# 目录

# 1 介绍

在一个单独的 OPC 概述文档（(OPCOVW.DOC ）对 OPC 作了全面的介绍，这个特殊的文档只是对 OPC 数据访问接口做详细的解说

## 1.1 读者

这个规范是特地为做 OPC 客户端和服务器端开发者作的一个参考手册。他认为读者已经对 MS OLE/COM 技术和工业过程控制需求非常熟悉。

这个规范是为熟悉并选择使用 C 和 C++语言来开发 OPC 客户程序提供的资料。所以开发者要对这些方面的技术和特别的组件要相当熟悉。

## 1.2 获得的资料

从 OPC 基金会可获得的 OPC 借口访问相关的资料包括他 OPC 规范本身以外还有 OPC IDL 文件（在这本文档的附录里）和 OPC 错误头文件（在这个文档里）。你可以方便地在 OPC 基金会的网站上可以找到标准的代理存根的 DLLS 和从 OPC 接口访问 IDL 文件产生的头文件，会员可以在 OPC 基金会的网站上下载 OPC 接口访问的例子的源代码。

这个 OPC 数据访问规范包括以下设计信息

1.OPC 数据访问自定义接口—这个文档描述了 OPC 组件和对象的接口和方法。

2.OPC 数据访问自动化接口—晚些时候将会提供一个可选择的单独文档（OPC 数据访问自动化接口规范 3。0）用来描述 OPC 自动化接口，这是为使用 VB Delphi 和其他具备自动化功能产品的软件来与 OPC 服务器对接编成提供的安定法。

# 2 OPC 数据访问基本原理

这个小节介绍 OPC 数据访问和涵盖 OPC 数据访问的主题的细节。另外的公共话题包括 Windows NT ， UNICODE，线程模式，等在 OPC 概述（OPCOVW.DOC）的资料里讨论。

## 2.1 OPC 概览

这个规范描述 OPC COM 组件和他们的接口通过 OPC 服务器的实现。一个 OPC 客户能连接到不同开发商开发的一个或多给 OPC 服务器上。

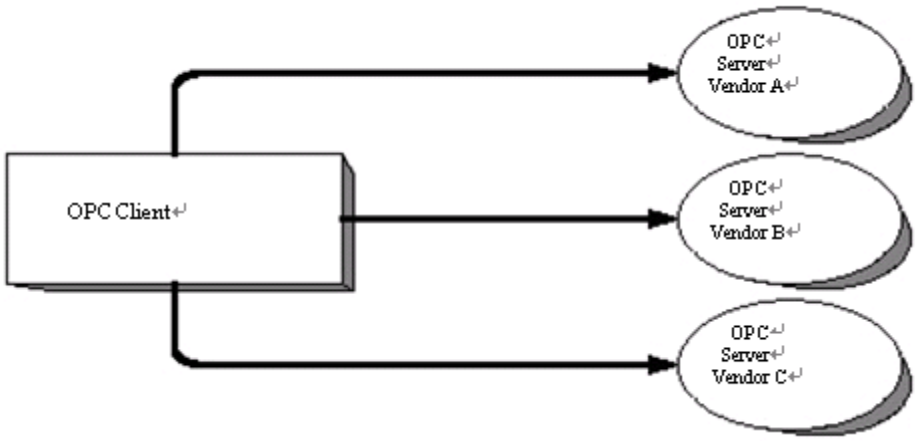


图 2-1 OPC 客户

不同的开发商提供不同的 OPC 服务器，开发商提供的代码决定了每个服务器访问的设备和数据，数据名称，和服务器访问的物理数据的细节。命名习惯在后来的章节提供。

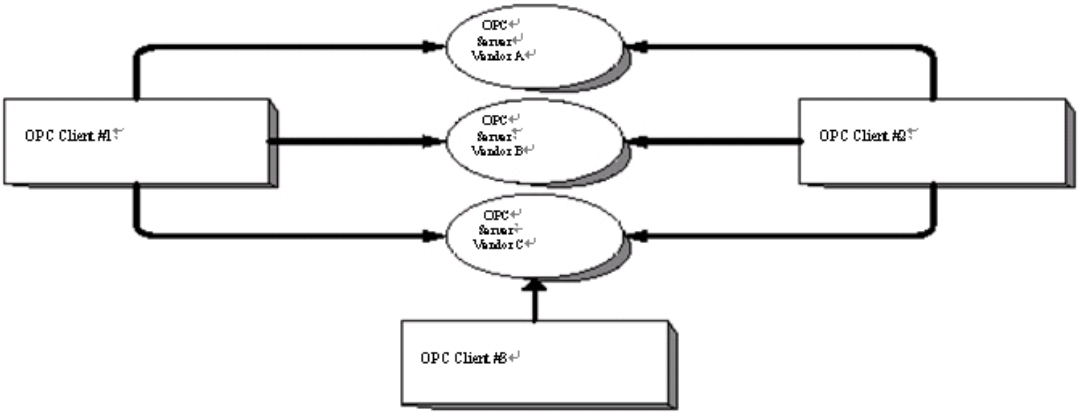


图 2-2 客户/服务器关系

水平高一点 一个 OPC 服务器可以由多个对象组成：服务器，组，和条目。OPC 服务器对象提供服务器的信息和作为容器为组对象服务。OPC 组对象提供组的信息和提供了包含和逻辑组织 OPC 条目的机制。

OPC 组对象为客户组织数据提供了一个方式。例如 组可也用一个特别的操作来展示和报告来表示条目。数据能被读写。基于连接的异常在客户与组里的条目之间被创建和根据需

使能或取消使能。OPC 客户端可以配置 OPC 服务器为客户端提供数据变化的速率。

再每一个组里，客户端可以定义一个或多个 OPC 条目。

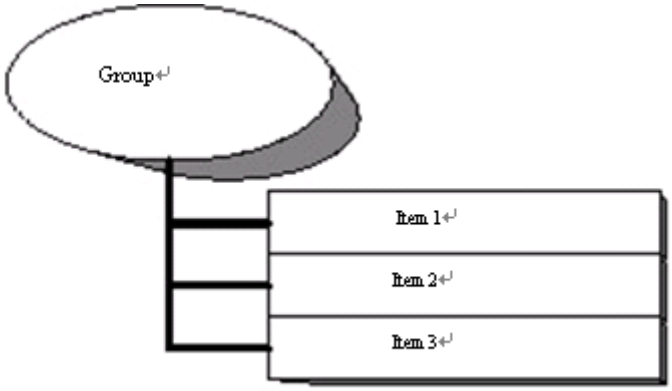


图 2-3 组/条目关系

OPC 条目表示同服务器的数据源连接。从自定义的接口来看，一个 OPC 条目不能被 OPC 客户端理解为一个对象。因此，没有为 OPC 条目定义的接口。所有的对 OPC 条目的访问都是通过 OPC 组对象来实现的，组对象包含了 OPC 条目，和 OPC 条目的简单定义。

每一个条目都是由数值，性质，和时间戳组成。数值是 VARIANT 类型的，性质很像 Fieldbus 里面的定义规范。

标注：条目不是数据源，他们仅仅提供连接。例如，一个 DCS 系统里的变量的存在跟客户端是否正在访问他们没有关系。OPC 条目应该被认为一个简单的数据地址的标示，不能认为地址所指实际的物理数据源。

## 2.2 OPC 适用范围

虽然 OPC 首先是为从网络服务器来访问数据而设计的，可是 OPC 接口能被应用到很多地方。在低端的应用水平，他们可也从物理设备里获得原始数据并写入 SCADA 或 DCS，或者从 SCADA 和 DCS 系统里到应用程序里。他的结构和设计使构造一个允许一个客户端访问多个不同开发者开发的运行在不同节点的服务器提供的数据的客户应用程序成为可能。

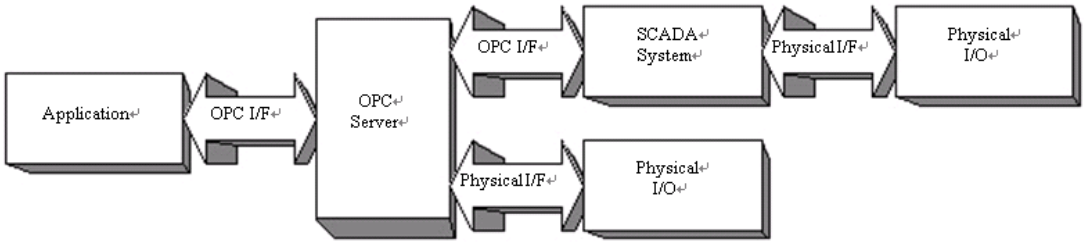


图 2-4 OPC 客户/服务器关系

## 2.3 一般的 OPC 体系结构与组成

OPC 是两套接口的规范：OPC 自定义接口和 OPC 自动化接口，一个修订的自动化接口随着 3.0 版 OPC 规范的发布一起提供。

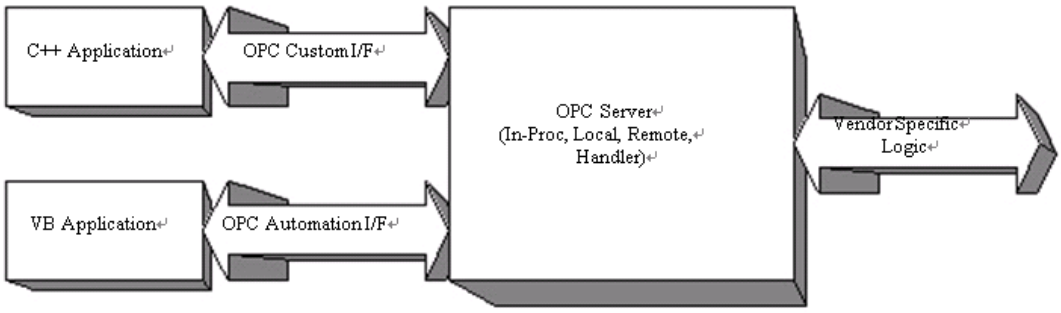


图 2-5 OPC 接口

OPC 说明书指定了 COM 接口，不是接口的实现，他指定了客户期望接口提供的功能行为。

包括架构的描述和适合这些架构接口，象所有 COM 的实现一样，OPC 的架构是客户/服务器模型，OPC 服务器组件为 OPC 组件提供接口和管理他们。

在 OPC Server 的实现里有几个综合的考虑。主要的问题是超越不可分享物理设备路径的数据传输的频率。因而，我们期望 OPC Server 是一个本地的或远程的 EXE 代码,能负责有效的物理设备数据的收集.

一个 OPC 客户程序通过规范的 OPC 接口与一个 OPC Server 进行通信.OPC Server 必须实现自定义的接口,可选的自动化接口如果定义了有时也会实现.

一个 OPC 句柄会用来聚合接口和提供 OPC 自动化接口的额外的条目功能.

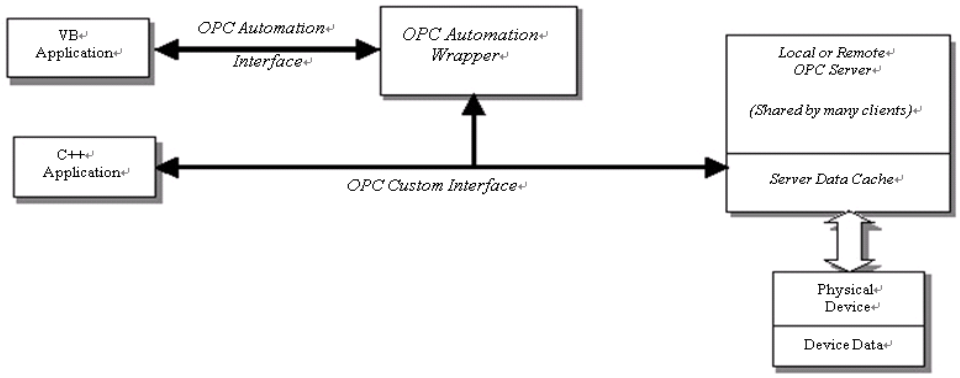


图 2-6 典型的 OPC 构架.

服务器被期望巩固和使所有的客户的数据访问最优化用来提高同物理设备的通信性能。对输入，从设备返回的数据会被暂存为了各种 OPC 客户的异步分发和同步采集。对输出，OPC 服务器会为 OPC 客户程序更新物理设备的数据。

## 2.4 OPC 数据访问架构的伴随规范

OPC 数据访问提供基本的功能同过一套标准的的接口来从各种网络上的设备访问数据。这些接口使客户和服务器发现对方和通信容量（功能，命名空间，命名空间条目信息）的相互协作变得更容易，另外提供了一套完善的接口定义来促进各种装置根据客户应用程序的需要来读写数据条目。OPC 数据访问的首要意图是为支持纵向的构架的数据服务提供接口（在高层的计算机上提供从设备到客户应用程序的服务）

OPC 基金会朝着扩展 OPC 数据访问的能力方向努力工作着。伴随的接口被加进来为提供需要的扩展数据访问功能。这些包括在不同的工业网络分享服务器数据的能力。

伴随的构架包括：

OPC 公共的定义和接口包含公共规则和设计标准和公共的数据访问接口的规范。

OPC 复杂的数据，详细说明了怎么使用数据访问来交换数据结构。他提供了一种机制来象传送实际值一样来传送数据的结构。综合的数据没有集成到 OPC Da 3.0 规范中

OPC 数据交换， OPC DX 已经被设计用来在 OPC 服务器间横向移动设备层数据，为了表达这种新的技术， OPC DX 能够实现在 Ethernet 系统包括 PLC HMI/SCADA 设备和 PC 之间的数据的互用.

## 2.5 对象和接口综述

OPC 服务器对象提供了同一系列数据源进行通信和数据访问的一种方式，可用的资源的类型是关于服务器实现的一个功能。

一个 OPC 客户程序同一个 OPC 服务器连接和通信是通过接口来完成的。OPC 服务器对象为客户程序提供了创建和操作 OPC 组对象的功能。这些组对象允许客户们组织她们想访问的数据。一个组可以是活动的或者非活动的单位，一个组也可以提供一个方式让客户用来定制条目列表从了在条目发生变化时得到通知。

注：所有的 COM 对象都是通过接口访问的。客户看到的仅仅是接口。因此，这里描述的对象是逻辑上的表示，没有对服务器内部作任何的实现。下面的图是 OPC 对象的和他们的接口的摘要。注一些接口是可选择的 [] 中的。

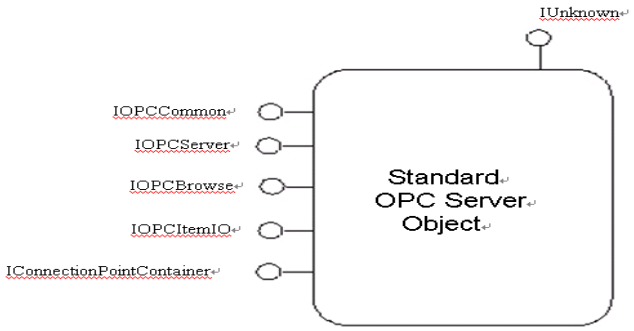


图 2 - 7 标准 OPC 服务器对象

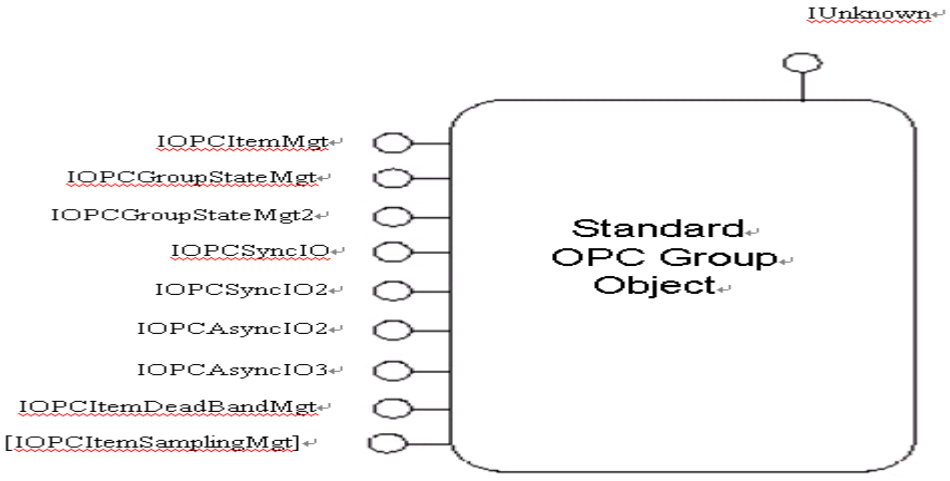


图 2 - 8 -标准 OPC 组对象

## 2.6 服务器的地址空间和配置

这个版本的 OPC 说明书设定服务器的地址空间的配置是用 IPersistfile 接口管理和长期存储的。只有服务器的特殊的信息会被长期存储。所用的客户配置信息（组和条目的定义）必须由用户程序分别自己永久存储。所有的系统定义的句柄在客户和服务器的会话过程中拥有相同的值。

把服务器空间和他的特殊用途的子空间区分开来是很重要的（做为服务器配置）客户的详细的组的维护细节会在这个说明书里讨论。组的永久存储是各个客户程序的事。服务器地址空间的定义和配置的细节特别没有说明。例如地址空间可能如下。

全部固定。（如，对于特别的设备接口象测量设备）

再 OPC 环境以外完全配置。（如，外部存在的 DCS 系统的接口）

一个智能的服务器能智能在启动时自动配置，让服务器安装硬件和接口。

根据客户程序当前的请求的数据条目的名称服务器自动地配置

服务器的地址空间期望是稳定的和能在服务器内进行管理。客户程序会更具需要不时地定义和管理被叫做组的相关的小的条目列表。通过 OPCGroups 得使用，这里描述的接口给客户程序提供了定义，管理，重新创建这些需要的列表的能力.

## 2.7 应用级别服务器和网络节点选择

OPC 数据访问支持在服务器内部管理客户组的请求的概念。这些组能包含仅从从一个特殊的 OPC 服务器对象数据请求。为了能访问数据，一个客户应用程序需要详细说明下列能容：

OPC 数据访问服务器的名称（CoCreateInstance ,CoCreateInstanceEx,使用）

OPC 数据访问服务器主机的名称（CoCreatInstanceEx）

开发商的 OPC 条目的定义详细说明（在地址空间里的数据条目的名称），说明书为讨论这些架构里的具体实现和客户程序里的接口留一些余地。

## 2.8 同步和串行问题

在单独的传输里的客户读写数值和属性的性能被称为同步传输。例如，很多应用想确保特殊的条目的数值，性能，和时间戳是同步的。还有，一个报告数据包被希望能确保一个组的几个值的读写能作为一个绑定的报告捆绑在一起。最后，一个配方的下载包也会被期望组里的所有的值在一起发送，数值没有接收完配方不会被执行。仅有一少部分的关于同步重要性的例子。

OPC 自身不能够保证所有的同步任务被完成只是一少部分结果。服务器和客户程序之间还加入了传送握手信号和标志来标示准备好和完成状态的请求。还有一些东西需要被详细说明，也就是 OPC 服务器的行为要确保这些同步能被完成。

以后会看到 OPC 允许直接读写组里的条目和单独的条目也希望能在连接之上。不用回头太远就能够观察到这些问题和服务器的行为。

1． 一般地，OPC 服务器会尝试在一个对数据条目和属性的单独的读写操作时保持同步。单独的条目的读和单独的条目的写操作要求保持同步是没有必要的。显然，从不同的物理设备进行数据读操作是很难保持同步的。

2． 能被不同线程访问的数据条目的读与写必须实现线程的安全，在这个方面这本说明书里对数据同步作了详细的说明。例如重要性包括：在一个服务器内部的逻辑，当另一个

线程完成物理传输和对一个与第一个线程共享的缓冲区放进数据时一个服务线程方法被执行。另外一个例子可能一个句柄的逻辑或者一个隐藏的 OPC 线程服务 OnDataChange 方法的代理服务器在对共享缓冲器写数据时一个客户线程也可能在读该缓冲器。

3． 线程问题经常是很重要的，尤其在 SMP 系统更是重要。

串行通信我们的意思是在写操作之行时客户程序控制顺序的功能。

1．前列推荐在任何服务器的实现里对同一个设备的写请求安顺序控制。例如一用程序会使用配方完成标志，在一个单独的配方条目发送以后一个配方下载完成标志被应用程序置位。在这种情况下，数据会被按照同样的顺序传输到物理设备，他会保证在所用数据传送完之前完整标志不会被置位。在服务器缓冲器里输出数据和执行单独的传送关系线程来发送输出到物理设备，服务器的实现必须特别注意来确保输出的次序的保持。

2 .客户程序在读数值要非常明确同时在接收通过回调函数更新时特别注意给出准确地定义回调函数发生和没有发生。这些会在以后讨论的更多。

所有这些问题会在以下的方法的具体实现的细节里被澄清。

## 2.9 永久存储过程

OPC 服务器会实现一个可选择的接口来使 OPC 客户告诉 OPC 服务器保存 OPC 服务器的配置信息更方便。服务器信息可能包括同设备和使数据源同 OPC 服务器通信变得更方便的必要的数据源信息。客户程序的配置信息包括组，条目被服务器永久存储。

客户程序对被他们请求使用的组和条目的配置和永久存储有责任。

# 3 OPC 数据访问快速参考

这一节包括自定义接口方法的快速参考。在以后的参考章节定义了详细的接口行为和参数。

## 3.1 自定义接口

注：这个章节不说明多余的标准的 COM 接口例如 IUnknown，IEnumString 和 IEnumUnknown 这些被 OPC 数据访问使用的接口。

OPCServer
IOPCServer
IOPCBrowse (new 3.0)
IOPCItemIO (new 3.0)
IConnectionPointContainer
IOPCCommon
OPCGroup
IOPCGroupStateMgt
IOPCGroupStateMgt2 (new 3.0)
IOPCASyncIO2
IOPCAsyncIO3 (new 3.0)
IOPCItemMgt
IOPCItemDeadbandMgt (new 3.0)
IOPCItemSamplingMgt (new 3.0, optional)
IConnectionPointContainer
IOPCSyncIO
IOPCSyncIO2 (new 3.0)
EnumOPCItemAttributes
IEnumOPCItemAttributes

# 4．OPC 自定义接口

## 4.1　OPC 自定义接口概述

在 OPC 自定义接口的对象包括以下自定义对象：

• OPCServer

•OPCGroup

本章对这些对象的接口和行为进行了详细介绍。 OPC 服务器的开发者需要通过提供这一章中所定义的功能来实现 OPC 对象。

本章还引用和定义了标准的 OLE 接口的预期行为。在构建 OPC 标准组建时，OPC 服务器和 OPC 客户端必须被实现的接口。

此外，创建标准和自定义枚举对象和接口对这些对象返回。一般来说枚举对象和接口进行了简要说明，因为他们的行为是由 OLE 明确定义的。

OPC 规范是首选方法,枚举的创建和返回是由通过对象上的方法得到的,而不是通过 QueryInterface 得到的。枚举成员如下:

- 组枚举- (see IOPCServer::CreateGroupEnumerator)
- 项属性枚举 - (see IOPCItemMgt::CreateEnumerator)

在某些情况下，你会注意到，通过枚举和在其他情况下为项目的简单列表返回的东西列表。我们的选择取决于返回的项目的预期数量。通过调查员，而“小”名单是更容易，并通过明确列出了有效恢复'大'列表是最好的恢复。

## 4.2　一般资料

本节提供了有关 OPC 接口的一般信息，以及有关 OPC 的设计师如何预期和实现这些接口的一些背景资料。

### 4.2.1　版本的互操作性

数据访问服务器可能是版本 1.0a 的， 2.x 或 3.0 的规范，或与数据存取规范版本的任意组合的要求相兼容。数据访问客户端也可能与版本 1.0a 的， 2.x 或 3.0 的规范，或与数据存取规范版本的任意组合的要求相兼容。

最好的迁移策略服务器和客户端的厂商将取决于其特定的业务情况。例如，一个供应商谁多卖了自己的客户端和服务器组件作为一个打包的系统，并为他们的 OPC 兼容性代表一项长期的战略将有较少需要支持多个版本的接口。

作为一般指引，建议现有的服务器厂商加入版本 3.0 的支持，并将继续支持版本 1.0A 和 2.x ，让现有客户迁移到自己的步调。

| 数据访问服务器<br>所需的接口 | **1.0** | **2.0** | **3.0** |
|---|---|---|---|
| OPCServer | | | |
| IUnknown | Required | Required | Required |

| | | | |
|---|---|---|---|
| IOPCServer | Required | Required | Required |
| IOPCCommon | N/A | Required | Required |
| IConnectionPointContainer | N/A | Required | Required |
| IOPCItemProperties | N/A | Required | N/A |
| IOPCBrowse | N/A | N/A | Required |
| IOPCServerPublicGroups | Optional | Optional | N/A |
| IOPCBrowseServerAddressSpace | Optional | Optional | N/A |
| IOPCItemIO | N/A | N/A | Required |
| | | | |
| OPCGroup | | | |
| IUnknown | Required | Required | Required |
| IOPCItemMgt | Required | Required | Required |
| IOPCGroupStateMgt | Required | Required | Required |
| IOPCGroupStateMgt2 | N/A | N/A | Required |
| IOPCPublicGroupStateMgt | Optional | Optional | N/A |
| IOPCSyncIO | Required | Required | Required |
| IOPCSyncIO2 | N/A | N/A | Required |
| IOPCAsyncIO2 | N/A | Required | Required |
| IOPCAsyncIO3 | N/A | N/A | Required |
| IOPCItemDeadbandMgt | N/A | N/A | Required |
| IOPCItemSamplingMgt | N/A | N/A | Required |
| IConnectionPointContainer | N/A | Optional | Required |
| IOPCAsyncIO | Required | Optional | N/A |
| IDataObject | Required | Optional | N/A |

### 4.2.2 内存的管理

按照 COM 规范，客户端必须释放与'out'或'in/out'参数相关联的所有内存。这包括由任何结构内的元素所指向的内存。客户端的开发者必须了解这些，否则会遇到很难找到内存泄漏。可以通过 IDL 文件，确定哪些参数是输出参数。推荐的方法是让客户端来创建一个用于正确释放每种类型结构的子程序。.

不管成功还是失败，服务器必须为’out’参数返回定义好值。释放已分配的内存是客户端的责任。

注意:如果错误结果是任何失败的错误,比如 E_OUTOFMEMORY OPC 服务器应该返回 NULL 为所有“了”(这是标准 COM 行为)的指针。这个规则也适用于错误返回的数组(ppErrors)下面的许多功能。在一般情况下,一个健壮的 OPC 客户端应该检查每一个释放前或/指针为 NULL。

### 4.2.3 标准接口

按照 COM 规范，每个必要接口上的所有方法必须被实现。

按照 COM 规范，任何可选的接口,支持必须实现该接口中的所有功能,即使实现只是返回 E_NOTIMPL 存根实现。

### 4.2.4 空字符串和空指针

这两个术语在下面使用。他们是不一样的东西。一个空指针是无效指针（0）如果使用将导致异常。一个 NUL 字符串是一个有效的（非零）指向一个 1 字符数组，其中的字符是 NUL（即 0）。如果一个 NUL 字符串作为[out]参数（或作为一个结构的元素）从一个方法返回，它必须释放，否则包含 NUL 的内存将会丢失。还要注意的是，由于 COM 封送限制，一个 NULL 指针不能传递给一个[in，string] 参数。在这种情况下，一个指向一个字符串 NUL 的指针应该作为省略的参数来传递。只要有可能，NUL 串使用这种规范。 C＃不处理 NUL 指针或 NUL 字符串。

### 4.2.5 返回的数组

你会注意到在与指针的指针结合使用 IDL 的语法 size_is(dwCount)。这表明返回的项目是一个指针所指示的类型的实际的阵列,而不是一个指针数组的指针所指示的类型的项目。这简化了编组，创建和由服务器和客户端的数据的访问。

### 4.2.6 缓存数据、设备数据和时间戳

数据源定义为 CACHE 和 DEVICE，在 OPC 规范中没有明确的说明，只作为一个抽象的概念在通信中使用。数据是从 CACHE 还是从 DEVICE 中读取，仅仅是由已定义的各种接口定义给定，具体实现规范中不做定义。

一般情况下，认为服务器把数据读到缓存中，而客户通过多种通信机制从缓存中获取数据。对 DEVICE 的访问被认为速度较慢，但是更准确，通常只在诊断测试或者在某些特殊严格的操作中使用。

缓存应该反映数据的最新值（可为以后讨论，更新率和死区优化），以及质量和时间戳。时间戳应表明其值和质量是由设备所获得的时间（如果可用）或时间服务器更新或验证在其缓存中的值和质量。注意，如果一个设备或服务器检查值每隔 10 秒那么预期行为将是该值的时间戳将每 10 秒进行更新（即使该值实际上不改变）。因此，时间戳反映的是服务器知道对应的值是准确的时间。

这也是真实的,无论物理设备到系统的接口是否是基于异常。例如,假设它是已知的(a)一种基于异常设备检查值每 0.5 秒和（B）连接到设备是好的，（c）该设备发送的用于项目

的更新 FIC1013 分钟前一个值的 1.234。在这种情况下，从缓存读取的返回值将是 1.234，更重要的是，时间戳返回这个值将是当前时间（0.5 秒内），因为它是已知的该项目的价值实际上仍然 1.234 为 0.5 秒前。

### 4.2.7 时间序列值

OPC 数据访问接口的设计主要采取当前实时处理或自动化数据的快照。与值一起返回的时间戳主要目的是作为一个指示当前的数据的质量。这些接口并不是真正为了处理等单点缓冲时间序列数据历史数据。如果服务器选择实现 IOPCItemSamplingMgt 接口,缓冲的数据是被允许的。详细信息见 IOPCItemSamplingMgt 接口。

### 4.2.8 异步和同步接口

架设很多客户程序希望访问暂存器的数据，会有几种方法客户用来访问服务器数据。

可以用同步方式从缓冲器里读数据。这种方式适合很简单的一些客户程序，只读一些相当小数量的数据和最大效率不予考虑。这种方式操作的客户程序本质上是对服务器上已经存在的扫描的一种复制。

可以用 IOPCDataCallback 预订缓冲器里的数据,这个是客户程序被推荐使用的一种方式，因为它会消耗 CPU 和网络资源最少。

### 4.2.9 活动标志，死区和更新率

这些组和条目的属性用来降低客户和服务器的消耗的资源。在稍后的 GROUPS 里会做详细的讨论。一般的，他们会影响缓冲区数据和质量信息的更新频率和客户程序 IOPCDataCallback 的访问频率。

### 4.2.10 错误和返回代码

OPC 说明书描述了 OPC 服务器实现的和 OPC 客户应用程序依靠的接口和相应的行为。在这个说明书的 OPC 错误代码摘要一章中包含了一个 OPC 详细的错误列表和返回代码。在每个被描述的方法后面都有一个所有可能出现的 OPC 错误代码也包括最平常的 OLE 错误代码。客户程序有可能遇到额外的错误代码象 RPC 和实际的安全代码,他们应该做好预处理这些代码的工作。

在两种情况下（读/写）允许服务器返回开发商特殊的错误代码。这些代码被传递给 GetErrorString 方法。以后会详细讨论这个。

所有的情况下 E 错误代码表示 FAILED 类型的错误和 S 错误代码表示有局部成功。

### 4.2.11 启动问题

当条目被加到组里面以后，OPC 服务器会花费一些时间为这些条目获得一些实时的值。在这些情况下，客户端就会完成一次读缓存，或者建立一个 AdviseSink 或者基于预约好的

连接点，和在这些建立在预约上的值可用之前对他们进行一次刷新。你会在后边对预约的讨论中你会看到，在一个组中会期望有一个包含所有数值刷新的回调函数。当条目被加到组中的同时，在这种条件下期望的行为就会被概括描写。这些条目的初始状态会设为被PC_QUALITY_WAITING_FOR_INITAL_DATA(20)屏蔽的 OPC_QUALITY_BAD。任何客户端对这个组的操作会被表现为好和坏的质量评估。如果服务器在添加条目时不能迅速判断数据类型，它就会返回 VT_EMPTY 的规范数据类型。注意在同步读和异步 2 操作时，服务器可能返回开发者定义的特殊的错误信息来提示开发者的一个特殊的错误例如“服务器初始化错误”。

### 4.2.12 VARIANT 数据类型和通用性

为了增强通用性，推出了下边的规则和建议。

**一般的建议：**

- RARIANT 类的 VT_I2, I4, R4, R8, CY, DATE, BSTR, BOOL, UI1 还有这些类型的单数组类希望能被经常作为通用类型使用。（部分地因为它们是 VB 里的合法类型）。
- 推荐无论什么情况下客户端请求的数据为这些数据类型，服务器也同样返回的是这些数据类型的数据。
- 如果想对一些扩展的数据类型进行使用在服务器和客户端都是同一个开发者开发的软件中进行不可移动的数据返回到客户端，为了通用性这些东西尽量少用。
- 在实际用于中法像一些服务器在条目加入和删除的时候不能断定不活动的数据类型（例如 那些远程和本地的连接）。作为不活动的数据类型返回的 VT_EMPTY 类型在这些服务器中已经得到了实践。当数据可用和他们的实际数据类型可以断定的时候，这些服务器会保持请求的数据类型和将以请求的数据类型返回数据，推荐客户端处理 AddItems 和 ValidateItems 返回的初始的 VT_EMPTY。

**通用规则：**

- 在推荐的数据类型之上服务器被允许维护和返回任何合法的通用数据类型（任何的对 VT_flags 的合法更换）
- 客户端被允许请求在推荐的数据类型之上的 Variant 数据类型
- 服务器应该准备以极好的方法来处理那些不能被转化为他们被请求类型的数据。那就是，他们不应该失效，或返回错误的结果，或丢失内存，就像提到的，任何地方它们都可能返回各种错误包括微软 VariantChangeType.返回的错误类型。
- 客户端应该随时准备处理那些不能处理请求数据类型的服务器。也就是，当错误被返回时它们不应该失效和丢失内存。
- 请求 VT_EMPTY（习惯被表示返回他自己的通用类型）的客户端应该同样准备处理任何被返回的类型。也就是，即使发现他们不能用或不能显示返回的数据。他们应该正确释放数据。和正确地给用户提示一个不能被客户程序识别的数据类型被返回了。
- Variant 的实数值（VT_R4,VT_R8）将包含 IEEE 浮点数。标注，IEEE 标准浮点数包含无数值的值（NAN）以这种格式存储。即使这些值相当少用，它们也是允许使用的。如果这些值被返回，就需要 QUALITY 标志位被设置为 OPC_QUALITY_BAD。
- 虽然 IEEE 标准允许 NAN 被存储到 VT_R4 和 VT_R8 格式，这些值仅可用目标条目的额外的固定格式来进行读写。他们不能与其他类型进行相互转换。当这些数值被读时

QUALITY 标志为必须被服务器返回 OPC_QUALITY_BAD。如果这些值被写时 QUALITY 标志必须被客户提供位 OPC_QUALITY_BAD，一些特殊的服务器是否返回和接受这些值就看服务器这方面是否特殊说明了。

**数据转换的公认额外规则**

OPC 服务器必须支持至少是下面的在规范的和请求的数据类型之间的转换。读和写必须同步。注意早期的版本实现这个功能是同沟在 COM 库中的 VariantChangeEx（）来提供的。在下面的表中，标志为 OK 的时可用的，其他的能与不能与数据源的特殊性有关。

象在这个规范的其他地方注释的一样，客户程序都会明确一个 localID 来使用。服务起会把他传递给 VariantChangeTypeEx()在所有的会话中使用。注意有喜恶客户会通过控制面板的区域设置覆盖默认的区域设定。例如，英文的设置可能是 MM/DD/YY 或 YY/MM/DD 也有其他的格式。显然 03/02/01 在这种情况下是含糊不清的。终端用户应该确保对给定的 local ID 在网路上不同的机子上是兼容的。

| FROM… | I1 | UI1 | I2 | UI2 | I4 | UI4 | R4 | R8 | CY | DATE | BSTR | BOOL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| TO… | | | | | | | | | | | | |
| I1 | OK | OK(7) | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1)(3) | (4) | OK |
| UI1 | OK(7) | OK | (1) | (1) | (1) | (1) | (1) | (1) | (1) | (1)(3) | (4) | OK |
| I2 | OK | OK | OK | OK(7) | (1) | (1) | (1) | (1) | (1) | (1)(3) | (4) | OK |
| UI2 | (1) | OK | OK(7) | OK | (1) | (1) | (1) | (1) | (1) | (1)(3) | (4) | OK |
| I4 | OK | OK | OK | OK | OK | OK(7) | (1) | (1) | (1) | OK | (4) | OK |
| UI4 | (1) | OK | (1) | OK | OK(7) | OK | (1) | (1) | (1) | OK | (4) | OK |
| R4 | OK | OK | OK | OK | OK | OK | OK | (1) | OK | OK | (4) | OK |
| R8 | OK | OK | OK | OK | OK | OK | OK | OK | OK | OK | (4) | OK |
| CY | OK | OK | OK | OK | OK | OK | (1) | (1) | OK | OK | (4) | OK |
| DATE | OK | OK | OK | OK | (1) | (1) | (1) | (1) | (1) | OK | (4) | OK |
| BSTR | OK | OK | OK | OK | OK | OK | OK | OK | OK | OK | OK | OK |
| BOOL | OK | OK | OK | OK | OK | OK | OK | OK | OK | OK | (4) | OK |

**标注：**

1. 转换的“死亡“， 例如，从 I4 到 I2 或者 从 R8 到 R4 是允许的，即使溢出了也是可能的。如果溢出出现了，一个溢出错误值就会被返回，这种情况下读的质量被设定为 BAD， 这种情况下写的目标值不会改变。
2. 注意内部的存储类型 VT_BOOL 是短整数，必须得值为 VARIANT_TRUE(0xFFFF-例如 ‘-1’ 被保存为短整数类型）和 VARIANT_FALSE（0）。当转换成布尔型时任何非零值被转换为 VARIANT_TRUE, 布尔型向其他有符号类型转换，VARIANT_TRUE 转换为-1，或者-0.1 无符号类型的话是转换为最大值。提醒，OPC 标准的转换是从布尔到 BSTR 不是真和假分别是 0 和-1，入宫服务器选择转换为真和假必须告诉本地机器原因。同样应该注意标准的 C++关键字 true 当转换为其他类型的值时总是被转换为 1。因此当设定真到 boolVal 时，总是被设定为 VARIANT_FALSE 或 VARIANT_TRUE。
3. 注意 日期类型是作为双精度数存储的，整数部分是日，小数部分是时间，对日期来说 0.0 是 1899 年 12 月 30,（例如,1900 年 1 月 1 日是 2.0 和 2001 年 12 月 4 日是 37229.0）对时间来说，小数部分是从午夜 0 点开始计算来表示的,（例如 0.2500 表示上午 6:00，0.400 表示 上午 9：36：00 ） 这个小数是不会受到日期的符号影响的，无论是正值和负值都是从午夜 0 点算起的。例如 -1.4 是 1899 年 12 月 29 日 上午 9：36：00 这些转换是 VariantChangeTypeEx()支持的。一般当转换到 UI1，I1，UI2 或者 I2 时总会发生溢出错误。当日期转换到整数时作为时间的小数部分会丢失。
4. 串无意转换到目标类型时 BSTR 转换会给出 DISP_E_TYPE 错误。例如 1234 转换为日和数字类型（例如 UI1 会产生溢出）“12/04/2001”转换到 DATE（依据地区）不会转换为任何数字类型，ABCD 不转换为任何其他类型。

5. 从非整数到整数的变化必须考虑小数第一位大于0.5的进位。例如 1.6要进位到2,-1.6要进位到-2。除此之外，一般是期望小数大于或等于 0.5。然而，客户端和服务器端的开发者要注意 VariantChange 函数是不支持等于 0.5 进位规律的。实际应用的经验是浮点数和双精度数可以等于 0.5 进位的其他的会丢弃。等于 0.5 时为了一致无论进位和丢弃都要接受。
6. 要记住，货币类型时作为扩展的 8 字节整数形式存储的。小数点后为四位的精度。例如￥12.34 保存为 123400。
7. 在有符号和无符号整数之间转换的是时候，如果要求的类型不能对存放数值一个溢出就会产生。（例如：I1=-1 转换为 UI1 时就会溢出， UI1 类型的 255 转换为 I1 时就会溢出）。然而，一些通过 ChangeVariantType 进行的类似的转换是不正确的。 特别 是转换相同位的数据，是不不会检测溢出的。 例如 一个 I1 类型的-1 转换为 UI1 是 255，相似地 一个 UI1 类型的 254 转换为 I1 类型为-2。对 I2 和 I4 也是一样的。通过这个函数这样的行为是不正确的。 客户端的程序员和用户应该特别注意很多服务器是使用这个函数的，会出现这种现象的。推荐纠正这些但是不时需要 OPC 都一样。对不同位数的类型的转换这个函数是正确的 。
8. 各种类型之间的转换可能失去精度，（例如从 R8 到 R4,从 R4 到 I4 或 I2）但是不会有溢出，和返回的质量是 GOOD 。
9. 注意 ChangeVariantTypeEx 不能应用于数组，支持数组的服务器必须有另外的代码来实现逻辑转换。对数组说当任何元素出现了转换错误，第一错误检测就会返回（DISP_E_OVER 或 DISP_E_TYPE）,如果读质量被置为 BAD，一个空的变量就会返回。对写,如果任何一个元素出现了错误那么没有一个元素会被更改,第一错误产生就会返回 。

## 4.2.13 位置区域和区域 ID

就像这个文档中提到的一样，服务器的对位置区域的适应对开发者不是是什么难度。然而这些问题需要讨论一下。位置区域不仅地字符串错误和信息重要。他对值作为字符串读写同样重要。数字，日期，货币等都的格式都依靠位置。一般会期望客户端会查询服务器对位置区域的支持然会使用 SetlocalID()或者相似的函数来选择一个。注意:在有些 DataAcess 的使用状况里，组的 LocaleID 会根据服务器的需要,通过 IOPCCommon::SetLocaleID().能被设定的与默认的 LocaleID 不一样 。

客户端会希望服务器返回已经根据 LocaleID 转换和格式过的字符串，为了使这些数据对组的影响最小。这些字符串包括那些因为是 VARIANT 类型要转化为需要的数据类型的。服务器会用函数 VariantChangertypeEx()很轻易的完成这些变换。

同样地任何 OPC 服务器对象会希望客户端会传递一些根据客户端已经告诉服务器要对对象使用的 LocaleID 而被转换和格式过的字符串。这些包括 BSTRs 变成服务器需要转换的活动的数据类型 VARIANT。同样 服务器能用 VariantChangertypeEx()来实现这个功能。

例如 如果客户端告诉服务器当从一个特殊的对象读数据时要返回德语格式的字符串。那么自然服务器希望当客户端向服务器端的对象写数据时要传送德语格式的字符串。

## 4.2.14 条目的属性

**概览**

这些属性能被合法的 IOPCItemProperties 接口和被方法 IOPCBrowse::GetProperties

访问，这些接口被客户用来浏览与 ITEM ID 有关的可用属性和读这些属性的当前数值。有些方面与 EnumItemAttributes 和 SyncIO Read 函数提供的功能相似。这些接口有两个重要的区别 a)要求很容易使用。b)不可用作大量的数据访问。 既使期望允许程序用来很容易地浏览和读取特殊条目的额外详细的信息。

这个接口的设计是基于如多条目是由表示相关的值类似工程单位区间，描述，报警状态得条目组成的假设而设计的。比如系统会自己创建一个对当前复杂对象（PID 控制器，计时器，计数器等）的一个记录，这些记录条目会有些属性。(当前值，上限，下限，描述等)。

结果 ，这些方法允许灵活的，方便得方式去浏览，定位和读取相关的信息，而不用在底层做任何特殊的结构设计。

不用创建和管理任何 OPCGroups，这些信息同样允许被读写。

**属性怎样与 ItemIDs 关联的**

如多情况下希望能这些属性通过 ItemIDs 被访问例如 FIC101.HI_EU, FIC101.DESC, FIC101.ALMSTAT, 等 这些相关的 ITEMIDs 可能在一个 OPCGroup. 里被使用。这个方法提供一个方式来决定是否这个可选择的访问方法用来访问属性,如果要访问大量的信息被有效的获取的话。

象上面所述的一个系统（例如，一个有内部的记录组成的系统）有可能也会暴漏一个高等级的地址空间以作为分支的 A100 和作为子项的 A100.CV, A100.CV, A100.SP, A100.OUT, A100.DESC 的形式给 OPC。也就是说，一个条目的被记录的属性通常会被映射进一个较低及的 ITEMIDS. 另外一种被认为是这样的方法是 象 A100，有一些属性做为分支的的节点会在 OPC 浏览器里被显示，和一些属性作为子节点在 OPC 浏览器里被显示。

注：A100 条目可能实际上被嵌入一个高层次的 Plant.Building.Line 级别，然而这时我们不考虑，这个文章中没有涉及这些。

这个方法的目的是通过以下功能：

1. 被给出一个任何一个与属性相关的数字的 ITEMID（象 A100.CV 或 A100.DESC 或 甚至 A100），返回一个其他相关属性的清单。
2. 给一个条目属性的 ID 清单，返回一个当前数据值的清单。
3. 给一个条目属性 ID 的清单，返回一个可以在调用 AddGroup 里使用的条目清单。

需要注意的是前面的 8 个属性（OPC 的特殊被设定为 1 的属性）是特例的 如果这些条目被加的一个 OPC 组里他们表示数据会存在 OPCServer 里 或者 不能表示在一个底层系统实际的变量记录被命名的属性。

在这个设定里的一些属性与 OPCITEMATTRIBUTES 结构的子项保持一致。这些包括 CanonicalDataType, AccessRights, EUType and EUInfo，这些属性，还有那些表示数值的，时间戳的，质量的属性 应用于一个单独的和可用的条目（一个被大于 99 的属性 ID 表示），因此，OPC 的被设定为 1 的属性在接口的这个方法的行为不一样。参考每个方法的行为的描述定义。

**典型的使用**

这些方法的典型的客户应用可能用来获取一个条目甚至用那个过 IopCBrowse 来获得一个子项或者通过被使用者直接输入到一个编辑框。如果使用合法的 IOPCItemProperties，接着那个条目会被传递到 QueryAvailableProperties()。产生的结果会展示给用户。他会从列表里选择他想要看到的属性。客户程序会把这个设定传给 GetItemProperties ()来得到一个数据的快照。客户程序可以选择把这个设定传递给 LookupItemIDs 何使用条目设定的结果来创建个 OPCGroup 来反复的获得数据。 如果使用 IOPCBrowse::GetProperties 那么这个方法能被用来或其这个信息。

**例子**

这仅仅是个例子。不打算在任何服务器的实现上用任何特殊的结构。

一个典型的 OPC 条目可能是 FIC101.CV，这个可能是表示一个变量的当前的值，或者是被叫做是 FIC101 的功能块。这个功能块一般有一些其他的属性和他有关，例如，工程单位，循环描述等。这个功能块肯能也会有一些报警的界限和状态，设定值，调整参数和文档交叉索引一样，维护信息，帮助显示，默认的操作显示和一个其他属性无限的设定。所有的这些属性都通过他们与 FIC101 的关联而相互有关联。这个接口提供了一个方便的快捷方式来访问这些相关的属性。

例如一个模拟量输入的功能函数块可能有一个值，高低限值和状态。他们可能具有 F100.CV, F100.HI, F100.LO, F100.STAT 这样的条目。把这些条目传递给 IOPCItemProperties::QueryAvailableProperties 或者 IOPCBrowse::GetProperties 都会可能产生相同的结果。例如，一个 4 个属性 ID 的列表（另外 8 保留个属性 ID）每一个属性都保持一致。相同的属性 ID 有可能被返回无论 ItemID 是否被传递。传递这 4 个属性 ID 给 IOPCItemProperties::LookupItemIDs 或 IOPCBrowse::GetProperties 将也会每次都给出同样的结果因此他会返回这里提到的 ItemIDs。传递这些属性 ID 给 IOPCItemProperties::GetItemProperties 或者 IOPCBrowse::GetProperties 也会产生相同的特性的值。一个不同的东西会是，传递 8 个保留的属性 ID 中的任何一个都会获得与特殊的 ItemID 有关的信息(例如,会返回与作为 GetItemProperties 的第一个参数传递特殊的 ItemID 有关的数值，质量和时间戳等

一个 MMI 包 例如会用这些方法允许用户来表达 高低工程单位值用来缩放一个用来表示数据的棒图。

注意由于这些联系可能太多了和也可能是循环的，一个客户应用程序不希望自动地把他们都做出来。不期望这些属性的浏览是分层的。

另外的一个相似的例子可能是一个功能块例如每个对象都由各种属性组成的一个高端 PLC 里的计时器或者计数器

**属性 ID**

服务器需要给属性分配双字的 ID 代码。这个允许客户端很容易地管理它想要访问的属性列表。这些属性被分三系列（一些是分层的）。OPC 固定的设定包含了通过 OPCITEMATTRIBUTE 的返回来区分的那些属性，建议的设定是对大多数服务器都是通用的，开发者自定义的是包含适合的额外的属性。对头两个属性 ID 是固定的，用户自定义的属性使用的 ID 应该超过 5000.

**OPC 属性设定**

这时一套属性 ID 对大多是服务器都一样的，提供一致的属性的服务器必须使用一下列表中的 ID 编码。等他这些属性的代号在 OPCProps。H 文件里提供的有（查看这个文档的附录）

再次注意 这个接口不期望允许对大量数据的有效访问。

服务器的 LocaleID（通过 IOPCCommon::SetLocaleID 设定）被服务器用来定位返回的数据条目字符串的格式。字符串的描述不会被重新改变位置区域。

ID 设定为 1－-OPC 特殊的属性－-这些包括对系统来说直接与 OPC 服务器有关信息。

| ID（编号） | 返回的 VARIANT 格式 的数据类型 | 标准的解释 |
|---|---|---|
| 1 | VT_I2 | 条目的规定类型，条目变量以 I2 类型保存 |
| 2 | <各种类型> | 条目的数值，<br>（VARIANT）注意 返回的数据类型有以上的条目规 |

| | | |
|---|---|---|
| | | 定类型决定，并以来条目本身，这个像在读取设备数据的时候会表现出来。 |
| 3 | VT_I2 | 条目的品质<br>（OPCQUALITY 以 I2 格式存在），这个像在读取设备数据的时候会表现出来。 |
| 4 | VT_DATE | 条目的时间戳<br>FILETIME 格式转换来的。这个像在读取设备数据的时候会表现出来。 |
| 5 | VT_I4 | 条目的访问权限。<br>（OPCACCESSRIGHTS 以 I4 格式存在） |
| 6 | VT_R4 | Server 的扫描频率<br>以毫秒为单位，这代表了服务器从基础数据源中获取数据的最快速度。基础数据源没有被定义，但通常是 DCS 系统，SCADA 系统，plc，串口或者网络中的设备等，如果 item 加入到一个 group 中，这个值表示“最好情况”最快的刷新频率。<br>系统的负载和其他因素可以极大的影响此值的精度（服务器达到“最佳情况”的性能的能力） |
| 7 | VT_I4 | Item EU 类型<br>（OPCEUTYPE 存储在 I4）详细信息参见见 IOPCItemAttributes |
| 8 | VT_BSTR VT_ARRAY | ? |
| 9-99 | | 保留 OPC 将来使用 |

ID Set 2（ID 设定是 2）是推荐的属性，其实是多余的一个建议，这个对 ITEM 来说是最常用的属性，它也包括条目值的范围也被保留在将来其他的说明中会用到。想了解其他的新的 ID 设定的话请查询其它相关的基金会的说明。

OPC 基金会的职能就是，如果你有一个属性看上去与下面描述的很像的话，建议你使用下面的描述肯 ID 通过接口来表述你的属性。

**一个服务器能提供这些数值的任何子项（或者一个也不提供）**

| ID（编号） | 返回的 VARIANT 格式 的数据类型 | 标准的解释 |
|---|---|---|
| 100 | VT__BSTR | EU 单位<br>例如， DEGC 或者 GALLONS |
| 101 | VT__BSTR | 条目描述<br>例如： 蒸发 6 冷却剂温度 |
| 102 | VT__R8 | 高 EU<br>模拟量 上限的表达时使用， 在正常操作时的最大值，也在自动放缩一个棒图时会用到。例如： |

| | | |
|---|---|---|
| | | 1400.0 |
| 103 | VT__R8 | 低 EU<br>下限的表达时使用， 在正常操作时的最<br>大值，也在自动放缩一个棒图时会用到。例如<br>-200.0 |
| 104 | VT__R8 | 仪表最高范围<br>只有 模拟量数据会用到，用来表达仪表返回的最大<br>数值，例如 9999.9 |
| 105 | VT__R8 | 仪表最低范围<br>只有 模拟量数据会用到，用来表达仪表返回的最小<br>数值，例如 -9999.9 |
| 106 | VT_BSTR | 触点关闭标签<br>仅用来表示 离散的数据，表示一个与这个触点有关<br>的字符串，当它在关闭（非 0）状态。例如： 运行，<br>关闭 使能 等。 |
| 107 | VT_BSTR | 触点打开标签，<br>仅用来表示 离散的数据，表示一个与这个触点有关<br>的字符串，当它在关闭（0）状态。例如： 停止，打<br>开 非使能 等。 |
| 108 | VT_I4 | 时区条目<br>当你获取他的数值时 UTC 时间戳和 本地时间戳的<br>分钟格式是有区别的，查看 OPCGroup TimeBias<br>属性，也可看 WIN32 TIME ZONE INFORMATION<br>结构。 |
| 109-199 | | 保留 OPC 将来使用 |
| | | |
| | | 报警和状态值相关属性<br>Id 300-399 是用于 OPC 报警和事件的。<br>详细信息参见<br>OPC Alarm and Events specification |
| 300 | VT_BSTR | 条件状态<br>与 item 相关的当前的报警或条件状态<br>例如：“正常，活动，高报警” |
| 301 | VT_BSTR | 报警快速帮助<br>报警产生时，为操作员提供一套的简单的操作指令的<br>一段字符串 |
| 302 | VT_BSTR<br>\|VT_ARRAY | 报警域列表<br>包含 TeamID 的字符串数组，代表设备或报警区域<br>的， |
| 303 | VT_BSTR | 主报警区域<br>包含了 TeanID 的字符串，代表主设备或报警区域 |
| 304 | VT_BSTR | 逻辑条件<br>一个任意的字符串描述所执行的测试。<br>例如：超过最高限制，TAG.PV>=TAG.HILIM |

| | | |
|---|---|---|
| 305 | VT_BSTR | 极限条件<br>多状态报警条件<br>如：高高 高 低低 低 |
| 306 | VT_R8 | 死区 |
| 307 | VT_R8 | 高高限 |
| 308 | VT_R8 | 高限 |
| 309 | VT_R8 | 低限 |
| 310 | VT_R8 | 低低限 |
| 311 | VT_R8 | 变化率的限制 |
| 312 | VT_R8 | 偏差限制 |
| 313 | VT_BSTR | 声音文件<br>如 C:\MEDIA\FIC101.WAV, or .MID |
| 314-399 | | 保留用于 opc AE |
| 400-4999 | | 保留用于 opc，在不修改接口的情况下增加 ID |

注：OPC 基金会有权扩大这一名单时的权利。客户应做好应对这一点。

ID 设定为 3 供应商特定的属性

| | | |
|---|---|---|
| 5000 | VT_xxx | 供应商特定属性<br>这些属性的 ID 号必须大于等于 5000，可以是非连续的。但是数据类型必须与 VARIANT 对应 |

客户端必须小心的处理这些供应商特定的属性 ID，不能假定它们。不同的供应商可能无法为 5000 以上的 ID 提供相同的信息。

## 4.2.15 IOPCSyncIO（同步）

| 接口函数 | 数据源 | 使用订阅回调函数 | Group 的活动状态 | Item 的活动状态 | 服务器反应 |
|---|---|---|---|---|---|
| Read | Cache | NA | Actice | Actice | 从服务器缓存中返回客户所需 Item 的值与质量 |
| Read | Cache | NA | Actice | InActice | 返回 Item 的值时，同时返回 OPC_QUALITY_OUT_OF_SERVICE 的质量 |
| Read | Cache | NA | InActice | NA | 返回 Item 的值时，同时返回 OPC_QUALITY_OUT_OF_SERVICE 的质量 |
| Read | Device | NA | NA | NA | 返回 Item 的值和质量，服务器以指定的更新率从设备更新数据 |

注：活动状态与数据源对同步读的影响

## 4.2.16 IOPCASyncIO2（异步）

| 接口函数 | 数据源 | 使用订 | Group 的 | Item 的活 | 服务器反应 |
|---|---|---|---|---|---|

| | | 阅回调函数 | 活动状态 | 动状态 | |
|---|---|---|---|---|---|
| Read | NA | NA | NA | NA | 经 IOPCDataCallback::OnReadComplete 从服务器缓存中返回客户所需项的值与质量，从设备读取数据，并更新缓存 |
| Refresh2 | Cache | NA | Actice | Actice | 通过 IOPCDataCallback::OnDataChange 从服务器缓存中返回处于活动状态的全部项 |
| Refresh2 | Cache | NA | InActice | NA | 非活动状态的 Items 不返回，若全部 Items 都为不活动状态，返回 E_FAIL |
| Refresh2 | Cache | NA | NA | NA | 返回 E_FAIL |
| Refresh2 | Device | NA | Active | Active | 经 IOPCDataCallback::OnReadComplete 从服务器缓存中返回客户所需项的值与质量，从设备读取数据，并更新缓存 |
| Refresh2 | Device | NA | Active | Active | 非活动状态的项不返回，若全部项都为不活动状态，返回 E_FAIL |
| Refresh2 | Device | NA | InActive | NA | 返回 E_FAIL |

注：活动状态与数据源对 IOPCAsyncIO2 的影响

## 4.2.17 通过 IOPCDataCallback 订阅

| 接口函数 | 数据源 | 使用订阅回调函数 | Group 的活动状态 | Item 的活动状态 | 服务器状态 |
|---|---|---|---|---|---|
| **OnDataChange** | NA | TRUE | Active | Active | 通过 IOPCDataCallback::OnDataChange 返回客户所需项的值与质量，从设备读取数据，并更新缓存 |
| **OnDataChange** | | TRUE | Active | InActive | 从物理设备获取处于活动状态的 Items |
| **OnDataChange** | | TRUE | Inactive | NA | 从物理设备获取处于非活动状态的 Group 的活动状态项 |
| **OnDataChange** | NA | FALSE | active | active | 通过 IOPCDataCallback::OnDataChange 返回客户所需项的值与质量，从设备读取数据，并更新缓存 |
| **OnDataChange** | NA | FALSE | active | Inactive | 从物理设备获取处于活动状态的 Items |
| **OnDataChange** | NA | FALSE | Inactive | NA | 从物理设备获取处于非活动状态的 |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | Group 的活动状态项 |

# 4.3 OPCServer 对象

## 4.3.1 概述

该 OPCServer 的对象是 OPC 服务器提供的主要对象。这个对象提供的接口包括：

- IUnknown
- IOPCServer
- IOPCBrowse (new)
- IOPCItemIO (new)
- IconnectionPointContainer

本节将描述上述接口所提供的功能。

注：规范 1.0 中 IenumUnkown 作为一个 opcServer 的接口列出的，这个错误已删除。QueryInterface 的语法是不允许这样实现的，正确的方法是通过 IOPCServer::CreateGroupEnumerator 获得一组枚举的。

## 4.3.2 IUnknown

服务器必须提供一个标准的 IUnknown 接口。由于这是一个良好定义的接口，不详细讨论。更多的信息参见 OLE 程式设计人员参考。微软要求必须实现接口的所有功能。..

## 4.3.3 IOPCCommon

Opc 规范如 Alarms and Events 等 opcServer 对象的公共接口，通过该接口函数，可以设置和查询组件应用程序的位置标识 localID,从而实现客户应用程序与服务器的有效会话，且客户程序间不受干扰。

下面提供了这个接口快速参考。更详细的讨论在 OPC 概述文档中可以找到。

```
HRESULT SetLocaleID (
[in] LCID dwLcid
);
HRESULT GetLocaleID (
[out] LCID *pdwLcid
);
HRESULT QueryAvailableLocaleIDs (
[out] DWORD *pdwCount,
[out, sizeis(, *pdwCount)] LCID **ppdwLcid
);
HRESULT GetErrorString(
[in] HRESULT dwError,
```

[out, string] LPWSTR *ppString
);
HRESULT SetClientName (
[in, string] LPCWSTR szName
);

### 4.3.4 IOPCServer

IOPCServer 是 opcServer 的主接口，通过它实现 OPCServer 在操作系统中的安装和注册。此接口是必须被实现的，其方法也必须被实现。

#### 4.3.4.1 IOPCServer::AddGroup

```
HRESULT AddGroup(
        [in, string] LPCWSTR szName,
        [in] BOOL bActive,
        [in] DWORD dwRequestedUpdateRate,
        [in] OPCHANDLE hClientGroup,
        [unique, in] LONG *pTimeBias,
        [in] FLOAT * pPercentDeadband,
        [in] DWORD dwLCID,
        [out] OPCHANDLE * phServerGroup,
        [out] DWORD *pRevisedUpdateRate,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN * ppUnk
        );
```

描述：
增加一个 Group 到 Server 中

| 参数 | 描述 |
|---|---|
| szName | Group 的名称。此名称必须唯一，如果为空，服务器为生成一个唯一的名称 |
| bActive | FALSE Group 被定义为不活动的<br>TRUE Group 被定义为活动的 |
| dwRequestedUpdateRate | 客户程序定义的Group最快的刷新速率，单位为毫秒，主要在 OnDataChage 中使用。同样说明了要求缓存数据的期望速率。如果此值为0，那么服务器必须以最快的实际速率刷新。 |
| hClientGroup | 客户程序提供的 Group 句柄。【更多信息参看数据类型、参数、和结构体的介绍】 |
| pTimeBias | 时间戳指针。如果指针为空表示你希望使用系统默认时间戳<br>时间戳的常规属性请参阅下面注释内容 |
| pPercentDeadband | 定义了死区参数。这个参数仅仅对本组内的模拟量有效。当模拟 |

| | |
|---|---|
| | 量的值变化超出死区，必须产生一个回调给客户程序。NULL 值代表0.0，即没有死区。 |
| dwLCID | 本组字符串操作时，服务器使用的语言 |
| phServerGroup | 服务器为新的组产生的句柄。客户程序将采用服务器提供的句柄来要求服务器对组进行其它的操作。 |
| pRevisedUpdateRate | 服务器返回的实际刷新速率，这个值可能与客户程序要求的刷新速率不一样。注意：这个值可能也比服务器实际从设备获取数据和刷新CACHE 的速率要慢。 |
| riid | 期望接口类型 (如 IID_IOPCItemMgt) |
| ppUnk | 返回接口指针，如果操作失败，返回NULL |

返回码

| 返回码 | 描述 |
|---|---|
| S_OK | 操作成功 |
| E_FAIL | 操作失败 |
| E_OUTOFMEMORY | 没有足够的内存空间 |
| E_INVALIDARG | 功能非法 |
| OPC_E_DUPLICATENAME | 名称重复 |
| OPC_S_UNSUPPORTEDRATE | 服务器不支持指定的刷新速率，服务器返回实际支持的刷新速率（pRevisedUpdateRate） |
| E_NOINTERFACE | 服务器不支持请求的接口 |

**注：**

一个 Group 是为客户组织操作数据项的逻辑容器。

服务器创建一个 Group 对象，并且返回一个客户要求的接口指针。如果客户要求一个任选的接口，而服务器并不支持这个接口，服务器会返回一个接口不支持的错误码。

被请求的刷新速率和修正后的刷新速率必须确定在服务器和客户的对话间。客户创建的组对象，按照修正后的刷新速率刷新，而不依赖于项的多少，Item 的添加，删除也没有影响。

**注释：**

对象的生存周期按以下定义。即使所有的接口都被释放，Group 对象仍然不会被删除，直到 RemoveGroup 被调用。在所有的界扩没有被释放前，客户不应该调用 RemoveGroup 来删除所有的私有组。

由于服务器被认为组的容器，因此服务器可以在服务器的接口被释放时强制删除所有存在的组对象。

DwLCID 完全由服务器指定。服务器可以忽略掉这个参数，如果不支持动态的语言定义。

### 4.3.4.2 IOPCServer::GetErrorString

```
HRESULT GetErrorString(
      [in] HRESULT dwError,
      [in] LCID dwLocale,
      [out, string] LPWSTR *ppString
      );
```

描述：返回一个服务器的特定错误代码的错误字符串

| 参数 | 描述 |
|---|---|
| DwError | 客户应用程序重服务器接口返回的错误码 |
| dwLocale | 返回的字符串的语言环境。 |
| ppString | 指向服务器提供的保存结果的指针 |

返回码

| 返回码 | 描述 |
|---|---|
| E_FAIL | 操作失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 功能非法 |
| S_OK | 操作成功 |

**注释：**

这基本上是一个相同的函数作为新的发现，IOPCCCommon

如果这个函数在远程函数上被调用，可能会产生一个 RPC 错误。这可能是由于客户端尝试调用本地的 Win32 函数失败造成的。

客户端必须释放返回的字符串。

建议在服务器把任何 OPC 特定字符串到一个外部资源，以简化翻译。

为了获得系统的缺省值，dwLocale 必须是 LOCALE_SYSTEM_DEFAULT.

### 4.3.4.3 IOPCServer::GetGroupByName

```
HRESULT GetGroupByName(
        [in, string] LPCWSTR szName,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN * ppUnk
        );
```

描述：通过 Group 名字来获得组的接口

| 参数 | 描述 |
|---|---|
| szName | Group 的名称。Group 必须已被调用者创建 |
| Niid | 期望接口类型 (如 IID_IOPCItemMgt) |
| ppUnk | 返回接口指针，返回 NULL 除了 S_OK 以外 |

返回值

| 返回码 | 描述 |
|---|---|
| E_FAIL | 操作失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 功能非法 |
| S_OK | 操作成功 |
| E_NOINTERFACE | 服务器不支持此接口 |

**注释：**

此函数可以用于连接到一个所有已发布的 Group 的接口，在完成后客户端必须释放返回的接口

如果需要，客户可以通过 OPCGroupStateMgt::GetState.获得 hServerGroup 句柄。

### 4.3.4.4 IOPCServer::GetStatus

```
HRESULT GetStatus(
        [out] OPCSERVERSTATUS ** ppServerStatus
        );
```

描述：返回服务器额当前状态信息。

| 参数 | 描述 |
|---|---|
| ppServerStatus | 只向OPCSERVERSTATUS 结构体指针，结构体又服务器创建 |

| 返回值 | 描述 |
|---|---|
| E_FAIL | 只向OPCSERVERSTATUS 结构体指针，结构体又服务器创建 |

**注释：**

OPCSERVERSTATUS 在本章后面介绍。

客户端必须释放结构以及该结构内的 VendorInfo 字符串

客户端定期调用 **GetStatus** 方法可以确定服务器连接正常。

### 4.3.4.5 IOPCServer::RemoveGroup

```
HRESULT RemoveGroup(
        [in] OPCHANDLE hServerGroup,
        [in] BOOL bForce
        );
```

描述：删除 Group 对象

| 参数 | 描述 |
|---|---|
| HserverGroup | 被删除Group的句柄。 |
| Bforce | 强制删除标志，即使 Group 对象仍然在引用中。 |

返回值：

| 参数 | 描述 |
|---|---|
| E_FAIL | 操作失败。 |
| E_OUTOFMEMO RY | 没有足够的内存。 |
| E_INVALIDARG | 功能非法。 |
| S_OK | 操作成功。 |
| OPC_S_INUSE | Group没有被删除，因为引用存在。Group将被标记为删除，当Group的所有引用释放后，服务器会自动删除Group。 |

**注释：**

如前所述，一个组不会被删除，当所有的客户接口被释放后。客户程序可以调用 GetGroupByName 在所有的接口被释放后。RemoveGroup()可以使服务器释放对组最后的引用，从而真正的删除组。

一般而言，一个好的客户程序只在释放所有接口后才调用此功能。如果接口还存在，Remove group 将把组标记为删除。任何这个组的接口调用将会返回 E_FAIL。当所有的接口被释放后，组将会被自动删除。如果 bForce 的值为 TRUE,那么组将被无条件的删除，即使引用仍然存在，那些接口的使用将会导致一个非法存取。这个功能不能被公用的组调用。

## 4.3.4.6 IOPCServer::CreateGroupEnumerator

```
HRESULT CreateGroupEnumerator(
        [in] OPCENUMSCOPE dwScope,
        [in] REFIID riid,
        [out, iid_is(riid)] LPUNKNOWN* ppUnk
        );
```

描述：服务器上所提供的 Group 建立不同的列举器

| 参数 | 描述 |
|---|---|
| dwScope | 指示要列举的类组<br>OPC_ENUM_PRIVATE_CONNECTIONS或<br>OPC_ENUM_PRIVATE枚举所有客户端创建的Group<br>OPC_ENUM_ALL_CONNECTIONS或<br>OPC_ENUM_ALL枚举所有的私有Group。。 |
| riid | |
| ppUnk | |

返回值：

| 参数 | 描述 |
|---|---|
| E_FAIL | 操作失败。 |
| E_OUTOFMEMO RY | 没有足够的内存。 |
| E_INVALIDARG | 功能非法。 |
| S_OK | 操作成功。 |
| S_FALSE | 没有什么枚举(没有满足请求的组)。然而空枚举器仍返回,必须被释放。注意：在规范的早期版本中出现了关于S_FALSE的情况下的行为有些含糊不清。由于这个原因,建议当服务器返回S_FALSE, 客户端就可以在发布Release之前测试返回的接口指针是否为NULL |
| E_NOINTERFACE | 服务器不支持此接口 |

**注释：**

连接是指一个接口指针存在

IEnumUnknown 为每个组中的枚举创建一个额外的接口指针（即使客户端已经和组建立了连接)。在 IEnumUnknown 的情况下(COM 规范),客户端必须释放所有返回的 IUnknown 指针。

## 4.3.5 IConnectionPointContainer (on OPCServer)

这个接口提供了 IOPCShutdown 访问连接点。在此不再赘述 ConnectionPoints 的一般原则,在 Microsoft 文档中有详细说明。假设读者熟悉这一技术。OPC DA 标准服务器,由 OPC2.0 或更高，都必须支持这个接口。同样，IEnumConnectionPoints 的细节，IConnectionPoint 和 IEnumConnections 接口由微软定义，在此不再赘述。

**注**：在服务器和客户端之间，OPC 标准服务器不需要支持多个连接。考虑到服务器客户端特定实体预计单个连接几乎所有应用程序就足够了。出于这个原因（按照 COM 规范）用于 IOPCShutdown 的 IConnectionPoint 接口的 EnumConnections 方法被允许返回 E_NOTIMPL。

### 4.3.5.1 IConnectionPointContainer::EnumConnectionPoints

```
HRESULT EnumConnectionPoints(
        IEnumConnectionPoints **ppEnum
        );
```

描述：为 OPC Group 和客户端之间的连接点创建枚举器

| 参数 | 描述 |
|---|---|
| ppCP | 存储连接点的指针 |
| riid | 连接点的IID（如：IID_IOPCShutdown ） |

返回值：

| 返回值 | 描述 |
|---|---|
| S_OK | 操作成功 |
| 其他返回直接减OLE文档 | |

**注释**：OPC 服务器必须支持 IID_IOPCShutdown，其他供应商特定的回调也是允许的

### 4.3.5.2 IConnectionPointContainer:: FindConnectionPoint

```
HRESULT FindConnectionPoint(
    REFIID riid,
    IConnectionPoint **ppCP
    );
```

描述：找到客户端和服务器之间特定的连接点

| 参数 | 描述 |
|---|---|
| ppCP | 存储连接点的指针 |
| riid | 连接点的IID（如：IID_IOPCShutdown ） |

| 返回值 | 描述 |
|---|---|
| S_OK | 操作成功 |

| | |
|---|---|
| 其他返回直接减OLE文档 | |

**注释**：OPC 服务器必须支持 IID_IOPCShutdown，其他供应商特定的回调也是允许的

## 4.3.6 IOPCBrowse

IOPCBrowse 接口提供了改进的浏览服务器地址空间的方法和获取条目属性。这个接口中的方法被设计成镜中相应的方法 XML-DA 接口

### 4.3.6.1 IOPCBrowse:: Browse

```
HRESULT Browse(
        [in, string] LPWSTR szItemID,
        [in,out, string] LPWSTR * pszContinuationPoint,
        [in] DWORD dwMaxElementsReturned,
        [in] OPCBROWSEFILTER dwBrowseFilter,
        [in, string] LPWSTR szElementNameFilter,
        [in, string] LPWSTR szVendorFilter,
        [in] BOOL bReturnAllProperties,
        [in] BOOL bReturnPropertyValues,
        [in] DWORD dwPropertyCount,
        [in, size_is(dwPropertyCount)] DWORD * pdwPropertyIDs,
        [out] BOOL * pbMoreElements,
        [out] DWORD * pdwCount,
        [out, size_is(,*pdwCount)] OPCBROWSEELEMENT ** ppBrowseElements
    );
```

描述：浏览地址空间的一个分支,并返回零个或多个 OPCBROWSEELEMENT 结构
详细见原文

| 参数 | 返回值 |
|---|---|
| szItemID | 浏览时层次结构中的分支名称，如果根分支被浏览，则传递一个 Null 字符串 |
| pszContinuationPoint | 见原文 |
| dwMaxElementsReturned | 见原文 |
| dwBrowseFilter | 枚举{All, Branch, Item }指定浏览元素返回的子集。请参阅下表在下面的评论部分，以确定在 OPCBROWSEELEMENT 位的组合:: dwFlagValue 返回在 dwBrowseFilter 每个值。 |
| szElementNameFilter | 符合 Visual Basic 的 LIKE 操作符，这将被用于过滤元素名称的通配符字符串。一个 NUL 字符串意味着没有过滤器。 |
| szVendorFilter | 服务器特定的过滤器字符串。这是完全自由的形式，而可以由用户通过编辑字段中输入。一个指向 NUL 字符串表示没有过滤。 |
| bReturnAllProperties | 服务器必须返回所有这些可为每个返回的元素的属性。如果为 true， |

| | |
|---|---|
| | pdwPropertyIDs 被忽略。 |
| bReturnPropertyValues | 除了属性名外的所有属性。 |
| dwPropertyCount | pdwPropertyIDs 数组的大小 |
| pdwPropertyIDs | 每个元素的属性 ID 的数组。如果 bReturnAllProperties 是 TRUE，pdwPropertyIDs 被忽略，所有的属性都返回。 |
| pbMoreElements | 如果服务器不支持一个延续点，那么服务器将设置 pbMoreElements 为 True，如果有比 dwMaxElementsReturned 更多的元素。 |
| pdwCount | ppBrowseElements 数组返回的大小 |
| ppBrowseElements | 服务器提供的 OPCBROWSEELEMENT 数组 |

返回值

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| S_FALSE | 操作成功，但是在**ppItemProperties**中有错误。<br>OPCITEMPROPERTIES 结构是 S_FALSE.则返回 S_FALSE |
| OPC_E_UNKNOWNITEMID | Item在服务器的地址不可用 |
| OPC_E_INVALIDITEMID | Item ID 错误 |
| OPC_E_INVALIDFILTER | 过滤字符串无效 |
| OPC_E_INVALIDCONTINUATIONPOINT | 指定的接续点是无效的 |
| E_OUTOFMEMO RY | 没有足够的内存。 |
| E_INVALIDARG | 功能非法。 |
| E_FAIL | 操作失败。 |

**内容:**

具体内容见 6.7.10 的 OPCBROWSEELEMENT structure.章节

如果指定的级别 szItemID 是有效的,但没有任何子节点,然后浏览会成功,但结果将是一个零长度 ppBrowseElements 数组。

如果过滤标准导致一个空的结果，那么浏览仍将成功

下表提供了 OPCBROWSEELEMENT:: dwFlagValue 和 dwBrowseFilter 之间的关系值，OPC_BROWSE_HASCHILDREN 和 OPC_BROWSE_ISITEM 缩写为 HasChildren 和 IsItem:

| **dwFlagValue** | | | **dwBrowseFilter** | | |
|---|---|---|---|---|---|
| IsItem | HasChildren | 描述 | All | Branch | Item |
| 0 | 0 | 空的分支 | • | • | |
| 0 | 1 | 有子节点的分支 | • | • | |
| 1 | 0 | 没有分支的 Item | • | | • |
| 1 | 1 | 一个有子节点分支或分支的 Item | • | • | • |

## 4.3.6.2 IOPCBrowse::GetProperties

```
HRESULT GetProperties(
    [in] DWORD dwItemCount,
```

```
        [in, string, size_is(dwItemCount)] LPWSTR * pszItemIDs,
    [in] BOOL bReturnPropertyValues,
    [in] DWORD dwPropertyCount,
    [in, size_is(dwPropertyCount)] DWORD * pdwPropertyIDs,
    [out, size_is(,dwItemCount)] OPCITEMPROPERTIES **ppItemProperties
);
```

描述：返回 OPCITEMPROPERTIES 的数组，一个 item 一个

| 参数 | 描述 |
|---|---|
| dwCount | pszItemIDs 和ppItemProperties数组的大小 |
| pszItemIDs | item IDs 属性数组 |
| bReturnPropertyValues | 服务器返回出属性名外的属性值。 |
| dwPropertyCount | pdwPropertyIDs 数组的大小，0则表示pdwPropertyIDs 数组为空 |
| pdwPropertyIDs | 属性id数组,返回数据。如果为空,则将返回所有可用的属性 |
| ppItemProperties | 服务器返回的OPCITEMPROPERTIES数组 |

返回值

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| S_FALSE | 操作成功，但是在**ppBrowseElements**中有错误。 |
| E_OUTOFMEMO RY | 没有足够的内存。 |
| E_INVALIDARG | 功能非法。 |
| E_FAIL | 操作失败。 |

**注释**：详细信息见 OPCITEMPROPERTIES structure

## 4.3.7 IOPCItemIO

这个接口为访问 OPC 数据提供了一个简单的方法。程序员应该知道,在大多数服务器上，基于 opc 接口的设计可以提供比这个接口更好的性能。在性能方面，该接口的用户应该假设它的行为就好像他要创建一个 Group，添加 Item，执行一个读或写，然后删除该 Group。这个接口的另一个目的是提供一种方法以便写作时间戳和质量信息到服务器。

### 4.3.7.1 IOPCItemIO::Read

```
HRESULT Read (
        [in] DWORD dwCount,
        [in, sizeis(dwCount)] LPCWSTR *pszItemIDs,
        [in, sizeis(dwCount)] DWORD *pdwMaxAge,
        [out, sizeis(,dwCount)] VARIANT **ppvValues,
        [out, sizeis(,dwCount)] WORD **ppwQualities,
        [out, sizeis(,dwCount)] FILETIME **ppftTimeStamps,
```

[out, sizeis(,dwCount)] HRESULT **ppErrors
);

描述：读取一个或多个 Item 的值、质量戳、时间戳。类似于 IOPCSyncIO::Read 的方法。

| 参数 | 描述 |
|---|---|
| dwCount | 将要读取的Item数目 |
| pszItemIDs | 可能会从IOPCBrowse:: GetProperties获得完全合格的ItemIDs.These列表 |
| dwMaxAge | 读取Item的更新时间，以毫秒计算。服务器将会计算现在和上一个时间戳之间的毫秒数，如果在超过MaxAge，必须从底存获取。或者不能从缓存获取的，也必须从底存设备上获取。0相当于OPC_DS_DEVICE，0xFFFFFFFF相当于OPC_DS_CACHE。不在服务器缓存的，将从设备直接读取，这就和maxage无关了。一些服务器为所有客户维护全局缓存。如果在这个缓存中有需要的Item，服务器利用它来检查的MaxAge值。服务器不应该自动创建或更改缓存的值。 （注：由于这是毫秒一个DWORD，最大的MaxAge价值将约49.7天）。 |
| ppvValues | 返回结果的数组指针，在接受后客户端必须释放它包含的variants |
| ppwQualities | 存储每个结果质量的数组，这些必须被客户端释放。 |
| ppftTimeStamps | 存储每个结果的时间戳数组。这些必须被客户端释放。 |
| ppErrors | 一系列HRESULTs表示个别项目的成功读取。对应的错误pszItemIDs ItemIDs传递。这表明阅读是否成功地获得定义值,质量和时间戳。注意任何失败错误代码显示相应的价值,质量和时间戳是未定义的。注意,这些必须被客户端释放。。 |

返回值：

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| S_FALSE | 该操作成功，但也有在ppErrors一个或多个错误。请参阅单独的错误获得更多信息。 |
| E_INVALIDARG | 参数无效 |
| E_FAIL | 操作失败。 |
| E_OUTOFMEMO RY | 没有足够的内存。 |

**ppErrorCodes**

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| E_FAIL | 操作失败。 |
| OPC_E_BADRIGHTS | Item不可读 |
| OPC_E_UNKNOWNITEMID | Item在服务器的地址不可用 |
| OPC_E_INVALIDITEMID | Item ID 错误 |
| S_xxx<br>E_xxx | S_xxx可以有供应商提供，不仅仅是GOOD<br>E_xxx供应商特定的错误，如果这个项目不能被访问。<br>这些供应商的特定代码可以被传递到GetErrorString（）。 |

**注释：**

MaxAge 只会子在这个函数被调用时才测试，例如 MaxAge 为 1000 毫秒 3 个 Item 被读取，其中两个在 MaxAge 范围内，那第三个 Item 将从设备获取。一旦获得了该 Item 的时间可能比要求的的 MaxAge 长，那么第三个 Item 将被打包返回到客户端。在这种情况下，还在 MaxAge 内的两个 Item 就是“旧”的。该测试下的 MaxAge 不会被重新计算，因此，两个“过时”的项目将与直接从设备得到的产品被退回。此功能在的地方，防止服务器试图递归获取值。
如果设备=缓存，一些服务器可能返回实际值。
如果返回值是 S_OK 则 ppErrors 将被忽略。
如果返回值是 S_FALSE，ppErrors 将代表每个 Item 的状态。
如果 HRESULT 是任何失败的代码，那么所有 OUT 参数的内容，包括 ppErrors 是不确定的。
对于任何 S_xxx 的结果，ppError 客户端可能会有相应的值，质量和时间戳被定义良好的，虽然质量不确定的。我们建议（但不是必须）服务器供应商在这里提供有关不确定 Item 的其他信息。
客户端必须释放所有输出的参数。

### 4.3.7.2 IOPCItemIO::WriteVQT

```
HRESULT WriteVQT (
        [in] DWORD dwCount,
        [in, sizeis(dwCount)] LPCWSTR *pszItemIDs,
        [in, sizeis(dwCount)] OPCITEMVQT *pItemVQT,
        [out, sizeis(,dwCount)] HRESULT **ppErrors
        );
```

描述：为指定的 Item 写一个或多个值,质量和时间戳。这是在功能上类似于 IOPCSyncIO2:WriteVQT 除了没有相关的 Group。如果客户端试图写 VQ,VT 或 VQT，它应该预料服务器将它们写全或无的。

| 参数 | 描述 |
|---|---|
| dwCount | 将要写入的Item的ID |
| pszItemIDs | 要写入ItemIDs列表。 |
| pItemVQT | OPCITEMVQT结构的列表。 |
| ppErrors | 产生的错误列表(一个Item一个)。注意,这些必须被客户端释放。 |

返回值：

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| S_FALSE | 该操作成功，但也有在ppErrors一个或多个错误。请参阅单独的错误获得更多信息。 |
| E_INVALIDARG | 参数无效 |
| OPC_E_NOTSUPPORTED | 如果客户端尝试写任何价值，质量，时间戳组合和服务器不支持请求的组合（这可能是一个单一的量，如只是时间戳），那么服务器将不执行任何写操作，将返回此错误代码。 |
| E_FAIL | 操作失败。 |

| | |
|---|---|
| E_OUTOFMEMO RY | 没有足够的内存。 |

**ppErrorCodes**

| 参数 | 描述 |
|---|---|
| S_OK | 操作成功。 |
| E_FAIL | 操作失败。 |
| OPC_S_CLAMP | 值被接受，但被夹住 |
| OPC_E_RANGE | 值超限了 |
| OPC_E_BADTYPE | 这个Item不支持数据类型 |
| OPC_E_BADRIGHTS | Item不可写 |
| OPC_E_UNKNOWNITEMID | Item在服务器的地址不可用 |
| OPC_E_INVALIDITEMID | Item ID 错误 |
| E_xxx<br>S_xxx | 可以返回供应商特定的返回码。 |

**注释：**如果返回 S_OK，ppErrorCodes 可以忽略（所有的结果保证是 S_OK），然后如果 HRESULT 任何 FAILED 的代码的内容 ppErrors 是未定义的。
作为 OPC_E_BADTPYE 的替代，可以接受服务器返回由 VariantChangeType 或 VariantChangeTypeEx 返回任何失败的错误。

## 4.4 OPC 的 Group 对象

OPCServer 向 OPCGroup 对象投递管理 Item 对象的请求。Group 对象支持的接口为：

- IUnknown
- IOPCGroupStateMgt
- IOPCGroupStateMgt2(new)
- IOPCItemMgt
- IOPCSyncIO
- IOPCSyncIO2 (new)
- IOPCAsyncIO2
- IOPCAsyncIO3(new)
- IOPCItemDeadbandMgt (new)
- IOPCItemSamplingMgt(new/optional)
- IconnectionPointContainer

本章详细定义了以上接口规范，并且定义了关于 OLE 实现 COM 接口嵌入的一些技术细节。

### 4.4.1 OPCGroup

对象包含了特定的属性和行为会影响接口的一些操作，这里提前介绍以防赘述。

#### 4.4.1.1 Group 对象命名

每个 Group 对象必须保证在本 OPC 客户端内不存在重名的情况。并且本客户端范围内可以重新命名 Group 对象。对象命名是区分大小写的。比如 Group1 和 group1 就是两个不同的 Group 名称。

#### 4.4.1.2 数据缓冲

一些方法可以指定数据操作是在缓存中还是在设备中执行。实际上大部分的 OPCServer 实现了一系列的缓存机制。前面已经讨论过，这些主题也是接口规范的一部分。然而两种方式下函数的功能性差别甚微，具体的实现细节不同的 OPC 提供商也不尽相同。多数情况下，使用缓存机制的 OPCserver 效率更高而直接操作设备的机制相对效率较低但更精确。友情提示：尽管本文强烈建议使用缓存机制但是非本规范强制要求。

#### 4.4.1.3 激活标志

Group 对象和 Item 对象都有对应的 Active Flag。两者的标志是相互独立的，改变 Group 对象的标志并不会影响到 Item 对象。

规范中的大部分内容将 Active Flag 解释为一种抽象表示。然后该标志将会在可控的范围内影响接口的一些行为。具体的实现细节方面本规范不做要求。

实际上大部分服务器程序使用 Activeflag 来优化 Cpu 使用率和通讯占用的资源量。例如处于非激活态的 Group 或者 Item 项可以移除缓存区。

对客户端程序而言，可以通过设置该标志来避免执行添加和移除操作。比如一个仅仅用来显示数据的应用处于最小化状态的情况下，简单的让 Items 处于非激活态即可。

参考前面的 Data Acquisition and Active State Behavior Summary 一节以获取更加详细的信息。

客户端数据的“变更”会影响到激活的 Group 内的激活的 Items.所谓的变更是值上一次发给客户端的数据值现在发生了改变或者质量戳发生变化。OPCServer 端通过 OnDataChange 回调函数传递这些数据。后面章节会详细描述。

#### 4.4.1.4 刷新频率

客户端程序可以配置 Group 对象的刷新频率，即数据的检测周期。周期一到，缓存区就会更新。服务端会最大限度的确保数据更新。同时这也会影响到客户端程序的回调函数的被调用率。当然，服务端程序可以非常肯定的保证不会在客户端请求的周期内调用回调接口。

重点内容：

需要指出的是，这种操作不应当影响到 Server 端后台程序的处理。比如一个 OPC 设备以 0.05s 的速率执行一个 PID 控制的工作，如果此时 MMI 给 OPC 请求了一个 5s 的更新频率，这当然不会影响到哪个 PID 的 0.05s。

另外，不管是同步还是异步读取数据，仅仅要求采集速度不得低于要求的 5s 而已。服务端程序可以在至少使用 5s 的要求下运行。这种要求是 Client 端给服务端发来的一份建议，

服务端只要尽可能的满足该请求即可。

每一个单独的 Item 对象必须有一个可配置的不同的采集频率。该采集频率不影响回调函数的调用周期。换言之，如果 Group 对象是 10s 的更新频率，而 Group 内的某条 Item 的更新频率为 2s，但回调函数的调用频率将仍然不少于 10s。这种情况下的 Item 的采集频率决定了 Server 端后台程序从设备采集数据的速度。

如果 Item 的数据的值或者质量戳比 Group 设置的更新率快的话，Server 端有必要做缓存,等待下一次调用回调函数时投递出去。缓存的数目是有每一个 Server 程序自己决定的。详细内容查阅 IOPCItemSamplingMgt 接口。

### 4.4.1.5 时区

我们有时候碰到这种情况，就是设备采集的数据与 Client 端的数据不是一个时区。这时候就有必要知道设备采集的数据是哪一个时区的。(时区决定了时间上的一个偏移的大小)。

时区的信息不常用，而且设备也很少知道自己所在的时区。因此时区信息的额外开销不会负担到所有的数据交互过程中，而是由 OPCGroup 对象提供了一个存放时区信息的内存 TimeBias。客户端程序可以修改或读取时区信息。另外，默认的时区信息来自产生 Group 的计算机。时区信息对 Server 端来说意义不大，只有 Client 端可能会使用到。

时区的设置仅用来标示数据产生的时间，和 Server 端或者 Client 端无关。组内默认的时区信息来自产生 Group(调用 AddGroup 传递一个 Null 指针)对象的计算机(Server 端)。时区信息的行为类似于 Win32 中 TIME_ZONE_INFORMATION 结构中的 Bias 域（不考虑 DST)。只有创建 Group 或者调用 SetState 两种方式设置时区信息。客户端 yong TimeStamp+ TimeBias + DSTBias（可选）来生成数据时间。

### 4.4.1.6 数据死区

数据死区为一个常数,从 0.0 到 100.0 之间的任意值均可。只有类型为 dwEUType 的 Item 模拟量才能起作用。Item 的 EU Low 成员和 EU High 成员用来计算死区应用的区间。计算的表达式为：

```
IF    |LastValue-CurValue|>(DeadBand/100.0)*(EUHigh-EULow)
THEN UPDATE
```

死区在 AddGroup 时作为参数传递到 Group 对象中，Group 内的 Item 继承了 Group 的死区的参数。新版本的 Da3.0 已经支持单独设置 Item 的 DeadBand。

当超过死区规定的区间之后，缓存中的最新值将会更新，进一步回调函数得到数据变更的通知。服务程序的死区参数必须是可以配置的，默认情况下为 0，也就是说有数据变更就会更新缓存中的数据。另外，时间戳的更新是独立的，不收数据更新的影响。数据不管变不变，时间一直都在变。如果服务程序不支持死区特性，简单建议设置死区为 0 是不允许的，而应该返回 OPC_E_DEADBANDNOTSUPPORTED 的错误码。

Group 的更新率和 Item 的采集频率决定了检测死区的时间。死区的意义在于对数据噪声的优化。

#### 4.4.1.7 客户端句柄

ClientHandle 作为回调函数的返回值，唯一的标识了数据所在的 Group 对象。要求客户端程序进行异步操作诸如调用 IOPCAsyncIO2 之类的接口函数时，保证一个 ClientHandle 的唯一性。

#### 4.4.1.8 数据操作

客户端可有通过 6 种方式读取数据。

- IOPCSyncIO::Read (fromcache or device)
- IOPCAsyncIO2::Read (fromdevice)
- IOPCCallback::OnDataChange() (exception based) which can alsobe triggered by IOPCAsyncIO2::Refresh.
- IOPCItemIO::Read (fromcache or device as determined by “staleness” of cache data)
- IOPCSyncIO2::ReadMaxAge (from cache or device as determined by “staleness” of cache data)
- IOPCAsyncIO3::ReadMaxAge (from cacheor device asdetermined by “staleness” of cache data)

这 6 种方式相互独立，不会产生“边际效应“，推荐信的应用程序使用 OnDataChange, IOPCSyncIO2 ,IOPCAsynchIO3 读取数据。

下面是 5 种写数据的方式

- IOPCSyncIO::Write
- IOPCAsyncIO2::Write
- IOPCItemIO::WriteVQT
- IOPCSyncIO2::WriteVQT
- IOPCAsyncIO3::WriteVQT

### 4.4.2 IOPCItemMgt

为客户端提供控制 Item 对象的 Add，remove 和 Control the behavior 等行为的方法。

#### 4.4.2.1 IOPCItemMgt::AddItems

```
HRESULT AddItems(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCITEMDEF*pItemArray,
[out, size_is(,dwCount)]OPCITEMRESULT** ppAddResults,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

**描述：**

向 Group 中添加 Item，可以添加多条。

| | |
|---|---|
| dwCount | 需要添加的 Item 的个数 |
| pItemArray | OPCITEMDEF 结构的数组，其中需要包含数据源路径，详细定义，数据类型等信息。 |
| ppAddResults | OPCITEMRESULT 结构的数据，包含 Item 的信息。 |
| ppErrors | 提供每一个 Item 添加是否成功的返回值。如果错误将提供详细信息。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | Item 添加成功 |
| OPC_E_INVALIDITEMID | ItemID 无效 |
| OPC_E_UNKNOWNITEMID | 未找到 ItemID |
| OPC_E_BADTYPE | 不支持的数据类型 |
| E_FAIL | 函数执行失败 |
| OPC_E_UNKNOWNPATH | 未找到数据 |

**注释：**

允许多次添加同一数据源的数据，并且会产生两个不同的具有各自唯一 Server 句柄的 Item。当 ppErrors 返回 Failed 后证明添加 Item 失败，并且 OPCITEMRESULT 不会包含任何有用的信息。

允许服务端程序用 VariantChangeType or VariantChangeTypeEx 代替 OPC_E_BADTYPE 作为返回值。

注意 Item 句柄只是在所属的 Group 内是唯一的，不保证多个 Group 之间的唯一性。Server 可以在 Item 被删除的时候回收这些句柄。（注：所谓的句柄其实是一个唯一标示 Item 的整数值）

OPCITEMRESULT 结构由一个[OUT]参数提供，所以客户端程序有必要对其中包含 BLOB 域的项做内存的清理工作，避免内存泄露。

Server 端支持 BLOB 的话需要在 OPCITEMRESULT 中返回，但是并不要求与 OPCITEMDEF 中投递的完全一致。（否则的话还有返回的必要么？）

需要注意的是，如果 AddItem 之后 Advise 处于激活状态，此时可能会收到 Server 端的回调通知，甚至 Client 端根本还没来的及处理接收到的数据。为了避免这种情况的发生，Client 端有必要在执行 AddItem 以及存储和等待操作结果之前使 Group 对象的激活标志失效。

4.4.2.2 IOPCItemMgt::ValidateItems

```
HRESULT ValidateItems(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCITEMDEF*pItemArray,
[in] BOOL bBlobUpdate,
[out, size_is(,dwCount)]OPCITEMRESULT** ppValidationResults,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：

决定一个 Item 是否有效（影响 Add 请求是否成功返回）。同时也会返回类型信息。该操作不会对 Group 产生影响。

| | |
|---|---|
| dwCount | 需要无效化的 Item 的个数 |
| pItemArray | OPCITEMDEF 结构的数组，其中需要包含数据源路径，详细定义，数据类型等信息。 |
| bBlobUpdate | 参数为任意非零值并且 Server 端支持 Blobs 的话 OPCITEMRESULT 数组中会包含 Blob 信息。否则反之。 |
| ppValidationResults | OPCITEMRESULT 数组。包含数据类型等其他详细信息。 |
| ppErrors | 返回操作结果。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | Item 添加成功 |
| OPC_E_INVALIDITEMID | ItemID 无效 |
| OPC_E_UNKNOWNITEMID | 未找到 ItemID |
| OPC_E_BADTYPE | 不支持的数据类型 |
| E_FAIL | 函数执行失败 |
| OPC_E_UNKNOWNPATH | 未找到数据 |

**注释：**

OPCITEMRESULT 结构由一个[OUT]参数提供，所以客户端程序有必要对其中包含 BLOB 域的

项做内存的清理工作，避免内存泄露。
允许服务端程序用 VariantChangeType or VariantChangeTypeEx 代替 OPC_E_BADTYPE 作为返回值。

### 4.4.2.3 IOPCItemMgt::RemoveItems

HRESULT RemoveItems(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
从 Group 对象中删除 Item。本质上这是 AddItems 的你操作。

| | |
|---|---|
| dwCount | 需要删除的 Item 的个数 |
| phServer | 需要删除的 Item 的句柄数组，从 AddItems 的返回值中得到 |
| ppErrors | 返回操作结果。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 相关的 Item 删除成功 |
| OPC_E_INVALIDITEMID | 相关的 Item 句柄无效 |

**注释：**
添加和删除 Item 的操作不会对 Server 端的数据空间或者设备造成影响，这仅仅是客户端在向 Server 端订阅和取消订阅特定的数据。
因为 Item 在客户端不是接口对象，所以不会维护它的引用计数。客户端在执行 Delete 操作前有责任保证不存在额外的引用。

### 4.4.2.4 IOPCItemMgt::SetActiveState

HRESULT SetActiveState(

[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in]BOOL bActive,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
设置一个或者多个 Item 的激活状态标志。该操作决定了 Group 是否能从缓存中通过 Read 到有效的数据或者通过 OnDataChange 回调的方式订阅到数据。

| | |
|---|---|
| dwCount | 受影响的 Item 的个数 |
| phServer | 受影响的 Item 的句柄数组，从 AddItems 的返回值中得到 |
| bActive | 设置 True 为激活，反之，你懂得。 |
| ppErrors | 返回操作结果。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 相关的 Item 删除成功 |
| OPC_E_INVALIDITEMID | 相关的 Item 句柄无效 |

**注释：**
关闭激活标志后再也不会收到回调通知，开启激活标志后将会在先一个更新周期后收到回调通知。

### 4.4.2.5 IOPCItemMgt::SetClientHandles

HRESULT SetClientHandles(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in,size_is(dwCount)] OPCHANDLE*phClient,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：

修改 Group 中任意个 Item 的句柄。

| | |
|---|---|
| dwCount | 受影响的 Item 的个数 |
| phServer | Server 端的 Item 的句柄数组，从 AddItems 的返回值中得到 |
| phClient | Client 端 Item 的句柄，不要求具有唯一性。 |
| ppErrors | 返回操作结果。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 相关的 Item 删除成功 |
| OPC_E_INVALIDITEMID | 相关的 Item 句柄无效 |

**注释：**

建议在 AddItems 时设置句柄而不是之后在改变句柄。

4.4.2.6 IOPCItemMgt::SetDatatypes

```
HRESULT SetDatatypes(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in, size_is(dwCount)]VARTYPE * pRequestedDatatypes,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：

改变 Group 内多个 Item 的数据类型。

| | |
|---|---|
| dwCount | 受影响的 Item 的个数 |
| phServer | Server 端的 Item 的句柄数组，从 AddItems 的返回值中得到 |
| pRequestedDatatypes | 指定的新的类型的数组 |
| ppErrors | 返回操作结果。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|

| | |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 相关的 Item 删除成功 |
| OPC_E_INVALIDITEMID | 相关的 Item 句柄无效 |
| OPC_E_BADTYPE | 相关的 Item 不支持该类型，原来的类型仍保留 |

**注释：**

建议 AddItems 时指定类型，而不是之后改变类型。

允许服务端程序用 VariantChangeType or VariantChangeTypeEx 代替 OPC_E_BADTYPE 作为返回值。

### 4.4.2.7 IOPCItemMgt::CreateEnumerator

```
HRESULT CreateEnumerator(
[in]REFIID riid,
[out,iid_is(riid)]LPUNKNOWN* ppUnk
);
```

描述：

创建一个可以枚举 Group 内 Item 的枚举器

| | |
|---|---|
| riid | 尽管 OPC 厂商可以提供扩展接口，但这里最好仅支持 IID_IEnumOPCItem |
| ppUnk | 返回接口。如果 HRESULT 不是返回的 S_OK，必须得到一个为 NULL 的 ppUnk |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

**注释：**

客户端使用完接口之后必须释放。

### 4.4.3 IOPCGroupStateMgt

该接口允许客户端控制Group对象的全局属性，比如Group更新率和Group的激活状态。

#### 4.4.3.1 IOPCGroupStateMgt::GetState

```
HRESULT GetState(
[out] DWORD * pUpdateRate,
[out] BOOL* pActive,
[out, string]LPWSTR * ppName,
[out] LONG* pTimeBias,
[out] FLOAT* pPercentDeadband,
[out]DWORD* pLCID,
[out] OPCHANDLE * phClientGroup,
[out] OPCHANDLE * phServerGroup
);
```

描述：

得到Group的当前状态。

| | |
|---|---|
| pUpdateRate | 以毫秒为单位的 Group 更新率 |
| pActive | Group 的激活状态 |
| ppName | Group 的名称 |
| pTimeBias | 分钟为单位的时区 |
| pPercentDeadband | 死区 |
| pLCID | Group 的 LCID |
| phClientGroup | 客户端 Group 对象句柄 |
| phServerGroup | 服务端的 Group 句柄 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| E_OUTOFMEMORY | 内存不足 |

**注释：**

该例程一般在调用 SetState 之前调用以获取当前状态。Group 对象创建成功后必将能够返回全部的信息，这对调试工作有很大帮助。

OUT 参数必须保证是指针的有效性，列集机制要求该指针行为的确定性。NULL 指针会应发一个 RPC 异常。

客户端有必要释放 ppName 字符串的资源。

### 4.4.3.2 IOPCGroupStateMgt::SetState

```
HRESULT SetState(
[unique,in]DWORD * pRequestedUpdateRate,
[out]DWORD* pRevisedUpdateRate,
[unique, in] BOOL*pActive,
[unique,in]LONG * pTimeBias,
[unique, in] FLOAT* pPercentDeadband
[unique,in]DWORD * pLCID,
[unique,in]OPCHANDLE *phClientGroup
);
```

描述：

客户端程序设置 Group 对象的属性。

IN 参数可以通过赋值为 NULL 来省略不关心的参数设置。pRevisedUpdateRate 作为唯一的 OUT 参数必须保证指针的有效性。

| | |
|---|---|
| pRequestedUpdateRate | 以毫秒为单位的新的 Group 更新率 |
| pRevisedUpdateRate | Server 支持的最接近的更新率 |
| pActive | TRUE 为激活态，反之。 |
| pTimeBias | 分钟为单位的时区 |
| pPercentDeadband | 死区 |
| pLCID | Group 的 LCID |
| phClientGroup | 客户端 Group 对象句柄 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| E_OUTOFMEMORY | 内存不足 |
| OPC_S_UNSUPPORTEDRATE | Server 端不支持该更新率，用最接近的替代 |

**注释：**

参考 Data Acquistion Section 一节以获取关于 OPCserver 同步、异步接口以及激活 Group 的详细信息。

AddGroup 中提到了关于 dwLCID 是 Server 端独有的一种属性的信息。如果 Server 不支持的话可以忽略该参数。

### 4.4.3.3 IOPCGroupStateMgt::SetName

HRESULT SetName(
[in,string]LPCWSTR szName,
);
描述：
改变 Group 对象的名称。Client 有必要保证命名的唯一性。

| szName | 新的 Group 名称 |
|---|---|
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| E_OUTOFMEMORY | 内存不足 |
| OPC_E_DUPLICATENAME | 命名冲突 |

**注释：**
Client 端必须保证在本客户端内部 Group 命名空间内名字具有唯一性。

### 4.4.3.4 IOPCGroupStateMgt::CloneGroup

HRESULT CloneGroup(
[in,string]LPCWSTR szName,
[in] REFIID riid,
[out,iid_is(riid)]LPUNKNOWN * ppUnk
);

描述：
创建一个唯一 Group 对象的副本。该副本具有源对象的所有的属性。副本包含相同的更新率，Item 数据项，Group 和 Item 的句柄，设定的数据类型等等。该副本与源 Group 是独立的，一旦对副本内的 Item 执行 Add 或者 Delete 等操作并不会影响原始 Group。
注意有些属性并不会拷贝到新的副本内部，包括：
副本的激活标志为初始状态 FALSE；
副本在 Client 端具有唯一的服务端句柄；
副本的 Item 拥有新的服务端句柄，客户端在用到的时候需要首先获取该句柄。
副本没有 Advise 和 Connection Point 对象。需要的话需要显式的创建。

| | |
|---|---|
| szName | 新的 Group 名称。Client 端保证该名字的唯一性。如果不指定名称可以传递一个空串（一个 NULL 字符串），服务端会自动生成一个唯一的名称。 |
| riid | 请求的接口 |
| ppUnk | 返回的接口指针。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| E_OUTOFMEMORY | 内存不足 |
| OPC_E_DUPLICATENAME | 命名冲突 |
| E_NOINTERFACE | 不支持的接口类型 |

**注释：**

该副本完全独立于原始 Group。参考 AddGroup 函数的说明获取 Group 的生存期的一些详细信息。比如 AddGroup 调用之后需要执行 RemoveGroup 以便于在不需要改 Group 的时候释放他。同时释放返回的接口指针也是客户端的责任。

## 4.4.4 IOPCGroupStateMgt2

该接口时 IOPCGroupStateMgt 的增强版。

IOPCGroupStateMgt2 继承了 IOPCGroupStateMgt 的所有方法，在此不在赘述（参考上一节获取详细信息）。只有支持 DA3.0 的服务器可以实现该接口作为 IOPCGroupStateMgt 的替代品。该接口增加了心跳检测的订阅功能。当订阅指定了一个非零的心跳周期后，Server 端将在指定的周期内调用 Client 端的一个回调接口。这样的话 Client 端再也没有必要循环调用 GetStatus 函数来维持链路了。

### 4.4.4.1 IOPCGroupStateMgt2::SetKeepAlive

```
HRESULT SetKeepAlive(
[in] DWORD dwKeepAliveTime,
[out]DWORD *pdwRevisedKeepAliveTime
);
```

描述：

如上所述，当服务端没有任何事务通知的时候，Client 端在也没有必要调用 GetStatus 来诊断 Server 端的在线状态了，通过指定一个心跳周期，Client 端就可以接到该通知。

服务端在接收数据的同时是可以改变心跳周期的。

dwKeepAliveTime　　以毫秒为单位的心跳周期，指定为 0 可关闭心跳检测

pdwRevisedKeepAliveTime　返回 Server 端实际支持的周期

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| OPC_S_UNSUPPORTEDRATE | 不支持的心跳周期，实际使用的值在数 2 指定 |

**注释：**

心跳回调指定了一个 dwCount 参数为 0 的 IOPCDataCallback:: OnDataChange()调用。

订阅无效的时候回调不会起作用。

心跳回调不会影响 IOPCServer::GetStatus()返回的 ftLastUpdateTime 的数值。

#### 4.4.4.2 IOPCGroupStateMgt2::GetKeepAlive

```
HRESULT GetKeepAlive(
[out]DWORD *pdwKeepAliveTime
);
```

描述：

返回心跳检测周期

| | |
|---|---|
| pdwKeepAliveTime | 以毫秒为单位的心跳周期，为 0 标示无心跳检测 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| S_OK | 执行成功 |

**注释：**

默认情况如果未调用 SetKeepAlive，返回心跳周期为 0。

### 4.4.5 IOPCSyncIO

该接口支持同步读写操作，完成后返回。

参考 Data Acquisition and Active State Behavior 以及本章之前的 Serialization andSyncronizationissues 小节获取关于对数据采集的配置对本接口可能造成的影响的详细信息。

### 4.4.5.1 IOPCSyncIO::Read

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
);
描述：
读取多个单个组内的多个 Item 的值，时间错，质量戳等信息。该函数在完成之前会一直阻塞。数据也许是以 Group 对象的更新率为基准周期性的从缓存中读取数据，或许每次操作设备的物理内存获取数据。但具体的实现方式本规范不做要求。
只有 Group 和 Item 同时处于激活态才能获取有效的数据，否则质量戳为 OPC_QUALITY_OUT_OF_SERVICE。参考后面章节的质量戳掩码一节获取必要信息。
当然，直接读取设备数据的方式不会受激活标志的影响，只有缓存模式受该标志影响。

| | |
|---|---|
| dwSource | 指定数据源类型。OPC_DS_CACHE or OPC_DS_DEVICE |
| dwCount | 需要读取的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| ppItemValues | 返回的数据列表 |
| ppErrors | 错误信息 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 读成功 |
| OPC_E_BADRIGHTS | 不是可读的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 读失败 |

S_xxx | 厂商定制的专有信息
E_xxx | 厂商专有的错误信息

**注释：**
返回 S_OK 的话可以忽略 ppError 的结果。当返回 S_FALSE 时，ppError 将包含每个 item 操作的详细错误信息。
如果返回的是其他错误码而不是 S_FALSE，所有的 OUT 参数必须得到一个 NULL 指针，包括 ppErrors。
对于 ppError 的其他 S_xx 的返回应该作为定义良好的返回来处理。为了便于列集和执行 VariantClear，Server 端必须在这种情况下将 VARIANT 变量赋值为 VT_EMPTY。
单个 Item 返回的 OPC_E_INVALIDHANDLE 不会影响其他 Item 的操作结果，但会导致整个函数返回 S_FALSE。
一般情况下读缓存数据很快，是毫秒级的。但是读设备数据往往很慢，几秒甚至更多。有时候单个 Client 执行的读操作会影响其他客户端的读操作，这完全依赖于 Server 端具体的实现。
所有大部分情况下期望从缓存读数据，只有在诊断设备的情况下才直接读取设备的数据。
注意释放 ppItemValues 和 ppErrors 的相关资源，并且对 ITEMRESULT 中的 variant 变量执行 VariantClear 操作。

### 4.4.5.2 IOPCSyncIO::Write

```
HRESULT Write(
[in]DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in,size_is(dwCount)] VARIANT*pItemValues,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：
设置 Group 内的多个变量值。该函数为阻塞函数。直接影响设备中的数据。任何激活标志不能影响写操作。

| | |
|---|---|
| dwCount | 需要设置的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| ppItemValues | 需要设置的数据列表，不要求设置的数据类型与实际类型一致，最多返回不能转换的错误。 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |

| | | |
|---|---|---|
| E_INVALIDARG | | 错误的参数 |
| S_OK | | 执行成功 |
| S_FALSE | | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 写成功 |
| OPC_E_BADRIGHTS | 不是可写的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 写失败 |
| S_xxx | 厂商定制的专有信息 |
| E_xxx | 厂商专有的错误信息 |
| OPC_S_CLAMP | 数值合理但不会被保留太久 |
| OPC_E_RANGE | 数据越界 |
| OPC_E_BADTYPE | 不支持的数据类型 |

**注释：**

返回 S_OK 的话可以忽略 ppError 的结果。当返回 S_FALSE 时，ppError 将包含每个 item 操作的详细错误信息。

如果返回的是其他错误码而不是 S_FALSE，所有的 OUT 参数必须得到一个 NULL 指针，包括 ppErrors。

单个 Item 返回的 OPC_E_INVALIDHANDLE 不会影响其他 Item 的操作结果，但会导致整个函数返回 S_FALSE。

写设备的操作耗时是不言而喻的，是否排斥其他客户端的写操作与具体的实现相关。

因此，推荐使用异步写代替同步写。同样，注意释放 ppErrors 的资源。

## 4.4.6 IOPCSyncIO2

IOPCSyncIO 的增强版。与 IOPCSyncIO 是继承关系，IOPCSyncIO 已有的函数不做介绍。支持 OPC DA3.0 的 Server 端必须实现该接口。该接口增加了从 Group 级别访问时间戳和质量戳的能力。他可以从一个 Group 中基于 MaxAge 来读取数据，这里与 IOPCSyncIO 最大的不同就是可以控制作用域为 Group 的范围而不是全局。

### 4.4.6.1 IOPCSyncIO2::ReadMaxAge

```
HRESULT ReadMaxAge (
[in] DWORD dwCount,
[in, sizeis(dwCount)]OPCHANDLE *phServer
[in, sizeis(dwCount)]DWORD *pdwMaxAge,
```

[out,sizeis(,dwCount)] VARIANT **ppvValues,
[out, sizeis(,dwCount)] WORD **ppwQualities,
[out,sizeis(,dwCount)] FILETIME **ppftTimeStamps,
[out,sizeis(,dwCount)] HRESULT **ppErrors
);
描述：
读取指定的 Item 的数据，质量戳，时间戳等信息。这个函数除了不能选择数据源（缓存还是设备）之外和同步读的函数 OPCSyncIO::Read 在功能上一致。
该函数指定了一个最大延迟时间，如果超过该延迟就会直接从设备读取数据，否则将会读缓存数据。

| | |
|---|---|
| dwCount | 需要读的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| pdwMaxAge | 指定每个 item 数据延迟的数组。Server 端需要计算每个 item 的延迟时间，方法是用现在的时间和数据的当前时间戳做减法，所得结果与延迟时间作比较，如果该结果超过了延迟时间，需要从设备更新数据，否则直接从缓存数据局。指定该值为 0 相当于设置 OPC_DS_DEVICE，同理指定该值为 0xFFFFFFFF 等效于设置了 OPC_DS_CACHE。如果客户端没有同时激活 Group 和其中的 item 的话，就不能够做 Server 端存在缓存的假设。然而有些 Server 包含全局的缓存，此时就会从全局的缓存中取数据进行对比。另外，Server 端不应该依赖于包含 MaxAge 请求的读操作来更新缓存，从其是一个 DWORD 类型的参数可以看出来（以毫秒为单位的话最大可以到 49.7 天）。 |
| ppItemValues | 返回的数据列表。客户端必须释放相关资源。 |
| ppwQualities | 返回的质量戳数据。客户端必须释放相关资源。 |
| ppftTimeStamps | 返回的时间戳数据（FILETIMES 结构）。客户端必须释放相关资源。 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 读成功 |
| OPC_E_BADRIGHTS | 不是可读的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |

OPC_E_UNKNOWNITEMID | Item 已经过期
E_FAIL | 读失败
S_xxx | 厂商定制的专有信息
E_xxx | 厂商专有的错误信息

**注释：**
MaxAge 在仅仅调用结束之前做一次比较。比如三个 item，其中两个个在 MaxAge 要求的 1000ms 以内，那么在所有三个 item 返回之前，必须等待另外一个从设备读取成功后，才能一起打包返回调用源。这样就阻止了递归的从设备中获取数据。
如果服务器的缓存模式和设备模式一致的话，服务器总是返回实际值。
返回 S_OK 的话可以忽略 ppErrors 的信息。反之，从 ppErrors 进一步获取每个 item 的错误信息。

### 4.4.6.2 IOPCSyncIO2::WriteVQT

```
HRESULT WriteVQT (
[in] DWORD dwCount,
[in, sizeis(dwCount)]OPCHANDLE *phServer,
[in, sizeis(dwCount)]OPCITEMVQT*pItemVQT,
[out,sizeis(,dwCount)] HRESULT **ppErrors
);
```

描述：
设置多个指定的 Item 的质量戳，时间戳，值。该函数基本与 IOPCSyncIO::Write 一致，但是可以设置时间戳和质量戳。

| | |
|---|---|
| dwCount | 需要设置的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| pItemVQT | OPCITEMVQT 结构的列表。结构体的成员包括与 ItemID 相关的质量戳，时间戳，变量值等信息。入股变量值为 VT_EMPTY，表明不需要设置。与质量戳和时间戳都有相关的一个 BOOL 域，表明是否设置。为 TRUE 代表需要设置。 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |
| OPC_E_NOTSUPPORTED | 说明服务器不支持该组合的设置。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 写成功 |
| OPC_E_BADRIGHTS | 不是可写的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 写失败 |
| S_xxx | 厂商定制的专有信息 |
| E_xxx | 厂商专有的错误信息 |
| OPC_S_CLAMP | 数值合理但不会被保留太久 |
| OPC_E_RANGE | 数据越界 |
| OPC_E_BADTYPE | 不支持的数据类型 |

**注释：**

返回 S_OK 可以或略 ppErrors 接口。

注意：

没有把时间戳写入设备的方案。有些服务器支持设置时间戳到设备而不是缓存的机制。这些情况下需要这样做是因为当时处于一种手动配置的状态，或者设备使用了一种保持寄存器。一般的情况写时间戳是没有意义的，因为时间戳肯定会被设备实时更改。

## 4.4.7 IOPCAsyncIO2

IOPCAsyncIO2 支持异步读写操作。函数会立刻返回不影响 Client 端的执行，发送的请求将进行排队等待完成。每个操作将作为一个事务并且获得一个事务 id。完成之后 Client 端会获得一个 IOPCDataCallback 的回调。回调函数的参数中将会包含该事务 id 和操作的结果。异步读、写、刷新的操作都是单独的返回一个 IOPCDataCallback 调用。

Server 具备单独对每一个组内的每一种请求排队的能力。如果相同的客户端相同的 Group 的相同类型的请求多次发生，Server 端必须返回 CONNECT_E_ADVISELIMIT 的错误码。如果厂商乐意的话，他们也可以支持额外的排队机制。

所有的请求都需要得到完成通知，所以“超时”的观念在这里很不实用。如果客户需要超时的机制的话可以通过返回特殊的错误码来实现。

Client 端实现需要注意的事项：

事务 ID 是有 Client 端在发送异步读写或者刷新请求的时候分配的，并且在请求完成后返回以便通知 Client 端响应的请求执行完。确保 Client 端能够正确处理，请保证事务 id 在 Client 中的唯一性和非零的特性。其他 Client 中的事务 id 不会对本 Client 有影响，实际上 Server 端不对 ID 做过多的检测，仅仅返回即可。另外 Group 对象的 ClientHandle 也会在回调中返回，便于标示返回的数据所属。

重点提示：

Server 或者 Client 端的线程模型如果是混合模式的话 ， IOPCDataCallBack 可以在 Read, Write orRefresh 等动作返回之前在同一线程中再次被调用。因此如果 Client 想要

保存一个事务 id 在某些定义好的“未决队列”中的话，必须先分配一个唯一的 id 并且保存他，然后再发送请求。

应用中很少会有这样一个未决队列，所以没有必要记录该 id。

#### 4.4.7.1 IOPCAsyncIO2::Read

```
HRESULT Read(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in] DWORD dwTransactionID,
[out] DWORD *pdwCancelID,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：

读 Group 内的多个 item。通过 IconnectionPointContainer：IOPCDataCallback 返回结果。

该操作是从设备直接读取，并且不受 Group 和 item 激活标志的影响。

| | |
|---|---|
| dwCount | 需要读的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| dwTransactionID | 事务 ID。在 OnReadComplete 回调函数的完成通知中可以获得该 id |
| pdwCancelID | Server 端生成的一个取消操作用的事务 id，将得到完成通知 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 有 item 不能读取。 |
| CONNECT_E_NOCONNECTION | 未通过 IConnectionPoint::Advise.注册回调函数。 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 读成功 |
| OPC_E_BADRIGHTS | 不是可读的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |

E_FAIL | 读失败
S_xxx | 厂商定制的专有信息
E_xxx | 厂商专有的错误信息
———————————————————————————

**注释：**
注意有些 Server 比较智能的及时的返回错误，而有些确实把请求排队，所有的错误信息在回调函数中返回，客户端程序必须处理这种情形。成功返回 S_OK 的话应该忽略 ppErrors 信息。
OnReadComplete 中将返回成功读取的 item 信息。
所有的信息必须在一次回调中返回。即使请求了多个设备的数据，也需要等待数据全部获取到之后一次性返回 OnReadComplete 回调函数。

## 4.4.7.2 IOPCAsyncIO2::Write

HRESULT Write(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in,size_is(dwCount)] VARIANT*pItemValues,
[in] DWORD dwTransactionID,
[out] DWORD *pdwCancelID,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
写 Group 内的多个 item。通过 IconnectionPointContainer：IOPCDataCallback 返回结果。

| | |
|---|---|
| dwCount | 需要写的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| pItemValues | 写入的数据列表。类型不匹配自动转换，除非转换失败并且返回错误码。 |
| dwTransactionID | 事务 ID。在 OnReadComplete 回调函数的完成通知中可以获得该 id |
| pdwCancelID | Server 端生成的一个取消操作用的事务 id，将得到完成通知 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：
HRESULT | 返回： |
———————|———————————————
E_FAIL | 失败 |
E_OUTOFMEMORY | 内存不足 |
E_INVALIDARG | 错误的参数 |
S_OK | 执行成功 |
S_FALSE |有 item 不能读取。
CONNECT_E_NOCONNECTION|未通过 IConnectionPoint::Advise.注册回调函数。

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 读成功 |
| OPC_E_BADRIGHTS | 不是可读的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 读失败 |
| S_xxx | 厂商定制的专有信息 |
| E_xxx | 厂商专有的错误信息 |

**注释：**

注意有些 Server 比较智能的及时的返回错误，而有些确实把请求排队，所有的错误信息在回调函数中返回，客户端程序必须处理这种情形。成功返回 S_OK 的话应该忽略 ppErrors 信息。

OnWriteComplete 中将返回成功写入的 item 信息。

所有的信息必须在一次回调中返回。即使请求了多个设备的数据，也需要等待数据全部获取到之后一次性返回 OnWriteComplete 回调函数。

### 4.4.7.3 IOPCAsyncIO2::Refresh2

```
HRESULT Refresh2(
[in] OPCDATASOURCE dwSource,
[in] DWORD dwTransactionID,
[out] DWORD *pdwCancelID
);
```

描述：

强制对某一个 Group 内处于激活态的 item 进行一次 IOPCDataCallback::OnDataChange 回调（不管数据是否改变）。

| | |
|---|---|
| dwSource | 指定是缓存操作还是设备操作。如果从设备读取将会更新缓存。 |
| dwTransactionID | 事务 ID。在 OnDataChange 回调函数的完成通知中可以获得该 id |
| pdwCancelID | Server 端生成的一个取消操作用的事务 id，将得到完成通知 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |

| | | |
|---|---|---|
| E_INVALIDARG | | 错误的参数 |
| S_OK | | 执行成功 |
| S_FALSE | 有 item 不能读取。 | |
| CONNECT_E_NOCONNECTION | 未通过 IConnectionPoint::Advise.注册回调函数。 | |

**注释：**

HRESULT 返回错误码之后，将不会产生回调通知。一个 Group 处于非激活态或者一个激活态的 Group 组内的 item 都处于非激活态，该调用的解决均会返回 E_FAIL。

Advise 初始化的时候会触发 OnDataChange 回调，但是回调参数中的事务 id 为 0.所以客户端想要区分是 Refrush2 调用还是初始化通知的情况请给事务 ID 分配一个非零值。功能上该方法等效于读取所有激活态的 item 数据。

注意：所有的数据均在一次回调中返回，多个设备中的多个事务必须等待一起返回。理想的情况是该调用不应该影响基于 Group 更新率的 OnDataChange 回调。比如更新率设置为 1 小时，但是该调用在 45 分钟的时候发生了，但是这不能到打断预期的一小时一次的 OnDataChange 调用。唯一受影响的是接下来的 15 分钟内的数据发生了改变的才能以 OnDataChange 调用的方式返回。

### 4.4.7.4 IOPCAsyncIO2::Cancel2

```
HRESULT Cancel2(
[in] DWORD dwCancelID
);
```

描述：

要求取消一个未决的事务。

| | |
|---|---|
| dwCancelID | 投递请求是获得的取消 id |
| | |

函数返回值：

| | | |
|---|---|---|
| HRESULT | | 返回： |
| E_FAIL | | 取消 id 无效货已经过期 |
| S_OK | | 执行成功 |

**注释：**

改掉用的实现是与具体的 server 实现相关的并且依赖去取消调用的时间。回调函数的触发也与时间相关。应用场合仅限于关闭 server 之前做一些清理工作。

该调用成功后会以 OnCancelComplete 回调的方式返回。

### 4.4.7.5 IOPCAsyncIO2::SetEnable

```
HRESULT SetEnable(
```

[in] BOOL bEnable
);

描述：
禁止所有事物 id 为 0 的 OnDataChange 回调（不包括 Refrsh 调用触发的）。

| bEnable | FALSE-禁止。TRUE-允许 |
|---|---|
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 取消 id 无效货已经过期 |
| S_OK | 执行成功 |
| CONNECT_E_NOCONNECTION | 未通过 IConnectionPoint::Advise.注册回调函数。 |

**注释：**
Group 对象初始化时该值为 TRUE。
该函数用来建立一个不需要 OnDataChange 通知的连接。比如一个偶尔才用 Refresh 更新数据的客户端。
就算客户端不打算用 OnDataChange 接收请求，但是仍然应该处理该请求，以便于处理未禁止改掉用之前收到的回调，至少你要释放 OUT 参数的相关资源（默认情况是不禁止的哦）
如果客户端要禁止回调并且一点也不想处理回调，请按照下面的步骤操作。创建一个 Group。将 Group 非激活。连接到 Group。调用本例程。激活 Group。
该例程不会影响 Refresh2 函数的调用，会触发 OnDataChange 的回调，相当于一次异步读缓存的操作。

## 4.4.7.6 IOPCAsyncIO2::GetEnable

HRESULT GetEnable(
[out] BOOL *pbEnable
);

描述：
上一次 SetEnable.设置的状态。

| pEnable | 返回的状态 |
|---|---|
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|

CONNECT_E_NOCONNECTION|未通过 IConnectionPoint::Advise.注册回调函数。
E_FAIL | 取消 id 无效货已经过期 |
S_OK | 执行成功 |

## 4.4.8 IOPCAsyncIO3

IOPCAsyncIO2 的增强版。重复的不再赘述。提供基于 Group 范围的 MaxAge 控制的操作模式。

### 4.4.8.1 IOPCAsyncIO3::ReadMaxAge

```
HRESULT ReadMaxAge (
[in] DWORD dwCount,
[in, sizeis(dwCount)]OPCHANDLE *phServer
[in, sizeis(dwCount)]DWORD *pdwMaxAge,
[in]DWORD dwTransactionID,
[out]DWORD *pdwCancelID,
[out,sizeis(,dwCount)] HRESULT **ppErrors
);
```

描述：
指定的 item 中读值，质量戳，时间戳。该函数与 OPCSyncIO::Read 的区别在于本函数为异步调用并且不能指定数据源类型。数据源类型有 Server 端参考 MaxAge 指定。

| | |
|---|---|
| dwCount | 需要读的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| pdwMaxAge | 指定每个 item 数据延迟的数组。Server 端需要计算每个 item 的延迟时间，方法是用现在的时间和数据的当前时间戳做减法，所得结果与延迟时间作比较，如果该结果超过了延迟时间，需要从设备更新数据，否则直接从缓存数据局。指定该值为 0 相当于设置 OPC_DS_DEVICE，同理指定该值为 0xFFFFFFFF 等效于设置了 OPC_DS_CACHE。如果客户端没有同时激活 Group 和其中的 item 的话，就不能够做 Server 端存在缓存的假设。然而有些 Server 包含全局的缓存，此时就会从全局的缓存中取数据进行对比。另外，Server 端不应该依赖于包含 MaxAge 请求的读操作来更新缓存，从其是一个 DWORD 类型的参数可以看出来（以毫秒为单位的话最大可以到 49.7 天）。 |
| ppItemValues | 返回的数据列表。客户端必须释放相关资源。 |
| ppwQualities | 返回的质量戳数据。客户端必须释放相关资源。 |
| ppftTimeStamps | 返回的时间戳数据（FILETIMES 结构）。客户端必须释放相关资源。 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |
| CONNECT_E_NOCONNECTION | 未注册回调 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 读成功 |
| OPC_E_BADRIGHTS | 不是可读的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 读失败 |
| S_xxx | 厂商定制的专有信息 |
| E_xxx | 厂商专有的错误信息 |

**注释：**

MaxAge 在仅仅调用结束之前做一次比较。比如三个 item，其中两个个在 MaxAge 要求的 1000ms 以内，那么在所有三个 item 返回之前，必须等待另外一个从设备读取成功后，才能一起打包返回调用源。这样就阻止了递归的从设备中获取数据。

如果服务器的缓存模式和设备模式一致的话，服务器总是返回实际值。

返回 S_OK 的话可以忽略 ppErrors 的信息。反之，从 ppErrors 进一步获取每个 item 的错误信息。

### 4.4.8.2 IOPCAsyncIO3::WriteVQT

HRESULT WriteVQT (
[in] DWORD dwCount,
[in, sizeis(dwCount)]OPCHANDLE *phServer,
[in, sizeis(dwCount)]OPCITEMVQT*pItemVQT,
[in] DWORD dwTransactionID,
[out]DWORD *pdwCancelID,
[out,sizeis(,dwCount)] HRESULT **ppErrors
)

描述：

设置多个指定的 Item 的质量戳，时间戳，值。数据通过 IconnectionPointContainer：

IOPCDataCallback 接口返回。该函数基本与 IOPCAsyncIO2::Write 一致，但是可以设置时间戳和质量戳。VQ, VT, or VQT 必须一起设置。

| | |
|---|---|
| dwCount | 需要设置的 Item 总数 |
| phServer | Server 端指定的 Item 的列表 |
| pItemVQT | OPCITEMVQT 结构的列表。结构体的成员包括与 ItemID 相关的质量戳，时间戳，变量值等信息。入股变量值为 VT_EMPTY，表明不需要设置。与质量戳和时间戳都有相关的一个 BOOL 域，表明是否设置。为 TRUE 代表需要设置。 |
| ppErrors | 错误信息，任何错误码都表明设备未接受该参数。 |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |
| OPC_E_NOTSUPPORTED | 说明服务器不支持该组合的设置。 |
| CONNECT_E_NOCONNECTION | 未注册回调 |

ppErrors 返回值：

| ppErrors | 返回： |
|---|---|
| S_OK | 写成功 |
| OPC_E_BADRIGHTS | 不是可写的 |
| OPC_E_INVALIDHANDLE | Item 句柄无效 |
| OPC_E_UNKNOWNITEMID | Item 已经过期 |
| E_FAIL | 写失败 |
| S_xxx | 厂商定制的专有信息 |
| E_xxx | 厂商专有的错误信息 |
| OPC_S_CLAMP | 数值合理但不会被保留太久 |
| OPC_E_RANGE | 数据越界 |
| OPC_E_BADTYPE | 不支持的数据类型 |

**注释：**

返回 S_OK 可以或略 ppErrors 接口。

注意：

某些更智能的 Server 可以及时返回操作结果，但是有些需要排队请求，只有回调函数能返回操作结果。

### 4.4.8.3 IOPCAsyncIO3:: RefreshMaxAge

HRESULT RefreshMaxAge(
[in] DWORD dwMaxAge,
[in] DWORD dwTransactionID,
[out] DWORD *pdwCancelID
);

描述：
强制获得一个 Group 内处于激活态的 item 的一次 IOPCDataCallback::OnDataChange 通知。是否返回数据有 MaxAge 做判断。

| | |
|---|---|
| dwMaxAge | 指定每个 item 数据延迟的数组。Server 端需要计算每个 item 的延迟时间，方法是用现在的时间和数据的当前时间戳做减法，所得结果与延迟时间作比较，如果该结果超过了延迟时间，需要从设备更新数据，否则直接从缓存数据局。指定该值为 0 相当于设置 OPC_DS_DEVICE，同理指定该值为 0xFFFFFFFF 等效于设置了 OPC_DS_CACHE。如果客户端没有同时激活 Group 和其中的 item 的话，就不能够做 Server 端存在缓存的假设。然而有些 Server 包含全局的缓存，此时就会从全局的缓存中取数据进行对比。另外，Server 端不应该依赖于包含 MaxAge 请求的读操作来更新缓存，从其是一个 DWORD 类型的参数可以看出来（以毫秒为单位的话最大可以到 49.7 天）。 |
| dwTransactionID | 事务 id |
| pdwCancelID | 事务取消 id |
| | |

函数返回值：

| HRESULT | 返回： |
|---|---|
| E_FAIL | 失败 |
| E_OUTOFMEMORY | 内存不足 |
| E_INVALIDARG | 错误的参数 |
| S_OK | 执行成功 |
| S_FALSE | 执行中遇到错误返回。 |
| OPC_E_NOTSUPPORTED | 说明服务器不支持该组合的设置。 |
| CONNECT_E_NOCONNECTION | 未注册回调 |

**注释：**
HRSULT 返回失败后不会收到任何完成通知。
如果 Group 处于非激活态或者 Group 内的 item 均处于非激活态的话，该调用返回失败。
注意：所有的数据均在一次回调中返回，多个设备中的多个事务必须等待一起返回。
理想的情况是该调用不应该影响基于 Group 更新率的 OnDataChange 回调。比如更新率设置

为 1 小时，但是该调用在 45 分钟的时候发生了，但是这不能到打断预期的一小时一次的 OnDataChange 调用。唯一受影响的是接下来的 15 分钟内的数据发生了改变的才能以 OnDataChange 调用的方式返回。

## 4.4.9 IOPCItemDeadbandMgt

设置死区。

### 4.4.9.1 IOPCItemDeadbandMgt::SetItemDeadband

```
HRESULT SetItemDeadband(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in,size_is(dwCount)]FLOAT*pPercentDeadband,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

详情看参考文档。

### 4.4.9.2 IOPCItemDeadbandMgt:: GetItemDeadband

```
HRESULT GetItemDeadband(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[out, size_is(,dwCount)]FLOAT **ppPercentDeadband,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

获得死区参数。
详情看参考文档。

### 4.4.9.3 IOPCItemDeadbandMgt:: ClearItemDeadband

```
HRESULT ClearItemDeadband(
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

清除死区设置，恢复 Group 中的默认值。
详情看参考文档。

## 4.4.10 IOPCItemSamplingMgt (optional)可选的。

该接口允许客户端配置数据的采集频率。不影响客户端的数据回调通知。

举个例子：

一个 Group 有 ABC 三个 item。Group 的更新率为 10 秒。采集率分别为 2s、15s、无采集率。

图像省略，赶进度。

那么根据 Group 的更新率进行数据更新的规则是：

1） 如果从上次更新以来质量戳发生改变的话那么就更新到客户端。

2） 如果从上次更新以来值发生改变并且超过了死区的话就会更新到客户端。

如果采集频率高于更新频率的话，Server 端需要做特殊处理，已决定下次更新的数据。

如果没有做缓冲的机制并且至少满足上诉的两条中的一条将会通知客户端。这种情况最后一次采集的结果会返回及时最新的值并不满足以上两条。

如果 server 做了缓存并且只有采集的数据满足了以上两条之后才进行缓存，此时数据会被更新到缓存，并且与上次缓存的作比较。如果仅仅是时间不同的话，那么只用更新时间戳就可以了。

总之，返回客户端的数据集是采集到的具有不同时间戳和质量戳的值而已。

如果一个 item 有多于一条的数据要返回的话，那么会分配重复的客户端句柄。

### 4.4.10.1 IOPCItemSamplingMgt::SetItemSamplingRate

```
HRESULT SetItemSamplingRate (
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[in, size_is(dwCount)] DWORD* pdwRequestedSamplingRate,
[out, size_is(,dwCount)]DWORD **ppdwRevisedSamplingRate,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：

单独设置某个 item 的采集率

详情看参考文档。

### 4.4.10.2 IOPCItemSamplingMgt::GetItemSamplingRate

```
HRESULT GetItemSamplingRate (
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[out, size_is(,dwCount)]DWORD * ppdwSamplingRate,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
```

描述：

获得单个 item 的采集率

详情看参考文档。

### 4.4.10.3 IOPCItemSamplingMgt::ClearItemSamplingRate

HRESULT ClearItemSamplingRate (
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE*phServer,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
清除之前设置的采集率，用 Group 默认的更新率覆盖。
详情看参考文档。
4.4.10.4 IOPCItemSamplingMgt::SetItemBufferEnable
HRESULT SetItemBufferEnable (
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE * phServer,
[in,size_is(dwCount)] BOOL*pbEnable,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
关闭或开启 Server 端的 item 的 buff 的缓冲。
详情看参考文档。

### 4.4.10.5 IOPCItemSamplingMgt::GetItemBufferEnable

HRESULT GetItemBufferEnable (
[in] DWORD dwCount,
[in,size_is(dwCount)] OPCHANDLE * phServer,
[out, size_is(dwCount)]BOOL ** ppbEnable,
[out, size_is(,dwCount)]HRESULT** ppErrors
);
描述：
查询特定 item 的缓冲开启状态。
详情看参考文档。

## 4.4.11 IConnectionPointContainer (on OPCGroup)

本接口提供的功能与 IdataObject 类似，但是本接口使用和实现起来都相对简单，并且提供了 IdataObject 中不支持的一些借口。客户端必须使用通过本接口建立的连接点对象使用新的 IOPCAsyncIO2 接口采集数据。遗留的 IOPCAsyncIO 接口时通过 IdataObject 来建立连接点的。

连接点的概念微软的文档描述的很详细，此处不做描述。这里假设读者已经详细的了解

了该技术。

IEnumConnectionPoints,IConnectionPointand IenumConnections 接口也是微软的定义范畴。

IconnectionPointContainer 的实现是基于 Group 的而非 item。

这实现了客户端和 Group 指端通过 IOPCDataCallback 接口高效的传输数据。

翻译不动了，略。

### 4.4.11.1 IConnectionPointContainer::EnumConnectionPoints

HRESULT EnumConnectionPoints(
IEnumConnectionPoints **ppEnum
);
描述：
创建一个 Group 到 Client 的连接点枚举器。
详细信息请参考文档。

### 4.4.11.2 IConnectionPointContainer:: FindConnectionPoint

HRESULT FindConnectionPoint(
REFIID riid,
IConnectionPoint **ppCP
);
描述：
找到一个 Group 和 Client 之间的枚举器。
详细信息请参考文档。

## 4.4.12 IEnumOPCItemAttributes

IEnumOPCItemAttributes 接口允许 Client 找到 Group 的内容和 item 的属性。

### 4.4.12.1 IEnumOPCItemAttributes::Next

HRESULT Next(
[in]ULONG celt,
[out, size_is(,*pceltFetched)] OPCITEMATTRIBUTES ** ppItemArray,
[out] ULONG * pceltFetched
);
描述：
找到下 Group 内的一个 item

#### 4.4.12.2 IEnumOPCItemAttributes::Skip

HRESULT Skip(
[in]ULONG celt
);
描述：
跳过没用的内容。

#### 4.4.12.3 IEnumOPCItemAttributes::Reset

HRESULT Reset(
void
);
描述：
复位枚举器到首个 item

#### 4.4.12.4 IEnumOPCItemAttributes::Clone

HRESULT Clone(
[out]IEnumOPCItemAttributes**ppEnumItemAttributes
);
描述：
创建枚举器的一个拷贝。

## 4.5 客户端接口规范

### 4.5.1

OPC 客户端使用 Connection Points 连接 OPCserver 的方法是，生成一个支持 IUnknown 和 IOPCDataCallback 接口的 COM 对象。客户端把 Iunknown 接口的指针（注意不是 IOPCDataCallback 的指针）传递给 IconnectionPoint（通过 IConnectionPointContainer:: FindConnectionPointorEnumConnectionPoints 获取）接口的 Advise 例程。OPC 服务器自动调用 QueryInterface 方法向客户端查询该回调接口。通过使用这种 COM 中被称作列集的技术保证接口可以跨越进程的访问。

客户端必须实现对下面所有方法的支持：

当指定的组内的数据改变时（OnDataChange 方法中）或者用户显式的调用 IOPCAsyncIO2 接口时，该接口（IOPCDataCallback）会被调用。

Note：当一个异步调用处于未决状态时，OPC 客户端可以改变 Group 或者 Items 的实时状态（Active Status），但是并不推荐这样做。服务器必须以一种合理的方式处理这种未规范化的情况。

Note：内存管理方式与 COM 标准一致。也就是说，服务端程序需要分配所有的 IN 参数，并且在使用完之后释放对应的内存。客户端程序释放所有的 OUT 参数分配的内存。在回调例程中不存在 OUT 参数，当然客户端程序不需要做额外的工作，由服务端程序负责内存的管理。

#### 4.5.1.1 IOPCDataCallback::OnDataChange

```
HRESULT OnDataChange(
[in]DWORD dwTransid,
[in] OPCHANDLE hGroup,
[in]HRESULT hrMasterquality,
[in]HRESULT hrMastererror,
[in]DWORD dwCount,
[in, sizeis(dwCount)] OPCHANDLE *phClientItems,
[in, sizeis(dwCount)]VARIANT * pvValues,
[in, sizeis(dwCount)] WORD *pwQualities,
[in,sizeis(dwCount)] FILETIME * pftTimeStamps,
[in,sizeis(dwCount)] HRESULT *pErrors
);
```

描述：

客户端程序使用该例程向 OPC 提交基础数据的变更和刷新请求。

| | |
|---|---|
| DwTransid | //0 参数表明是一个原始数据的订阅请求。其他参数表明是一个对 Refresh2 函数的调用，本参数作为 Refresh2 函数的参数传递。 |
| hGroup | 客户端程序保留的 Group 对象的句柄 |
| HrMasterquality | 只有当所有数据的 OPC_QUALITY_MASK 均为 OPC_QUALITY_GOOD 时才能返回 S_OK，否则返回 S_FALSE |
| HrMastererror | 只有错误信息都为 S_OK 时，才返回 S_OK。否则返回 S_FALSE。 |
| dwCount | 客户端保留的 Items 对象句柄的数目。为 0 表明本次调用是一个激活通知（参考 IOPCGroupStateMgt2::SetKeepAlive()） |
| phClientItems | 客户端投递的需要获取数据改变通知的 Items 对象句柄列表 |
| pvValues | 用来接收服务器发来的最新数据的 VARIANTS 变量列表(需要填充请求变量的类型) |
| pwQualities | 用来接收服务端程序发来的最新数据的质量戳列表 |
| pftTimeStamps | 同上（时间戳列表） |
| pErrors | 错误信息列表（返回每一项的错误信息） |

返回值：

服务端程序必须返回 S_OK。

pErrors 返回码

| | |
|---|---|
| S_OK | 返回数据质量戳为 GOOD |
| E_FAIL | 对 Item 的操作返回失败。 |
| OPC_E_BADRIGHTS | Item 不可读 |
| OPC_E_UNKNOWNITEMID | 服务端不存在该数据 |

| | |
|---|---|
| OPC_S_DATAQUEUEOVERFLOW | 表明数据更新过多导致缓冲区溢出，只能返回最新记录 |
| S_XXX，E_XXX | S_XXX 厂商特有的质量戳信息。E_XXX 厂商特有的错误信息。可以通过 GetErrorString 获取该信息。 |

**注释：**

对于返回的 S_XXX 类的错误码，客户端程序必须假设时间错，质量戳，变量值都是有效的。强烈建议 OPCserver 提供商提供对于 UNCERTAIN 或者 BAD 之类的质量戳的额外信息。

杜宇其他的 pError 返回的 FAILED 信息必须假定相关的那个 item 的的值，时间戳，质量戳都是无效的，需要将 Value 的 VARIANT 赋值为 VT_EMPTY，保证可以被正确的列集。

下面举例说明客户端程序需要回调例程的情形：

1. 变量改变事件。一个已经激活的组内处于激活状态的 Items 质量戳或者值发生改变才会调用回调接口。一般频率不会高于整个组的刷新频率。参数 TransactionID 应该赋值为 0.
2. 通过 AsyncIO2 接口异步获取数据。请求投递完成后会尽快执行，这种情况下 transactionID 应该是一个非零值。

当服务端程序获取某一个 Item 的数据发生错误时，可以通过 pErrors 返回足够的错误信息。信息中应当包含通讯故障或者设备状态方面的信息，当然也允许偷懒而仅仅返回 E_FAIL。

参考 IOPCItemSamplingMgt::SetItemSamplingRate 来获取更多关于 OPCSERVER 对于数据缓冲的处理细节方面的参考。

在做 Cleanup 期间，服务端不要忘了为每一个 Variants 调用 VariantClear（），避免内存泄露。

IOPCDataCallback::OnReadComplete

```
HRESULT OnReadComplete(
[in]DWORD dwTransid,
[in] OPCHANDLE hGroup,
[in]HRESULT hrMasterquality,
[in]HRESULT hrMastererror,
[in]DWORD dwCount,
[in, sizeis(dwCount)] OPCHANDLE *phClientItems,
[in, sizeis(dwCount)]VARIANT * pvValues,
[in, sizeis(dwCount)] WORD *pwQualities,
[in,sizeis(dwCount)] FILETIME * pftTimeStamps,
[in,sizeis(dwCount)] HRESULT *pErrors
);
```

描述：

客户端程序提供的 OPC Group 对象的异步读取数据的完成通知例程。

| | |
|---|---|
| DwTransid | 返回读请求初始化后的事务号 |
| hGroup | 客户端程序保留的 Group 对象的句柄 |
| HrMasterquality | 只有当所有数据的 OPC_QUALITY_MASK 均为 OPC_QUALITY_GOOD 时才能返回 S_OK，否则返回 S_FALSE |
| HrMastererror | 只有错误信息都为 S_OK 时，才返回 S_OK。否则返回 S_FALSE。 |
| dwCount | 客户端保留的 Items 对象句柄的数目。为 0 表明本次调用是一个激活通知（参考 IOPCGroupStateMgt2::SetKeepAlive()） |

| | |
|---|---|
| phClientItems | 客户端投递的需要获取数据改变通知的 Items 对象句柄列表 |
| pvValues | 用来接收服务器发来的最新数据的 VARIANTS 变量列表(需要填充请求变量的类型) |
| pwQualities | 用来接收服务端程序发来的最新数据的质量戳列表 |
| pftTimeStamps | 同上（时间戳列表） |
| pErrors | 错误信息列表（返回每一项的错误信息） |

返回值：
服务端程序必须返回 S_OK。
pErrors 返回码

| | |
|---|---|
| S_OK | 返回数据质量戳为 GOOD |
| E_FAIL | 对 Item 的操作返回失败。 |
| OPC_E_BADRIGHTS | Item 不可读 |
| OPC_E_UNKNOWNITEMID | 服务端不存在该数据 |
| OPC_S_DATAQUEUEOVERFLOW | 表明数据更新过多导致缓冲区溢出，只能返回最新记录 |
| S_XXX，E_XXX | S_XXX 厂商特有的质量戳信息。E_XXX 厂商特有的错误信息。可以通过 GetErrorString 获取该信息。 |

**注释：**
对于返回的 S_XXX 类的错误码，客户端程序必须假设时间错，质量戳，变量值都是有效的。
强烈建议 OPCserver 提供商提供对于 UNCERTAIN 或者 BAD 之类的质量戳的额外信息。
杜宇其他的 pError 返回的 FAILED 信息必须假定相关的那个 item 的的值，时间戳，质量戳都是无效的，需要将 Value 的 VARIANT 赋值为 VT_EMPTY，保证可以被正确的列集。
对于 AsyncIO2 的请求初始化过程中产生错误的 Item，在本例程中将不会返回结果。也就是说返回的列表有可能不是连续的，从而与投递的请求列表出现不匹配的情况。
只有 AsyncIO2 操作完成后本例程才会有可能被调用。
当服务端程序获取某一个 Item 的数据发生错误时，可以通过 pErrors 返回足够的错误信息。信息中应当包含通讯故障或者设备状态方面的信息，当然也允许偷懒而仅仅返回 E_FAIL。
IOPCDataCallback::OnWriteComplete
HRESULT OnWriteComplete(
[in]DWORD dwTransid,
[in]OPCHANDLE　hGroup,
[in]HRESULT hrMasterError,
[in]DWORD dwCount,
[in, sizeis(dwCount)] OPCHANDLE *phClientItems,
[in,sizeis(dwCount)] HRESULT * pError
);
描述：
客户端调用 AsyncIO2 进行写操作的完成通知例程。

| | |
|---|---|
| DwTransid | 返回写请求初始化后的事务号 |
| hGroup | 客户端程序保留的 Group 对象的句柄 |
| HrMasterquality | 只有当所有数据的 OPC_QUALITY_MASK 均为 OPC_QUALITY_GOOD 时才能返回 S_OK，否则返回 S_FALSE |
| HrMastererror | 只有错误信息都为 S_OK 时，才返回 S_OK。否则返回 S_FALSE。 |

| | |
|---|---|
| dwCount | 客户端保留的 Items 对象句柄的数目。为 0 表明本次调用是一个激活通知（参考 IOPCGroupStateMgt2::SetKeepAlive()） |
| phClientItems | 客户端投递的需要获取数据改变通知的 Items 对象句柄列表 |
| pvValues | 用来接收服务器发来的最新数据的 VARIANTS 变量列表(需要填充请求变量的类型) |
| pwQualities | 用来接收服务端程序发来的最新数据的质量戳列表 |
| pftTimeStamps | 同上（时间戳列表） |
| pErrors | 错误信息列表（返回每一项的错误信息） |

返回值：
服务端程序必须返回 S_OK。
pErrors 返回码

| | |
|---|---|
| S_OK | 返回数据质量戳为 GOOD |
| E_FAIL | 对 Item 的操作返回失败。 |
| OPC_E_BADRIGHTS | Item 不可写 |
| OPC_E_UNKNOWNITEMID | 服务端不存在该数据 |
| OPC_S_DATAQUEUEOVERFLOW | 表明数据更新过多导致缓冲区溢出，只能返回最新记录 |
| S_XXX，E_XXX | S_XXX 虽然可写但是包含厂商的警告信息。E_XXX 写失败，诸如设备已经下线等情况。可以通过 GetErrorString 获取该信息。 |
| | |
| | |

**注释：**
对于 AsyncIO2 的写请求初始化过程中产生错误的 Item，在本例程中将不会返回结果。也就是说返回的列表有可能不是连续的，从而与投递的请求列表出现不匹配的情况。
只有 AsyncIO2 写操作完成后本例程才会有可能被调用。
当服务端程序获取某一个 Item 的数据发生错误时，可以通过 pErrors 返回足够的错误信息。信息中应当包含通讯故障或者设备状态方面的信息，当然也允许偷懒而仅仅返回 E_FAIL。
IOPCDataCallback::OnCancelComplete
HRESULT OnCancelComplete(
[in]DWORD dwTransid,
[in]OPCHANDLE hGroup
);
描述：
客户端接收服务端的异步操作取消通知。

| | |
|---|---|
| DwTransid | 返回对应的读请求，写请求，刷新请求初始化后的事务号 |
| hGroup | 客户端程序保留的 Group 对象的句柄 |
| | |

返回值：
服务端程序必须返回 S_OK。
AsyncIO2 的 cancel 请求执行成功后才有可能接收到该通知。

## 4.5.2 IOPCShutdown

OPC 客户端使用 Connection Points 连接 OPCserver 的方法是，生成一个支持 IUnknown 和 IOPCShutdown 接口的 COM 对象。客户端把 Iunknown 接口的指针(注意不是 IOPCShutdown 的指针）传递给 IconnectionPoint（通过 IConnectionPointContainer:: FindConnectionPointorEnumConnectionPoints 获取）接口的 Advise 例程。OPC 服务器自动调用 QueryInterface 方法向客户端查询该回调接口。通过使用这种 COM 中被称作列集的技术保证接口可以跨越进程的访问。

服务端程序推出前将会调用该接口的 ShutdownRequest 方法。客户端程序要在该例程中执行清理连接和接口的一些操作。

支持多连接的客户端需要为每个 OPCServers（比如数据 server、报警或者事件 serveer 等等此类）管理单独的 shutdown 回调对象。

IOPCShutdown::ShutdownRequest

HRESULT ShutdownRequest (

[in]LPWSTR szReason

);

客户端提供该例程以便于服务端请求断开连接。之后客户端要做一些必要的清理工作，释放连接对象，Group 对象以及其他接口占用的资源。

| | |
|---|---|
| szReason | 一个可配置的字符串，包含断开连接的原因，可以提供一个空串表明无特别原因 |
| | |

返回值：

服务端程序必须返回 S_OK。

**注释：**

Shutdown 的连接点对象是一个 per COM object（标准的 COM 对象），也就是说，他代表了用 CoCreate…类函数创建的对象。一个支持多 OPCserver 的客户端需要为每一个连接维护该对象区别对待。

# 5 安装问题

假定服务器厂商会提供一个他们设备服务器所需组件的 SETUP.EXE.这个将不作进一步讨论.不同于实际的组件，影响的 OLE 软件的主要问题是 Windows 注册表和组件类别管理。问题是(a)需要做什么 (b)他们应该怎么做。

同样，某些常见的安装和注册表的论题，包括自动注册，自动代理/存根注册和登录引用计数都在 OPC 概述文档讨论。

## 5.1 组件类别

OPC Data Access 接口定义了以下组件类别.下面的列表是 CATIDs，描述符和数据访问符。
"OPC Data Access Servers Version 1.0"
CATID_OPCDAServer10 = {63D5F430-CFE4-11d1-B2C8-0060083BA1FB}
"OPC Data Access Servers Version 2.0"
CATID_OPCDAServer20 = {63D5F432-CFE4-11d1-B2C8-0060083BA1FB}
"OPC Data Access Servers Version 3.0"
CATID_OPCDAServer30 = {CC603642-66D7-48f1-B69A-B625E73652D7}

预计服务器将首先创建它使用的任何类别，然后将注册该类别。取消注册的话，服务器应该将它从该类别中删除。见 ICatRegister 有关其他信息文档。

这些 CATIDs 在其相关的 IDL 文件中定义。对于数据访问 1.0，2.0，3.0 和 XMLDA 时，CATIDs 都在 opcda.idl 定义。

## 5.2 注册表项的自定义接口

下列条目以支持自定义接口的 OPC 兼容服务器的最低要求。

1. HKEY_CLASSES_ROOT\Vendor.Drivername.Version = A Description of your server
2. HKEY_CLASSES_ROOT\Vendor.Drivername.Version\CLSID = {Your Server's unique CLSID}
3. HKEY_CLASSES_ROOT\CLSID\{Your Server's unique CLSID} = A Description of your server
4. HKEY_CLASSES_ROOT\CLSID\{Your Server's unique CLSID}\ProgID = Vendor.Drivername.Version

One or more of the following lines (inproc and/or local/remote and/or handler)

5. HKEY_CLASSES_ROOT\CLSID\{Your Server's unique CLSID}\InprocServer32 = Full Path to DLL
6. HKEY_CLASSES_ROOT\CLSID\{YourServer's unique CLSID}\LocalServer32 = Full Path to EXE
7. HKEY_CLASSES_ROOT\CLSID\{YourServer's unique CLSID}\InprocHandler32 = Full Path to DLL

1. 这个项只是建立在你的 ProgID(进程 ID)为根的子项下，并且其他子项可以进入。说明(此键的“值”)可以呈现给用户一个可用的 OPC 服务器的名称(请参见下面的例子)。它应该与第 6 行的描述相匹配。
2. CLSID 行使 CLSIDFromProgID 功能工作。即 I.e.允许系统打开一键给定的进程 id ，并取得 CLSID 作为键的值。请看下面的例子。
3. 在 OPC 行创建了一个没有值的'标志'子项。这是用于 Data Access 1.0 允许客户端浏览可

用的 OPC 服务器。在 2.0 版中，首选的方法是客户端和服务器使用组件类别。 3.0 版的方法要求客户端和服务器使用组件类别。

4. 厂商线是可选的。很简单，就是找出谁提供的供应商的一种手段，尤其是 OPC 服务器。

5. 此行只是建立你的 CLSID 作为子项关闭的 ROOT \ CLSID 下的其他子项可以进入。说明（此键的'值'）应该是一个用户服务器的说明。它应该与上面第 1 项相匹配。

6 .该进程 id 线使 ProgIDFromCLSID 功能工作。即允许系统打开一键给定的 CLSID ，并取得进程 id 作为键的值。（此功能不常用）。

7.该 InprocServer32 行或 LocalServer 32 行或 InprocHandler32 行允许 CoCreateInstance 定位一个给定 CLSID 的 DLL 或 EXE 。供应商应定义这些的至少一个。

一般情况下，作为微软的文档中所描述的自我注册是推荐两个 DLL 和 EXE 服务器来简化安装。

## 5.3 注册表项的 Proxy/Stub DLL

代理/存根 DLL 被用于从编组接口到本地或远程服务器。它直接从 IDL 代码产生，每一个 OPC 服务器应该相同。在一般的 Proxy/Stub 将使用自注册。(在生成过程中定义 REGISTER_PROXY_DLL)。因为这是完全自动和透明的，所以就不进一步讨论。

还要注意的是一个预构建和测试 proxy/stub DLL 将在 OPC Foundation 网站上提供，因此不必为供应商重建这个 DLL。

尽管供应商被允许自己的接口添加到 OPC 对象(与任何 COM 对象)，但他们从来不能修改标准的 OPC IDL 或 Proxy/Stub DLLs 此类接口。这样的接口应该总是在一个单独的供应商特定的 IDL 文件中定义，应该由一个独立的供应商特定的 Proxy/Stub DLL 进行封存处理。

# 6.数据类型、参数和结构体描述

## 6.1 Item 定义

在服务器中，ItemID 是完全限定定义的一个数据项。没有其他的信息被要求识别这个数据项，以确定该数据项的客户端能够读/写值。

ItemID(条目标识)是在 OPCITEMDEF 中被使用的，并且在其他地方是以一个空结束的字符串来标识唯一一个 OPC 数据项。标识符的语法是依赖服务器（尽管它应该只包括可打印的 UNICODE 字符），它提供了一个参考或“钥匙”在一个'项目'数据源。该项目是可以由一个 VARIANT 表示，虽然它是通常表示任何一个单个值，如模拟，数字或字符串值。

例如，如 FIC101 中的一条项目可以代表整个记录诸如一个现场总线，HartFoundation 或 Profibus 数据结构。这种行为被明确允许，但并不要求通过 OPC - 数据结构的回归被认为是供应商特定的行为。另外 FIC101.PV 可能表示如过程值的一个记录属性。采取 double 类型被任何客户端使用。

作为一个极端的例子，因为该项目的 ID 的语法是服务器特定的附加信息，如计数，工程单位定标和信号调理的信息可以被嵌入到定义的字符串中(虽然不建议这样做)。

例如：

一个支持访问现有的 DCS 的服务器，可能支持一个简单的语法，如”TIC101.PV”

一个支持低级别访问 PLC 的服务器，可能支持的语法如”COM1.STATION:42.REG:40001;0,4095,-100.0,+1234.0”。

## 6.2 访问路径

访问途径是作为一种方式，它为客户端提供一个到服务器的建议数据路径（如

尤其是调制解调器或网络接口），并说明了如何获取数据。

该 ITEM ID 提供了所有的定位和处理数据项所需要的信息。访问路径是可选的信息，可以由客户端提供。服务器被准许客户端提供一种”建议”到服务器去观察一下如何获取数据。举一个例子，如果 ItemID 表示一个电话号码，访问路径可能代表一个请求通过卫星(或跨大西洋电缆或 microwaveLink)呼叫的路线。呼叫将经过建议的路径，不论是否是您指定的访问路径，也不论该服务器能否使用。

例如，假设你想访问一个 RTU 的值，对高速调制解调器 COM1

和一个低速的调制解调器 COM2。你可以指定 COM1 作为首选的访问路径。无论哪一个工作，无论 COM2 性能多好，但你宁愿使用 COM1。

在任何情况下，由服务器和客户端的使用的访问路径是可选的。即使访问路径被提供，服务器也不需要提供功能，并且客户机不必使用它。

不支持访问路径的服务器会完全忽略任何已通过的访问路径(客户端也不会将此视为错误)。此外，查询时，对于所有项目，这些服务器将始终返回一个空的访问路径(即一个 NULL 字符串)。

## 6.3 Blob

BLOB (binary large object)，二进制大对象，是一个可以存储二进制文件的容器。在计算机中，BLOB 常常是数据库中用来存储二进制文件的字段类型。(此段是在网上找的)

我们将讨论 Blob 为什么存在，以及 Blob 的特性。

BLOB 是服务器与项目相关联的区域，服务器用它来提升访问和处理 Items 速度或这些项目的处理。这种方法已使用在特定服务器中。

这个想法是，客户端通过参考 ASCII 字符串项目，而在内部加速访问，则服务器可能需要解决这个字符串转换成一些内部服务器的具体地址，网络地址，指针到一个表中，一组指数或文件或注册号等，这地址解析可能需要相当多的时间，并且所得到的内部地址可以采用空间的任意量。此 Blob 允许服务器返回此内部地址，并允许客户端将其保存，并提供 BLOB 备份到服务器，此文件以备将来参考。该服务器可以使用 BLOB 作为'提示'，以帮助更快速地从这条项目找到下一个项目，”将 BLOB 说，我最后一次看这个标签，我发现它'这里'所以让我们看看它仍然在该位置”。然而，在所有情况下，该项目的 ID 仍然是“键”的数据。不管 BLOB 其中的内容，服务器需要确保它实际上是在引用被提及的 ITEM ID 项目。

以下是 BLOB 的特性。

通过客户端和服务器的使用是可选的。服务器可以执行”AddItems”快速建立该项目的定义，通常不会返回一个 Blob。服务器返回一个 Blob 情况下，客户端可以自由地忽略这些 Blob（虽然这可能会影响其服务器性能）。

将 BLOB 传递给 AddItems 和 ValidateItems，它是由服务器任何时间返回的 AddItems 或 ValidateItems 或 EnumItemAttributes 完成。从一个传递内容来看，返回 Blob 的大小和内容可能会有所不同。

请注意，服务器可以在任何时间更新一个 Item 的 Blob, (包括，例如，只要在客户端改变一个项目的属性).

Proper behavior of a client that wishes to support the Blob is to Enumerate the item attributes to get afresh copy of the Blobs for each item prior to deleting an item or group and to save that updated copyalong with the other application data related to the items.

## 6.4 时间戳

在 FILETIME 里的时间戳，它比其他有效的标准时间结构更简洁。Win32 函数中有许多包括不同的时间格式和时间之间的转换区。时间戳指的是 UTC 时间,这种格式是有利的,因为它总是不断增加并且是能很清晰看到。如本文前面所讨论的，时间戳应反映最佳估计时间点的相应值，被称为是准确的。如果这不是设备本身提供，那么它应该被服务器所提供。

## 6.5 Variant 数据类型的 OPC 数据项

在 NT 4.0 和 Windows95 与 DCOM 支持，所有的 VARIANT 数据类型可以通过标准编组封送处理。在自动化，类型将被强制转换为已知的自动化数据类型。

注释：

variant（VT_R4，VT_R8）实际值将包含 IEEE 浮点数。注意，这个 IEEE 标准允许某些非数字值（称为 NaN 的）以这种格式存储，使用如此的值是比较罕见的，但他们的确是被允许的。如果有这个 NaN 的值被返回，则这个质量戳将被设置成 OPC_QUALITY_BAD。

虽然 IEEE 标准允许 NANS 存储在 VT_R4 和 VT_R8，但这样的值只能被准确的原类型进行读写。他们将不会被转换到其他类型或从其他类型转换过来。当这样的值被读取(为原类型)，那么服务器一定会被返回 OPC_QUALITY_BAD 质量戳。如果这样的一个值写入（如原类型），那么客户端一定会被返回 OPC_QUALITY_BAD 质量戳。一个特定的服务器是否曾返回或接受这个值是服务器特定的。

# 6.6 常量

## 6.6.1 OPCHANDLE

OPC 句柄在 groups 和 groups 中的 items 都有使用，这个句柄的作用是能够快速地访问客户端和服务器上的 various 对象。

服务器的 handles 的内部实现处理完全是设备厂商特定的。

### 6.6.1.1 Group Handles

OPCGroups 由一个客户端和一个服务器句柄相关联组成。

服务器组句柄是整个服务器唯一的，在创建组时返回，然后由客户端的不同方法进行传递。服务器组句柄可以被假定为保持有效，直到客户端删除组和释放所有的接口。

它不应该被客户端永久地存储，因为它可能与下一次创建的 OPC 组不同。

客户端组句柄是由客户端向服务器提供。它可以是任何值，并且不是唯一的。它被包含在以帮助客户端识别回调的数据源。

实际上，如果打算使用任何客户端将在 OPC 接口（包括 IOPCAsyncIO2 和的 IConnectionPoint 的异步函数接口）给它的句柄分配一个唯一值，因为这是通过服务器给回客户端 IconnectionPoint 接口的唯一方法。

### 6.6.1.2 Item Handles

OPCItems 由一个客户端和一个服务器句柄相关联组成。

服务器项句柄在群（group）里是唯一的，在创建项时返回，然后由客户端的不同方法进行传递。服务器项句柄可以被假定为保持有效，直到客户端删除组和释放所有的接口。

它不应该被客户端永久地存储，因为它可能与下一次创建的 OPC 组不同。

客户端项句柄是由客户端向服务器提供。它可以是任何值，并且不是唯一的。它被包含在 IconnectionPoint 回调中，以帮助客户端快速识别客户端应用中的哪个对象影响了变化的数据。

实际上，如果打算使用任何客户端将在 OPC 接口（包括 IOPCAsyncIO2 和的 IConnectionPoint 的异步函数接口）给它的句柄分配一个唯一值，因为这是通过服务器给回客户端 IconnectionPoint 接口的唯一方法。

## 6.7 Structures and Masks

### 6.7.1 OPCITEMSTATE

这个结构体被用在 IOPCSyncIO::Read

```
typedef struct {
        OPCHANDLE hClient;
        FILETIME
        ftTimeStamp;
        WORD
        wQuality;
        WORD
        wReserved;
        VARIANT
        vDataValue;
  } OPCITEMSTATE
```

| Member | 描述 |
| --- | --- |
| hClient | 客户端为 Item 提供的句柄 |
| ftTimeStamp | Item 值的 UTC 时间戳，如果设备不能提供一个时间戳，那么服务器应该提供一个 |
| wQuality | Item 质量戳 |
| vDataValue | 变量值 |

**评论：**

客户端应该调用 VariantClear()去释放关联变量的内存。

variant（VT_R4，VT_R8）实际值将包含 IEEE 浮点数。注意，这个 IEEE 标准允许某些非数字值（称为 NaN 的）以这种格式存储，使用如此的值是比较罕见的，但他们的确是被允许的。如果有这个 NaN 的值被返回，则这个质量戳将被设置成 OPC_QUALITY_BAD。

虽然 IEEE 标准允许 NANS 存储在 VT_R4 和 VT_R8，但这样的值只能被准确的原类型进行读写。他们将不会被转换到其他类型或从其他类型转换过来。当这样的值被读取（为原类型），那么服务器一定会被返回 OPC_QUALITY_BAD 质量戳。如果这样的一个值写入（如原类型），那么客户端一定会被返回 OPC_QUALITY_BAD 质量戳。一个特定的服务器是否曾返回或接受这个值是服务器特定的。

### 6.7.2 OPCITEMDEF

```
typedef struct {
        [string] LPWSTRszAccessPath;
        [string] LPWSTRszItemID;
        BOOLbActive ;
        OPCHANDLEhClient;
        DWORD       dwBlobSize;
```

[size_is(dwBlobSize)] BYTE * pBlob;
VARTYPEvtRequestedDataType;
WORDwReserved;
} OPCITEMDEF;

这个结构体被使用在 IOPCItemMgt::AddItems and ValidateItems，在' Used by'下面列中之处这两个函数的每个使用成员。

| Member | Used by | 描述 |
|---|---|---|
| szAccessPath | both | 访问路径的服务器应关联此项目。按照惯例，一个指向一个 NUL 字符串<br>指针指向服务器应选择访问路径。支持访问路径是可选的。<br>注：版本 1 指出一个 NULL 指针将允许服务器挑选路径，然而在 proxy/stub 里代码传递一个 NULL 指针会导致故障的，因此是不允许的。 |
| szItemID | both | 一个空结束的字符串，OPC 数据项的唯一标识。看到 ItemID 讨论 AddItems 功能关于这个字段内容的具体信息。 |
| bActive | add | 这个 Boolean 值影响变量属性方法，如在此其他地方有所描述。 |
| hClient | add | 客户端希望关联 Item 的句柄，关于这个字段的更具体内容信息见 OPCHANDLE。 |
| dwBlobSize | both | Item 中 pBlob 大小 |
| pBlob | both | pBlob 是指向 Blob 的指针 |
| vtRequestedDataType | both | 客户端请求的数据类型。如果服务器不能提供此格式的 Item，一个错误是返回的(见 Additems ValidateItems)。<br>传递 VT_EMPTY 意味着客户端将接受服务器的典型数据类型。 |

**评论：**

关于数据类型，往往是相同的值可以返回一个以上的格式。例如，一个数值可能会返回文本（VT_BSTR）或实数（VT_R8）。这种转换通常是服务器通过 VariantChangeType 在处理。同样一个状态（扫描状态，自动/手动，报警，等）可能会返回一个整数（VT_I4），它要使用在动画或颜色选择或作为一个字符串（VT_BSTR）直接显示给用户。这第二种情况也是作为一

种枚举类型被知道，是厂商指定的。客户端厂商应该注意，本规范没有规定什么枚举存在或服务器如何将值映射到字符串。服务器厂商都极力鼓励遵循这一领域的标准，如现场总线。有关此主题的更多信息见 IEnumOPCItemAttributes。

## 6.7.3 OPCITEMRESULT

```
typedef struct {
        OPCHANDLEhServer;
        VARTYPEvtCanonicalDataType;
        WORDwReserved;
        DWORDdwAccessRights;
        DWORDdwBlobSize;
        [size_is(dwBlobSize)] BYTE * pBlob;
} OPCITEMRESULT;
```

这个结构体被使用在 IOPCItemMgt::AddItems() and ValidateItems()

| Member | Used by | 描述 |
|---|---|---|
| hServer | add | 适用于 Item 使用的服务器句柄。 |
| vtCanonicalDataType | both | 原生数据类型。仍然保持在服务器这个项目的数据类型。一个服务器不知道该规范类型中的项目（例如在启动时）必须返回 VT_EMPTY。这是一个说明该数据类型是不是客户端当前已知的，但会在某个未来的时间确定。 |
| dwAccessRights | both | 指出这个 Item 可能是只读、只写或读写的。这跟安全性不相关，而是底层硬件的性质。请参阅下面的访问权限部分 |
| dwBlobSize | both | Blob 在这个 Item 的大小。注意对于不要求和不支持的服务器来说，这个大小可能为 0 |
| pBlob | both | 指向 Blob 的指针 |

**评论：**

对于 AddItems，始终支持此功能的服务器将返回 pBlob。对于 ValidateItems ，如果 dwBlobUpdate 参数 ValidateItems 是 TRUE，它只会被退回。

在释放 OPCITEMRESULT 结构之前，客户端软件必须释放 BLOB 内存。

## 6.7.4 OPCITEMATTRIBUTES

```
typedef struct {
        [string] LPWSTRszAccessPath;
        [string] LPWSTR szItemID;
        BOOL    bActive;
        OPCHANDLEhClient;
        OPCHANDLE hServer;
        DWORDdwAccessRights;
        DWORDdwBlobSize;
        [size_is(dwBlobSize)] BYTE * pBlob;
        VARTYPEvtRequestedDataType;
        VARTYPEvtCanonicalDataType;
        OPCEUTYPEdwEUType;
        VARIANTvEUInfo;
} OPCITEMATTRIBUTES;
```

| Member | 描述 |
|---|---|
| szAccessPath | 访问客户端指定路径，如果服务器不支持这个访问路径，则返回一个空字符串的指针。 |
| szItemID | Item 唯一标识 |
| bActive | 如果 Item 当前没有激活，则返回 FALSE，否则返回 TRUE。 |
| hClient | 客户端关联 Item 的句柄 |
| hServer | 服务器引用 Item 的句柄 |
| dwAccessRights | 指出这个 Item 是只读、只写或者读写方式，这与安全性不相关，而是底层硬件性质所决定的，详见下面访问权限部分。 |
| dwBlobSize | Item 的 pBlob 大小，注意这在不支持或不需要特性的服务器的大小可能为 0。 |
| pBlob | 指向 Blob 的指针 |
| vtRequestedDataType | Item 值返回的数据类型，注意，如果这个请求的数据类型被拒绝的话，这块将返回标准类型。 |
| vtCanonicalDataType | Item 值的数据类型是仍然保存到服务器上的数据类型。服务器不知道一个 Item 的标准类型就必须返回 VT_EMPTY，这就指出客户端的数据类型是未知的，在将来某个时间会被确认。 |
| dwEUType | 指出包含在 vEUInfo 的 EU 信息的类型。<br>0：无效的 EU 信息(vEUInfo 为 VT_EMPTY)<br>1：Analog-vEUInfo 将包含一个 SAFEARRAY 的两个浮点型数(VT_ARRAY \| VT_R8)正好与 EU 的 LOW 范围与 HI 范围相对应。 |

| | |
|---|---|
| | 2 ： Enumerated-vEUInfo 将包含一个 SAFEARRAY 的字符串(VT_ARRAY \| VT_BSTR), 这个字符串包含一个字符串列表(Example: “OPEN”, “CLOSE”, “IN TRANSIT”,etc),这个列表中的字符串与连续的数字(0, 1, 2,etc.)相对应。 |
| vEUInfo | VARIANT 包含 EU 的信息，详见下面评论 |

评论：

EU 的支持是可选的。不支持此服务器将始终返回 EUType 为 0， EUInfo 为 VT_EMPTY。EU 信息（模拟或枚举类型）可以返回规范类型是任意的值：VT_I2，I4，R4，R8，BOOL，显然 UI1 在实践中的一些组合比其他更容易。该包含数据的数组的条目（VT_ARRAY），EU 信息将应用到该数组中的所有项（就像要求和规范的数据类型适用于数组中的所有项目一样）。

EU 信息由服务器向客户端提供，基本上是只读的。 OPC 不提供客户端超过 EU 设定的任何控制。

对于模拟 EU 返回的信息表示该条目值的'正常'范围。传感器或仪器故障或失灵由超出此范围的返回条目值导致。客户端软件必须能处理这样的情况。类似地，一个客户端也可以尝试写入超出此此范围的值并由服务器返回。服务器出现的确切行为（接受，拒绝，堵塞等），该情况下，一般的服务器应该能处理这个问题。

对于返回' string lookup table“这样的枚举 EU 信息表示从 0 开始的连续整数值。值的数量是由 SAFEARRAY 大小中确定。同样，强大的客户端应能处理超出列表范围的条目值和强大的服务器应能处理非法的写入值。

服务器可以有选择的枚举支持本地化。在这种情况下，服务器应该使用当前组的 ID 值。见 IOPCServer:: ADDGROUP 和 IOPCGroupStateMgt::getstate andSetState。

客户端负责释放包含任何 SafeArrays 元素的 OPCITEMATTRIBUTES 结构体中的变量。

客户端希望创建并使用一个通用的函数，如 FreeOPCITEMATTRIBUTES（PTR），以减少内存泄漏。

## 6.7.5 OPCSERVERSTATUS

```
Typedefstruct {
        FILETIMEftStartTime;
        FILETIMEftCurrentTime;
        FILETIMEftLastUpdateTime;
        OPCSERVERSTATEdwServerState;
        DWORDdwGroupCount;
        DWORDdwBandWidth;
        WORDwMajorVersion;
        WORDwMinorVersion;
        WORDwBuildNumber;
        WORDwReserved;
        [string] LPWSTRszVendorInfo;
} OPCSERVERSTATUS;
```

这个结构体被用来服务器到客户端的状态通讯，这个信息由 IOPCServer::GetStatus() 调用的

服务器提供。

| Member | 描述 |
| --- | --- |
| ftStartTime | 服务器开始时间(UTC),当服务器更改状态时，服务器实例是恒定且不复位。这个过程开始时，服务器的每个实例应该保持这个时间。 |
| ftCurrentTime | 服务器的当前时间(UTC) |
| ftLastUpdateTime | 服务器发送最后的数据区更新客户端的时间(UTC)，此值保持在一个实例基础上。 |
| dwServerState | 服务器的当前状态，OPC Server State values below 有提及。 |
| dwGroupCount | 服务器实例控制的 Groups 总数量，这主要是帮助进行诊断。 |
| dwBandWidth | 服务器特有的行为。当前使用的服务器带宽比。如果有多重连接被使用，就可能返回可以返回的“worst case”链接。请注意，超过100％的任何值表示该项目的总结合的更新率 UpdateRate 太高。如果这个值是未知，该服务器也可能返回为 0xFFFFFFFF。 |
| wMajorVersion | 服务器软件的主要版本 |
| wMinorVersion | 服务器软件的次要版本 |
| wBuildNumber | 服务器软件的 build number |
| szVendorInfo | 厂家提供关于服务器的字符串附加信息。建议在此提供该公司的名称和设备（S）的支持类型。 |

| OPCSERVERSTATE Values | 描述 |
| --- | --- |
| OPC_STATUS_RUNNING | 服务器正常运行，这是服务器的正常状态。 |
| OPC_STATUS_FAILED | 发生在服务器里的致命错误。服务器不再工作。这种情形下，厂商会有特有的恢复过程。一个 E_FAIL 的错误代码通常从其他的服务器方式返回。 |
| OPC_STATUS_NOCONFIG | 服务器没有加载配置信息将不能正常运行。注意，这就意味着服务器运行需要配置信息，如果服务器不需要配置信息，则就不会返回这个状态。 |
| OPC_STATUS_SUSPENDED | 服务器是临时暂停状态，并且不能发送或接收数据。注意，质量戳将返回 OPC_QUALITY_OUT_OF_SERVICE。 |
| OPC_STATUS_TEST | 服务器是测试模式。输出是与硬件断开连接。输入可以是真实的或可能根据供应商的实现进行模拟，质量戳一般将返回正常 |
| OPC_STATUS_COMM_FAULT | 服务器运行正常，但 |

| | |
|---|---|
| | 从数据源访问数据还是有困难。这可能是由于通信问题，或其他一些底层设备问题或者控制系统问题等。从返回的有效数据看，这可能是完全失败，<br>这意味着没有数据可用，或者部分失败，这意味着某些数据仍然可用。预期<br>受该故障影响的 Items 将单独返回一个 BAD 的质量戳。 |

## 6.7.6 Access Rights(访问权限)

这代表了服务器访问单个的 OPC 数据项的能力。注意 DWORD 的低 16 位是保留给 OPC 使用，目前包括 OPC 访问权限定义在 IDL 中，并说明如下。DWORD 的高 16 位可用于供应商特定用途。

该 OPC_READABLE 和 OPC_WRITABLE 位用来表示该 Item 是否是可读写。例如一个物理输入值一般是可读但不可写，但一个物理输出或可调参数，如值设定值或报警限值一般是可读可写的。这是可能的表示值没有回读功能的物理输出可能会被标记为可写，但无法读取。推荐在客户端应用程序使用此信息仅作为东西被用户查看。用户要尝试读取或写入的值，应始终由客户端程序被传递到服务器，当这个 Item 加入以后，不用管被返回的访问权限。如果需要，该服务器能返回 E_BADRIGHTS

另外，返回的访问权限值是不相关的安全问题。服务器预期会安全验证当前登录用户的任何读写数据，因为他们在相应的读写时，如果在出现问题时将返回在响应一个合适的供应商特定的 HRESULT。服务器应该返回两个值（OPC_READABLE 和 OPC_WRITEABLE）以指示该访问权限是当前“未知”。

| AccessRights Values | 描述 |
|---|---|
| OPC_READABLE | 客户端能读数据 Item 的值 |
| OPC_WRITEABLE | 客户端能改变数据 Item 的值 |

## 6.7.7 OPCITEMPROPERTY

一个指向这个结构体数组的指针是 OPCITEMPROPERTIES 结构体的基本元素。

```
typedef struct tagOPCITEMPROPERTY {
        VARTYPE         vtDataType;
        WORD            wReserved;
        DWORD               dwPropertyID;
        [string] LPWSTR szItemID;
        [string] LPWSTRszDescription;
        VARIANTvValue;
        HRESULThrErrorID;
} OPCITEMPROPERTY;
```

| Member | 描述 |
|---|---|

| | |
|---|---|
| vtDataType | 此属性的标准数据类型 |
| dwPropertyID | 该属性 ID 用于定义一个列表属性，请参阅表中”theItem Properties section” |
| szItemID | 一个完全合格的 ItemID，可用于访问此属性。如果一个空字符串返回，则该属性无法通过 ItemID 访问. |
| szDescription | 该属性的非本地化文本描述 |
| vValue | 当前属性值。如果不需要此值，vValue 类型设为 VT_EMPTY |
| hrErrorID | 如果失败，只有 dwPropertyID 包含一个已知的有效值，所有其他结构体成员可能包含无效数据。 |

‘Errors’ Codes for hrErrorID

| Return Code | 描述 |
|---|---|
| S_OK | 相应的属性 ID 是有效的并且结构体包含着准确的信息 |
| OPC_E_INVALID_PID | 对于 Item，传过去的属性 ID 没有定义。 |

**评论：**

调用者必须释放 szItemID 和 szDescription 字符串，并调用 VVALUE 的 VariantClear。

## 6.7.8 OPCITEMPROPERTIES

这个结构体被 IOPCBrowse::GetProperties()使用，并作为 OPCBROWSEELEMENT 结构体的一个基本元素。

```
typedef struct tagOPCITEMPROPERTIES {
        HRESULThrErrorID;
        DWORDdwNumProperties;
        [size_is(dwNumProperties)] OPCITEMPROPERTY *pItemProperties;
} OPCITEMPROPERTIES;
```

| Member | 描述 |
|---|---|
| hrErrorID | 如果失败，dwNumProperties 必须是 0 并且 pItemProperties 必须为 NULL |
| dwNumProperties | pItemProperties 数组的元素号。如果 HRESULT 失败，返回 0 |
| pItemProperties | 一个指向 OPCITEMPROPERTY 数组的指针，如果 HRESULT 失败，返回 NULL |

‘Errors’ Codes for hrErrorID

| Return Code | 描述 |
|---|---|
| S_OK | 项目 ID 所请求的属性被返回。 |
| S_FALSE | 一个或多个请求的属性都为这个无效的 ItemID |
| OPC_ E_INVALIDITEMID | 项目名称不符合服务器的语法 |

| | |
|---|---|
| OPC_E_UNKNOWNITEMID | 该项目在该服务器的地址空间不识别 |

**评论：**

调用者必须释放 pItemProperties.

## 6.7.9 OPCBROWSEELEMENT 项目 ID 所请求的属性

这个结构体被 IOPCBrowse::Browse()使用

```
typedef struct tagOPCBROWSEELEMENT {
        [string] LPWSTRszName;
        [string] LPWSTRszItemID;
        DWORDdwFlagValue;
        OPCITEMPROPERTIES ItemProperties;
} OPCBROWSEELEMENT;
```

| Member | 描述 |
|---|---|
| szName | 命名空间指向与短用户友好的部分<br>元素。这是在将用于显示目标的一个树控件.. |
| szItemID | Item 的唯一标识，它能跟 AddItems, Browse 或 GetProperties 一块使用 |
| dwFlagValue | 当前位有效设置是：<br>OPC_BROWSE_HASCHILDREN = 0x01<br>OPC_BROWSE_ISITEM = 0x02<br>如果第一个位被设置（OPC_BROWSE_HASCHILDREN）那么这表明返回的元素有子项并且可以用于后续的浏览。如果用太多时间消耗在一台服务器上，以确定一个元素是否有子项，那么这个值应该被设置为 TRUE，以使该客户端被给予机会试图浏览潜在的子项。<br>如果第二位被置位（OPC_BROWSE_ISITEM），那么这个基本元素是一个 Item 可以用来读，写，订阅。<br>如果第二位被设置，szItemID 是一个空字符串，则<br>这个元素是一个“hint”，与一个有效的 Item 相对。 |
| ItemProperties | 与 Item 相关联的 Item 属性 |

**评论：**

在 Browse()调用中使用的 szItemID 必须是 szItemID 的返回值，这个值是从以前调用浏览或空字符串（用于表示一个高级浏览）的一个浏览元素值。

如果 dwFlagValue 的第二位（OPC_BROWSE_ISITEM）被设置，该 szItemID 是一个完全合格的描述符，它可以被用于进一步浏览元素，并作为项目的 id。注意，有可能作为单一的元素包含一个项目，有子项（如复杂的数据项）。

调用者必须释放字符串 szName 的和 szItemID。

### 6.7.10 OPCITEMVQT

这个结构体被 IOPCItemIO::WriteVQT 使用

```
typedef struct {
        VARIANTvDataValue;
        BOOLbQualitySpecified;
        WORDwQuality;
        BOOLbTimeStatmpSpecified;
        FILETIMEftTimeStamp;
} OPCITEMVQT;
```

| Member | 描述 |
|---|---|
| vDataValue | 值本身作为变量，如果初始化为 VT_EMPTY，则没有值去写 |
| bQualitySpecified | true 值表示写入一个有效的质量。<br>false 值表示没有质量来写 |
| wQuality | Item 的质量戳 |
| bTimeStampSpecified | true 值表示写入一个有效的时间。<br>false 值表示没有时间来写 |
| ftTimeStamp | Item 的时间戳 |

**评论：**

这个结构体被用来写入一组变量值、时间、质量戳到设备。

## 6.8 OPC 质量标志

这些标志代表了一个项目的数据值的质量状态。这样做的目的是相似的，但比现场总线数据质量指标稍微简单（在 H1 的最终规格第 4.4.1 节）。这种设计使得它很容易为服务器和客户端应用程序来决定他们想要实现多少功能。

质量的低 8 位标记当前定义的形式三个位字段：质量，子状态和极限状态。 8 质量位分布如下:

QQSSSSLL

质量的高 8 位词可用于特定于厂商的使用。如果这些位被使用，标准的 OPC 质量位仍必须尽可能准确设置客户端返回的数据。此外，它是任何客户端解释供应商特有的品质信息，以确保该服务器提供它使用相同的“规则”作为客户的责任。这种协商的细节没有被指定为该标准，虽然 QueryInterface 调用服务器 IMyQualityDefinitions 等特定于供应商的接口是一种可能的方法。

OPC 标准质量的细节部分:

| QQ | BIT VALUE （位值） | DEFINE （定义） | DESCRIPTION （描述） |
|---|---|---|---|
| 0 | 00SSSSLL | Bad | 值没有用不是子状态的原因 |

| | | | |
|---|---|---|---|
| 1 | 01SSSSLL | Uncertain | 该值的质量不确定不是子状态的原因。 |
| 2 | 10SSSSLL | N/A | 没有用到 OPC |
| 3 | 11SSSSLL | Good | 该值的质量是好的。 |

**评论：**

服务器不支持质量信息必须返回3(好)。也可以接受服务器简单地返回坏或好(0 x00 或0 xc0)并总是返回 0 的子状态和限制。

建议客户检查质量最小位域的所有结果(即使他们不检查 substatus 或限制字段)。

即使表示“坏”值,值字段的内容仍然必须定义良好的变体,尽管它不包含一个准确的值。这是在客户端应用程序简化了错误处理。例如,客户总是希望调用 VariantClear()的结果同步阅读。

如果服务器没有已知值返回,那么一些合理的默认值应该返回一个空字符串 NULL 或数值 0。

子状态位域

这一领域的布局取决于质量字段的值。

子状态的坏质量：

| SSSS | BIT VALUE | DEFINE | DESCRIPTION |
|---|---|---|---|
| 0 | 000000LL | Non-specific （非特异性） | 该值是不好的，但没有具体的原因是众所周知的。 |
| 1 | 000001LL | Configuration Error （配置错误） | 有一些服务器的具体配置出现问题。例如有问题的项目已经从配置中删除。 |
| 2 | 000010LL | Not Connected | 输入需要逻辑上连接而不是东西。这可能反映了质量没有价值,价值等原因可能没有提供的数据源。 |
| 3 | 000011LL | Device Failure | 设备故障被检测到。 |
| 4 | 000100LL | Sensor Failure | 传感器故障被检测到（在'限制'字段可以在某些情况下提供更多的诊断信息)。 |
| 5 | 000101LL | Last Known Value | 通讯失败。然而，最后的已知值是可用的。需要注意的是该值的“年龄”可能在 OPCITEMSTATE 的时间戳确定。 |
| 6 | 000110LL | Comm Failure | 通讯失败。没有最后 |

| | | | |
|---|---|---|---|
| | | | 已知值可用。 |
| 7 | 000111LL | Out of Service | 该区块是关闭扫描或以其他方式锁定。这质量也用来当项目或包含该项目组的激活状态无效。 |
| 8 | 001000LL | Waiting for Initial Data | 之后的项目被添加到一个组，它可能需要一些时间让服务器真正获得这些项目的值。在这种情况下，客户端可能会执行一个读（从缓存），或建立的 ConnectionPoint 根据认购及/或该等认购执行一个刷新的值可用之前。此子状态只能从 OPC DA3.0 或更高版本的服务器。 |
| 9-15 | | N/A | 留给未来的 OPC 使用 |

**评论：**

不支持子状态的服务器应该返回 0。需要注意的是一个'老'值可能与质量设置为 BAD（0）返回与子状态设置为 5。这是用于与现场总线一致性。

规范。这是其中一个客户端可以假设一个'坏'的值仍然是由应用程序使用的唯一情况。

子状态的不确定质量：

| SSSS | BIT VALUE | DEFINE | DESCRIPTION |
|---|---|---|---|
| 0 | 010000LL | Non-specific | 没有具体的理由的值是不确定的。 |
| 1 | 010001LL | Last Usable Value | 无论是在写这个值已经停止这样做。返回值应该被视为“过时”。请注意，这不同于坏值与子状态 5(最后一次值)。该状态是与上一个'取'值可检测的通信错误特别相关。这个错误是与一些外部源故障，“把'到的东西的价值在可接受的时间内有关。需要注意的是该值的 |

| | | | |
|---|---|---|---|
| | | | “年龄”可以从OPCITEMSTATE时间戳来确定。 |
| 2-3 | | N/A | Not used by OPC |
| 4 | 010100LL | Sensor Not Accurate | 任一值“挂”在传感器的限制（在这种情况下，限制字段应该被设置为 1 或 2）或传感器也被称为是出于校准的通过某种形式的内部诊断（1 在这种情况下限制字段应为 0）。 |
| 5 | 010101LL | Engineering Units Exceeded | 返回的值是该参数定义的范围之外。请注意，在这种情况下（根据现场总线规格）的'限制'字段指示已超出其限制，但并不一定意味着该值不能移动得更远超出范围。 |
| 6 | 010110LL | Sub-Normal | 该值是从多个来源得到并且具有小于的良好来源规定的数量。 |
| 7-15 | | N/A | 保留供将来使用的OPC |

**评论**:
不支持子状态的服务器应该返回 0。
子状态质量好：

| SSSS | BIT VALUE | DEFINE | DESCRIPTION |
|---|---|---|---|
| 0 | 110000LL | Non-specific | 该值是好的。没有特别的条件。 |
| 1-5 | | N/A | Not used by OPC |
| 6 | 110110LL | Local Override | 价值已被覆盖。这通常意味着输入已经断开连接和手动输入值“强迫”。 |
| 7-15 | | N/A | reserved for future OPC use |

评论：
不支持子状态的服务器应该返回 0。

极限位域
极限位域是有效的，无论质量和子状态的。在某些情况下，如传感器故障，它可以提供有用的诊断信息。

| LL | BIT VALUE | DEFINE | DESCRIPTION |
|---|---|---|---|
| 0 | QQSSSS00 | Not Limited | The value is free to move up or down. |
| 1 | QQSSSS01 | Low Limited | The value has 'pegged' at some lower limit. |
| 2 | QQSSSS10 | High Limited | The value has 'pegged' at some high limit. |
| 3 | QQSSSS11 | Constant | The value is a constant and cannot move. |

**评论**:
不支持限制服务器应该返回 0。
符号等式定义的值和“质量”部分的 OPC 头文件。

# 7 OPC 错误码概述

我们试图通过确定共同的一般性问题和定义，可以在很多情况下被重用的错误代码，尽量减少唯一错误的次数。 OPC 服务器应仅返回列在本文中的各种错误码或是标准的微软错误码。需要注意的是 OLE 本身也会经常返回除了那些在本规范中列出的错误码（如 RPC 错误）

客户端最重要的是要检查返回失败的任何错误码。除此之外，一个强大的、用户友好的客户端应该假设服务器可以返回任何错误代码，调用 GetErrorString 功能，以提供有关这些错误的用户可读信息。

| 这是常用的 OPC 服务器标准的 COM 错误 | 描述 |
|---|---|
| E_FAIL | 未指定的错误 |
| E_INVALIDARG | 一个或多个参数的值无效。这一般用在预期问题不大可能或是容易确定（例如，当只有一个参数）这里更具体错误的位置中。 |
| E_NOINTERFACE | 不支持此接口 |
| E_NOTIMPL | 未实现 |
| E_OUTOFMEMORY | 没有足够的内存来完成请求的操作。这可以发生在服务器需要分配内存来完成所请求操作的任何时间 |
| CONNECT_E_ADVISELIMIT | 溢出提醒限值 |
| OLE_E_NOCONNECTION | 不存在的连接 |
| DV_E_FORMATETC | 在 FORMATETC 指定了无效或未注册的格式 |

| OPC 的特定错误 | 描述 |
|---|---|
| OPC_E_INVALIDHANDLE | 句柄值无效。注：客户端不会传递无效句柄值给服务器。若发生此错误，这可能由于在客户端或者在服务器中的编程错误。 |
| OPC_E_BADTYPE | 服务器无法转换指定格式/请求数据类型和规范数据类型之间的数据。 |
| OPC_E_PUBLIC | 在一个公共组中所请求的操作不能完成。 |
| OPC_E_BADRIGHTS | 该项目 AccessRights 不允许操作。 |

| | |
|---|---|
| OPC_E_UNKNOWNITEMID | 该项目的 ID 没有在服务器地址空间中定义（附加或验证）或在服务器的地址空间中不存在（读或写）。 |
| OPC_E_INVALIDITEMID | 项目 ID 不符合服务器的语法。 |
| OPC_E_INVALIDFILTER | 过滤字符串无效 |
| OPC_E_UNKNOWNPATH | 该项目的到服务器的访问路径是未知的。 |
| OPC_E_RANGE | 数值溢出。 |
| OPC_E_DUPLICATENAME | 不允许重复名称。 |
| OPC_S_UNSUPPORTEDRATE | 服务器不支持请求的数据速率，但会使用最接近的有效的数据速率。 |
| OPC_S_CLAMP | 写值可执行，但输出被钳制。 |
| OPC_S_INUSE | 该对象被引用，操作不能被执行 |
| OPC_E_INVALIDCONFIGFILE | 服务器的配置文件格式无效 |
| OPC_E_NOTFOUND | 请求的对象（例如，一个公共组）未找到 |
| OPC_E_INVALID_PID | 项目传递的属性 ID 无效。 |
| OPC_E_DEADBANDNOTSET | 该条目死区尚未设定。 |
| OPC_E_DEADBANDNOTSUPPORTED | 该条目不支持死区。 |
| OPC_E_NOBUFFERING | 当采样速率比组更新速率更快，服务器不支持的需要采集的数据项的缓冲 |
| OPC_E_INVALIDCONTINUATIONPOINT | The continuation point 无效。 |

| | |
|---|---|
| OPC_S_DATAQUEUEOVERFLOW | 该条目不是每个检测到的变化都已正确返回。即服务器缓存达到极限，需清除最早的数据。只有最近的数据被提供。服务器应只删除那些有在缓冲区可用较新的样本项目的最早的数据。这将允许单一取样的更旧的条目被返回到客户端。 |
| OPC_E_RATENOTSET | 有一个特定的条目没有设置采样率。在这种情况下，该条目默认为组的更新率。 |
| OPC_E_NOTSUPPORTED | 如果客户端尝试写任意数值，质量和时间戳的组合而不是服务器支持的请求组合（可能是一个单一的量如只是时间戳），那么服务器将不执行任何写操作，并返回此错误值。 |

您将在附录中看到，这些错误代码使用 ITF_FACILITY。这意味着，他们是特定的上下文（即特定的 OPC）。调用应用程序应先检查与服务器提供的错误（即调用 GetErrorString）。

0x0000H~0x0200H 的错误码（HRESULT 的低位字）是给微软使用保留的（尽管有些是在 OPC 1.0 错误码无意中被使用）； 0x0200H~0x7FFFH 为 OPC 使用而保留的；0x8000H~0xFFFFH 用于特定厂商。

# 8 附录 A - opcerror.h

下面信息和相关的文本仅用于作为最终的参考，并且可从说明书进行更新。该信息应该从相关的文本文件，复制到名为 opcerror.h。为了确保您获得属于规范的特定版本相关联的文本文件，使用的命名约定如下：

opcerror_V.xy.h 其中 V 表示主版本和 XY 代表副版本号。例如，如果在规范版本为 3.0，文本文件的名称将是 opcerror_3.00.h。如果规范版本是 3.15，文本文件的名称将是 opcerror_3.15.h

```
1997-2003 The OPC Foundation
//
// DISCLAIMER:
// This code is provided by the OPC Foundation solely to assist in
// understanding and use of the appropriate OPC Specification(s) and
// may be used as set forth in the License Grant section of the OPC
// support of any sort and is subject to the Warranty and Liability
// Disclaimers which appear in the printed OPC Specification.
// MODIFICATION LOG:
//
// Date By Notes
// 1997/05/12 ACC Removed Unused messages
// Added OPC_S_INUSE, OPC_E_INVALIDCONFIGFILE,
// OPC_E_NOTFOUND
// 1997/05/12 ACC Added OPC_E_INVALID_PID
// 2002/08/12 CRT Added new error codes for DA3.0
// resource block.
//
#ifndef __OPCERROR_H
#define __OPCERROR_H
#if _MSC_VER >= 1000
#pragma once
#endif
// The 'Facility' is set to the standard for COM interfaces or
// FACILITY_ITF (i.e. 0x004) The 'Code' is set in the range defined OPC
// Commmon for DA (i.e. 0x0400 to 0x04FF)
// Note that for backward compatibility not all existing codes use this
// range.
//
//
// Values are 32 bit values layed out as follows:
//
// 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
// 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
// +---+-+-+----------------------+-------------------------------+
```

```
// |Sev|C|R| Facility | Code |
// +---+-+-+----------------------+------------------------------+
//
// where
//
// Sev - is the severity code
//
// 00 - Success
// 01 - Informational
// 10 - Warning
// 11 - Error
//
// C - is the Customer code flag
//
// R - is a reserved bit
//
// Facility - is the facility code
//
// Code - is the facility's status code
//
//
// Define the facility codes
//
//
// Define the severity codes
//
//
// MessageId: OPC_E_INVALIDHANDLE
//
// MessageText:
//
// The value of the handle is invalid.
//
#define OPC_E_INVALIDHANDLE ((HRESULT)0xC0040001L)
//
// MessageId: OPC_E_BADTYPE
//
// MessageText:
// The server cannot convert the data between the specified format and/or
requested data type and the canonical data type.
//
#define OPC_E_BADTYPE ((HRESULT)0xC0040004L)
//
// MessageId: OPC_E_PUBLIC
```

```
//
// The filter string was not valid.
// MessageId: OPC_E_UNKNOWNPATH
#define OPC_E_UNKNOWNPATH ((HRESULT)0xC004000AL)
//
// MessageText:
//
// The requested operation cannot be done on a public group.
//
#define OPC_E_PUBLIC ((HRESULT)0xC0040005L)
//
// MessageId: OPC_E_BADRIGHTS
//
// MessageText:
//
// The item's access rights do not allow the operation.
//
#define OPC_E_BADRIGHTS ((HRESULT)0xC0040006L)
//
// MessageId: OPC_E_UNKNOWNITEMID
//
// MessageText:
// The item ID is not defined in the server address space or no longer
exists in the server address space.
//
#define OPC_E_UNKNOWNITEMID ((HRESULT)0xC0040007L)
//
// MessageId: OPC_E_INVALIDITEMID
//
// MessageText:
//
// The item ID does not conform to the server's syntax.
//
#define OPC_E_INVALIDITEMID ((HRESULT)0xC0040008L)
//
// MessageId: OPC_E_INVALIDFILTER
//
// MessageText:
//
//
#define OPC_E_INVALIDFILTER ((HRESULT)0xC0040009L)
//
//
// MessageText:
```

```
//
// The item's access path is not known to the server.
//
//
// MessageId: OPC_E_RANGE
//
// MessageId: OPC_E_DUPLICATENAME
// A value passed to write was accepted but the output was clamped.
// MessageId: OPC_S_INUSE
//
// MessageId: OPC_E_INVALIDCONFIGFILE
#define OPC_E_INVALIDCONFIGFILE ((HRESULT)0xC0040010L)
//
// MessageText:
// The value was out of range.
//
#define OPC_E_RANGE ((HRESULT)0xC004000BL)
//
//
// MessageText:
//
// Duplicate name not allowed.
//
#define OPC_E_DUPLICATENAME ((HRESULT)0xC004000CL)
//
// MessageId: OPC_S_UNSUPPORTEDRATE
//
// MessageText:
//
// The server does not support the requested data rate but will use the
closest available rate.
//
#define OPC_S_UNSUPPORTEDRATE ((HRESULT)0x0004000DL)
//
// MessageId: OPC_S_CLAMP
//
// MessageText:
//
//
#define OPC_S_CLAMP ((HRESULT)0x0004000EL)
//
//
// MessageText:
```

```
// The operation cannot be performed because the object is bering
referenced.
//
#define OPC_S_INUSE ((HRESULT)0x0004000FL)
//
//
// MessageText:
//
// The server's configuration file is an invalid format.
//
// MessageText:
// The continuation point is not valid.
// MessageId: OPC_E_NOTFOUND
//
// MessageText:
//
// The requested object (e.g. a public group) was not found.
//
#define OPC_E_NOTFOUND ((HRESULT)0xC0040011L)
//
// MessageId: OPC_E_INVALID_PID
//
// MessageText:
//
// The specified property ID is not valid for the item.
//
#define OPC_E_INVALID_PID ((HRESULT)0xC0040203L)
//
// MessageId: OPC_E_DEADBANDNOTSET
//
// MessageText:
//
// The item deadband has not been set for this item.
//
#define OPC_E_DEADBANDNOTSET ((HRESULT)0xC0040400L)
//
// MessageId: OPC_E_DEADBANDNOTSUPPORTED
//
//
// The item does not support deadband.
//
#define OPC_E_DEADBANDNOTSUPPORTED ((HRESULT)0xC0040401L)
//
// MessageId: OPC_E_NOBUFFERING
```

```
//
// MessageText:
//
// The server does not support buffering of data items that are collected
at a faster rate than the group update rate.
//
#define OPC_E_NOBUFFERING ((HRESULT)0xC0040402L)
//
// MessageId: OPC_E_INVALIDCONTINUATIONPOINT
//
// MessageText:
//
//
#define OPC_E_INVALIDCONTINUATIONPOINT ((HRESULT)0xC0040403L)
//
//
//
#define OPC_E_NOTSUPPORTED ((HRESULT)0xC0040406L)
// MessageId: OPC_S_DATAQUEUEOVERFLOW
//
// MessageText:
//
// Data Queue Overflow - Some value transitions were lost.
//
#define OPC_S_DATAQUEUEOVERFLOW ((HRESULT)0x00040404L)
//
// MessageId: OPC_E_RATENOTSET
//
// MessageText:
//
// Server does not support requested rate.
#define OPC_E_RATENOTSET ((HRESULT)0xC0040405L)
//
// MessageId: OPC_E_NOTSUPPORTED
//
// MessageText:
//
// The server does not support writing of quality and/or timestamp.
#endif // ifndef __OPCERROR_H
```

# 9 附录 B -数据访问 IDL 规范

下面信息和相关的文本仅用于作为最终的参考，并且可从说明书进行更新。该信息应该从相关的文本文件，并复制到名为 opcda.idl。为了确保您获得属于规范的特定版本相关联的文本文件，使用的命名约定如下：

opcda_V.xy.idl 其中 V 表示主版本和 XY 代表副版本号。例如,如果在规范版本为 3.0，文本文件的名称将是 opcda_3.00.idl。如果规范版本是 3.15，文本文件的名称将是 opcda_3.15.idl

使用命令行 MIDL / / Oicf opcda.idl。

由此产生的 OPCDA.H 文件应该包含在所有客户端和服务器。

由此产生的 OPCDA_I.C 文件定义的接口 ID，应该链接到所有的客户端和服务器。

注：此 IDL 文件和代理/存根不应该被以任何方式修改。如果您添加了供应商特定的接口到你的服务器（这是允许的），你必须生成一个单独的供应商特定的 IDL 文件来描述只有那些接口和一个单独的供应商特定 ProxyStub 的 DLL 来安排那些接口。

注：OPCCOMN.IDL 的讨论，请参阅在 OPC 概述文件（OPCOVW.DOC）。

```
//============================================================================
===========
// TITLE: opcda.idl
// // Interface declarations for the OPC Data Access specifications. //
understanding and use of the appropriate OPC Specification(s) and may be
// in the printed OPC Specification. // // to avoid conflict with 'old'
OPCDA Automation uuids
```

```
// CONTENTS:
//
//
// (c) Copyright 1997-2002 The OPC Foundation
// ALL RIGHTS RESERVED.
//
// DISCLAIMER:
// This code is provided by the OPC Foundation solely to assist in
// used as set forth in the License Grant section of the OPC Specification.
// This code is provided as-is and without warranty or support of any sort
// and is subject to the Warranty and Liability Disclaimers which appear
//
// MODIFICATION LOG:
// Date By Notes
// ---------- --- -----
// 1997/05/12 ACC fixed UNCERTAIN bits, add AsyncIO2, OPCDataCallback,
// OPCItemProperties, BROWSE_TO
//
// 1998/06/19 ACC change V2 uuids prior to final release
// Change name of 3 methods on AsyncIO2 to
// Cancel2,SetEnable,GetEnable to eliminate conflicts
//
import "ocidl.idl";
cpp_quote("#define CATID_OPCDAServer30 IID_CATID_OPCDAServer30")
cpp_quote("#define CATID_XMLDAServer10 IID_CATID_XMLDAServer10")
OPC_DS_DEVICE
OPC_LEAF,
OPCBROWSETYPE;
{
// 2002/02/10 CRT Added IOPCItemDeadbandMgt Interface
// Added IOPCItemSamplingMgt Interface
// Added IOPCBrowse Interface
// Added IOPCServer2 Interface // 2002/08/21 RSA Added asynchrounous UUIDs.
Fixed formatting. // Added IOPCServer2::RemoveGroupEx //
//================================================================
===========
// Added IOPCItemIO Interface
// Added new BAD Quality status mask
//
// 2002/08/09 CRT Added IOPCITEMVQTstructure
// Moved #defines to Library section and “typed” them
// Added definition for Category Ids
//
//
```

```
// 2002/08/30 CRT Added IOPCSyncIO2::Read
//
// 2002/09/20 CRT Added bit masks for Browse method
// 2002/10/03 CRT Added IOPCAsyncIO3 Interface
// Added IOPCGroupStateMgt2 Interface
//
// 2003/03/03 RSA Added fields to ensure natural byte alignment for new
structures.
//
import "oaidl.idl";
import "objidl.idl";
// Category ID declarations (defined as interfaces to ensure they show
up in the typelib).
[uuid(63D5F430-CFE4-11d1-B2C8-0060083BA1FB)] interface
CATID_OPCDAServer10 : IUnknown {}
[uuid(63D5F432-CFE4-11d1-B2C8-0060083BA1FB)] interface
CATID_OPCDAServer20 : IUnknown {}
[uuid(CC603642-66D7-48f1-B69A-B625E73652D7)] interface
CATID_OPCDAServer30 : IUnknown {}
[uuid(3098EDA4-A006-48b2-A27F-247453959408)] interface
CATID_XMLDAServer10 : IUnknown {}
cpp_quote("#define CATID_OPCDAServer10 IID_CATID_OPCDAServer10")
cpp_quote("#define CATID_OPCDAServer20 IID_CATID_OPCDAServer20")
//=================================================================
===========
// Structures, Typedefs and Enumerations.
typedef DWORD OPCHANDLE;
typedef enum tagOPCDATASOURCE
{
OPC_DS_CACHE = 1,
}
OPCDATASOURCE;
typedef enum tagOPCBROWSETYPE
{
OPC_BRANCH = 1,
OPC_FLAT
}
typedef enum tagOPCNAMESPACETYPE
OPC_NS_HIERARCHIAL = 1,
OPC_NS_FLAT
}
OPCNAMESPACETYPE;
OPC_BROWSE_DOWN,
OPCBROWSEDIRECTION;
```

```
typedef enum tagOPCSERVERSTATE
OPC_STATUS_FAILED,
OPC_STATUS_TEST,
OPC_ENUM_PRIVATE,
}
HRESULT hrStatus;
typedef enum tagOPCBROWSEDIRECTION
{
OPC_BROWSE_UP = 1,
OPC_BROWSE_TO
}
typedef enum tagOPCEUTYPE
{
OPC_NOENUM = 0,
OPC_ANALOG,
OPC_ENUMERATED
}
OPCEUTYPE;
{
OPC_STATUS_RUNNING = 1,
OPC_STATUS_NOCONFIG,
OPC_STATUS_SUSPENDED,
OPC_STATUS_COMM_FAULT
}
OPCSERVERSTATE;
typedef enum tagOPCENUMSCOPE
{
OPC_ENUM_PRIVATE_CONNECTIONS = 1,
OPC_ENUM_PUBLIC_CONNECTIONS,
OPC_ENUM_ALL_CONNECTIONS,
OPC_ENUM_PUBLIC,
OPC_ENUM_ALL
OPCENUMSCOPE;
typedef struct tagOPCGROUPHEADER
{
DWORD dwSize;
DWORD dwItemCount;
OPCHANDLE hClientGroup;
DWORD dwTransactionID;
}
OPCGROUPHEADER;
typedef struct tagOPCITEMHEADER1
{
OPCHANDLE hClient;
```

```
DWORD dwValueOffset;
WORD wQuality;
WORD wReserved;
FILETIME ftTimeStampItem;
}
OPCITEMHEADER1;
typedef struct tagOPCITEMHEADER2
{
OPCHANDLE hClient;
DWORD dwValueOffset;
WORD wQuality;
WORD wReserved;
}
OPCITEMHEADER2;
typedef struct tagOPCGROUPHEADERWRITE
{
OPCHANDLE hClient;
OPCITEMHEADERWRITE;
}
typedef struct tagOPCSERVERSTATUS
FILETIME ftCurrentTime;
WORD wBuildNumber;
}
typedef struct tagOPCITEMDEF
[string] LPWSTR szItemID;
WORD wReserved;
[size_is(dwBlobSize)] BYTE* pBlob;
OPCEUTYPE dwEUType;
OPCITEMATTRIBUTES;
DWORD dwItemCount;
OPCHANDLE hClientGroup;
DWORD dwTransactionID;
HRESULT hrStatus;
}
OPCGROUPHEADERWRITE;
typedef struct tagOPCITEMHEADERWRITE
{
HRESULT dwError;
}
typedef struct tagOPCITEMSTATE
{
OPCHANDLE hClient;
FILETIME ftTimeStamp;
WORD wQuality;
```

```
WORD wReserved;
VARIANT vDataValue;
OPCITEMSTATE;
{
FILETIME ftStartTime;
FILETIME ftLastUpdateTime;
OPCSERVERSTATE dwServerState;
DWORD dwGroupCount;
DWORD dwBandWidth;
WORD wMajorVersion;
WORD wMinorVersion;
WORD wReserved;
[string] LPWSTR szVendorInfo;
OPCSERVERSTATUS;
{
[string] LPWSTR szAccessPath;
BOOL bActive ;
OPCHANDLE hClient;
DWORD dwBlobSize;
[size_is(dwBlobSize)] BYTE* pBlob;
VARTYPE vtRequestedDataType;
}
OPCITEMDEF;
typedef struct tagOPCITEMATTRIBUTES
{
[string] LPWSTR szAccessPath;
[string] LPWSTR szItemID;
BOOL bActive;
OPCHANDLE hClient;
OPCHANDLE hServer;
DWORD dwAccessRights;
DWORD dwBlobSize;
VARTYPE vtRequestedDataType;
VARTYPE vtCanonicalDataType;
VARIANT vEUInfo;
}
}
typedef enum tagOPCBROWSEFILTER
OPC_BROWSE_FILTER_BRANCHES,
typedef struct tagOPCITEMRESULT
{
OPCHANDLE hServer;
VARTYPE vtCanonicalDataType;
WORD wReserved;
```

```
DWORD dwAccessRights;
DWORD dwBlobSize;
[size_is(dwBlobSize)] BYTE* pBlob;
OPCITEMRESULT;
typedef struct tagOPCITEMPROPERTY
{
VARTYPE vtDataType;
WORD wReserved;
DWORD dwPropertyID;
[string] LPWSTR szItemID;
[string] LPWSTR szDescription;
VARIANT vValue;
HRESULT hrErrorID;
DWORD dwReserved;
}
OPCITEMPROPERTY;
typedef struct tagOPCITEMPROPERTIES
{
HRESULT hrErrorID;
DWORD dwNumProperties;
[size_is(dwNumProperties)] OPCITEMPROPERTY* pItemProperties;
DWORD dwReserved;
}
OPCITEMPROPERTIES;
typedef struct tagOPCBROWSEELEMENT
{
[string] LPWSTR szName;
[string] LPWSTR szItemID;
DWORD dwFlagValue;
DWORD dwReserved;
OPCITEMPROPERTIES ItemProperties;
}
OPCBROWSEELEMENT;
typedef struct tagOPCITEMVQT
{
VARIANT vDataValue;
BOOL bQualitySpecified;
WORD wQuality;
WORD wReserved;
BOOL bTimeStampSpecified;
DWORD dwReserved;
FILETIME ftTimeStamp;
}
OPCITEMVQT;
```

```
{
OPC_BROWSE_FILTER_ALL = 1,
OPC_BROWSE_FILTER_ITEMS,
}
OPCBROWSEFILTER;
//====================================================================
===========
// IOPCServer
[
object,
{
uuid(39c13a4d-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCServer : IUnknown
{
HRESULT AddGroup(
[in, string] LPCWSTR szName,
[in] BOOL bActive,
[in] DWORD dwRequestedUpdateRate,
[in] OPCHANDLE hClientGroup,
[unique, in] LONG* pTimeBias,
[unique, in] FLOAT* pPercentDeadband,
[in] DWORD dwLCID,
[out] OPCHANDLE* phServerGroup,
[out] DWORD* pRevisedUpdateRate,
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
HRESULT GetErrorString(
[in] HRESULT dwError,
[in] LCID dwLocale,
[out, string] LPWSTR* ppString
);
HRESULT GetGroupByName(
[in, string] LPCWSTR szName,
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
HRESULT GetStatus(
[out] OPCSERVERSTATUS** ppServerStatus
);
HRESULT RemoveGroup(
[in] OPCHANDLE hServerGroup,
```

```
[in] BOOL bForce
);
HRESULT CreateGroupEnumerator(
[in] OPCENUMSCOPE dwScope,
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
}
//====================================================================
===========
// IOPCServerPublicGroups
[
object,
uuid(39c13a4e-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCServerPublicGroups : IUnknown
HRESULT GetPublicGroupByName(
[in, string] LPCWSTR szName,
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
HRESULT RemovePublicGroup(
[in] OPCHANDLE hServerGroup,
[in] BOOL bForce
);
}
{
);
[out] LPENUMSTRING* ppIEnumString
//====================================================================
===========
// IOPCBrowseServerAddressSpace
[
object,
uuid(39c13a4f-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCBrowseServerAddressSpace: IUnknown
HRESULT QueryOrganization(
[out] OPCNAMESPACETYPE* pNameSpaceType
HRESULT ChangeBrowsePosition(
[in] OPCBROWSEDIRECTION dwBrowseDirection,
[in, string] LPCWSTR szString
```

```
);
HRESULT BrowseOPCItemIDs(
[in] OPCBROWSETYPE dwBrowseFilterType,
[in, string] LPCWSTR szFilterCriteria,
[in] VARTYPE vtDataTypeFilter,
[in] DWORD dwAccessRightsFilter,
);
HRESULT GetItemID(
[in] LPWSTR szItemDataID,
[out, string] LPWSTR* szItemID
);
HRESULT BrowseAccessPaths(
[in, string] LPCWSTR szItemID,
[out] LPENUMSTRING* ppIEnumString
);
}
//====================================================================
===========
// IOPCGroupStateMgt
[
object,
uuid(39c13a50-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCGroupStateMgt : IUnknown
{
HRESULT GetState(
[out] DWORD* pUpdateRate,
[out] BOOL* pActive,
[out, string] LPWSTR* ppName,
[out] LONG* pTimeBias,
[out] FLOAT* pPercentDeadband,
[out] DWORD* pLCID,
[out] OPCHANDLE* phClientGroup,
[out] OPCHANDLE* phServerGroup
);
HRESULT SetState(
[unique, in] DWORD* pRequestedUpdateRate,
[out] DWORD* pRevisedUpdateRate,
[unique, in] BOOL* pActive,
[unique, in] LONG* pTimeBias,
[unique, in] FLOAT* pPercentDeadband,
[unique, in] DWORD* pLCID,
[unique, in] OPCHANDLE* phClientGroup
```

```
);
HRESULT SetName(
[in, string] LPCWSTR szName
);
HRESULT CloneGroup(
[in, string] LPCWSTR szName,
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
}
//=====================================================================
===========
// IOPCPublicGroupStateMgt
[
object,
uuid(39c13a51-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCPublicGroupStateMgt : IUnknown
{
HRESULT GetState(
[out] BOOL* pPublic
);
HRESULT MoveToPublic(
void
);
}
//=====================================================================
===========
// IOPCSyncIO
[
object,
uuid(39c13a52-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCSyncIO : IUnknown
{
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
);
```

```
HRESULT Write(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] VARIANT* pItemValues,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
}
//=================================================================
===========
// IOPCAsyncIO
[
object,
uuid(39c13a53-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCAsyncIO : IUnknown
{
HRESULT Read(
[in] DWORD dwConnection,
[in] OPCDATASOURCE dwSource,
);
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out] DWORD* pTransactionID,
[out, size_is(,dwCount)] HRESULT** ppErrors
HRESULT Write(
[in] DWORD dwConnection,
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] VARIANT* pItemValues,
[out] DWORD* pTransactionID,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT Refresh(
[in] DWORD dwConnection,
[in] OPCDATASOURCE dwSource,
[out] DWORD* pTransactionID
);
HRESULT Cancel(
[in] DWORD dwTransactionID
);
}
```

```
//====================================================================
===========
// IOPCItemMgt
[
object,
uuid(39c13a54-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCItemMgt: IUnknown
{
HRESULT AddItems(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCITEMDEF* pItemArray,
[out, size_is(,dwCount)] OPCITEMRESULT** ppAddResults,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT ValidateItems(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCITEMDEF* pItemArray,
[in] BOOL bBlobUpdate,
[out, size_is(,dwCount)] OPCITEMRESULT** ppValidationResults,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT RemoveItems(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT SetActiveState(
[in] DWORD dwCount,
[in] BOOL bActive,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT SetClientHandles(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] OPCHANDLE* phClient,
[out, size_is(,dwCount)] HRESULT** ppErrors

[in] ULONG celt,
[in] DWORD dwTransid,
);
HRESULT SetDatatypes(
[in] DWORD dwCount,
```

```
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] VARTYPE* pRequestedDatatypes,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT CreateEnumerator(
[in] REFIID riid,
[out, iid_is(riid)] LPUNKNOWN* ppUnk
);
}
//====================================================================
===========
// IEnumOPCItemAttributes
[
object,
uuid(39c13a55-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IEnumOPCItemAttributes : IUnknown
{
HRESULT Next(
[out, size_is(,*pceltFetched)] OPCITEMATTRIBUTES** ppItemArray,
[out] ULONG* pceltFetched
);
HRESULT Skip(
[in] ULONG celt
);
HRESULT Reset(
void
);
HRESULT Clone(
[out] IEnumOPCItemAttributes** ppEnumItemAttributes
);
}
//====================================================================
===========
// IOPCDataCallback
[
object,
uuid(39c13a70-011e-11d0-9675-0020afd8adb3),
pointer_default(unique)
]
interface IOPCDataCallback : IUnknown
{
HRESULT OnDataChange(
```

```
[in] DWORD dwTransid,
[in] OPCHANDLE hGroup,
[in] HRESULT hrMasterquality,
[in] HRESULT hrMastererror,
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phClientItems,
[in, size_is(dwCount)] VARIANT* pvValues,
[in, size_is(dwCount)] WORD* pwQualities,
[in, size_is(dwCount)] FILETIME* pftTimeStamps,
[in, size_is(dwCount)] HRESULT* pErrors
);
HRESULT OnReadComplete(
uuid(39c13a71-011e-11d0-9675-0020afd8adb3),
[out] DWORD* pdwCancelID,
[in] OPCHANDLE hGroup,
[in] HRESULT hrMasterquality,
[in] HRESULT hrMastererror,
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phClientItems,
[in, size_is(dwCount)] VARIANT* pvValues,
[in, size_is(dwCount)] WORD* pwQualities,
[in, size_is(dwCount)] FILETIME* pftTimeStamps,
[in, size_is(dwCount)] HRESULT* pErrors
);
HRESULT OnWriteComplete(
[in] DWORD dwTransid,
[in] OPCHANDLE hGroup,
[in] HRESULT hrMastererr,
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* pClienthandles,
[in, size_is(dwCount)] HRESULT* pErrors
);
HRESULT OnCancelComplete(
[in] DWORD dwTransid,
[in] OPCHANDLE hGroup
);
}
//=====================================================================
===========
// IOPCAsyncIO2
[
object,
pointer_default(unique)
]
```

```
interface IOPCAsyncIO2 : IUnknown
{
HRESULT Read(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in] DWORD dwTransactionID,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT Write(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] VARIANT* pItemValues,
[in] DWORD dwTransactionID,
[out] DWORD* pdwCancelID,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT Refresh2(
[in] OPCDATASOURCE dwSource,
[in] DWORD dwTransactionID,
[out] DWORD* pdwCancelID
);
HRESULT Cancel2(
[in] DWORD dwCancelID
);
HRESULT SetEnable(
[in] BOOL bEnable
);
HRESULT GetEnable(

pointer_default(unique)
HRESULT SetItemDeadband(
[in, size_is(dwCount)] FLOAT* pPercentDeadband,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out] BOOL* pbEnable
);
}
//====================================================================
===========
// IOPCItemProperties
[
object,
uuid(39c13a72-011e-11d0-9675-0020afd8adb3),
]
interface IOPCItemProperties : IUnknown
```

```
{
HRESULT QueryAvailableProperties (
[in] LPWSTR szItemID,
[out] DWORD* pdwCount,
[out, size_is(,*pdwCount)] DWORD** ppPropertyIDs,
[out, size_is(,*pdwCount)] LPWSTR** ppDescriptions,
[out, size_is(,*pdwCount)] VARTYPE** ppvtDataTypes
);
HRESULT GetItemProperties (
[in] LPWSTR szItemID,
[in] DWORD dwCount,
[in, size_is(dwCount)] DWORD* pdwPropertyIDs,
[out, size_is(,dwCount)] VARIANT** ppvData,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT LookupItemIDs(
[in] LPWSTR szItemID,
[in] DWORD dwCount,
[in, size_is(dwCount)] DWORD* pdwPropertyIDs,
[out, string, size_is(,dwCount)] LPWSTR** ppszNewItemIDs,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
}
//=============================================================================
===========
// IOPCItemDeadbandMgt
[
object,
uuid(5946DA93-8B39-4ec8-AB3D-AA73DF5BC86F),
pointer_default(unique)
]
interface IOPCItemDeadbandMgt : IUnknown
{
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT GetItemDeadband(
[in] DWORD dwCount,
[out, size_is(,dwCount)] FLOAT** ppPercentDeadband,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT ClearItemDeadband(
[in] DWORD dwCount,
```

```
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
}
HRESULT GetItemSamplingRate (
[out, size_is(,dwCount)] DWORD** ppdwSamplingRate,
HRESULT SetItemBufferEnable(
]
//=================================================================
===========
// IOPCItemSamplingMgt
[
object,
uuid(3E22D313-F08B-41a5-86C8-95E95CB49FFC),
pointer_default(unique)
]
interface IOPCItemSamplingMgt : IUnknown
{
HRESULT SetItemSamplingRate (
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] DWORD* pdwRequestedSamplingRate,
[out, size_is(,dwCount)] DWORD** ppdwRevisedSamplingRate,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT ClearItemSamplingRate(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] BOOL* pbEnable,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT GetItemBufferEnable(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] BOOL** ppbEnable,
[out, size_is(,dwCount)] HRESULT** ppErrors
```

```
);
}
//====================================================================
===========
// IOPCBrowse
[
object,
uuid(39227004-A18F-4b57-8B0A-5235670F4468),
pointer_default(unique)
interface IOPCBrowse : IUnknown
{
[in] DWORD dwItemCount,
[in, string, size_is(dwItemCount)] LPWSTR* pszItemIDs,
HRESULT GetProperties(
[in] BOOL bReturnPropertyValues,
[in] DWORD dwPropertyCount,
[in, size_is(dwPropertyCount)] DWORD* pdwPropertyIDs,
[out, size_is(,dwItemCount)] OPCITEMPROPERTIES** ppItemProperties
);
HRESULT Browse(
[in, string] LPWSTR szItemID,
object,
[out, size_is(,dwCount)] WORD** ppwQualities,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
[in] DWORD dwCount,
[in,out, string] LPWSTR* pszContinuationPoint,
[in] DWORD dwMaxElementsReturned,
[in] OPCBROWSEFILTER dwBrowseFilter,
[in, string] LPWSTR szElementNameFilter,
[in, string] LPWSTR szVendorFilter,
[in] BOOL bReturnAllProperties,
[in] BOOL bReturnPropertyValues,
[in] DWORD dwPropertyCount,
[in, size_is(dwPropertyCount)] DWORD* pdwPropertyIDs,
[out] BOOL* pbMoreElements,
[out] DWORD* pdwCount,
[out, size_is(,*pdwCount)] OPCBROWSEELEMENT** ppBrowseElements
);
}
// IOPCItemIO
uuid(85C0B427-2893-4cbc-BD78-E5FC5146F08F),
//====================================================================
===========
```

```
[
pointer_default(unique)
]
interface IOPCItemIO: IUnknown
{
[in] DWORD dwCount,
[in, size_is(dwCount)] LPCWSTR* pszItemIDs,
[out, size_is(,dwCount)] VARIANT** ppvValues,
HRESULT Read(
[in, size_is(dwCount)] DWORD* pdwMaxAge,
[out, size_is(,dwCount)] FILETIME** ppftTimeStamps,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT WriteVQT(
[in] DWORD dwCount,
[in, size_is(dwCount)] LPCWSTR* pszItemIDs,
[in, size_is(dwCount)] OPCITEMVQT* pItemVQT,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
}
//====================================================================
===========
// IOPCSyncIO2
[
object,
pointer_default(unique)
]
uuid(730F5F0F-55B1-4c81-9E18-FF8A0904E1FA),
interface IOPCSyncIO2: IOPCSyncIO
{
HRESULT ReadMaxAge(
[in] DWORD dwCount,
[in, size_is(dwCount)] DWORD* pdwMaxAge,
[out, size_is(,dwCount)] VARIANT** ppvValues,
[out, size_is(,dwCount)] FILETIME** ppftTimeStamps,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[out, size_is(,dwCount)] WORD** ppwQualities,
HRESULT WriteVQT (
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] OPCITEMVQT* pItemVQT,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
pointer_default(unique)
[in, size_is(dwCount)] OPCITEMVQT* pItemVQT,
```

```
);
[out] DWORD* pdwKeepAliveTime
}
//====================================================================
===========
// IOPCAsyncIO3
[
object,
uuid(0967B97B-36EF-423e-B6F8-6BFF1E40D39D),
]
interface IOPCAsyncIO3: IOPCAsyncIO2
{
HRESULT ReadMaxAge(
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in] DWORD dwTransactionID,
[out] DWORD* pdwCancelID,
);
[in] DWORD dwCount,
[in, size_is(dwCount)] OPCHANDLE* phServer,
[in, size_is(dwCount)] DWORD* pdwMaxAge,
[out, size_is(,dwCount)] HRESULT** ppErrors
HRESULT WriteVQT(
[in] DWORD dwTransactionID,
[out] DWORD* pdwCancelID,
[out, size_is(,dwCount)] HRESULT** ppErrors
);
HRESULT RefreshMaxAge(
[in] DWORD dwMaxAge,
[in] DWORD dwTransactionID,
[out] DWORD* pdwCancelID
);
}
//====================================================================
===========
[
// IOPCGroupStateMgt2
uuid(8E368666-D72E-4f78-87ED-647611C61C9F),
pointer_default(unique)
]
interface IOPCGroupStateMgt2 : IOPCGroupStateMgt
{
HRESULT SetKeepAlive(
[in] DWORD dwKeepAliveTime,
```

```
[out] DWORD* pdwRevisedKeepAliveTime
HRESULT GetKeepAlive(
);
}
//=========================================================================
===========
// Type Library
[
uuid(3B540B51-0378-4551-ADCC-EA9B104302BF),
version(3.00),
helpstring("OPC Data Access 3.00 Type Library")
]
library OPCDA
{
//=========================================================================
=======
// Category IDs
const WORD OPC_QUALITY_BAD = 0x00;
const WORD OPC_QUALITY_NOT_CONNECTED = 0x08;
const WORD OPC_QUALITY_LAST_USABLE = 0x44;
const WORD OPC_QUALITY_SUB_NORMAL = 0x58;
interface CATID_OPCDAServer10;
interface CATID_OPCDAServer20;
interface CATID_OPCDAServer30;
interface CATID_XMLDAServer10;
//=========================================================================
=======
// Constants
module Constants
{
// category description strings.
const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA10 = L"OPC Data Access Servers
Version 1.0";
const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA20 = L"OPC Data Access Servers
Version 2.0";
const LPCWSTR OPC_CATEGORY_DESCRIPTION_DA30 = L"OPC Data Access Servers
Version 3.0";
const LPCWSTR OPC_CATEGORY_DESCRIPTION_XMLDA10 = L"OPC XML Data Access
Servers Version 1.0";
// values for access rights mask.
const DWORD OPC_READABLE = 0x01;
const DWORD OPC_WRITEABLE = 0x02;
// values for browse element flags.
const DWORD OPC_BROWSE_HASCHILDREN = 0x01;
```

```
const DWORD OPC_BROWSE_ISITEM = 0x02;
}
//==========================================================================
=======
// Qualities
module Qualities
{
// Values for fields in the quality word
const WORD OPC_QUALITY_MASK = 0xC0;
const WORD OPC_STATUS_MASK = 0xFC;
const WORD OPC_LIMIT_MASK = 0x03;
// Values for QUALITY_MASK bit field
const WORD OPC_QUALITY_UNCERTAIN = 0x40;
const WORD OPC_QUALITY_GOOD = 0xC0;
// STATUS_MASK Values for Quality = BAD
const WORD OPC_QUALITY_CONFIG_ERROR = 0x04;
const WORD OPC_QUALITY_DEVICE_FAILURE = 0x0c;
const WORD OPC_QUALITY_SENSOR_FAILURE = 0x10;
const WORD OPC_QUALITY_LAST_KNOWN = 0x14;
const WORD OPC_QUALITY_COMM_FAILURE = 0x18;
const WORD OPC_QUALITY_OUT_OF_SERVICE = 0x1C;
const WORD OPC_QUALITY_WAITING_FOR_INITIAL_DATA = 0x20;
// STATUS_MASK Values for Quality = UNCERTAIN
const WORD OPC_QUALITY_SENSOR_CAL = 0x50;
const WORD OPC_QUALITY_EGU_EXCEEDED = 0x54;
// STATUS_MASK Values for Quality = GOOD
const WORD OPC_QUALITY_LOCAL_OVERRIDE = 0xD8;
// Values for Limit Bitfield
const WORD OPC_LIMIT_OK = 0x00;
const WORD OPC_LIMIT_LOW = 0x01;
const WORD OPC_LIMIT_HIGH = 0x02;
const WORD OPC_LIMIT_CONST = 0x03;
}
//==========================================================================
=======
// Properties
module Properties
{
// property ids.
const DWORD OPC_PROPERTY_DATATYPE = 1;
const DWORD OPC_PROPERTY_VALUE = 2;
const DWORD OPC_PROPERTY_QUALITY = 3;
const DWORD OPC_PROPERTY_TIMESTAMP = 4;
const DWORD OPC_PROPERTY_ACCESS_RIGHTS = 5;
```

```
const DWORD OPC_PROPERTY_SCAN_RATE = 6;
const DWORD OPC_PROPERTY_EU_TYPE = 7;
const DWORD OPC_PROPERTY_EU_INFO = 8;
const DWORD OPC_PROPERTY_EU_UNITS = 100;
const DWORD OPC_PROPERTY_DESCRIPTION = 101;
const DWORD OPC_PROPERTY_HIGH_EU = 102;
const DWORD OPC_PROPERTY_LOW_EU = 103;
const DWORD OPC_PROPERTY_HIGH_IR = 104;
const DWORD OPC_PROPERTY_LOW_IR = 105;
const DWORD OPC_PROPERTY_CLOSE_LABEL = 106;
const DWORD OPC_PROPERTY_OPEN_LABEL = 107;
const DWORD OPC_PROPERTY_TIMEZONE = 108;
const DWORD OPC_PROPERTY_CONDITION_STATUS = 300;
const DWORD OPC_PROPERTY_ALARM_QUICK_HELP = 301;
const DWORD OPC_PROPERTY_ALARM_AREA_LIST = 302;
const DWORD OPC_PROPERTY_PRIMARY_ALARM_AREA = 303;
const DWORD OPC_PROPERTY_CONDITION_LOGIC = 304;
const DWORD OPC_PROPERTY_LIMIT_EXCEEDED = 305;
const DWORD OPC_PROPERTY_DEADBAND = 306;
const DWORD OPC_PROPERTY_HIHI_LIMIT = 307;
const DWORD OPC_PROPERTY_HI_LIMIT = 308;
const DWORD OPC_PROPERTY_LO_LIMIT = 309;
const DWORD OPC_PROPERTY_LOLO_LIMIT = 310;
const DWORD OPC_PROPERTY_CHANGE_RATE_LIMIT = 311;
const DWORD OPC_PROPERTY_DEVIATION_LIMIT = 312;
const DWORD OPC_PROPERTY_SOUND_FILE = 313;
// property descriptions.
const LPCWSTR OPC_PROPERTY_DESC_DATATYPE = L"Item Canonical Data Type";
const LPCWSTR OPC_PROPERTY_DESC_VALUE = L"Item Value";
const LPCWSTR OPC_PROPERTY_DESC_QUALITY = L"Item Quality";
const LPCWSTR OPC_PROPERTY_DESC_TIMESTAMP = L"Item Timestamp";
const LPCWSTR OPC_PROPERTY_DESC_ACCESS_RIGHTS = L"Item Access Rights";
const LPCWSTR OPC_PROPERTY_DESC_SCAN_RATE = L"Server Scan Rate";
const LPCWSTR OPC_PROPERTY_DESC_EU_TYPE = L"Item EU Type";
const LPCWSTR OPC_PROPERTY_DESC_EU_INFO = L"Item EU Info";
const LPCWSTR OPC_PROPERTY_DESC_EU_UNITS = L"EU Units";
const LPCWSTR OPC_PROPERTY_DESC_DESCRIPTION = L"Item Description";
const LPCWSTR OPC_PROPERTY_DESC_HIGH_EU = L"High EU";
const LPCWSTR OPC_PROPERTY_DESC_LOW_EU = L"Low EU";
const LPCWSTR OPC_PROPERTY_DESC_HIGH_IR = L"High Instrument Range";
const LPCWSTR OPC_PROPERTY_DESC_LOW_IR = L"Low Instrument Range";
const LPCWSTR OPC_PROPERTY_DESC_CLOSE_LABEL = L"Contact Close Label";
const LPCWSTR OPC_PROPERTY_DESC_OPEN_LABEL = L"Contact Open Label";
const LPCWSTR OPC_PROPERTY_DESC_TIMEZONE = L"Item Timezone";
```

```
const LPCWSTR OPC_PROPERTY_DESC_CONDITION_STATUS = L"Condition Status";
const LPCWSTR OPC_PROPERTY_DESC_ALARM_QUICK_HELP = L"Alarm Quick Help";
const LPCWSTR OPC_PROPERTY_DESC_ALARM_AREA_LIST = L"Alarm Area List";
const LPCWSTR OPC_PROPERTY_DESC_PRIMARY_ALARM_AREA = L"Primary Alarm
Area";
const LPCWSTR OPC_PROPERTY_DESC_CONDITION_LOGIC = L"Condition Logic";
const LPCWSTR OPC_PROPERTY_DESC_LIMIT_EXCEEDED = L"Limit Exceeded";
const LPCWSTR OPC_PROPERTY_DESC_DEADBAND = L"Deadband";
const LPCWSTR OPC_PROPERTY_DESC_HIHI_LIMIT = L"HiHi Limit";
const LPCWSTR OPC_PROPERTY_DESC_HI_LIMIT = L"Hi Limit";
const LPCWSTR OPC_PROPERTY_DESC_LO_LIMIT = L"Lo Limit";
const LPCWSTR OPC_PROPERTY_DESC_LOLO_LIMIT = L"LoLo Limit";
const LPCWSTR OPC_PROPERTY_DESC_CHANGE_RATE_LIMIT = L"Rate of Change
Limit";
const LPCWSTR OPC_PROPERTY_DESC_DEVIATION_LIMIT = L"Deviation Limit";
const LPCWSTR OPC_PROPERTY_DESC_SOUND_FILE = L"Sound File";
}
//============================================================================
=======
// Synchronous Interfaces
interface IOPCServer;
interface IOPCServerPublicGroups;
interface IOPCBrowseServerAddressSpace;
interface IOPCGroupStateMgt;
interface IOPCPublicGroupStateMgt;
interface IOPCSyncIO;
interface IOPCAsyncIO;
interface IOPCDataCallback;
interface IOPCItemMgt;
interface IEnumOPCItemAttributes;
interface IOPCAsyncIO2;
interface IOPCItemProperties;
interface IOPCItemDeadbandMgt;
interface IOPCItemSamplingMgt;
interface IOPCBrowse;
interface IOPCItemIO;
interface IOPCSyncIO2;
interface IOPCAsyncIO3;
interface IOPCGroupStateMgt2;
};
```