス

6xx: MSD 固有のサービス

K00: COM モジュール関連のサービス（ハードウェア）

P01: 受動ファイルシステム用のサービス

パスとファイルの名前については，次に示すパラメータの説明で識別できます。

xxx = DIR または WPD（つまりディレクトリ）

yyyyy = ファイルまたはパス名の任意のストリング（最大で 25 文字）

zzz = ファイル拡張子の任意のストリング（最大で 3 文字）

■ 例

PI サービス SELECT を使い，パートプログラムを選択する例 :

```
PI_START
/NC
201
MPF_DIR/TEST_MPF
_N_SELECT
```

意味 : NC の第 1 チャネルでメインプログラム TEST.MPF を選択します。

定義したコマンドの概要については，「PI サービス」（PI.HLP）に述べられています。

19.3 その他のコマンド

■ ACTIONLOG

このコマンドには 2 つの役割があります。

- パラメータが送られると，ログファイルの末尾に挿入されます。このストリングには，改行もタブも入れることができません。
- パラメータが送られない場合，コマンドサーバは，これまでに追加されたすべてのデータがハードディスクに書き込まれるようにします。

■ PLC_MEMORY_RESET()

PLC 全体をリセットします。

まず PLC を停止する必要があります。

端末アドレス（通常は ”/PLC”）がパラメータとして送られます。

20 コマンドインタフェースで定義されているエラーソース

コマンドインタフェースには，エラーコードとして，定義済みのエラーコードが備えられています。

以下の表では，個々のエラーソースに含まれるエラーコードがリストされています。

エラーコードは，8 桁の 16 進数の形式で示されます。

2 桁目の値でエラーコードが定義されます。

”?” は，1 桁の 16 進数字を意味します。

20.1 エラーソース 1（サーバ）

次の表にリストされているエラーコードでは，”xx” には，“1” 以外の任意の数値を挿入できます。

表 20.1 エラーコード 1

エラーコード	説明
0x?1??01yy	NC/PLC が yy 秒内に応答しませんでした
0x?1??xx03	変数名が不明です
0x?1??xx06	引数の数値が間違っています
0x?1??xx07	引数を受け入れられません
0x?1??xx08	不明なコマンドです
0x?1??xx0a	変数は使用中です（たとえば，ファイル転送の状態報告のため）
0x?1??xx0b	変数タイプが間違っています（たとえば，ファイル転送の状態報告では，数値変数だけが受け入れられます）
0x?1??xx0c	変数を作成できませんでした
0x?1??xx11	データ型を変換できません
0x?1??xx12	アプリケーションで配布されたデータが破損しています
0x?1??xx13	LINK コマンドのデータ型が未定義です
0x?1??xx14	アプリケーションで配布されたデータが大きすぎます
0x?1??xx19	MPI/BTSS/L2DP/... 以外で改行され，マシンスイッチ中間改行が行われています
0x?1??xx1c	データを宛先タイプに変換できません
0x?1??xx1e	ファイル転送の停止に使われたエラーコードの少なくとも 1 バイトがゼロでした
0x?1??xx1f	サーバが終了します
0x?1??xx21	リソースがコンパイル時間外に構成されました
0x?1??xx22	MPI/BTSS/L2DP/... ドライバをオープンできません（インストールを調べてください）
0x?1??xx22	システムで使用可能なウィンドウまたはメモリが少なくなっています
0x?1??xx28	メモリが不足しています
0x?1??xx29	バッファがオーバフローしました
0x?1??xx2a	不明なコマンドです
0x?1??xx2c	mmc.ini が損傷を受けました
0x?1??xx2f	変数索引はサポートされていません
0x?1??xx30	変数名の構文が間違っています
0x?1??xx31	データの構文が間違っています
0x?1??xx32	LINK コマンド構文が間違っています
0x?1??xx33	ストリング構文が間違っています

表 20.1 エラーコード 1（続き）

エラーコード	説明
0x?1??xx34	アトムテーブルがオーバフローしています
0x?1??xx3c	サーバのバージョンが古すぎるか，ACC ファイル項目をマップできません
0x?1??xx3d	パイプ変数が使用中です
0x?1??xx3e	パイプ変数が使用中です。ファイル転送は終了します
0x?1??xx41	コマンドチャネルが不明です
0x?1??xx42	コマンドチャネルが存在します
0x?1??xx43	サーバへ接続できません
0x?1??xx44	コマンドチャネルが使用中なのですぐに削除できません。後で再試行してください
0x?1??xx45	サーバはエラーコードを受け取りませんでした
0x?1??xx46	アトムを使い切りました
0x?1??xx47	書き込み保護されています
0x?1??xx48	アドレス指定できるマシンが多すぎます
0x?1??xx49	ReadVarCyclic はサポートされていません
0x?1??xx4a	Read はサポートされていません
0x?1??xx4b	Write はサポートされていません
0x?1??xx4c	Execute はサポートされていません
0x?1??xx4d	装置が存在しません
0x?1??xx50	dll をロードできません。インストールを調べてください
0x?1??xx51	MPI/BTSS/L2DP/... ケーブルが抜かれているか，NC/PLC の電源が入っていません
0x?1??xx58	NC/PLC から送られたデータが LINK コマンドと一致しません
0x?1??xx59	NC/PLC から NAN（NotANumber）を受け取りました
0x?1??xx5c	メッセージ /ALARM_S/...ReadVarCyclics が多すぎます
0x?1??xx5e	NC/PLC への通信は現在ロックされています
0x?1??xx60	アクションログがオフに切り替えられました
0x?1??xxfe	アクションログファイルをオープンできません

20.2 エラーソース 2（MPI/BTSS/L2DP/... インタフェース）

一般に，このようなエラーコードはすべて，HMI と NC/PLC との間のレイヤ 4 の通信問題を記述したものです。中でも注目できるのは，以下のエラーコードです。

表 20.2 エラーソース 2

エラーコード	説明
0x?2??0304	リモート端末のリソースが少なくなっています
0x?2??030e	リモート端末が接続をクローズしました
0x?2??0310	リモート端末がトークンバスから消えました（たとえば，リセット，電源オフなどによって）
0x?2??0316	リモート端末が接続の試行を拒否しています
0x?2??031a	アドレスが間違っています。mmc.ini か netnames.ini を調べてください

20.3 エラーソース 3（NC/PLC）

これらのエラーコードは NC/PLC から送達されます。HRE-SULT のエラーコード部分には，レイヤ 7 のエラークラスとエラーコードが含まれています。NC からこれらのコードを受け取った場合，exerror.hh ファイルを使って合理的なエラー分析を行えます。

20.4 エラーコード 4（サーバ）

サーバ OSI のレイヤ 7 エラー（プロトコルの競合）の場合，サーバは自動的に回復します。

20.5 エラーコード 5（NC/PLC）

これらのエラーコードは NC/PLC によって送達され, ’ アクセスエラー’ と呼ばれます。

表 20.3 エラーソース 5

エラーコード	説明
0x?5??0901	ハードウェアエラーです
0x?5??0903	アクセスが許可されません
0x?5??0905	アドレスが間違っています
0x?5??0906	データ型がサポートされていません
0x?5??0907	データ型が矛盾しています
0x?5??090a	存在しません

20.6 エラーソース 8（サーバ）

これらのエラーコードは，ファイルを扱うときに，CRT ライブラリによって送達されたものです。最後の 16 ビット部分は，CRT のエラー番号コードに対応しています。

20.7 エラーソース 9（サーバ）

これらのエラーコードは，DDEML ライブラリによって予期せず送達されたものです。

20.7 エラーソース 9（サーバ）

第 7 部 : 例題

21 プログラム例

■ この章の目的

この章では，以下の言語で OPC サーバのクライアントアプリケーションをプログラムする方法について説明します。

- Visual Basic
- Visual C++

2 つのプログラム例を使用しています。

21.1 Visual Basic での OPC クライアント

■ 前提条件

以下のクライアントアプリケーション例をプログラムするには，Microsoft 社の Visual Basic 6.0 と Developer Studio 6.0 が必要です。

このクライアントアプリケーションは，Microsoft Visual Basic を使って作成します。この簡単なアプリケーションは，OLE オートメーションオブジェクトにアクセスして，ボタンなどのコントロールエレメントを使ったダイアログボックスを作成するだけでなく，そのコントロールエレメントを使用して値を表示します。

21.1.1 プログラムの説明

OPC Client では，以下の作業を実行します。

ダイアログボックスには 2 つのボタンがあります。最初のボタンは，OPC クライアントを OPC サーバに接続します。プログラムを書く際に，接続するサーバをどれにするか決める必要があります。2 番目のボタンは，OPC クライアントをサーバから切断します。

サーバへの接続を確立すると，サーバ内の変数が読み取られて表示されます。

後でプログラムをアップグレードする際，サーバにあるデータが変更されるとそのデータを更新するように計画しておきます。

この場合，ユーザインタフェースは以下のようになります。

図 21.1 OPC クライアントの例

21.1.2 ユーザインタフェースの作成

初めて OPC クライアントをテストするときには，以下のものを備えたユーザインタフェースをまず作成します。

- 2 つのボタン，および
- 表示フィールド

それから，次のようなプログラムを作成します。

- サーバに接続する
- サーバから切断する
- 変数を表示する

■ ユーザインタフェース作成のための操作順序

(1) Microsoft Visual Basic を始動します。

(2) 始動した後すぐに，Visual Basic は新規プロジェクトを作成するかどうか尋ねてきます。このダイアログが表示されない場合には，［ファイル］メニューで［新しいプロジェクト］メニューオプションを選択します。

(3) このフィールドで，“ 標準 EXE” をクリックし，［開く］で確認します。

(4) ボタンと表示フィールドを作成するには，Microsoft Visual Basic のツールボックスが必要です。これが表示されない場合には，［表示］メニューから［ツールバー］メニュー項目を選択します。

(5) ツールボックスのツールバーで，［Command Button］をクリックし，描画プログラムで長方形を描く要領で，ダイアログフォームにボタンを描きます。

(6) 2 番目のボタンを作成するため，ステップ 5 を繰り返します。

(7) 表示フィールドを描画するには，ツールボックスから［Label］コントロールエレメントを選択します。ボタンを描画したときと同じようにして，表示フィールドを描画します。

(8) ここで，特定のボックスおよび表示フィールドのプロパティを決定する必要があります。最初のボタン上でマウスの右ボタンをクリックし，コンテキストメニューの中から［プロパティ］オプションを選択します。すべてのボタンのプロパティを表示するダイアログボックスが開きます。このダイアログが開かない場合には，すでにプロパティダイアログは表示されています。

(9) （オブジェクト名）の右側にあるフィールドに，ボタンの名前を入力します。このボタンには，“ConnectButton” という名前を付けます。

(10) Caption の右側のフィールドに，ボタンの上に表示するテキスト（この場合は “Connect”）を入力します。

(11) ここで，2 番目のボタンをクリックします。このボタンのプロパティが，すぐにプロパティダイアログに表示されます。このボタンの名前は “DisconnectButton” とし，“Disconnect” と表示する必要があります。

Visual Basic でユーザインタフェースを作成した状態を，以下の図で示します。

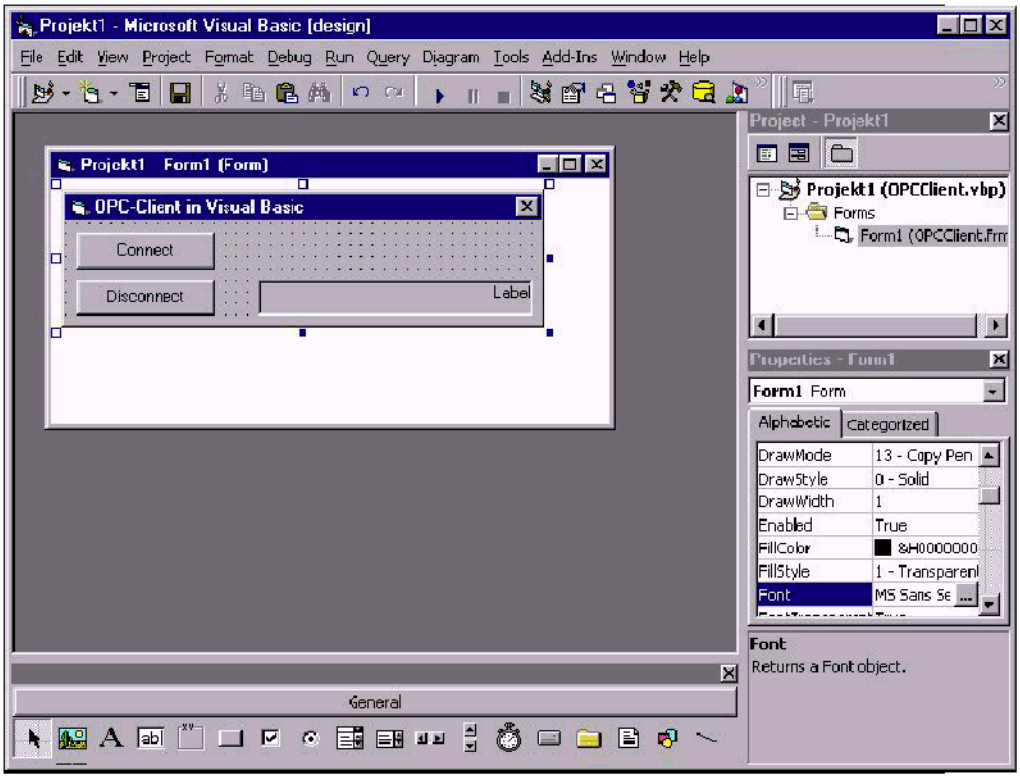


図 21.2 Visual Basic によるヒューマンマシンインタフェースの作成

(12) 表示フィールドのプロパティを定義するには，その表示フィールドをクリックします。このプロパティの名前は “Output” です。より見栄えの良いものにするため，テキストの位置を左寄せにして，テキストサイズを増やすことができます。

21.1.3 プログラムの作成

コントロールエレメントを作成し終えたら，プログラムを実装することができます。このプログラムは，以下の事柄を行います。

- OPC サーバオブジェクトを作成する
- OPC サーバに接続する
- OPC グループを生成し，OPC 項目を追加する。この項目から値が要求され，ダイアログボックスに表示されます。

■ 準備としてのステップ

(1) プログラミングを始める前に，Visual Basic で［コード］ウィンドウを表示する必要があります。Visual Basic で，［表示］メニューから［コード］メニュー項目を選択します。［コード］ウィンドウが表示されます。このエディタでプログラミングをすべて行います。

(2) クライアントプログラムを記述するには，その前にいくつかの準備作業を行う必要があります。

Visual Basic で OPC に属するオブジェクトを使用するには，Visual Basic にそのオブジェクトを認識させる必要があります。こうするには，［プロジェクト］メニューから［参照設定］メニュー項目を選択します。

ダイアログウィンドウが表示されます。リストで “Siemens OPCDAAutomation 2.0” への使用可能な参照を見つけます。

注

この参照が使用できない場合は，［参照］ボタンをクリックします。

こうすると，希望する参照が入ったライブラリ（DLL）を選択できます。

“Siemens OPC DAAutomation 2.0” は，“SOPCDAAuto.dll” ライブラリに入っています。Windows のシステムディレクトリのファイルダイアログでこの DLL を検索し，それを開きます。

これで，“Siemens OPC DAAutomation 2.0” の参照はこのリストに入っているはずです。

---

(3) この項目をマークする必要があります。［OK］ボタンをクリックして，ダイアログを閉じ，変更内容を適用します。

注

Visual Basic が必要な OPC オブジェクトを ’ 認識している ’ かどうかテストするには，Visual Basic でオブジェクトブラウザを開き，ツールバーにある該当するボタンをクリックするか，または［表示］メニューから［オブジェクトブラウザ］メニュー項目を選択します。

オブジェクトブラウザを使用することにより，Visual Basic が処理できるオブジェクトを判別することができます。

“OPCServer”，“OPCGroup”，および “OPCItem” オブジェクトを検索するために，オブ

ジェクトブラウザを使用してください。ここで説明されている準備作業が終了したら，これらのオブジェクトはここに入っているはずです。

オブジェクトが見つからない場合は，［プロジェクト］->［参照設定］メニュー項目で “Siemens OPC DAAutomation 2.0” がチェックされているかどうか，もう一度確かめてください。

■ 重要なオブジェクト

OPC サーバにアクセスするため，最初のクライアントアプリケーションで必要なのは少数のオブジェクトだけです。必要なオブジェクトは，以下のとおりです。

－ OPCServer

このオブジェクトは，実際の OPC サーバへの接続を表示します。

－ OPCGroup

OPC サーバでは，変数は常にグループ内でコンパイルおよび処理されます。クライアントは，サーバにこの OPC グループを作成します。

－ OPCItem

いわゆる項目は，OPC サーバにある変数またはデータソースで構成されます。クライアントは，この項目の値を読み取って表示します。

これらのオブジェクトを宣言するには，ソースコードエディタから左上のコンボボックスの［General］オブジェクトを選択します。Declaration メソッドが，直ちに右側に表示されます。Visual Basic では，“Dim” ステートメントでオブジェクトの宣言を実行します。

(4) これで，以下の構文を使用して，オブジェクトを宣言することができます。

```
Dim WithEvents  Server  As OPCServer
Dim WithEvents  Group   As OPCGroup
Dim             Item    As OPCItem
```

図 21.3 ソースコードの構文

注

オブジェクトサーバおよびグループは，WithEvents パラメータを使って宣言されます。

これにより，オブジェクトは Visual Basic オブジェクトとみなされ，標準のコントロールエレメント（たとえば，ボタン）と同じ方法でイベントに対して反応するようになります。

クライアントは，サーバおよびグループオブジェクトの両方に応答する位置に置くこともできます。

たとえば，グループ項目の値が変化すると，OPC グループはイベントに応答します。

新しいバージョンでは，クライアントアプリケーションはこれらのイベントを処理でき，表示される値は更新されるはずです。このメカニズムについては，後で詳しく説明します。

■ OPCServer オブジェクトの作成と接続

インストールされた OPC サーバに接続するには，まず OPCServer オブジェクトを作成する必要があります。

(5) ダイアログボックスの［Connect］ボタンをクリックすると，OPCServer オブジェクトが作成されます。

左上のコンボボックスの［Editor］ウィンドウから，オブジェクトとして ConnectButton ボタンを選択します。右側のコンボボックスで，イベントとして click が即時に選択されます。

“ConnectButton_Click()” メソッドがソースコードに自動的に追加されます。このメソッドは，［Connect］ボタンを押すと必ず呼び出されます。

```
'Handling routine for Connect button clicked
Private Sub ConnectButton_Click()
```

図 21.4 ソースコードの構文

注

これで，“ConnectButton_Click()” 処理ルーチンに以下のステートメントがあるはずです。

(6) Visual Basic では，オブジェクトは［New］コマンドで作成します。OPCServer オブジェクトは，以下のステートメントを作成します。

```
'Create New OPCServer Object
Set Server = New OPCServer
```

図 21.5 ソースコードの構文

OPC Automation Wrapper 内の OPCServer オートメーションオブジェクトは，［New］コマンドで作成されます。しかし，クライアントを接続するサーバで認識されていないため，実際に始動する OPC サーバはありません。

(7) OPC Automation Wrapper 内の OPC サーバに OPCServer オブジェクトを接続するため，OPCServer オブジェクトには，“Connect” メソッドが用意されています。その構文は，以下のとおりです。

OPCServer.Connect(ProgID As String, Optional Node As Variant)

このメソッドは，OPC サーバを始動し，OPC Automation Wrapper 内の OPCServer オブジェクトにそれを接続します。

いわゆる，プログラム識別子（”ProgID”）によって，始動すべきサーバが識別されます。ProgID は，このメソッドに文字列として転送されます。各 ProgID は，OLE オブジェクトのクラスに明確に割り当てられます。

この例の OPC サーバの ProgID は，“OPC.SINUMERIK.Machine-switch“ です。

注

“IOPC ServerList” インタフェース（“OPC COMMON INTERFACE” の一部）を照会すると，インストールされている OPC サーバを実行時に確認することができます。

ProgID に加え，“Node” の名前が付いた別の任意指定パラメータを Connect メソッドに転送することができます。OPC サーバが別のコンピュータのクライアントアプリケーションであれば，コンピュータ名か IP アドレスを転送することができます。

OPCServer オブジェクトを OPC サーバに接続するには，以下のステートメントをソースコードに指定します。

```
'Connecting server object with OPC server
Server.Connect( "OPC.840DI.Machineswitch" )
```

図 21.6 ソースコードの構文

このステートメントを実行し終えると，OPC サーバが始動します。

■ OPC グループオブジェクトの作成と初期化

(8) すでに述べたように，OPC サーバでは変数はグループとして処理されます。

OPC サーバへの接続が正常に確立されると，OPC サーバにグループを作成することができます。

この目的で，OPCServer には “OPCGroup” というコレクションがあります。

クライアントが作成する OPCGroup オブジェクトによって表されているすべてのグループは，クライアントが OPC サーバに接続されている限り，このコレクションに格納されます。

OPCGroup オブジェクトには，グループを作成するための AddGroup メソッドが用意されています。このメソッドの構文は，以下のとおりです。

OPCGroups.Add(Optional Name As Variant) As OPCGroup

このメソッドを使用して，グループ名は，グループを作成している OPC サーバに任意に送られます。

この名前は，クライアントごとに固有でなければなりません。

単一のクライアントが，同じ名前の 2 つのグループを作成することはできません。

名前が転送されない場合，OPC サーバは自動的に固有の名前でグループを作成します。

この例では，“FirstGroup” という名前のグループが作成されます。

メソッドは，ユーザが保管する新しく作成したグループに対する参照を戻します。

グループを作成するためにここで必要なのは，ソースコードに以下の命令を入力することだけです。

```
'Create an indepedent group in the server
Set Group = Server.OPCGroups.Add( "FirstGroup" )
```

図 21.7 ソースコードの構文

(9) グループは，項目の値が変化するとイベントプロシージャを実行します。

これにはグループへの “ 接続 ” が必要です。OPCGroup オブジェクトの “IsSubscribed” プロパティを “True” 値に設定して，変数の値が変化すると OPC サーバがグループに通知するようにする必要があります。

さらに，OPC サーバの最大間隔を定義して，変化が生じたら合図を出すように設定できます。“UpdateRate” プロパティを，このために使用します。SINUMERIK では，100 ミリ秒の更新率だけがサポートされています。

更新速度は，ミリ秒で入力します。この例では，変数の値が変化した場合，グループは最大 100 ミリ秒ごとにイベントを受信します。

関係するステートメントを以下に示します。

```
'Group signals when items change
Group.IsSubscribed = True
'The group update rate is 100
Group.UpdateRate = 100
```

図 21.8 ソースコードの構文

■ グループへの項目の追加

これで，クライアントは OPCGroup オブジェクトを所有し，プロセス変数への接続で構成される OPCItem オブジェクトを追加することができます。

(10) このため，各 OPCGroup オブジェクトには，“OPCItems” という名前のコレクションが含まれます。

このコレクションは，クライアントが作成した OPCItem オブジェクトによって表されているすべての項目を，OPC サーバに接続されている限り格納します。

項目を追加するため，OPCItem オブジェクトには AddItem メソッドが用意されています。その構文は以下のとおりです。

OPCItems.AddItem (ItemID As String, ClientHandle As Long) As OPCItem

このメソッドに転送される最初のパラメータは ItemID です。

ItemID は，サーバ上の項目に固有の識別子です。

サーバはその項目を階層構造で配置できるため，ItemID は多くの場合，ファイルシステムと同じようなタイプのパスで構成されます。

この例では，“/bag/state/opmode” 項目が示されます。

いわゆるクライアントハンドルが，転送される 2 番目のパラメータです。

グループと項目には，常に 2 つの異なるハンドルがあります。

- サーバハンドル
- クライアントハンドル

サーバハンドルは，グループをサーバに接続します。クライアントハンドルは，グループをクライアントに接続します。

注

たとえば，ビジネスレターで，Client_Handle を “Your Ref.” にすると，Server_Handle は “Our Ref.” になります。

クライアントハンドルは，固有なものにすべきです（必須条件ではない）。クライアントハンドルは，イベントが生じると，サーバからクライアントに送達されます。

このクライアントハンドルだけで，クライアントはどの項目が変化したかを識別することができます。

この例で追加された項目は，値が 21071977（ランダム値）のハンドルを受け取ります。

AddItem メソッドには, 新しく追加した OPCItem オブジェクトの格納用の参照が用意されています。

“FirstGroup” グループに項目を追加するには，以下のステートメントを使用します。

```
'Add an item to the new OPC group
Set Item = Group.CItems.AddItem( ”/bag/state/opmode”,
21071977
```

図 21.9 ソースコードの構文

■ 項目の値の読み取り

項目の現行値を読み取るには，Automation インタフェースを使って，OPCItem オブジェクトのプロパティ値にアクセスすれば十分です。

(11) この例では，ダイアログボックス内の項目の現行値は出力表示フィールドに表示されます。

表示フィールドに表示されるテキストは，表示フィールドの Caption プロパティによって決まります。

OPCitem の値を表示するには, 表示フィールドの Caption プロパティの OPCItem オブジェクトのプロパティ値を転送する必要があります。

そのためには，以下のステートメントが必要です。

```
'Read and display the value of this item
Output.Caption = Item.Value
```

図 21.10 ソースコードの構文

Automation インタフェースを介して，サーバに変数を読み込むために必要なステートメントはこれだけです。

■ グループでの項目の削除

クライアントアプリケーションを閉じるには，その前にいくらかのリソースを解放する必要があります。こうしないと，OPC サーバは閉じません。

これを避けるには，プライベートグループからすべての項目を削除し，OPC サーバを閉じる前にプライベートグループをそれぞれ削除する必要があります。

このクライアントアプリケーションは，ユーザが［Disconnect］ボタンをクリックする前に，リソースをすべて解放する必要があります。

(12) こうするには，コンボボックスの左上隅にある Visual Basic コードエディタから，DisconnectButton オブジェクトを選択します。その後，右側のコンボボックスで Click イベントを選択します。

DisconnectButton_Click() サブルーチンが，VisualBasic によって自動的に追加されます。この関数は，ユーザが［Disconnect］ボタンをクリックすると呼び出されます。

```
'Handling routine for Disconnect button clicked
Private Sub DisconnectButton_Click()
```

図 21.11 ソースコードの構文

注

この後のすべてのステートメントで，この関数を記述する必要があります。

(13) オブジェクトを削除するときは，作成する場合とは逆の順序に従う必要があります。

それで，まず最初に，“FirstGroup” グループから /bag/state/opmode 項目を削除します。

OPCGroup オブジェクトにある OPCItem のコレクションには, “Remove” メソッドが用意されています。その構文は，以下のとおりです。

OPCItems.Remove (Count As Long, ServerHandles() As Long, ByRef Er-rors()As Long)

このメソッドを使用すると，複数の OPCItem オブジェクトを一度に削除することができます。

OPC 数量操作は，作業をより効率的にするためにさまざまな処理に適用できます。また，同じコマンドを使って，それぞれの OPCItem オブジェクトを順番に削除していくのは，非常に骨の折れることでしょう。

このメソッドには，これから説明するように，3 つのパラメータが必要です。

カウントパラメータによって，このメソッドを呼び出す際に削除する OPCItem オブジェクトの数が決まります。この例では，グループに含まれる OPCItem オブジェクトは 1 つだけです。それで，転送されるパラメータは “1” です。

2 番目のパラメータは，Long の配列として転送されます。この配列のそれぞれの値は，削除する OPCitem のサーバハンドルです。

3 番目のパラメータは, Long の 2 番目の配列として転送されます。OPCItem を削除できない場合，この配列にエラーが保管されます。

この例では，1 つの OPCItem だけを削除するには，以下のステートメントが必要です。

```
Dim ServerHandles(1)            As Long
Dim Errors()                    As long
'Provide server handle of our OPC item
ServerHandles(1) = Item.ServerHandle
'Delete OPC_Item from our group
Group.OPCItems.Remove 1, ServerHandles(), Errors()
Set Item = Nothing
```

図 21.12 ソースコードの構文

最初のコマンドを使用して，1 つの Long の配列を宣言します。これは，1 つの OPCItem オブジェクトのサーバハンドルをちょうど格納できるスペースです。

2 番目のコマンドは，Long から大きなフィールド（未決定）を宣言します。このフィールドに，メソッドのエラーコードを格納します。

3 番目のステートメントを使用して，1 つの OPCItem オブジェクトのサーバハンドルの値を，［Server Handle］フィールドの最初の要素に割り当てます。

これで，OPCItem コレクションの Remove メソッドを呼び出し，両方の配列を転送することができます。

このコマンドの実行後に，/bag/state/opmode OPCItem オブジェクトが “FirstGroup” OPC グループから削除されます。

注

操作が正常に行われたかどうかを確認するため，戻されるエラーコードをさらに要求することができます。

エラー配列の最初の要素には，このエラーコードが入ります。値が 0 であれば，操作は正常に行われています。0 でなければ，エラーが生じています。

OPC Foundation の “Data Access AutomationInterface Standard, Version 2.02” という資料の第 5 章，“OPC Data Access Automation Definitions and Symbols” を参照してください。

■ OPC グループオブジェクトの削除

(14) ここで，FirstGroup OPC グループをサーバから削除する必要があります。

このことを行うため，OPCGroup には “RemoveAll” メソッドが準備されています。

このメソッドを使用すると，1 度呼び出すだけですべての OPCGroup オブジェクトを削除できます。

“FirstGroup” グループを削除するには，以下のコマンドが必要です。

```
'Delete your own group in the OPC server
Server.OPCGroups RemoveAll
Set Group = Nothing
```

図 21.13 ソースコードの構文

これで，サーバでの消去作業は終了です。

■ サーバの切断と OPCServer オブジェクトの削除

(15) 最後のステップとして，OPC サーバから OPCServer オブジェクトを切断します。

サーバで作成されたすべてのデータ（グループや項目など）がすべて削除されていれば，切断した後，サーバは OLE 機能によって自動的に閉じられます。

OPCServer オブジェクトには，サーバから切断するための “Disconnect” メソッドが用意されています。

“DisconnectButton_Click()” 関数で，以下の呼び出しを使用します。

```
'Disconnect the OPCServer object from the OPC sever
Server.Disconnect
Set Server = Nothing
```

図 21.14 ソースコードの構文

これで，OPC クライアントのプログラミングを終了します。

ここで，学んだのは以下の事柄です。

- Automation インタフェースを介して OPC サーバに接続する方法 _ サーバにグループと OPNItem を追加する方法
- OPC サーバから値を読み取る方法

さらに，以下の点を取り上げました。

- OPC クライアントを閉じる前に必要とされる消去作業について
- OPC サーバから切断する方法について

21.1.4 ユーザインタフェースとプログラムのテスト

ここまでで，プログラムをテストできるようになりました。

テストを始める前に，ご使用の OPC サーバの “OPC.SINUMERIK.Machineswitch” ProgID が正確であることを確認してください。

このクライアントアプリケーションを機能させるためには，OPC サーバが正しくインストールされている必要があります。

さらに，ポーリングする “/bag/state/opmode” 項目が，実際にサーバ上に存在することも確認してください。

(1) ［実行］メニューから［開始］メニュー項目を選び，プログラムを開始します。
(2) プログラムが開始され，ダイアログボックスが表示されます。［Connect］ボタンをクリックします。

    サーバが起動するまで，1，2 分かかります。

    プログラムが正常に実行されると，ダイアログボックスに項目の値が表示されます。

    プログラムが正常に実行されない場合，以下の点を調べてください。

    － OPC サーバの ProgID が正確かどうか。
    － サーバがコンピュータにインストールされているかどうか。これをテストするには，Windows のレジストリでサーバの ProgID を探します。
    － 要求した変数がサーバにあるかどうか。
    － Visual Basic が “OPC Automation 2.0” および “OPC-DA2.0 Type Library” ライブラリを参照しているかどうか。
(3) クライアントプログラムを終了するには，［Disconnect］ボタンをクリックし，ダイアログボックスを終了します。ついで，OPC サーバがクローズします。

21.1.5 プログラムの拡張

クライアントプログラムを，OPC 項目の値が変化するときにその表示が自動的に更新されるように拡張します（2.2.1「接続可能なオブジェクトおよびイベントインタフェース」も参照）。

■ グループ内での値の変化に対する応答

ご使用のクライアントアプリケーションは，値の変化の際に，OPC 項目の値の表示が自動的に更新されます。

項目の値が変化すると，その項目が見つかった OPC サーバのグループは，“DataChange” イベントを起動します。

このイベントに応答するには，OPC グループがイベントを生成したことと，これらのイベントを処理したいことを Visual Basic に通知しなければなりません。

この通知は，OPCGroup オブジェクトを宣言したときに，“WithEvents” キーワードを指定することによってすでに行っています。

そのため，OPCGroup オブジェクトは Visual Basic オブジェクトであるとみなされ，コントロールエレメントと同じ方法でイベントを生成することができます。

(4) このイベントの処理ルーチンをクライアントへ追加するには，Visual Basic ソースコードエディタの左上にあるコンボボックスから［Group］オブジェクトを選びます。

イベントは，すぐに右側のコンボボックスに “DataChange” として表示されます。

“Group_DataChange()” 処理ルーチンが, Visual Basic からソースコードへ自動的に挿入されます。

```
'Handling routine for DataChange event
Private Sub Group_DataChange(ByVal NumItems As Long,
ClientHandles() As Long,ItemValues() As Variant,Qua-
lities()As Long,TimeStamps() As Date)
```

図 21.15 ソースコードの構文

注

表示を更新するためのステートメントがこの関数に実装されました。

“DataChange” イベントには，以下のパラメータがあります。

－ “NumItems”

値または状況が変化した項目の番号

－ “ClientHandles”

値または状況が変化した OPCItem オブジェクトのクライアントハンドルのフィールド

－ “ItemValues”

OPCItem オブジェクトの新しい値

－ “Qualities”

状況が変化した各 OPCItem オブジェクトの品質値のフィールド

－ “TimeStamps”

変化した各 OPCItem オブジェクトの新しいタイムスタンプのフィールド

この関数で，どのように値が評価されるのでしょうか。

“NumItems” では，変化した OPCItem オブジェクトの数が分かります。

関数は，表示を更新する OPCItem について，Client Handle フィールドを調べます。

クライアントハンドルが 21071977 である，追加した OPCItem オブジェクトを再呼び出しします。

OPC 項目が見つかる場合（グループに OPCItem オブジェクトを 1 つしか追加していない場合などに生じる），この IPCItem オブジェクトの新しい値として，ItemValues フィールドの値が使われ，ダイアログボックスの表示フィールドに表示されます。

これで更新は終わりです。

“Group_DataChange()” 関数では，以下のソースコードが必要になります。

```
'All client handles search for the Client Handle of our
'OPC Item
'If available, display current value
For i = 1 To NumItems
 If ClientHandles(i) = Item.ClientHandle Then
     Output.Caption = ItemValues(i)
 End If
Next
```

図 21.16 ソースコードの構文

1 から変更した項目の番号までの FOR ループがあるため，変数 “i” が使われます。

このプロセスでは，次の行にある “i” ごとに，i 番目の ClientHandle が，OPC 項目の ClientHandle に合致するかどうか調べられます。これが当てはまる場合，i 番目の項目値が，ダイアログボックスの［Output display］フィールドに指定されます。

自動的な更新を作成する簡単な OPC クライアントには，これ以上のステートメントは必要ありません。

(5) ［実行］メニューから［開始］を選び，プログラムを再始動します。
(6) ［Connect］ボタンをクリックした後は，OPC 項目の値が変化したら必ず更新されるはずです。

注

値が変化しない場合，OPC 項目が定数になっていないことを確認してください。

おそらく，OPC サーバに接続されたマシンで値が変化するように操作する必要があります（たとえば，工作機械の軸を送るなど）。

この OPC の完全なソースコード（コメント付き）が，付録に載せられています。

21.2 Visual C++ および MFC を使った OPC クライアント

■ 前提条件

次に示されているクライアントアプリケーションの例をプログラムするには，ご使用の PC で以下のハードウェアとソフトウェアが必要です。

- Visual C++ 6.0 が付属した Microsoft 社の Developer Studio 6.0
- OPC サーバがインストールされていること
- NC に接続された MPI カード

注

よりよく理解するために，付録のソースを読みながらこの例をプログラムされるようお勧めします。

21.2.1 一般情報

前の節では，VisualBasic を使って OPC クライアントを作成する方法や，OPC サーバの Automation インタフェースについて説明しました。

この節では，OPC サーバの Custom インタフェースへ直接にアクセスするクライアントアプリケーションを，Visual C++ を使って作成する方法を説明します。

そのために，この節では，Automation インタフェースと Custom インタフェースの違いについて扱います。

Visual C++ を使ってクライアントアプリケーションを作成するまでにはさらに労力を要するので，Visual C++ と COM プログラムに十分に精通する必要があります。

また，Microsoft Foundation Classes（MFC）を扱った経験があればなおさら結構です。

■ Visual Basic との違い

Visual Basic でのプログラミングと C++ でのプログラミングには，次のような違いがあります。

- Visual Basic は，OLE の自動化機能を使って OPC サーバとやり取りしますが，Visual C++ で作成されたクライアントは，OPC サーバの Custom インタフェースへ直接にアクセスします。
- Visual Basic を使って OPC サーバの Automation インタフェースへアクセスする場合，クライアントには，プロパティとメソッドを伴う自動化オブジェクトが提供されます。そのため，ユーザは，メソッドを呼び出さずに，オブジェクト（たとえば，OPCitem）のプロパティを直接に変更したり読み込んだりすることができます。
- Visual C++ では，状況が全く異なっています。

  OPC サーバを表示するコンポーネントはいろいろあります（たとえば，OPC サーバオブジェクト）。

  これらのコンポーネントにはインタフェースがあります。

コンポーネントを作成したら，クライアントは，そのコンポーネントに必要なインタフェースを要求します。

各インタフェースには，特定数のメソッドが含まれており，クライアントはこれを利用して，コンポーネントにアクセスしたり，そのコンポーネントのプロパティを変更することができます。

コンポーネントのすべてのインタフェースが解放されたら，そのコンポーネントは自動的に削除されます。

- Visual Basic では，OPC 項目にアクセスし，オブジェクトのプロパティを直接に読み取ることができます。

  しかし，OPC サーバの OPC 項目コンポーネントのインタフェースは 1 つではありません。

  Visual C++ で項目にアクセスするには，IOPCItemMgt という OPCGroup のコンポーネントのインタフェースが必要です。

  このインタフェースを使うと，サーバの項目オブジェクトを管理しモニタすることができます。

- もう一つの違いは，Visual Basic はプログラマをいくつかの作業から解放してくれます。

  Visual Basic のプログラマは，COM システムの初期設定を心配する必要がないので，イベント処理はずっと簡単になります。

  しかし Visual C++ では，2，3 の初期設定ジョブを実行する必要があります。

  さらに，プログラマは割り振ったメモリの解放に注意している必要もあります。

  したがって，C++ でのプログラミングははるかに大変ですが，サーバへの接続のコントロールがほとんど明らかにされていない Visual Basic と比べ，サーバへの接続を非常に細かくコントロールすることができるという利点もあります。

21.2.2 プログラムの説明

OPC Client では，以下の作業を実行します。

- ダイアログボックスには 2 つのボタンがあります。
  - 最初のボタンは，OPC クライアントを OPC サーバに接続するものです。プログラムを作成するときに，サーバを決める必要があります。
  - 2 番目のボタンは，OPC クライアントをサーバから切断するときに使います。
- 変数を表示するテキストフィールドがあります。

サーバへの接続を確立すると，サーバ内の変数が読み取られて表示されます。

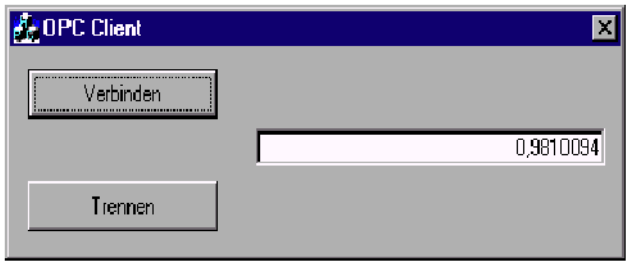


図 21.17 OPC クライアントアプリケーション

21.2.3 準備作業

OPC クライアントでプログラミングを始めるには，以下のようにします。

(1) Visual C++ を始動します。

(2) ［ファイル］メニューから［新規作成］メニュー項目を選び，新しいワークスペースを作ります。

(3) プロジェクトタイプとして，［MFC AppWizard（exe)］を選びます。プロジェクトには，任意の名前を付けられます（たとえば，“OPC Client”)。［OK］をクリックします。

(4) ［MFC AppWizard］の最初の画面で，アプリケーションタイプとして［ダイアログベース］アプリケーションを選びます。

ステップ 2 で，［Dialog］フィールドの［バージョン情報］チェックボックスと［ActiveX コントロール］のチェックを外します。どちらもこのアプリケーションでは使いません。

3 番目の画面では，ステップ 4 のときと同様に，すべてデフォルトのままにしておくことができます。

［Finish］をクリックします。

(5) MFC アプリケーションのウィザードが表示され，プロジェクトについての情報が示されます。了解するときには［OK］をクリックします。

これでプロジェクトが生成されます。

注

生成される OPC Client プロジェクトは，それぞれのプロジェクトディレクトリに作成されます。すでにたくさんのファイルが作成されています。最も重要なファイルは次のとおりです。

- OPC Client.h，OPC Client.cpp

  どちらのファイルにも，OPCClientApp クラスが定義されています。

  このクラスは，ダイアログボックスの背後で実行するアプリケーションを構成します。

  このようなダイアログベースのアプリケーションでは，このクラスには，重要な初期設定を計画することとダイアログを開始する役割があります。

- OPC ClientDlg.h，OPC ClientDlg.cpp

どちらのファイルも OPCClientDlg クラスを定義しています。

このクラスは，ユーザが見ることのできるダイアログボックス，すなわち操作ウィンドウを構成します。

このクラスには，リソースが組み込まれます。

ダイアログボックスは，ワークスペースの［Resource］ビューの［Dialog］メニュー項目の背後に設計されます。

ダイアログのリソースには，IDD_OPCCLIENT_DIALOG という ID があります。

最も重要なプログラミング作業は，このクラスで行われます。

(6) “opcda.h” および “opcda_i.c” ファイルを，C:\Siemens\HMI–Programming Package\Source Files\OPC_DATAACCESS”（OPC サーバを最初にインストールしたときに作成される）ディレクトリから，各自のプロジェクトディレクトリにコピーします。

ここには，OPC データアクセスサーバのインタフェース仕様があります。クライアントは，このインタフェースを認識していれば，このサーバだけにアクセスできます。

(7) ［File］登録タブ（左側のウィンドウの下）を押して，［File View］に移動します。

自分のプロジェクトでこれらのインタフェース仕様を活動化するには，次の組み込みステートメントを “StdAfx.cpp” ファイルに挿入してください。

```
#include "opcda_i.c"\par
```

図 21.18 “StdAfx.cpp” ソースコードの構文

さらに，“StdAfx.h” ファイルに次のステートメントを入力してください。

```
#include "opcda.h"\par
```

図 21.19 “StdAfx.h” ヘッダファイルの構文

(8) このクライアントには，次の 3 つのインタフェースが必要です。

－ IOPCServer

これは，OPC サーバで最も重要なインタフェースです。このインタフェースを使うと，データはサーバ内でソートされてグループ化され，状況情報が要求され，エラーメッセージが処理されます。

－ IOPCItemMgt

OPC グループでこのインタフェースを使うと，グループ内の項目オブジェクトが処理されます。

－ IOPCSyncIO

このインタフェースでは，1 つ以上の項目に対して，読み書き操作を同時に実行することができます。このような操作をすると，操作が完了するまで，アプリケーションがロックされます。

(9) このプログラムは Microsoft 社の DCOM テクノロジを使うため，コマンドおよび定数が完全に認識されるように，プリプロセッサ定義を行う必要があります。

［プロジェクト］メニューから，［設定］メニュー項目を選びます。

左のウィンドウにあるダイアログボックスに示されているプロジェクト名をク

リックし，リストフィールドで “Win32 Debug” を選びます。

次に［C/C++］索引タブをクリックします。

［カテゴリ］リストフィールドが［一般］であることを確認します。［プリプロセッサの定義］の値（“_WIN32_DCOM”）を入力フィールドに入力します。

左上のウィンドウにある［Win32 Release］のプロパティを選び，［Release Project Settings］でこの作業を繰り返します。

(10)［OK］をクリックして確認します。

これで OPC クライアントを作成する準備作業が完了しました。

次のステップでは，クライアントのユーザインタフェースを作成します。

21.2.4 ユーザインタフェースの作成

このクライアントのユーザインタフェースは，Visual Basic クライアントのユーザインタフェースに似たものにします。Visual C++ では，リソースエディタでのインタフェースの作成がサポートされています。

(1) ワークスペースの左の部分にある［Resources］タブをクリックし，ダイアログボックスのリソースエディタを起動します。次に，Resources/Dialog ディレクトリにある IDD_OPCCLIENT_DIALOG ダイアログリソースをダブルクリックします。

これで，Visual Basic と同じ方法でダイアログを設計できます。

次の図には，VisualC++ のリソースエディタを使ってユーザインタフェースを作成する方法が示されています。

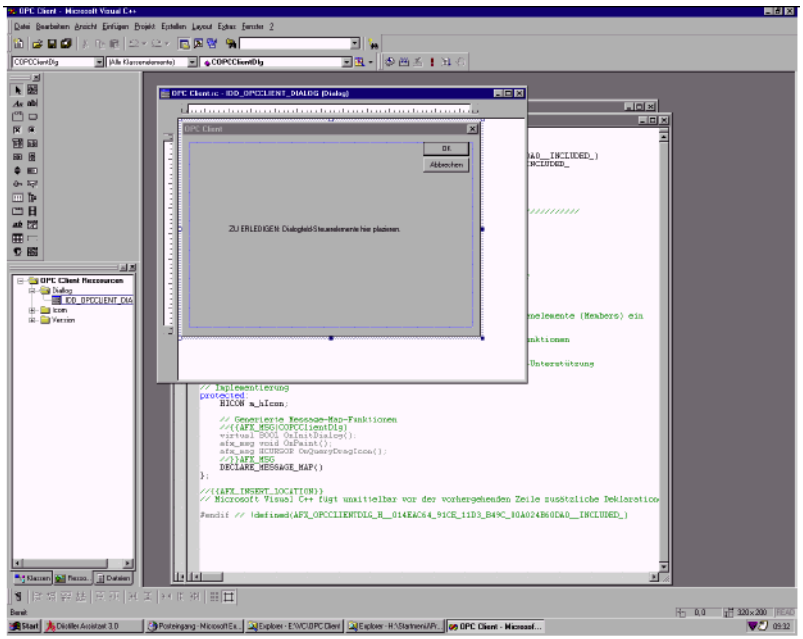


図 21.20 リソースエディタでの表示

(2) まず，既存のボタンと，ダイアログリソースのテキスト例を削除します。

(3) ツールバーで適切なツールを選び，アプリケーションの 2 つのボタンとテキスト入力フィールドを作ります。

(4) 作成したダイアログボックスの上でマウスの右ボタンをクリックし，最初のボタンのプロ

パティとして，“IDC_CONNECT_BUTTON” という ID と，“Connect” というキャプションを入力します。

2 番目のボタンの ID は “IDC_DISCONNECT_BUTTON” で，キャプションは “Disconnect” です。

テキスト入力フィールドの ID は，“IDC_OUTPUT” にします。

(5) 左側のウィンドウで IDD_OPC-CLIENT_DIALOG ダイアログリソースをクリックし，マウスの右ボタンを押して，OPC Client.rc ダイアログリソースを保管します。

これで，ユーザインタフェースが作成されました。

次のステップでは，アプリケーションをプログラミングします。

21.2.5 プログラムの作成

ダイアログボックスの外観を決めるダイアログリソースが作成されたら，そのダイアログボックスを指定するクラスへの接続を確立する必要があります。

Microsoft Foundation Classes では，簡単なデータタイプを使って，コントロールエレメントを表現できます。

- したがって，入力フィールドのテキストは，ストリングで表されます。
- さらに，2 つのボタンのいずれかが押されたときの処理を担当するダイアログクラスに，メンバ関数を追加する必要があります。

これらのジョブはどちらも，クラスウィザードで簡単に実行できます。

■ メッセージ処理およびメンバ変数

(1) ［表示］メニューから［ClassWizard］メニューオプションを選びます。

［MFC ClassWizard］の［メッセージマップ］タブをクリックします。

ダイアログが表示されるので，ここで，ダイアログクラスのメンバ関数に，特定のコントロールエレメントのメッセージを指定することができます。

(2) まだ選択していないのであれば，［クラス名］リストフィールドで COPCClientDlg クラスを選びます。

［オブジェクト ID］リストフィールドには，それぞれのダイアログリソースの ID で，すでに存在しているコントロールエレメントの ID すべてが表示されます。

(3) ［Connect］ボタンに属する IDC_CONNECT_BUTTON という ID をクリックします。

［メッセージ］リストフィールドに，このボタンで送信されるメッセージが表示されます。

ここでは，BN_CLICKED メッセージが必要です。

(4) このメッセージをクリックして次に進みます。

(5) ［関数の追加］ボタンをクリックします。

(6) 新しい “OnConnectButton” メンバ関数の名前を確認し，［OK］をクリックします。

ユーザが［Connect］ボタンをクリックするときに呼び出される COPCClientDlg クラスに，“OnConnectButton” メンバ関数が追加されました。

(7) 2 番目のボタンについても，IDC_DISCON-NECT_BUTTON という ID を使い，3 ～ 6 のス

テップを同じように繰り返し，“OnDisconnectButton” メンバ関数を追加します。

(8) 次のステップでは，メンバ変数として “string” を COPCClientDlg クラスへ追加します。このストリングは，ダイアログの入力フィールドの内容になります。
［MFC ClassWizard］で［メンバ変数］タブを選びます。

このビューには，それぞれのコントロールエレメントのオブジェクト ID も表示されます。

(9) テキスト入力フィールドを表す ID IDC_OUTPUT をクリックし，［変数の追加］ボタンをクリックします。

(10) 表示されるダイアログで，“m_szOutput” という新しいメンバ変数を指定します。
［OK］をクリックすると，新しいメンバ変数が追加されています。

(11) ［OK］をクリックし，クラスウィザードをクローズできます。

■ 他のメンバ変数

(12) COPCClientDlg では，他のメンバ変数が必要です。

前述のように，このクライアントでは，次の 3 つのインタフェースを使います。

- IOPCServer
- IOPCItemMgt
- IOPCSyncIO

インタフェースごとにポインタが要求されます。ポインタは，続く 3 つのメンバ変数に追加されます。

左側のウィンドウの［ClassView］タブを押して［Class］ビューへ移動し，COPCClientDlg クラスをダブルクリックします。

```
protected:
IOPCServer*      m_pIOPCServer;
IOPCItemMgt*     m_pIOPCItemMgt;
IOPCSyncIO*      m_pIOPCSyncIO;
```

図 21.21 ソースコードの構文

(13) COM を使うときには，パラメータ戻り値はメモリに割り当てられたままなので，終わるときに解放する必要があります。

Microsoft 社では，そのための IMalloc という COM インタフェースを提供しています。これを使うとこの作業が非常に楽になります。

したがって，COPCClientDlg クラスでは次のメンバ変数を作成します。

```
IMalloc*       m_pIMalloc;
```

図 21.22 ソースコードの構文

(14) 次の 4 つの新しい変数は，COPCClientDlg クラスの設計では，あらかじめ NULL に初期設定されます。

以下のステートメントが必要です。

```
m_pIMalloc         = NULL;
m_pIOPCServer      = NULL;
```

```
m_pIOPCItemMgt     = NULL;
m_pIOPCSyncIO      = NULL;
```

図 21.23 ソースコードの構文

(15) プログラムの終了時に（ユーザが［Disconnect］ボタンをクリックするのを忘れたとしても）すべてのインタフェースが解放されるようにするために，COPCClientDlg クラスのデストラクタで検査が実行され，該当するインタフェースが要求されていたかどうかが確認されます。

要求されていた場合，インタフェースは解放されます。

デストラクタは，次のようになります。

```
COPCClientDlg::~COPCClientDlg()
{
if ( m_pIMalloc )
{
m_pIMalloc->Release();
m_pIMalloc = NULL;
}
if ( m_pIOPCServer )
{
m_pIOPCServer->Release();
m_pIOPCServer = NULL;
}
if ( m_pIOPCItemMgt )
{
m_pIOPCItemMgt->Release();
m_pIOPCItemMgt = NULL;
}
if ( m_pIOPCSyncIO )
{
m_pIOPCSyncIO->Release();
m_pIOPCSyncIO = NULL;
}
}
```

図 21.24 ソースコードの構文

(16) OPC 項目と OPC グループは，それぞれのサーバハンドルによってのみ削除できるため，グループのサーバハンドルと項目のサーバハンドルは，COPCClientDlg クラスの次のメンバ変数に格納されます。

```
OPCHANDLE          m_hServerHandleGroup;
OPCHANDLE          m_hServerHandleItem;
```

図 21.25 ソースコードの構文

注

これらの変数は，コンストラクタでもあらかじめ “0” に初期設定しておくようにします。

■ COM システムの初期設定

OPC は Microsoft 社のコンポーネントテクノロジをベースにしているため，COM ライブラリを OPC クライアント用に初期設定する必要があります。

これは，“CoInitialize” 関数で行います。

(17) したがって，“InitInstance” メンバ関数に次のソースコードを追加するようにします。これは，関数を開始するときに行います。

```
HRESULT               hResult;
// COM-Bibliotheken initialisieren
hResult = CoInitialize(NULL);
if ( FAILED(hResult) )
{
return FALSE;
}
```

図 21.26 ソースコードの構文

これで，クライアントは COM を扱えます。

ここで，エラーメッセージが記されたメッセージボックスを表示することができます。

(18) 次に，指定したパラメータや戻り値を解放できるように，COM Memory Manager へのポインタが要求されます。

このために，COPCClientDlg の “OnInitDialog” メンバ関数を変更し，次のステートメントを追加します。

```
HRESULT hResult = 0;
// Request access to the COM Memory Manager
hResult = CoGetMalloc(MEMCTX_TASK, &m_pIMalloc);
if ( FAILED(hResult) )
{
CoUnitialize();
exit(0);
}
```

図 21.27 ソースコードの構文

これで，OPC クライアントのプログラミングを開始するための準備がすべて整いました。

■ OPC サーバの作成

(19) ユーザが［Connect］ボタンをクリックすると，OPC サーバが起動し，OPC サーバへの接続が確立されます。

ボタンを押したら，Windows のメッセージシステムによって，COPCClientDlg クラスの “OnConnectButton” メンバ関数が自動的に呼び出されます。

OPC サーバへの接続を確立するステートメントは次のようになります。

メンバ関数に以下のステートメントを追加します。

```
HRESULT hResult;              // results
CLSID clsid;                  // Class ID of the server
// Determine CLSID from ProgID
hResult = CLSIDFromProgID(L”OPC.840DI.Machine-
switch”,&clsid);
if ( SUCCEEDED(hResult) )
{
// Give OPC Server entity
hResult = CoCreateInstance( clsid,
```

```
                                        NULL,
                                        CLSCTX_SERVER,
                                        IID_IOPCServer,
                                        (void**)&m_pIOPCServer
                                        );
...
```

図 21.28 ソースコードの構文

“CLSIDFromProgID” ステートメントでは，サーバの ProgID でクラス ID を判別します。サーバに実体を持たせるために，このことが必要になります。

“CoCreateInstance” ステートメントによってサーバが作成されました。

以下のパラメータが転送されます。

－ “clsid”

このパラメータは，OPC サーバのクラス ID です。

各 OPC サーバには，世界的に統一された CLSID があります。

－ “NULL”

このパラメータは，OPCServer オブジェクトが別のオブジェクトの一部ではないことを定義します。

－ “CLSCTX_SERVER”

このパラメータでは，どのタイプのサーバを始動できるかを判別します。

この例では，プロセス内サーバ，ローカルサーバ，そしてリモートサーバを始動できます。

－ “IID_IOPCServer”

これは，サーバによって要求されたインタフェースの ID です。この例では，IOPCServer インタフェースが要求されています。

－ “&m_pIOPCServer”

インタフェースポインタへのポインタです。

IOPCServer へ戻されるインタフェースポインタがここに格納されます。

このステートメントを正常に実行したら，クライアントは，“m_pIOPCServer” 経由で OPC サーバの IOPCServer インタフェースにアクセスできます。

■ グループの追加

何度も繰り返しているように，変数（すなわち項目）は，OPC サーバではグループに組織されています。

この例では，OPC サーバで “FirstGroup” という名前のグループが作成されます。

(20) まず，次のようなステートメントを使い，余すところなくグループを作成します。

```
LONG      lTimeBias = 0;
FLOAT     fDeadband = 0;
DWORD     dwRevisedUpdateRate = 0;
hResult = m_pIOPCServer->AddGroup(
     L"FirstGroup",       //Name of the new group
     TRUE,                //Active switching
```

```
    100,                //desired update rate
    23111980,           //Client handle
    &lTimeBias,         //time difference
    &fDeadband,         //dead band reaction to server
    LOCALE_USER_DEFAULT, //LCID of the user location
    &m_hServerHandleItem,//Server handle of the group
    &dwRevisedUpdateRate,//cor. update rate
    IID_IOPCItemMgt,    //desired interface of group
    (LPUNKNOWN*)&m_pIOPCItemMgt
    );
if ( FAILED(hResult) )
{
    return;
```

図 21.29 ソースコードの構文

グループを作成するときには，IOPCServer インタフェースの AddGroup メソッドを使います。

そのために，多くのパラメータが転送されます。それぞれのパラメータを以下に説明します。

－ ”FirstGroup”

これは，作成するグループの名前です。

ストリングの前に文字 L があると，このストリングはユニコードストリングに変換されます。

－ “TRUE”

TRUE パラメータでグループが使用可能になります。

つまり，このグループのすべての項目は，基本のシステムのデータを利用していて，それらは継続的に更新されるということです。

－ “100”

これにより，サーバの更新率が 100 ミリ秒に設定されます。

サーバは，この更新率で項目の変更を表示します。さらに，この間隔で，使用可能な OPC 項目のインターバルキャッシュを更新します。

－ “23111980”

これはグループが受け取るクライアントハンドルです。これはグループからクライアントへの接続を構成します。

－ “lTimeBias”

これは，地方時と，グローバル時間の標準である協定世界時との時差です。

－ “fDeadband”

Deadband には，項目の値の許容限度が示されます。項目の値の変更が許容限度を超過した場合に，サーバは “IOPCDataCallback::OnDataChange” メソッドだけを呼び出して，内部キャッシュだけを更新できます。

－ “LOCALE_USER_DEFAULT”

このパラメータは，ユーザが使う音声言語を指定します。

－ “&m_hServerHandle”

サーバは，グループのサーバハンドルをここに保管します。

－ “&dwRevisedUpdateRate”

万が一，必要な更新率に達しない場合，サーバは修正した更新率をここに入力します。

－ “IID_IOPCItemMgt”

新しく作成したグループによって要求されるインタフェースの ID です。

この例では，項目がグループに追加されて処理されるように，サーバは IOPCItemMgt インタフェースを送ります。

－ “&m_pIOPCItemMgt”

新しく作成したグループの IOPCItemMgt へのインタフェースポインタは，ここに保管されます。

この関数が正常に実行されたら，項目は，この新しいインタフェースを介してグループに追加できます。正常に実行されない場合，“OnConnectButton” 関数は即座に終了します。

■ 読み書き操作のためのインタフェースの要求

(21) 項目を読み取るには，IOPCServer インタフェースだけでは不十分です。

この機能には，OPC グループの別のインタフェースが必要です。

－ IOPCSyncIO

このインタフェースは，次のステートメントを使って，IOPCItemMgt インタフェースを介して要求できます。

```
hResult = m_pIOPCItemMgt->QueryInterface(
                                IID_IOPCSyncIO,
                                (void**)&m_pIOPCSyncIO
                                );
if ( FAILED(hResult) )
{
    return;
}
```

図 21.30 ソースコードの構文

QueryInterface を使うと，特定オブジェクトで使える別のインタフェースがないかどうかを照会することができます。万が一，この照会が失敗する場合，OnConnectButton 関数は自動的に終了します。ここで，メッセージボックスでエラーメッセージを確認することもできます。

■ 項目の追加

項目をグループに追加する場合，関係するすべての局面を説明する必要があるので，グループをサーバに追加することよりも手間がかかります。

FirstGroup グループの IOPCItemMgt インタフェースの AddItems メソッドには，項目をグループへ追加する役割があります。

AddItems は，このメソッドを 1 回呼び出すだけで特定数の項目をグループに追加できるようにする数量演算子として働きます。

このメソッドの構文は次のとおりです。

```
HRESULT AddItems(
    [in] DWORD dwCount,
    [in, size_is(dwCount)]   OPCITEMDEF * pItemArray,
    [out, size_is(,dwCount)] OPCITEMRESULT ** ppAddResults,
    [out, size_is(,dwCount)] HRESULT ** ppErrors
    );
```

図 21.31 AddItems メソッドの構文

“dwCount” パラメータには，このグループに追加する項目数を指定します。

この例では，1 つの項目だけを追加します。

2 番目のパラメータには，OPCITEMDEF 構造の配列へのポインタを指定します。

項目は，OPCITEMDEF 構造体を使って定義できます。構造体は次のように宣言されます。

```
typedef struct {
      [string]            LPWSTR    szAccessPath;
      [string]            LPWSTR    szItemID;
                          BOOL      bActive ;
                          OPCHANDLE hClient;
                          DWORD     dwBlobSize;
      [size_is(dwBlobSize)] BYTE * pBlob;
                          VARTYPE   vtRequestedDataType;
                          WORD      wReserved;
} OPCITEMDEF;
```

図 21.32 OPCITEMDEF 構造体

構造体の変数には，次のような意味があります。

－ “szAccessPath”

このパラメータは，クライアントが変数へのアクセス方法を示唆できるようにするために備えられています。

たとえば，コンピュータが，COM1 の高速モデムと COM2 の標準モデム経由で，別のマシンに接続されているとします。

“szAccessPath” を使うと，サーバが高速モデムと低速モデムのどちらでマシンにアクセスするのかを，クライアント側が決定できます。

この情報はこのクライアントには関係しないので，ここは空ストリングのままです。

－ “szItemID”

ItemID は，サーバ内の項目の統一 ID です。

サーバは，項目を階層配列で提示できるので，ItemId は多くの場合，ファイルシステムと同じようなタイプのパスで構成されます。

この例では，”/bag/state/opmode” 項目が示されます。

ご使用の OPC サーバの項目については，メーカーにお尋ねになるか，サーバ

に付属の資料を参照してください。

－ “bActive”

このパラメータは，項目が下層のシステムからデータを受け取るのかどうかを宣言します。

この例でもそうですが，一般にこのパラメータはいつも TRUE にセットされます。

－ “hClient”

OPC 項目のクライアントハンドルです。

これは，項目のクライアントへの接続を構成します。

この例では，クライアントハンドルは 21071977 に設定されています。

－ “dwBlobSize”

続いて，Blob を使う理由を説明します。

’Blob’ は，サーバ内の項目へのアクセスを速めるために使われる，構造化されていないデータ域です。

通常は，クライアントは記号名（つまり項目 ID）を使って項目にアクセスします。

サーバでは，この名前は，ネットワークアドレス，テーブルへのポインタ，ファイル名または他の索引など，サーバ固有のアドレスに変換する必要があります。

サーバでこの変換をすると，時間がかかります。Blob を使えば，項目の内部アドレスをクライアントに転送することができます。次回にこの項目をアクセスするときに，この Blob を使うことができます。したがって，サーバはより速くデータへアクセスできるようになります。

Blob を使うことは，サーバでもクライアントでも任意選択です。

DwBlobSize は，Blob ストリングの長さだけを与えます。

このクライアントでは Blob は必要なく，“dwBlobSize” は 0 に設定されます。

－ “pBlob”

Blob そのものへのポインタです。

このクライアントは Blob を使わないので，ポインタは NULL に設定されます。

－ “vtRequestedDataType”

これらのパラメータを使い，クライアントは，サーバによって送信される変数のデータ型を判別します。

このクライアントではデータをストリングとして表示するので，該当データをすぐにストリングとして要求するのはもっともなことです。

したがって，このクライアントは，このパラメータに VT_BSTR を転送します。

－ “wReserved”

将来の利用のために予約されたパラメータです。

これは，OPCITEMDEF 構造体のパラメータでした。

“AddItems” 関数の 3 番目のパラメータとして，ポインタが OPCITEMRESULT 構造体から転送されます。

この構造体では，サーバは，サーバ側で追加された各項目のプロパティを書き込みます。

以下のプロパティが使われます。

－ “hServer”

これは，項目のサーバへの接続です。後で，項目をグループから削除するには，サーバハンドルが必要です。

－ “vtCanonicalDataType”

これは，サーバ内の項目のデータ型です。

－ “dwAccessRights”

クライアントが項目にアクセスするときのアクセス許可です。これにより，クライアントが読み取り専用なのか，書き込み専用なのか，それとも読み書きできるのかが決まります。

－ “dwBlobSize”

この項目の “Blob” の新しいサイズです。

－ –“pBlob”

新しい Blob データです。

4 番目および最後のパラメータとして，AddItems メソッドは，HRESULT データからフィールドへポインタを転送します。

ここでは，項目がグループへ正常に追加されたかどうかについて，項目ごとに記入されます。

”bag/state/opmode” ItemID を使って項目をサーバに追加するため，この項目の固有データが入った後続の OPCITEMDEF 構造体に記入され，AddItems メソッドが呼び出されます。

(22) 以下のステートメントでゴールです。

```
OPCITEMDEF        opcItemDef[1];
OPCITEMRESULT*    popcItemResult =NULL;
HRESULT*          phResultArray = NULL;
popcItemResult=(OPCITEMRESULT*)m_pIMalloc ->Alloc(
sizeof(OPCITEMRESULT));
phResultArray=(HRESULT*)m_pIMalloc ->Alloc(sizeof(HRESULT));
opcItemDef[0].szAccessPath=L"";
opcItemDef[0].szItemID=L"/bag/state/opmode";
opcItemDef[0].bActive=TRUE;
opcItemDef[0].hClient=21071977;opcItemDef[0].dwBlobSize=0;
opcItemDef[0].pBlob=NULL;
opcItemDef[0].vtRequestedDataType=VT_BSTR;
hResult = m_pIOPCItemMgt->AddItems( 1,
                          opcItemDef,
                          (OPCITEMRESULT**)&popcItemResult,
                          (HRESULT**)&phResultArray
```

```
                                );
if ( FAILED(hResult) )
{
    return;
}
```

図 21.33 ソースコードの構文

AddItems メソッドで 2 つの配列を割り振る場合で，評価後に必要がなくなったときには，忘れずにこれらの配列を解放するようにします。

もう一度グループから項目を削除できるように，新しい項目のサーバハンドルをあらかじめバッファーに入れておく必要があります。

OnConnectButton メソッドに，以下のステートメントを追加します。

```
m_hServerHandleItem = popcItemResult[0].hServer;
m_pIMalloc->Free( popcItemResult );
m_pIMalloc->Free( phResultArray );
```

図 21.34 ソースコードの構文

■ 項目の値の読み取り

OPCItem の現行値を読み取る場合，IOPCSncIO インタフェースの Read メソッドを使います。

このメソッドは，次のように宣言されます。

```
HRESULT Read(
[in]                        OPCDATASOURCE    dwSource,
[in]                        DWORD            dwCount,
[in, size_is(dwCount)]      OPCHANDLE*       phServer,
[out, size_is(,dwCount)]OPCITEMSTATE**   ppItemValues,
[out, size_is(,dwCount)]HRESULT**        ppErrors
);
```

図 21.35 “read” メソッドの構文

メソッドのパラメータには，次のような意味があります。

－ “dwSource”

このパラメータにより，OPC サーバが項目のデータを取得するときの取得元が決まります。

2 つの可能性があります。

定期的に更新されるサーバ内部のキャッシュから読み取るか，デバイスから直接にデータを読み取るかのいずれかです。

項目またはグループがアクティブ状態にない場合，アクティブ状態にないグループではデータキャッシュは更新されないので，デバイスから直接に読み取る場合に限り，データを受け取ることができます。

このクライアントは，デバイスから直接にデータを読み取り，

ついで，OPC_DS_DEVICE を転送します。

－ “dwCount”

このパラメータにより，一度に読み取られる項目の数が決まります。Read は数量操作であるということです。

このクライアントのグループで見つかった項目は 1 つだけなので，“1” がメソッドに送られます。

－ “phServer”

このパラメータでは，読み取る OPC 項目のサーバハンドルの配列が転送されます。

この場合，/bag/state/opmode 項目のサーバハンドルが転送されます。

－ “ppItemValues”

このパラメータでは，サーバは OPCITEMSTATE 構造体の配列を送ります。この配列には，現在のタイムスタンプ，クライアントハンドル，そして読み取る OPC 項目の値と性質が含まれています。

－ “ppErrors”

サーバは，各読み取り項目にエラー状況を送ります。読み取りのプロセスが正常に行われたかどうかが示されています。この配列には，発生したすべての読み取りエラーが記録されます。

(23) 項目を読み取るには，以下のステートメントが必要です。

```
OPCHANDLE          phServer[1];
OPCITEMSTATE*      pItemValues;
HRESULT*           pErrors;
phServer[0] = m_hServerHandleItem;
hResult = m_pIOPCSyncIO->Read(
                              OPC_DS_DEVICE,
                              1,
                              phServer,
                              &pItemValues,
                              &pErrors
                              );
```

図 21.36 ソースコードの構文

この操作が正常に行われれば，読み取られた値がダイアログボックスに表示されます。

クライアントから要求されるときに，サーバは，通常のストリングに変換する必要のあるユニコードストリングを送ります。これは，CString クラスを使うことにより解決されます。

さらに，戻された領域が再び解放されます。

```
if ( SUCCEEDED( hResult ) )
{
      m_szOutput=CString(pItemValues[0].vDataValue.bstrVal);
      UpdateData(FALSE);
}
m_pIMalloc->Free( pItemValues );
m_pIMalloc->Free( pErrors );
```

図 21.37 ソースコードの構文

■ 読み取り／書き込みインタフェースの解放

［Disconnect］ボタンが押されると，サーバへの接続がクローズされます。

サーバによって要求されるすべてのインタフェースが解放されます。

最初に解放されるインタフェースは，読み取り操作用のインタフェースです。

［Disconnect］ボタンをクリックすると，Windows のメッセージシステムによって，OnDisconnectButton メソッドが自動的に呼び出されます。

(24) COPCClientDlg 関数に，以下のステートメントを追加します。

```
if ( m_pIOPCSyncIO )
{
      m_pIOPCSyncIO->Release();
      m_pIOPCSyncIO = NULL;
}
```

図 21.38 ソースコードの構文

■ 項目の削除

OPC の仕様を満たすには，すべてのインタフェースを解放するだけでは不十分です。

インタフェースを解放する前に，グループから項目を削除し，サーバからグループを削除する必要があります。

FirstGroup グループの個々の項目は，項目の処理時にインタフェースから最初に削除され，解放されます。

システムから項目を削除するときには，IOPCItemMgt の RemoveItems メソッドが使われます。

このメソッドを使うと，複数の項目を一度に削除することができます。

(25) OnDisconnectButton メソッドでは，以下のステートメントが必要です。

```
OPCHANDLE        phServer[1];
HRESULT*         pErrors;
if ( m_pIOPCItemMgt )
{
      phServer[0] = m_hServerHandleItem;
      m_pIOPCItemMgt->RemoveItems( 1,phServer,&pErrors );
      m_pIMalloc->Free( pErrors );
      m_pIOPCItemMgt->Release();
      m_pIOPCItemMgt = NULL;
      m_hServerHandleItem = 0;
}
```

図 21.39 ソースコードの構文

■ グループの削除

グループ内のすべてのインタフェースが使える場合でも，OPC の仕様に従い，グループへのアクセス権を保持することができます。

IOPCServer の GetGroupByName メソッドを使うことにより，グループに参照を付加することができます。

その後，OPCServer は RemoveGroup メソッドを使ってグループを削除します。

(26) 最終的にグループを削除するためには，以下のステートメントが必要です。

```
if ( m_pIOPCServer )
{
      m_pIOPCServer->RemoveGroup(m_hServerHandleGroup,FALSE);
      m_hServerHandleGroup = 0;
...
```

図 21.40 ソースコードの構文

■ サーバからの切断

サーバは，クライアントによって要求された最後のインタフェースが解放されると，自動的に切断されます。

従って，前の「グループの削除」ステートメントの IF 要求に，以下のステートメントを追加します。

```
....
      m_pIOPCServer->Release();
      m_pIOPCServer = NULL;
}
```

図 21.41 ソースコードの構文

これにより，最後のインタフェースが解放され，サーバがクローズします。ここで，プログラムをテストしてください。

21.2.6 ユーザインタフェースとプログラムのテスト

［ビルド］–>［ビルド OPCClient.exe］メニューで OPCClient.exe プログラムを作成し，開始します。

［Connect］ボタンをクリックします。

OPC サーバが起動するまで，若干時間がかかります。クライアントがサーバへ正常に接続されたら，ダイアログボックスの入力フィールドに値が表示されます。

プログラムが機能しない場合，以下の点を調べてください。

- サーバの正しい ProgID を入力しましたか
- サーバに追加したい項目は存在していますか
- サーバが正しくインストールされていて，登録されていますか

プログラムを終了するには，［Disconnect］ボタンをクリックしてダイアログボックスを終了します。

これで，Visual C++ を使って，以下の機能を備えたクライアントアプリケーションを開発する方法を習得できました。

- OPC サーバへの接続を確立する
- サーバにグループを作成する
- グループに項目を追加する
- 項目の値を読み取る
- アプリケーションを終了する前に行う必要のある動作を実行する

Visual C++ でのプログラミングは，確かに費用がかさみますが，高度な処理ができるという利点があります。

索引

用語集

■ ADSI プロバイダ（ADSI Provider）

Active **D**irectory **S**ervice **I**nterface プロバイダ。ADS 定義のインタフェースを提供する。

■ アプリケーション（Application）

ユーザプログラムの指定

■ AS インタフェース（AS Interface）

Actor **S**ensor Interface – 簡単なバイナリセンサとアクタとを直接リンクするときの回線（転送できる情報の量は少ない）

■ COM

Component **O**bject **M**odel – OLE をベースにした，Microsoft 社による Windows オブジェクトの仕様

■ DCOM

Distributed **COM** – COM 規格を拡張したもので，COM オブジェクトを使って，ネットワーク内で COM オブジェクトを配布できるようにする

■ DLL

ダイナミックリンクライブラリ（**D**ynamic **L**ink **L**ibrary）– 複数のプログラムで使えるが，メモリに 1 度しかロードされない機能の集合（Windows/Windows NT のプロパティ）

■ 産業用イーサネット（Industrial Ethernet）

イーサネットベースの産業向けのバスシステム（以前の SINECH1）

■ MFC

Microsoft Foundation Classes

■ ネットワーク（Networks）

1 つのネットワークは，複数の相互接続サブネットワークの 1 つで形成される。ここには，任意の数のノードが含まれている。複数のネットワークを併存させることも可能。

■ OLE

Object **L**inking and **E**mbedding。さらに開発されて COM になっている。

■ OPC

OLE for **P**rocess **C**ontrol – OLE をベースにした，産業通信ネットワークへのメーカーに依存しないアクセスを定義する業界規格

■ OPC クライアント（OPC Client）

OPC インタフェースを使って処理データにアクセスするために，OPC サーバを使うユーザプログラム

■ OPC サーバ（OPC–Server）

OPC サーバは，産業ネットワークを使って通信するために，OPC クライアントへ高度な機能を提供する製品

■ PROFIBUS

Process **F**ield **B**us – 欧州方式のフィールドバス規格

Yaskawa Siemens CNC シリーズ

本製品の最終使用者が軍事関係であったり，用途が兵器などの製造用である場合には，「外国為替及び外国貿易法」の定める輸出規制の対象となることがありますので，輸出される際には十分な審査及び必要な輸出手続きをお取りください。

製品改良のため、定格、寸法などの一部を予告なしに変更することがあります。
この資料についてのお問い合わせは、当社代理店もしくは、下記の営業部門にお尋ねください。

製造

株式会社 安川電機　　シーメンスAG

販売

シーメンス・ジャパン株式会社

工作機械営業本部
東京都品川区大崎1-11-1 **ゲートシティ大崎ウエストタワー**　〒141-8644
TEL（03）3493-7411　FAX（03）3493-7422

アフターサービス

カスタマーサービス事業本部
TEL 0120-996095（フリーダイヤル）　FAX (03)3493-7433

シーメンス・ジャパン株式会社
http://www.siemens.co.jp

02- WSC02, 02-WSC09