# OPC Клиенты
# На C++ и C#
# Изд. 1

Федоренко Денис
Fedorenko.Dennis@gmail.com


www.opcfoundation.com

## ОГЛАВЛЕНИЕ

# ОСНОВЫ OPC

В этом разделе рассматривается технология OPC и самые общие принципы ее функционирования. Для людей, которые уже сталкивались в своей практической деятельности с OPC, здесь не откроется ничего нового и они могут приступать к рассмотрению материала по разработке OPC клиентов.

А для тех, кто только начинает свое знакомство с обменом данными между приложениями автоматизации знание этого крайне важно для дальнейшей разработки клиентов и серверов OPC.

OPC (OLE for Process Management) - промышленный стандарт, созданный консорциумом всемирно известных производителей оборудования и программного обеспечения, такими как Fischer Rosemount, Rockwell Software, Intellution и пр., при участии Microsoft, который описывает интерфейс обмена данными между различными источниками данных и программным обеспечением.

OPC основывается на технологии OLE/COM/DCOM компании Microsoft, Inc. (не считая OPC UA, в основу которой положено SOA).

Главной целью было предоставить разработчикам систем диспетчеризации некоторую независимость от конкретного типа источника данных. В настоящее время, OPC используется повсеместно: не только для обмена данными с аппаратным обеспечением, но и для связи одного приложения с другим, для связи с СУБД и пр.

Для того, чтобы описать что такое OPC, воспользуемся любимым примером OPC Foundation, а именно – использование принтеров.

Предположим, что Вы использует какой-либо высокоуровневый текстовый редактор для создания документов. После того, как документ создан, его необходимо распечатать посредством принтера, условно производства фирмы «А». Для того, чтобы это сделать, Ваш текстовый редактор должен уметь работать с принтером «А», более того, если производители текстового редактора хотят, чтобы он пользовался популярностью, то он должен иметь поддержку и принтеров «Б», «В» и так почти до бесконечности.

Реализовать возможность работы с каждым принтером в самом текстовом редакторе - подход совершенно непрактичный:

во-первых, растет сложность редактора;

во-вторых, с каждым новым принтером необходимо дополнять список поддерживаемых команд;

в-третьих, сколько еще на одной рабочей станции будет стоять программ, которые будут нуждаться в услугах принтеров и нести в себе эти же наборы команд.

Вместо такого подхода, как известно, все специфические команды принтера возлагаются на драйвер, в то время как конкретные приложения «общаются» уже с драйвером принтера, а не обращаются к нему напрямую.

Таким образом, текстовый редактор ничего не «знает» о тонкостях реализации принтера, а видит его лишь через призму драйвера, интерфейс которого стандартизирован. Теперь, если мы хотим добавить поддержку принтера в наше приложение необходимо лишь реализовать возможность работы с драйвером печатающих устройств.

По такому же принципу работает и OPC (см. рисунок). Основная идея заключается в том, что у нас есть некое программное приложение, которое здесь представлено компьютером Client, которому необходимо получать данные из определенного количества разнородных источников, например, ПЛК, интеллектуального полевого оборудования, другого программного обеспечения, СУБД и т.п.

Для этого приложению необходимо иметь большое количество встроенных драйверов и быть «жестко» привязанным к конкретному источнику данных. С ростом функциональности приложения будет расти и количество драйверов.

В случае же использования OPC, к источнику прилагается OPC сервер, который общается с источником на "понятном ему языке" (native API), а полученные от источника данные передает клиентам уже посредством интерфейсов OPC.

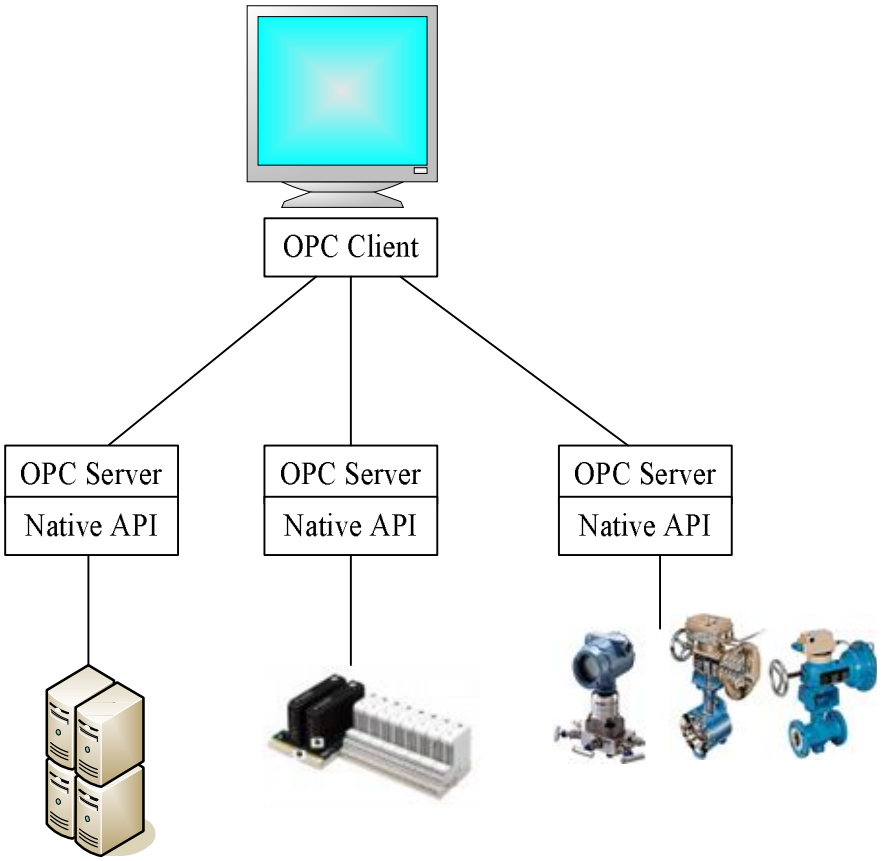



Рисунок 1. – Использование OPC

Таким образом, клиент находится в полном неведении о тонкостях реализации источника данных, зная лишь то, что он поддерживает спецификацию OPC для обмена данными.

OPC формирует лишь форму обмена данными, содержание регламентируется различными спецификациями, самые распространенные из которых:

OPC DA (Data Access). Самая распространенная спецификация, которая реализована в 99% OPC приложений. Интерфейсы OPC DA предназначены для чтения, записи и мониторинга переменных, представляющих собой текущее значение контролируемых параметров.

OPC A&E (Alarms and Events). Интерфейс обеспечивает получение событий и тревог из источника. Событие в рамках этой спецификации – это оповещение о возникновении какого-либо производственного события. Тревога ж – это оповещение, которое информирует клиента об изменениях в условиях протекания процесса. Некоторые тревоги и события требует подтверждения (квитирования), статус которого тоже можно передавать посредством OPC A&E.

OPC HDA (Historical Data Access). Интерфейс предназначен для доступа к историческим (архивным) данными. Различные источники исторических данных, такие как СУБД, архивы SCADA/DCS и могут быть опрошены единым образом

OPC UA (Unified Architecture). Платформо-независимая спецификация, основанная на SOA, которая обеспечивает унифицированный обмен любым типом данных: реального времени, историческими и тревогами между различными платформами

Существуют также спецификации XML-DA, Batch и пр., но они не так распространены.

## РАБОТА СО СПЕЦИФИКАЦИЕЙ OPC DA

Как уже упоминалось, спецификация OPC DA предназначена для обмена данными в реальном времени и является самой распространенной. На сегодняшний день практически каждая SCADA предоставляет функциональность как OPC DA сервера, так и клиента.

### Объектная модель OPC DA и краткое описание интерфейсов

В основу OPC DA положена объектная модель COM, поэтому, на самом деле, спецификация есть ни что иное как описание необходимых компонентов (объектов) и поддерживаемых ими интерфейсов.
Чтобы иметь возможность работы с приложениями, написанными посредством любого языка программирования OPC DA предоставляет два набора интерфейсов - Custom и Automation.

Первый используется для взаимодействия с компилируемыми языками программирования (например, C++), второй с интерпретируемыми (например, VBA). Custom-клиенты взаимодействуют с OPC напрямую, а OPC Automation через «обертку» OLE Automation.

На самом деле, такое разделение свойственно всем COM компонентам, которые "хотят" работать с обоими видами языков программирования. Причина в том, что COM основан на использовании таблицы виртуальных функций, которая поддерживается только компилируемыми языками.

Т.к. все примеры реализованы на компилируемых языках, то далее рассматривается взаимодействие с OPC через Custom интерфейс.

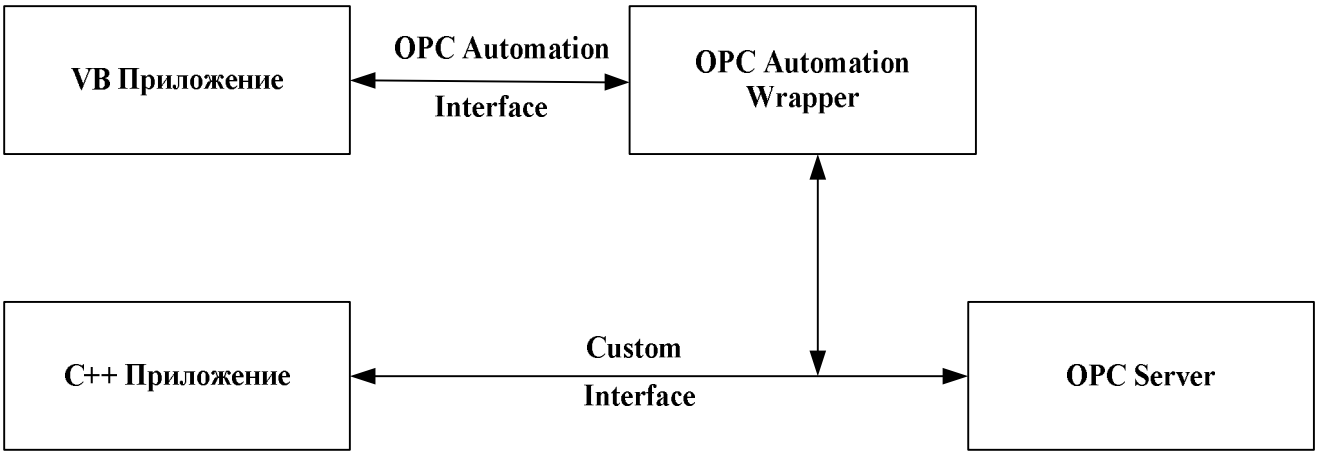



Рисунок 2. - Структура работы с OPC посредством различных языков программирования

Технология OPC непрерывно развивается сообществом и как результат, на сегодняшний день существует три основных версии спецификации OPC DA: DA 1.0, DA 2.0, DA 3.0.

От версии к версии происходит аннулирование и добавление интерфейсов, что естественно должно находит свое отражение и на реализации клиентов. Ситуацию спасает тот факт, что большая часть производителей OPC серверов обеспечивают поддержку всех доступных версий, что обеспечивает поддержку всего спектра клиентов.

Ниже приведено описание работы со спецификацией 2.0 (точнее 2.05), а также указаны замечания относительно спецификации 3.0. Спецификации 1.0 не рассматривается в силу своей давности

OPC предоставляет два вида объектов:

Объект OPC сервера

Объект OPC группы

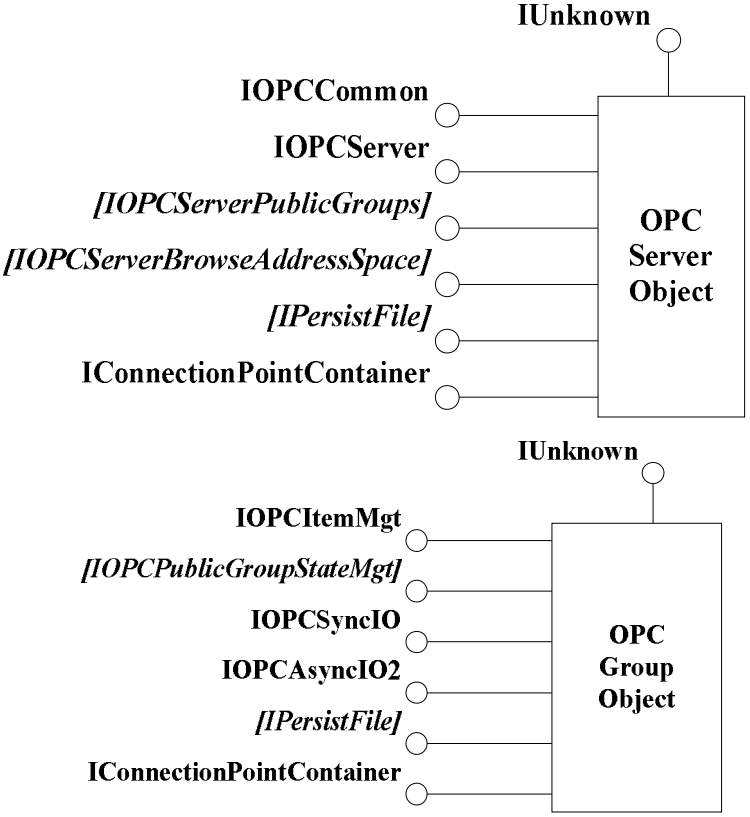
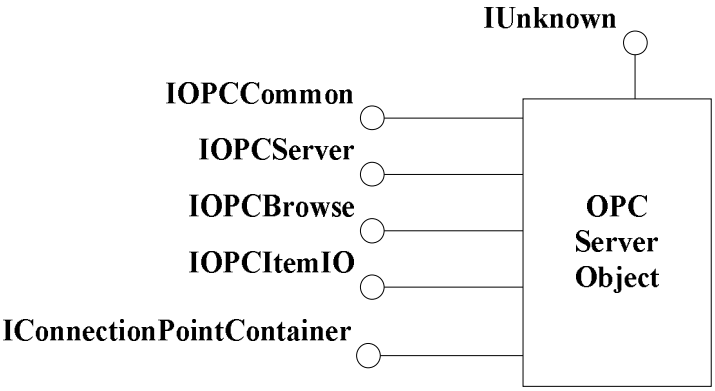



Рисунок 3.1. - Компоненты OPC сервера и OPC группы и их интерфейсы согласно спецификации OPC DA 2.0



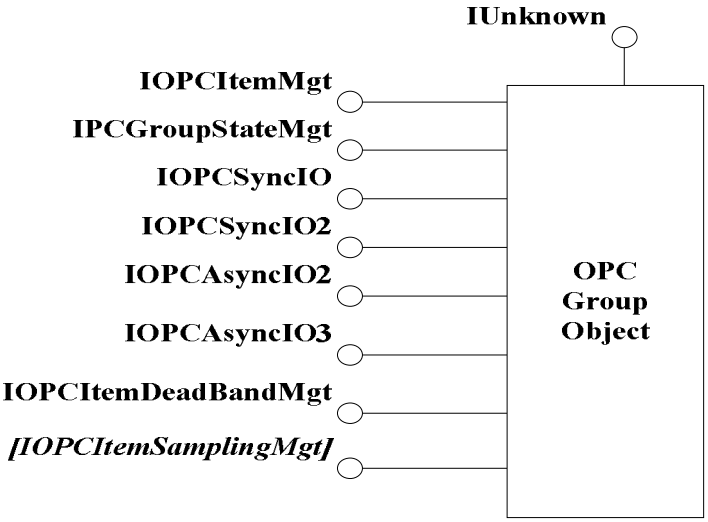



Рисунок 3.2. - Компоненты OPC сервера и OPC группы и их интерфейсы согласно спецификации OPC DA 3.0

При организации обмена с сервером используются оба объекта. Каждый объект предоставляет набор интерфейсов, которые можно разбить на две группы:

Обязательные интерфейсы - интерфейсы, которые сервер должен поддерживать каждый сервер, чтобы соответствовать спецификации и иметь право называться OPC-сервером.

Необязательные интерфейсы - набор интерфейсов, поддержка которых жестко не требуется спецификацией и предназначенные для предоставления клиентам дополнительных возможностей и функциональности.

На рисунках приведены указанные объекты и предоставляемые ими интерфейсы соответственно спецификации OPC DA 2.0 и OPC DA 3.0s.

Необязательные интерфейсы приведены курсивом и взяты в квадратные скобки

В приведенных примерах будут использоваться не все интерфейсы, которыми обладает тот или иной объект, но большую часть из них, все же придется задействовать.

Каждый интерфейс предоставляет набор функций, с помощью которых осуществляется манипуляция и доступ к данным сервера. Полное описание интерфейсов и содержащихся в них функций можно найти в спецификации OPC, где перечислены параметры и возвращаемые значения каждой функции.

## Адресное пространство OPC DA сервера

Адресное пространство OPC DA определяет каким образом хранятся данные, полученные от источника. Оно может быть

Плоским. Сервер состоит из набора элементов данных следующих друг за другом (организация в виде списка). Такая структура свойственна очень простым серверам и встречается крайне редко.

Иерархическим. Эта архитектура свойственна большинству OPC DA серверов. Обычно структура имеет вид приведенный на рисунке 4

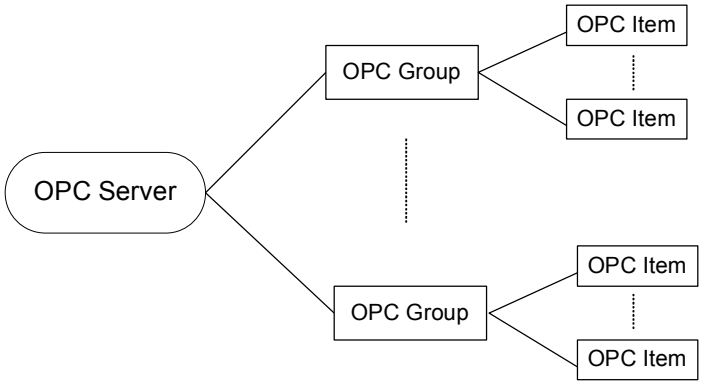



Рисунок 4. - Базовая структура иерархического адресного пространства.

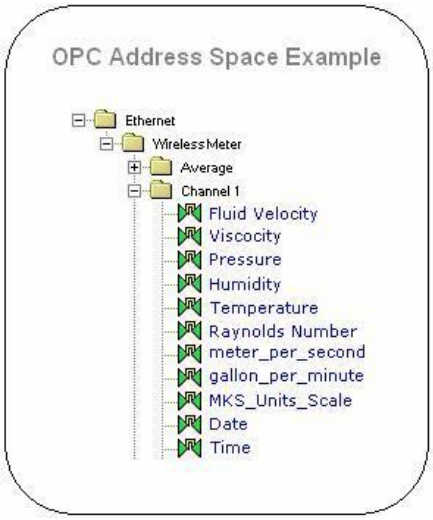



Рисунок 5. – Пример OPC сервера с иерархическим адресным пространством

Во главе адресации стоит сервер, в котором имеется некоторое количество групп, в каждой группе имеется несколько элементов данных.

Такая структуризация не является жестким стандартом и достаточно часто можно видеть вложенность групп или наличие элементов данных в узле сервера без использования групп.

Пример сервера с иерархическим адресным пространством приведен на рисунке 5 (источник http://www.codeproject.com/KB/COM/opctechnology.aspx)

## Способы чтения данных из OPC DA сервер

Чтение данных из OPC DA сервера возможно тремя способами

1. Синхронный обмен через интерфейс IOPCSyncIO
2. Асинхронный обмен через интерфейс IOPCAsyncIO2
3. Асинхронный обмен через подписку на изменения активных элементов активных группы через обратные вызовы интерфейса IOPCDataCallback.

## Синхронное чтение

Этот способ является наиболее простым с точки зрения программной реализации. В этом случае клиент запрашивает у сервера данные через интерфейс IOPCSyncIO и приостанавливает свое выполнение до окончания запроса, т.е. чтение происходит в основном потоке, там где и был послан запрос.

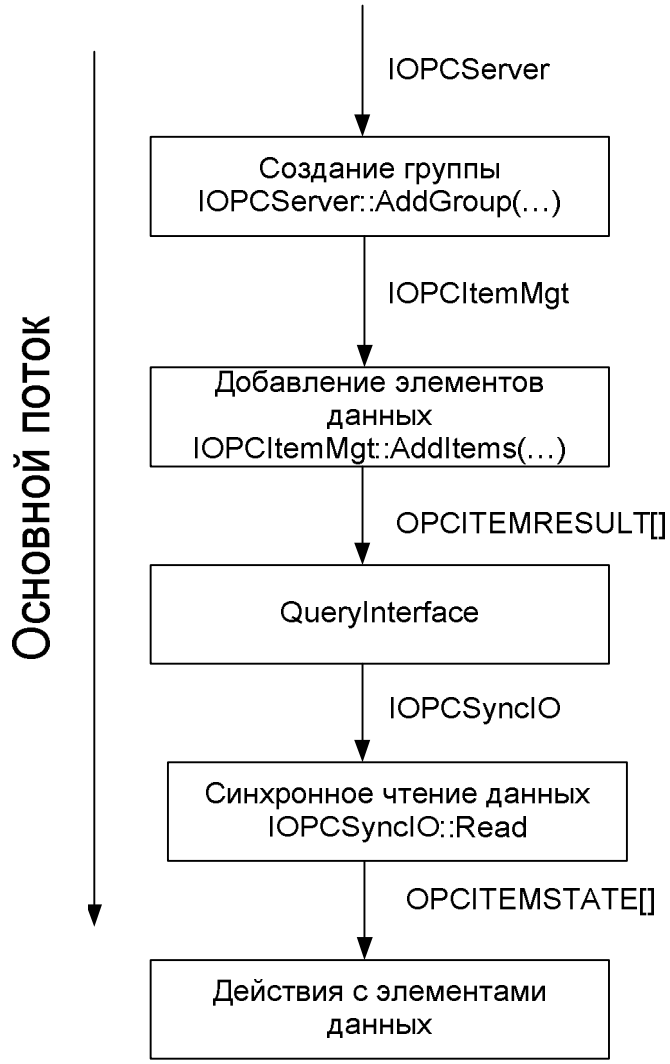



Рисунок 6.1 – Синхронное чтение данных из OPC сервера

Синхронное чтение состоит из шагов приведенных на рисунке

Вначале происходит добавление группы в сервер и установка ее настроек, с помощью вызова метода IOPCServer::AddGroup(). В результате возвращается интерфейс IOPCItemMgt.

Следующим шагом в группу добавляются необходимые элементы данных и устанавливаются их настройки, с помощью вызова метода IOPCItemMgt::AddItems().

После того, как группа добавлена и заполнена элементами данными клиент запрашивает интерфейс IOPCSyncIO и выполняет функцию IOPCSyncIO, получая в ответ информацию о запрашиваемых элементов данных в массиве структур OPCITEMSTATE. Как уже упоминалось поток, вызывающий функцию Read блокируется в ожидании ответа от сервера.

### Асинхронное чтение

Асинхронное чтение отличается тем, что основной поток, в котором происходит вызов запросов на чтение, не блокируется на ожидание ответа, а продолжает свое выполнение: ответы сервера выполняются в отдельном рабочем потоке.

Т.е. имеет место двунаправленное взаимодействие между COM клиентом и сервером, для организации которого используется технология точек подключения, которая коротко описана ниже.

### Точки подключения

В однонаправленном COM взаимодействии клиент является инициатором всех транзакций, а сервер лишь отвечая на запросы QueryInterface клиента, возвращает требуемые интерфейсы и реализует их функциональность.

Однако существуют случаи, и асинхронное взаимодействие с OPC сервером яркий пример, когда инициатором обмена должен стать сервер, а клиент должен среагировать на его запрос.

Такое взаимодействие называется двунаправленным, суть которого заключается в следующем. Пусть имеется COM сервер, который поддерживает некий интерфейс ISomeSink, реализующий методы, которые можно поставить в соответствие каким-либо событиям на сервере. Т.е. по возникновению события происходит вызов соответствующего метода. И имеется клиент, который хочет получать оповещения об этих событиях на своей стороне.

Решение этой задачи заключалось бы в том, что клиент на своей стороне реализовал бы интерфейс ISomeSink и заставил сервер запросить его у себя. А сервер, имея у себя клиентский интерфейс, вызывал бы соответствующие методы и тем самым информировал клиента о событиях. Осталось только заставить сервер выполнять функции запроса интерфейса.

Для этих целей существует технология точек подключения (соединения) – connection point (см. рисунок).

Она заключается в том, что клиент получает (каким образом будет рассмотрено ниже) у сервера интерфейс IConnectionPoint, который представлен как

```
interface IConnectionPoint : IUnknown {
HRESULT GetConnectionInterface ([out] IID* pIID);
HRESULT GetConnectionPointContainer (
```

```
[out] IConnectionPointContainer** ppCPC);
HRESULT Advise ([in] IUnknown* pUnkSink,
[out] DWORD* pdwCookie);
HRESULT Unadvise ([in] DWORD dwCookie);
HRESULT EnumConnections ([out] IEnumConnections** ppEnum);}
```

Этот интерфейс содержит метод Advise (подписка), который занимается тем, что говорит серверу запросить переданный в него в качестве параметра интерфейс у клиента и активировать его методы, когда необходимо.
На рисунке вызов Advise представлен виде замыкания ключа, который направляет вызовы клиенту через интерфейс ISomeSink.

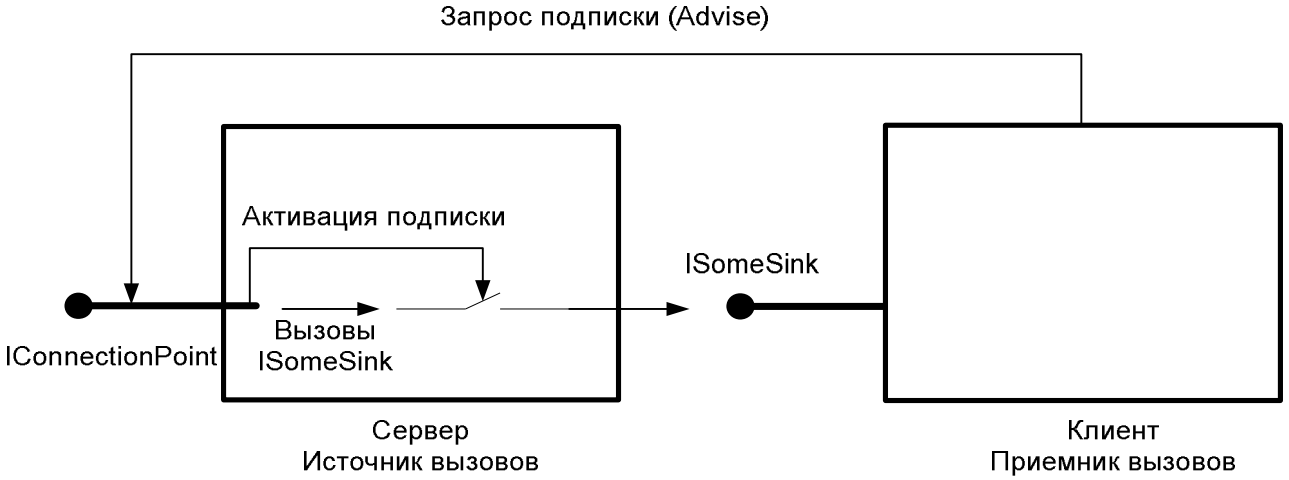



Рисунок 6.2. – Технология точек соединения COM сервера

Каждой точке подключения соответствует один интерфейс, т.е. сервер с одной точкой подключения поддерживает отправку лишь по одному интерфейсу. Такой факт, естественно является неудовлетворительным, т.к. в один интерфейс уложить все возможные методы противоречило бы самой концепции COM, которая создавалась для максимальной гибкости и возможности масштабирования программного обеспечения

Чтоб иметь возможность взаимодействовать по нескольким интерфейсам сервер должен обладать несколькими точками подключения.
Все точки подключения сгруппированы в контейнер, который представляется интерфейсом IConnectionpointContainer.

```
interface IConnectionPointContainer : IUnknown {
HRESULT EnumConnectionPoints (
[out] IEnumConnectionPoints ** ppEnum);
HRESULT FindConnectionPoint ([in] REFIID riid,
[out] IConnectionPoint** ppCP);
}
```

Вследствие такой группировки IConnectionPoint не является частью идентификации объекта-сервера, т.е. не может быть запрошен вызовом QueryInterface. Запрос точек подключения происходит через IConnectionPointContainer.

Т.е., чтобы получить IConnectionPoint, необходимо запросить у сервера IConnectionpointContainer, затем у него вызовом метода FindConnectionPoint запросить точку подключения.

В качестве первого параметра передается идентификатор интерфейса, через который поддерживается взаимодействие данной точкой подключения. Если сервер поддерживает связь по данному интерфейсу, то он вернет соответствующую точку подключения.

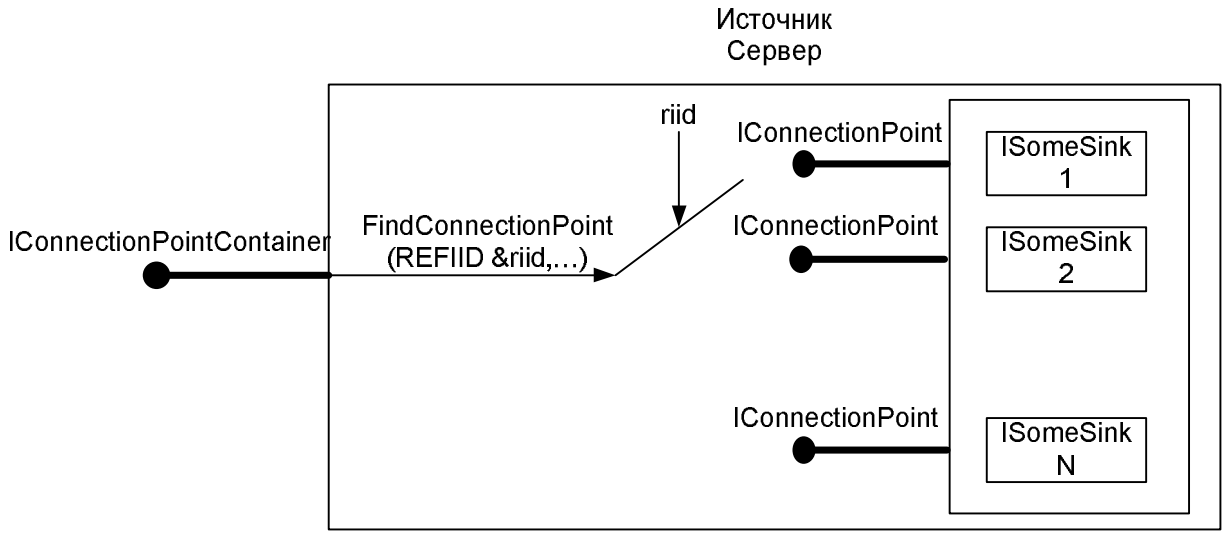



Рисунок 6.3 – Запрос точки подключения через IConnectionPointContainer

## OPC и точки подключения

Каждый из объектов OPC – сервер и группа поддерживают интерфейс IConnectionPointContainer и несколько точек подключения на различные события. Необходимо помнить, что в зависимости от того, кто вернул контейнер, сервера или группа, он будет содержать различные точки подключения.

Т.е. запрос точки подключения к интерфейсу IID_IOPCDataCallback, для оповещения о событиях в данных, через контейнер полученный у сервера будет неудачным, т.к. сервер не предоставляет такую точку соединения. Аналогично нет смысла ожидать от группы сообщений о закрытии сервера и т.п. событий свойственных компоненту сервера.

### Асинхронное чтение данных с помощью функции IOPCAsyncRead

Вначале выполняются операции аналогичные синхронному чтению (см. рисунок 6.4).

Добавление группы в сервер и установка ее настроек, с помощью вызова метода IOPCServer::AddGroup(). В результате возвращается интерфейс IOPCItemMgt.

Следующим шагом в группу добавляются необходимые элементы данных и устанавливаются их настройки, с помощью вызова метода IOPCItemMgt::AddItems().

С этого момента сходство заканчивается. Через интерфейс IOPCItemMgt необходимо получить интерфейс IOPCConnectionPointContainer, содержащий точки подключения компонента группы.

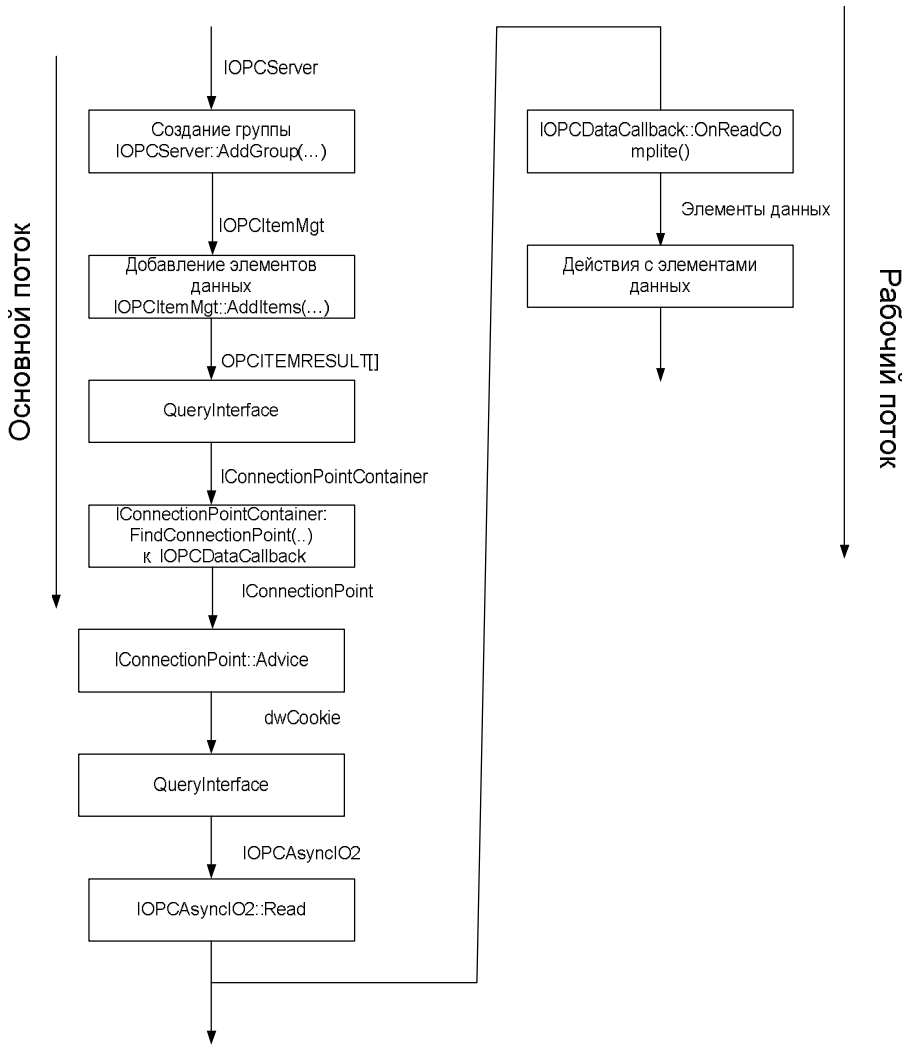



Рисунок 6.4 – Операция асинхронного чтения с помощью IOPCAsyncIO2

У него запросить точку подключения к интерфейсу IOPCDataCallback. После получения необходимой точки подключения IConnectionPoint необходимо вызвать функцию IConnectionPoint::Advise, куда в качестве параметра передать объект класса, поддерживающего интерфейс IOPCDataCallback.

Этот класс необходимо будет вручную создать в клиенте, унаследовав его от IOPCDataCallback и перегрузить методы интерфейса IOPCDatacallback и

IUnknown[1]

После того, как подписка была выполнена, вернется описатель dwCookie, в котором будет храниться идентификатор подписки, по которому впоследствии можно будет отписаться, вызвав метод Unadvise().

### Чтение данных по их изменению

В обоих предыдущих случаях инициатором чтения данных выступал клиент, т.е. он явно посылал запросы серверу на чтение данных. Однако чаще всего лучше читать данные только в том случае, если в них произошло изменение, таким образом снижать количество транзакций и тем самым повышать производительность. Особенно это касается дискретных величин, которые могут принимать значения 1/0 и, вообще говоря, не интересны пока не сменят своего состояния.

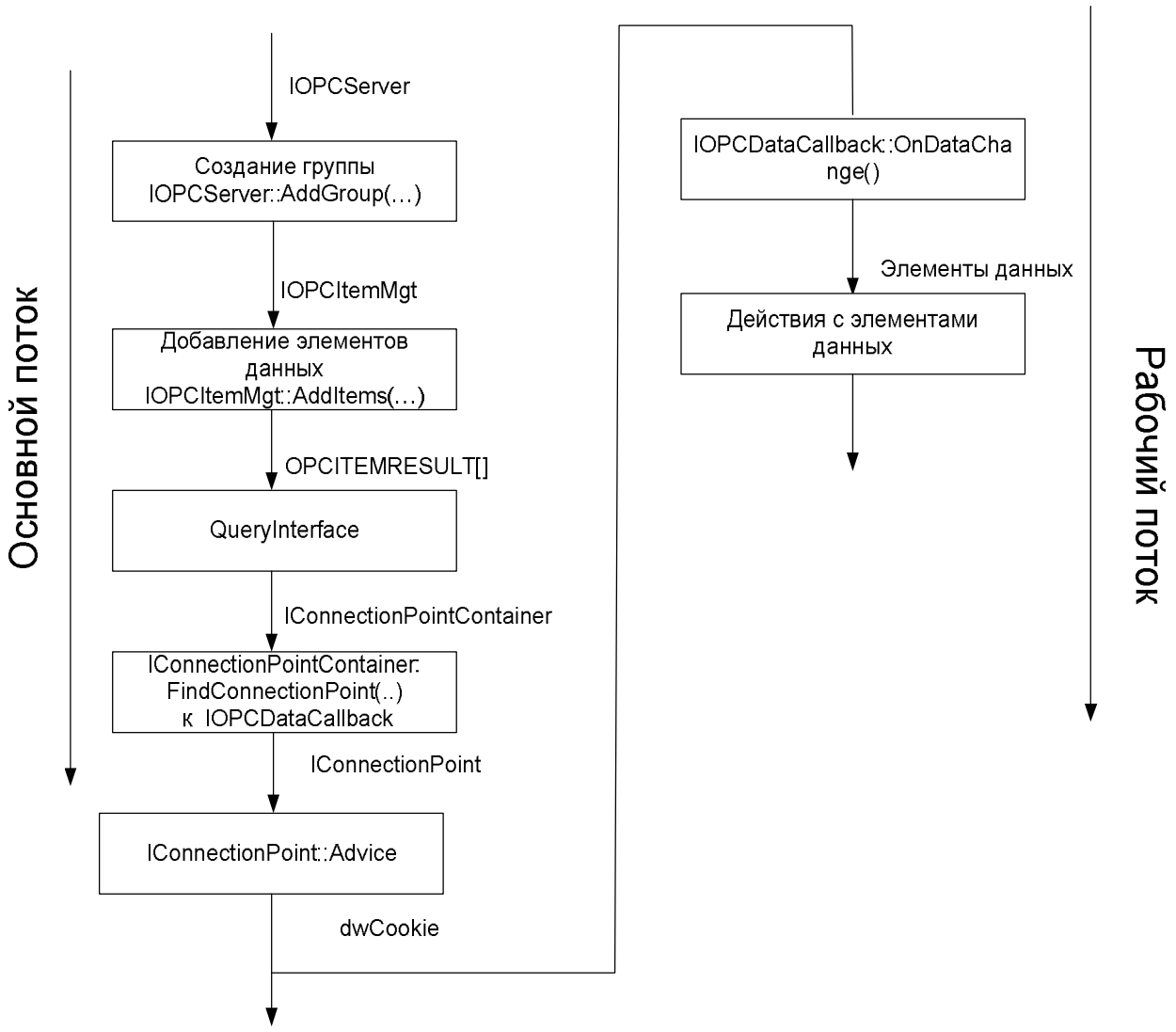



Рисунок 6.5 - Операция асинхронного обмена по подписке на изменение данных

[1] В реализации .NET перегружать методы QueryIntreface, AddRef и Release, относящиеся к IUnknown, нет необходимости, т.е. необходима перегрузка лишь методов IOPCDataCallback

Для таких целей спецификацией OPC предусмотрена операция чтения элементов данных по их изменению, представленная методом IOPCDataCallback::OnDataChange(). Она становится активной сразу после подписки на сообщения сервера и посылает значения всех активных элементов активных групп. Управлять вызовом OnDataChange() можно с помощью метода IOPCAsyncIO::SetEnabled(), в которую передается булевская переменная значение TRUE, которой соответствует активации, а FALSE деактивации данной функции.

Согласно спецификации OPC DA эта функция всегда активирована, т.е. при подписке нет смысла ее активировать и в предыдущем примере он наравне с методом OnReadComplete будут вызываться методы OnDataChange, пока не произойдет отписка или же не будет вызван метод IOPCAsyncIO::SetEnabled() с параметром FALSE.

Необходимо помнить, что посылаются лишь активные элементы данных в активных группах, т.е. если группа была создана с параметром bActive = false никаких вызовов происходить не будет. То ж касается и элементов данных.

Действия по этому способу аналогичны предыдущему примеру, за исключением того, что нет необходимости получать интерфейс IOPCAsyncIO, выполнять метод Read. Кроме того, происходит вызов метода OnDataChange.

## ПРОГРАММНАЯ РЕАЛИЗАЦИЯ OPC КЛИЕНТА

**Вариант C++/ MFC**

*Шаг 0. Подготовка диалога*

Первое, что сделаем - создадим диалоговый проект (MFC Dialog based) и назовем его OPCClient.

Разместим два элемента управления CListView и CTreeView. В первом мы будем отображать список доступным серверов, а в дереве будем показывать его структуру адресного пространства.

Т.к. нам часто придется обращаться к созданным элементам управления, то необходимо создать соответствующие им переменные.

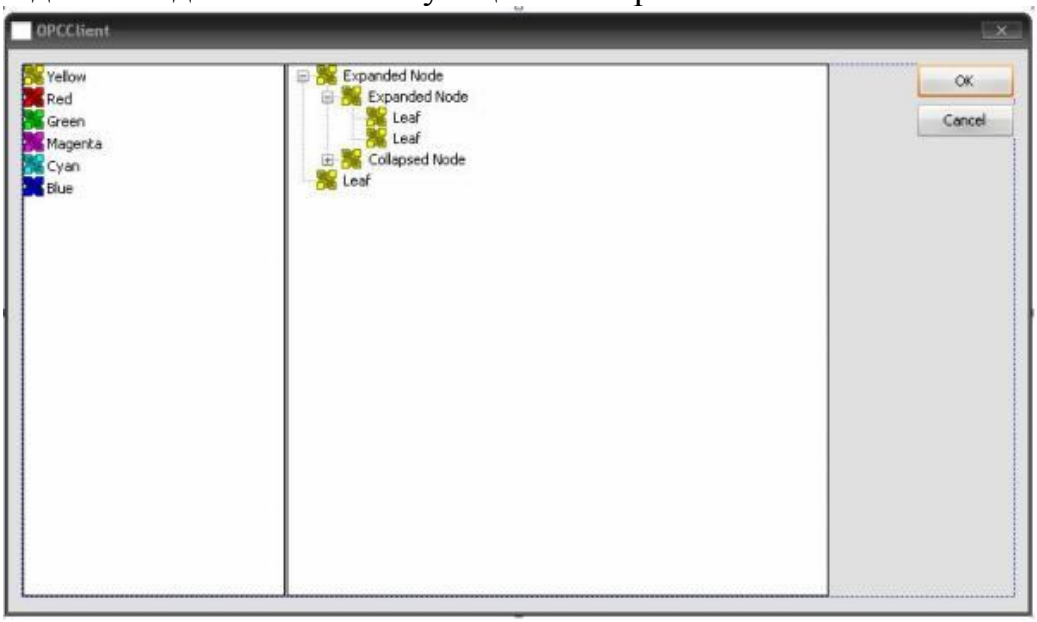



Рисунок 7. – Диалог с необходимыми элементами управления

```
private:
CListCtrl m_listOPCServers;
CTreeCtrl m_treeOPCServerBrowse;
```

В список мы будем отображать доступные на данном узле сервера, а в дерево будем развертывать адресное пространство выбранного сервера.

*Шаг 1. Просмотр установленных на локальной машине OPC-серверов*

Для решения задачи просмотра списка установленных OPC-серверов сообществом OPC Foundation была выпущена специализированная утилита OPCEnum.

Эта утилита находится в свободном пользовании и, потратив немного времени на поиск, ее можно без проблем найти в Интернете. Кроме того,

большинство SCADA систем, являясь OPC-клиентами по умолчанию, при установке инсталлируют указанную утилиту.
OPCCEnum является объектом COM и содержит в себе библиотеку типов (type library), описывающую инкапсулированные в ней интерфейсы (см. рис 8).

Любую информацию об OPCEnum можно получить просмотрев соответствующие ветки системного реестра или же воспользовавшись утилитой OleView.

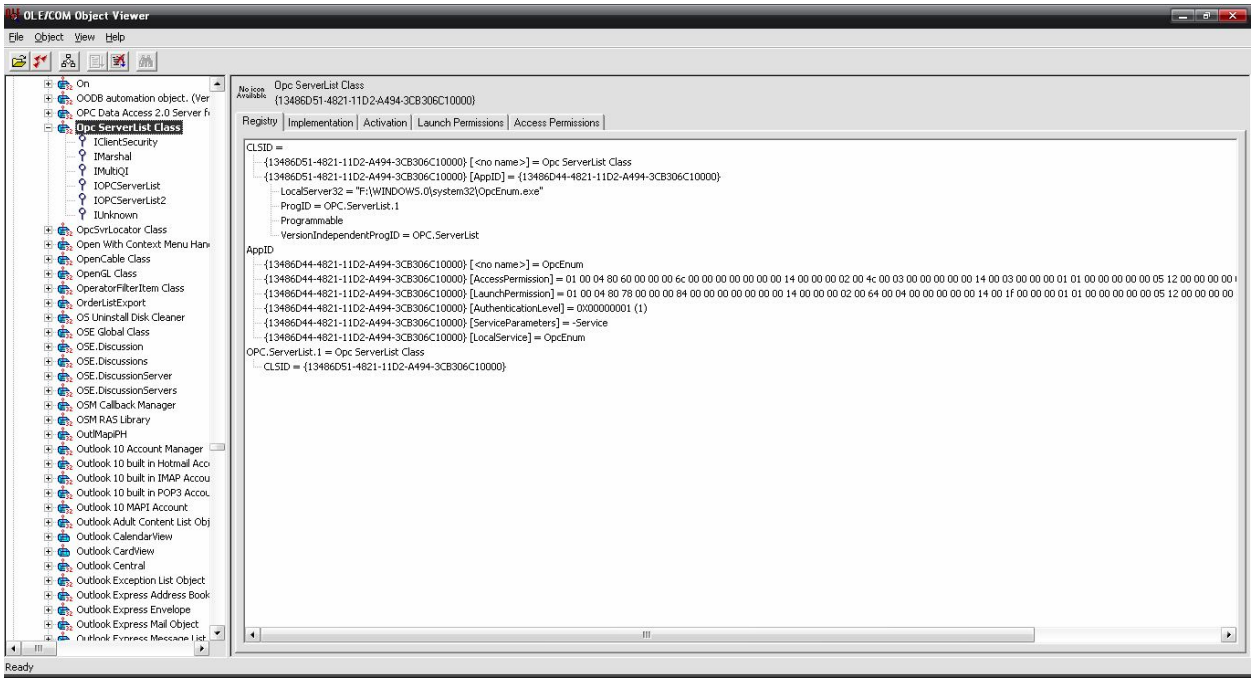


Рисунок 8. – Объект списка серверов при использовании утилиты OleView

Для удобства COM-компоненты могут быть сгруппированы в группу с индивидуальных идентификатором, именно так обстоит дело с OPC. Зарегистрированные группы и их идентификаторы могут быть просмотрены посредством утилиты OLEView(см. Рис 9)

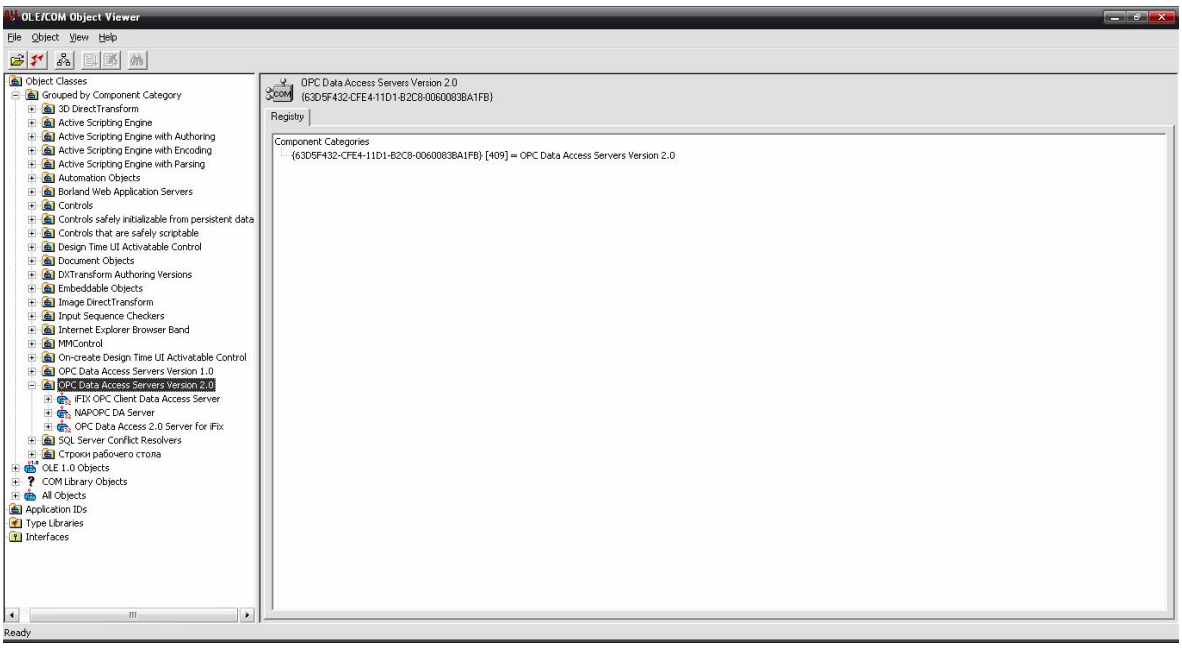



Рисунок 9. – Группа серверов OPC DA 2.0

Первое что необходимо – это импортировать объявления классов из библиотеки OpcEnum. Для этого необходимо найти директорию, где расположена утилита и в свой проект вставить директиву препроцессора import с указанием абсолютного пути к файлу OPCEnum.exe. Т.е. необходимо добавить экспорт типов в файл stdafx.h

```
#import "c:\\Windows\system32\opcenum.exe" \
rename_namespace OPCENUM
```

Путь к OPCEnum может отличаться для различных рабочих станций. Теперь добавим в класс COPCClientDlg объявление функции

```
private:
    int ShowRegisteredServers();
```

Функция будет добавлять в список установленные OPC сервера и возвращать их общее количество.

```
int COPCClient::ShowRegisteredServers()
{
CLSID clsid; // индентификатор OPCEnum
CLSID clsidcat; //идентифификатор категории OPC DA серверов
HRESULT hRes;
// Идентификатор категории OPC DA 2.0
hRes=CLSIDFromString(L"{63D5F432-CFE4-11D1-B2C8-
0060083BA1FB}",&clsidcat);
// Идентификатор компонента просмотра списка серверов
hRes=CLSIDFromProgID(L"OPC.ServerList",&clsid);
//Идентификатор интерфейса IOPCServerList
```

```
IID IID_IOPCServerList=__uuidof(IOPCServerList);
IOPCServerList *pServerList;
// запрос интерфейса у компонента должен вернуть S_OK
hRes=CoCreateInstance(clsid,NULL,CLSCTX_LOCAL_SERVER,
IID_IOPCServerList,(void**)&pServerList);
//перечислитель, в котором будут храниться GUID серверов
IOPCEnumGUID * pIOPCEnumGuid;
//запрос серверов спецификации OPC DA 2.0
pServerList->EnumClassesOfCategories(1, &clsidcat,0,
NULL,&pIOPCEnumGuid);
OLECHAR *pszProgID; // буфер для записи ProgID серверов
OLECHAR *pszUserType; // буфер для записи описания серверов
LVITEM lvItem; // подготовка элемента списка для вставки

ZeroMemory(&lvItem,sizeof(lvItem));
lvItem.cchTextMax=100;
lvItem.mask=LVIF_TEXT;

GUID guid; // Сюда будет записывать идентификатор текущего сервера
int nServerCnt=0; // общее количество доступных серверов
int iRetSvr; // количество серверов, предоставленных запросом
// получение первого доступного идентификатора сервера
pIOPCEnumGuid->Next(1,&guid,&iRetSvr);
     while (iRetSvr!=0)
 {
nServerCnt++;
pServerList->GetClassDetails(&guid,&pszProgID,&pszUserType);
lvItem.pszText=pszProgID;
int iItem=m_listOPCServers.InsertItem(&lvItem);
GUID *pGuid = new GUID;
//создаем область памяти, чтобы хранить идентификатор в привязке к
строке списка
memcpy(pGuid,&guid,sizeof(guid));
//связываем элемент списка и указатель на идентификатор
m_listOPCServers.SetItemData(iItem,(DWORD_PTR)pGuid);
pIOPCEnumGuid->Next(1,&guid,&iRetSvr); // получаем следующий сервер
}
 return nServerCnt;
}
```

Теперь необходимо вставить вызов функции в методе InitDialog нашего диалога и не забыть инициализировать COM. Функции, отвечающие за это, выделены курсивом.

```
BOOL COPCClientDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);
```

```
    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
         pSysMenu->AppendMenu(MF_SEPARATOR);
         pSysMenu->AppendMenu(MF_STRING,IDM_ABOUTBOX,
strAboutMenu);
        }
    }
    // Set the icon for this dialog.  The framework does this
automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);            // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon

    CoInitialize(NULL);

if (0==ShowRegisteredServers())
    MessageBox(L"Нет установленных серверов",L"Список
Серверов",MB_OK);

return TRUE;
}
```

Результат работы программы приведен на рисунке 10


Рисунок 10. - Результат получения списка зарегистрированных серверов

*Шаг 2. Просмотр содержимого OPC сервера*

Реализуем теперь обработчик, который при выделении соответствующего элемента списка будет показывать адресное пространство соответствующего сервера. Важно помнить, что мы имеем дело с обменом данными между приложениями реального времени, поэтому сервер, к которому происходит подключение, должен быть запущен.

Вначале нам необходимо подключить «обертку» функций для работы с OPC, для этого необходимо в файл stdafx.h вставить следующую директиву импорта

```
#import "c:\\Windows\system32\opcproxy.dll"\
rename_namespace("OPCDA")
using namespace OPCDA;
```

Спецификация OPC DA 2.0 и выше регламентирует необязательный интерфейс IOPCBrowseServerAddressSpace.

Не смотря на то, что спецификация не требует его обязательной реализации, у большинства серверов он реализован, что дает возможность клиенту просмотреть его внутреннее строение. Именно этот интерфейс мы и будем использовать2.

Далее будем предполагать, что сервер имеет иерархическое адресное пространство (в большей части серверов так и есть).

В сервере с иерархическим адресным пространством существует два типа узлов:

Ветви (OPC_BRANCH) – элементы у которых есть дочерние элементы

Листья (OPC_LEAF) – элементы у которых нет дочерних элементов

Так ветвями является элемент самого сервера (Server) и элементы групп (Group) в сервере, а листьями являются теги (Item).

Таким образом, необходимо разработать алгоритм, который бы рекурсивно проходил по элементам сервера и разворачивал их в дерево. Такой алгоритм у нас будет реализовывать функция

```
private:
void DisplayChildren(HTREEITEM hParent,
IOPCBrowseServerAddressSpace* pParent);
```

Функция в качестве аргументов принимает ссылку на родительский элемент в дереве и интерфейс IOPCBrowseServerAddressSpace спозиционированный на родительский элемент в сервере. Тело DisplayChildren приведено ниже

[2] В разделе замечания по поводу OPC DA 3.0 реализован просмотр адресного пространства с помощью IOPCBrowse

```
void COPCClientDlg::DisplayChildren(HTREEITEM hParent,
IOPCBrowseServerAddressSpace* pParent)
{
IEnumString *pEnum;
HTREEITEM hItem;
wchar_t * strName,*strTst;
unsigned long cnt;
TVINSERTSTRUCT tvInsert;
ZeroMemory(&tvInsert,sizeof(tvInsert));
pParent->BrowseOPCItemIDs(OPC_LEAF,L"",VT_EMPTY,0,&pEnum);
tvInsert.item.cchTextMax=100;
tvInsert.item.mask=TVIF_TEXT;
tvInsert.hParent=hParent;
pEnum->Next(1,&strName,&cnt);
LPWSTR lpItemID;
    while (cnt!=0)
    {
        tvInsert.item.pszText=strName;
        hItem=m_treeOPCServerBrowse.InsertItem(&tvInsert);
        pParent->GetItemID(strName,&lpItemID);//получает полный
идентификатор тега

m_treeOPCServerBrowse.SetItemData(hItem,(DWORD_PTR)lpItemID);
        pEnum->Next(1,&strName,&cnt);
    }
pParent->BrowseOPCItemIDs(OPC_BRANCH,L"",VT_EMPTY,0,&pEnum);
tvInsert.hParent=hParent;
pEnum->Next(1,&strName,&cnt);
HRESULT hRes;
tvInsert.item.iImage=0;
tvInsert.item.iSelectedImage=0;
while (cnt!=0)
   {
      tvInsert.item.pszText=strName;
      hItem=this->m_treeOPCServerBrowse.InsertItem(&tvInsert);
      hRes=pParent->ChangeBrowsePosition(OPC_BROWSE_DOWN,strName);
        if (S_OK==hRes)
            DisplayChildren(hItem,pParent);
       pParent->ChangeBrowsePosition(OPC_BROWSE_UP,strName);
        pEnum->Next(1,&strName,&cnt);
    }}
```

Следующим шагом добавим функцию, которая будет выполнять подключение к серверу и непосредственно вызывать экспорт адресного пространства

Добавим переменную

```
private:
IOPCServer m_pOPCServer;
```

в которой будем хранить интерфейс IOPCServer и через которую впоследствии получать все остальные интерфейсы OPC.
А также функцию

```
private:
void OnServerChange();
```

Тело, которой будет пустым и заполнится позже, когда будут рассмотрены операции чтения данных

```
void COPCClientDlg::OnServerChange()
{
}
```

Теперь функция подключения к серверу может быть записана как

```
private:
 int BrowseServer(const GUID *pGuid);
```

Функция принимает в качестве аргумента идентификатор текущего сервера

```
int COPCClientDlg::ConnectAndBrowseServer(const GUID *pGuid)
{
     //Если уже был подключен к серверу
if (m_pOPCServer!= NULL)
     {
          OnServerChange();
          m_pOPCServer->Release();
     }

IID IID_IOPCSERVER=__uuidof(IOPCServer);
//пробуем подключиться как к локальному dll серверу
HRESULT hRes=CoCreateInstance(*pGuid,NULL,CLSCTX_INPROC_SERVER,
IID_IOPCSERVER,(void**)&m_pOPCServer);

if (hRes!=S_OK)
{
     //пробуем подключиться как к локальному exe серверу
     hRes=CoCreateInstance(*pGuid,NULL,CLSCTX_LOCAL_SERVER,
IID_IOPCSERVER,(void**)&m_pOPCServer);
     if (hRes!=S_OK)
     {
     MessageBox(L"Не удалось подключиться к серверу",L"Ошибка
подключения",MB_OK);
    return -1;
     }
}
//Подключение установлено
IID IID_IOPCBrowseServerAddressSpace=
```

```
__uuidof(IOPCBrowseServerAddressSpace);
IOPCBrowseServerAddressSpace* pBrowse;

hRes= m_pOPCServer->QueryInterface(IID_IOPCBrowseServerAddressSpace,
(void**)&pBrowse);
if (hRes!=S_OK)
{
    MessageBox(L"Не удалось получить
IID_IOPCBrowseServerAddressSpace",
L"Ошибка просмотра",MB_OK);
    return -1;
}
// отображаем содержимое сервера, начиная с корневого узла
     DisplayChildren(TVI_ROOT,pBrowse);
      pBrowse->Release();
}
return 0;}
```

Теперь необходимо добавить обработчик сообщения TVN_ITEMHANGED элемента CListCtrl, который будет развертывать адресное пространство сервера

```
void COPCClientDlg::OnLvnItemchangedList1(NMHDR *pNMHDR,
LRESULT *pResult)
{
LPNMLISTVIEW pNMLV = reinterpret_cast<LPNMLISTVIEW>(pNMHDR);
BOOL bSelectedNow = ( pNMLV->uNewState  & LVIS_SELECTED);
BOOL bSelectedBefore = (pNMLV ->uOldState  & LVIS_SELECTED);
if ( bSelectedNow && !bSelectedBefore )
   {
     int nItem = pNMLV->iItem;
m_treeOPCServerBrowse.DeleteAllItems();
GUID * guid=(GUID*)m_listOPCServers.GetItemData(nItem);
BrowseServer(guid);
       }
    *pResult = 0;}
```

Ниже приведен результат работы клиента с серверами

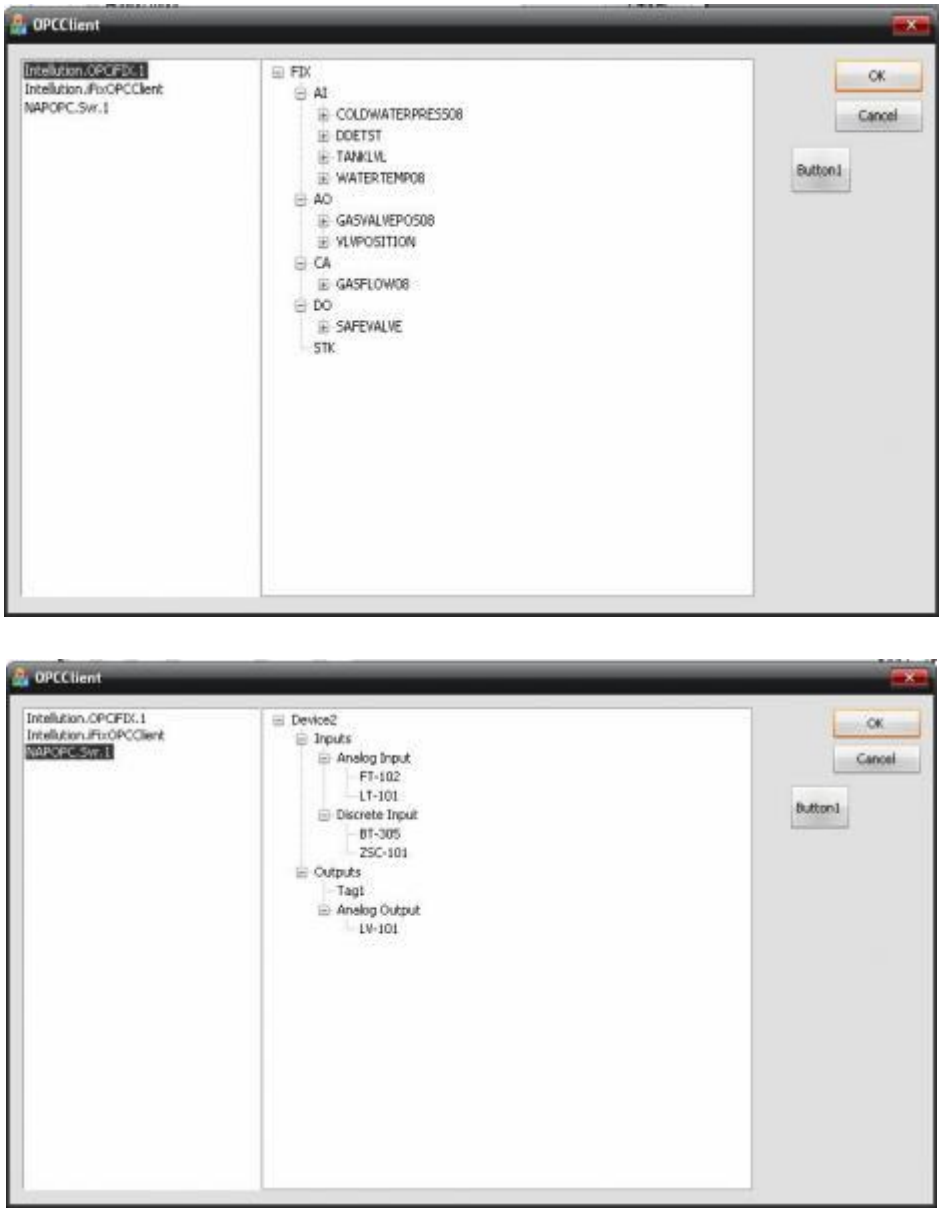



Рисунок 11. - Результат подключения и просмотра содержимого различных OPC DA серверов

*Шаг 3.1. Синхронное чтение данных из OPC сервера*

В качестве подготовки нашего приложения, добавим элемент CListCtrl и установим его свойство View = Report и создадим для него переменную m_valueView

```
CListCtrl m_valueView;
```

Кроме того, добавим кнопку, которой дадим заголовок «Значение». В итоге диалог должен принять вид, показанный на рисунке 12.

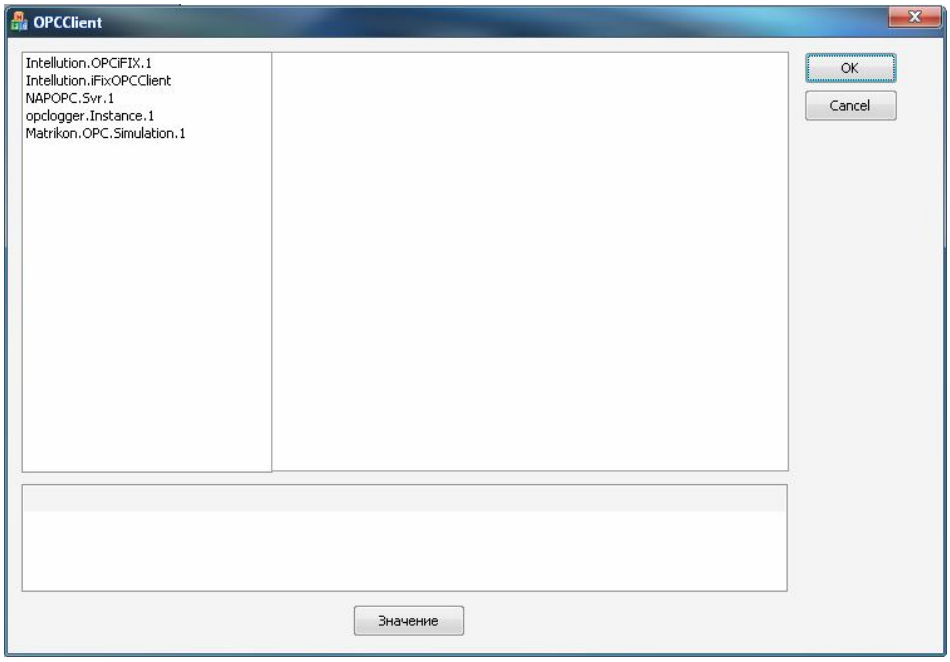


Рисунок 12. - Добавление CListCtrl для вывода свойств считанного элемента

Теперь напишем функцию, которая при старте диалоге будет добавлять в этот элемент управления столбцы с заголовками «Элемент», «Тип», «Значение», «Метка времени и «Качество».

```
void FormatValueView();
```

Реализация этой функции представлена ниже, а вызов ее необходимо вставить в InitDialog().

```
void COPCClientDlg::FormatValueView()
{
m_valueView.SetExtendedStyle(LVS_EX_GRIDLINES| LVS_EX_FLATSB |
LVS_EX_FULLROWSELECT);
LVCOLUMNW lvColumn;

lvColumn.cchTextMax=25;
lvColumn.cx=150;
lvColumn.mask=LVCF_TEXT|LVCF_WIDTH;
lvColumn.pszText=L"Элемент";

m_valueView.InsertColumn(0,&lvColumn);

lvColumn.cx=100;
lvColumn.pszText=L"Тип";

m_valueView.InsertColumn(1,&lvColumn);

lvColumn.cx=100;
lvColumn.pszText=L"Значение";
```

```
m_valueView.InsertColumn(2,&lvColumn);

lvColumn.cx=150;
lvColumn.pszText=L"Метка времени";

m_valueView.InsertColumn(3,&lvColumn);

lvColumn.cx=100;
lvColumn.pszText=L"Качество";

m_valueView.InsertColumn(4,&lvColumn);
}
```

В результате диалог при запуске должен принимать вид, изображенный на рисунке 13.

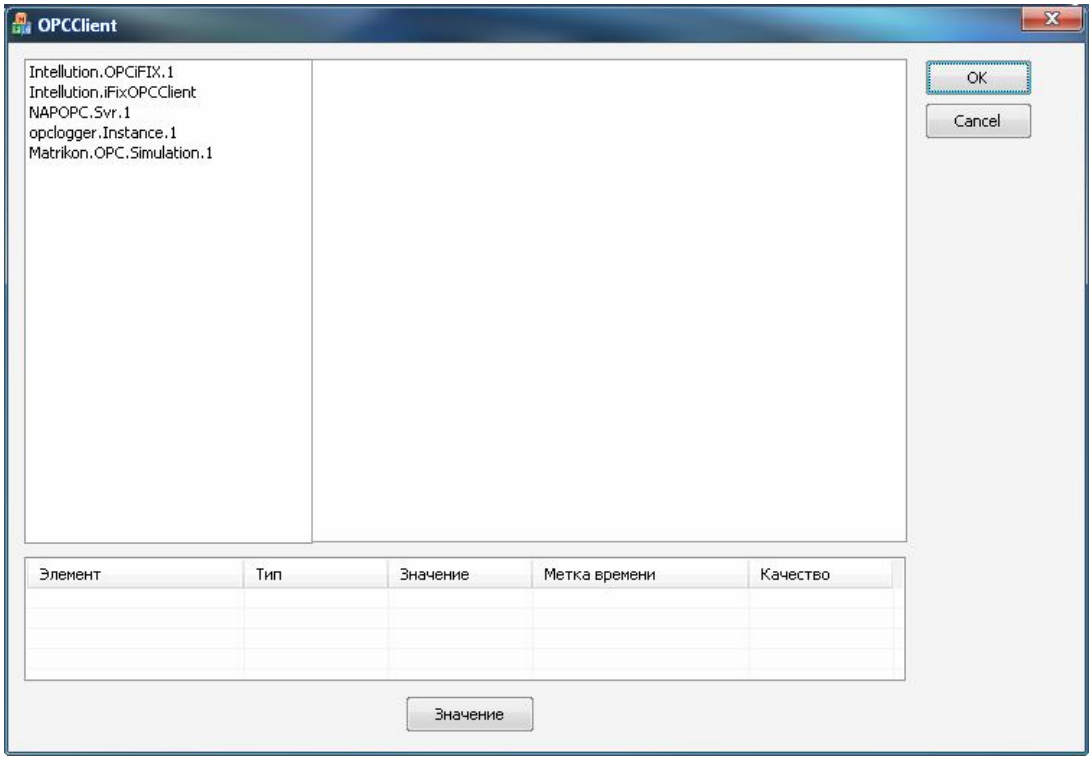



Рисунок 13. – Форматирование CListCtrl в стиле Report

Теперь нужно добавить переменную

```
private:
     DWORD m_hGroup; // описатель возданной группы
```

Следующим шагом добавим вспомогательный класс, методы которого будут переводить тип данных, метку времени и качество полученных данных в текстовый вид.

Для этого создадим два файла ToStringConverter.h и ToStringConverter.cpp, содержимое которых должно быть следующим

```
#pragma once
class CToStringConverter
{
public:
     static CString GetVTypeString(unsigned short usType);
     static CString GetQualityString(unsigned short usQuality);
     static CString GetFTString(_FILETIME ft);
};
```

```
// Преобразование типа в текст
CString CToStringConverter::GetVTypeString(unsigned short usType)
{
     switch (usType)
     {
     case VARENUM::VT_R4        : return CString("VT_R4");
     case VARENUM::VT_R8        : return CString("VT_R8");
     case VARENUM::VT_ARRAY     : return CString("VT_ARRAY");
     case VARENUM::VT_BOOL      : return CString("VT_BOOL");
     case VARENUM::VT_BSTR      : return CString("VT_BSTR");
     case VARENUM::VT_DECIMAL   : return CString("VT_DECIMAL");
     case VARENUM::VT_I1        : return CString("VT_I1");
     case VARENUM::VT_I2        : return CString("VT_I2");
     case VARENUM::VT_I4        : return CString("VT_I4");
     case VARENUM::VT_I8        : return CString("VT_I8");
     case VARENUM::VT_FILETIME : return CString("FILETIME");
     default                   : return CString("Unknown Type");
     }
}

CString CToStringConverter::GetQualityString(unsigned short
usQuality)
{
 switch (usQuality)
   {
     case 0x00: return L"Bad";
     case 0x04: return L"Config Error";
     case 0x08: return L"Not Connected";
     case 0x0C: return L"Device Failure";
     case 0x10: return L"Sensor Failure";
     case 0x14: return L"Last Known";
     case 0x18: return L"Comm Failure";
     case 0x1C: return L"Out of Service";
     case 0x20: return L"Initializing";
     case 0x40: return L"Uncertain";
     case 0x44: return L"Last Usable";
     case 0x50: return L"Sensor Calibration";
     case 0x54: return L"EGU Exceeded";
     case 0x58: return L"Sub Normal";
     case 0xC0: return L"Good";
     case 0xD8: return L"Local Override";
```

```
     default: return  L"Unknown";
     }
        }

CString CToStringConverter::GetFTString(_FILETIME ft)
{
 COleDateTime dt = COleDateTime(ft);
 return dt.Format(L"%Y-%m-%d %H:%M:%S");
}
```

Последним этапом подготовки будет функция, которая будет возвращать идентификатор текущего элемента данных, выбранного в дереве.

```
private:
LPWSTR GetCurrentItemID();
```

Реализация, которой имеет вид

```
LPWSTR COPCClientDlg::GetCurrentItemID()
{
     HTREEITEM hItem = m_treeOPCServerBrowse.GetSelectedItem();
     if (NULL == hItem)
          return NULL;
     LPWSTR szItemID =(LPWSTR)
m_treeOPCServerBrowse.GetItemData(hItem);
     return szItemID;
}
```

Подготовка завершена и можно приступать к реализации необходимых функций.

Добавим обработчик нажатия на кнопку Значение.

```
void COPCClientDlg::OnBnClickedButton2()
{
     HRESULT hRes;
/*
Этот участок на данном этапе должен быть закомментирован. Его
необходимо будет использовать тогда, когда добавим асинхронную
операцию чтения по подписке

     if (m_dwCookie!=0)
     {
          hRes = ((IConnectionPoint*)m_pDataCallback)->Unadvise(
                                        m_dwCookie);
          m_pDataCallback->Release();
          m_dwCookie = 0;
     }
*/

m_valueView.DeleteAllItems();
```

```
LPWSTR szItemID = GetCurrentItemID();
if (NULL == szItemID) return;

DWORD updateRate = 1000;
IOPCItemMgt *pItemMgt = NULL;
long bActive = 1;

if (m_hGroup!=0)
{
     hRes = m_pOPCServer->RemoveGroup(m_hGroup,1);
     m_hGroup = 0;
}
hRes = m_pOPCServer->AddGroup(OLESTR("MyGroup"), bActive,
updateRate,
1,NULL,NULL,0,
     &m_hGroup,&updateRate,(GUID*)&__uuidof(IOPCItemMgt),
                                        (IUnknown**)&pItemMgt);


if (FAILED(hRes))
{
     LPWSTR lpError;
     m_pOPCServer->GetErrorString(hRes,2,&lpError);
     MessageBox(lpError,L"Ошибка",MB_OK);
}

//Добавляем элементы в группу
DWORD          dwCount  = 1;
tagOPCITEMDEF*    pItems   =
(tagOPCITEMDEF*)CoTaskMemAlloc(dwCount*sizeof(tagOPCITEMDEF));
tagOPCITEMRESULT* pResults = NULL;
HRESULT*       pErrors  = NULL;

pItems[0].szItemID             = szItemID;
pItems[0].szAccessPath        = NULL;
pItems[0].bActive             = TRUE;
pItems[0].hClient             = 0;
pItems[0].vtRequestedDataType = VT_EMPTY;
pItems[0].dwBlobSize          = 0;
pItems[0].pBlob               = NULL;

hRes = pItemMgt->AddItems(1, pItems, &pResults, &pErrors);
if (FAILED(hRes))
{
     LPWSTR lpMsg = NULL;
     m_pOPCServer->GetErrorString(hRes,2,&lpMsg);
     MessageBox(lpMsg,L"Ошибка",MB_OK);
}
else
{
```

```
IOPCSyncIO * pSyncIO=NULL;
IID IID_IOPCSYNCIO=__uuidof(IOPCSyncIO);
hRes=pItemMgt->QueryInterface(IID_IOPCSYNCIO,(void**)&pSyncIO);
tagOPCITEMSTATE *pItemValue=NULL;

  //Считываем элементы
hRes=pSyncIO->Read(OPC_DS_CACHE,1,&pResults->hServer,&pItemValue,
&pErrors);

  //Отображаем данные
m_valueView.DeleteAllItems();

  LVITEM lvItem;
  ZeroMemory(&lvItem,sizeof(lvItem));

  lvItem.cchTextMax = 25;
  lvItem.mask = LVIF_TEXT;
  lvItem.pszText = szItemID;
  m_valueView.InsertItem(&lvItem);

  m_valueView.SetItem(0,1,LVIF_TEXT,
CToStringConverter::GetVTypeString(pResults-
>vtCanonicalDataType),0,0,0,0,0);

 m_valueView.SetItem(0,3,LVIF_TEXT,
CToStringConverter::GetFTString(pItemValue-
>ftTimeStamp),0,0,0,0,0);

m_valueView.SetItem(0,2,LVIF_TEXT,CString(pItemValue-
>vDataValue),0,0,0,
0,0);

m_valueView.SetItem(0,4,LVIF_TEXT,CToStringConverter::GetQualitySt
ring(pItemValue->wQuality),0,0,0,0,0);

 pSyncIO->Release();
CoTaskMemFree(pItemValue);

}
  pItemMgt->Release();

  CoTaskMemFree(pErrors);
  CoTaskMemFree(pResults);
  CoTaskMemFree(pItems);
}
```

Теперь каждый раз по нажатию на кнопку в элемент m_valueView будут выводиться идентификатор элемента, текущее значение, тип, метка времени и качество.

При этом обмен будет происходить синхронно, т.е. клиент посылая запрос серверу, приостанавливает свое выполнение и ожидает ответа на него.

*Шаг 3.2.Асинхронное чтение данных из OPC сервера*

Если Вы приступили к чтению этого раздела минуя предыдущий о синхронном обмене данных, то все же придется вернутся и выполнить подготовительные действия, т.к. они необходимы и для данного раздела.

Кроме того, необходимо добавить переменные для асинхронного обмена

```
private:
IConnectionPoint *m_pDataCallback; // Точка подключения к серверу
DWORD m_dwCookie; // идентификатор подписки на события сервера
CDataCallback *m_pSink; // Интерфейс, через который будут
проходить обратные вызовы
```

Объявление класса CDataCallback представлено ниже

```
class CDataCallback :public IOPCDataCallback
{
public:

    CDataCallback(CListCtrl * pValueView, CString szItemID)
    {
        m_pValueView = pValueView;
        m_szItemID = szItemID;
       m_ulRefs = 1;
    }

void SetItemID(CString szItemID)
{
        m_szItemID = szItemID;
}

//================================================================
==========
     // IUnknown

    // QueryInterface
     virtual  STDMETHODIMP QueryInterface(REFIID iid, LPVOID*
ppInterface);
     virtual STDMETHODIMP_(ULONG) AddRef();
    // Release
     virtual  STDMETHODIMP_(ULONG) Release();


//================================================================
==========
     // IOPCDataCallback
virtual STDMETHODIMP raw_OnDataChange (
```

```
        /*[in]*/ unsigned long dwTransid,
        /*[in]*/ unsigned long hGroup,
        /*[in]*/ HRESULT hrMasterquality,
        /*[in]*/ HRESULT hrMastererror,
        /*[in]*/ unsigned long dwCount,
        /*[in]*/ unsigned long * phClientItems,
        /*[in]*/ VARIANT * pvValues,
        /*[in]*/ unsigned short * pwQualities,
        /*[in]*/ struct _FILETIME * pftTimeStamps,
               /*[in]*/ HRESULT * pErrors );

      virtual HRESULT __stdcall raw_OnReadComplete (
        /*[in]*/ unsigned long dwTransid,
        /*[in]*/ unsigned long hGroup,
        /*[in]*/ HRESULT hrMasterquality,
        /*[in]*/ HRESULT hrMastererror,
        /*[in]*/ unsigned long dwCount,
        /*[in]*/ unsigned long * phClientItems,
        /*[in]*/ VARIANT * pvValues,
        /*[in]*/ unsigned short * pwQualities,
        /*[in]*/ struct _FILETIME * pftTimeStamps,
               /*[in]*/ HRESULT * pErrors );

      virtual  STDMETHODIMP raw_OnWriteComplete (
        /*[in]*/ unsigned long dwTransid,
        /*[in]*/ unsigned long hGroup,
        /*[in]*/ HRESULT hrMastererr,
        /*[in]*/ unsigned long dwCount,
        /*[in]*/ unsigned long * pClienthandles,
          /*[in]*/ HRESULT * pErrors );

      virtual  STDMETHODIMP raw_OnCancelComplete (
        /*[in]*/ unsigned long dwTransid,
          /*[in]*/ unsigned long hGroup );

 private:
      CListCtrl *m_pValueView;
       ULONG m_ulRefs;
      CString m_szItemID;
 };
```

А реализация имеет вид

```
#include"stdafx.h"
#include "DataCallback.h"
#include "ToStringConverter.h"

  STDMETHODIMP CDataCallback::QueryInterface(REFIID iid, LPVOID*
ppInterface)
    {
       if (NULL == ppInterface)
```

```
        {
           return E_INVALIDARG;
        }

        if (IID_IUnknown == iid)
        {
           *ppInterface = dynamic_cast<IUnknown*>(this);
           AddRef();
           return S_OK;
        }

        if (__uuidof(IOPCDataCallback) == iid)
        {
           *ppInterface = dynamic_cast<IOPCDataCallback*>(this);
           AddRef();
           return S_OK;
        }

        return E_NOINTERFACE;
     }

  //--------------------
       STDMETHODIMP_(ULONG) CDataCallback::AddRef()
     {
          return InterlockedIncrement((LONG*)&m_ulRefs);
     }
        //------------
        STDMETHODIMP_(ULONG) CDataCallback::Release()
     {
          ULONG ulRefs = InterlockedDecrement((LONG*)&m_ulRefs);

          if (ulRefs == 0)
          {
              delete this;
              return 0;
          }

          return ulRefs;
     }

STDMETHODIMP CDataCallback::raw_OnDataChange (
         /*[in]*/ unsigned long dwTransid,
         /*[in]*/ unsigned long hGroup,
         /*[in]*/ HRESULT hrMasterquality,
         /*[in]*/ HRESULT hrMastererror,
         /*[in]*/ unsigned long dwCount,
         /*[in]*/ unsigned long * phClientItems,
         /*[in]*/ VARIANT * pvValues,
         /*[in]*/ unsigned short * pwQualities,
         /*[in]*/ struct _FILETIME * pftTimeStamps,
                /*[in]*/ HRESULT * pErrors )
           {
```

```
                LVITEM lvItem;
                ZeroMemory(&lvItem,sizeof(lvItem));
                lvItem.cchTextMax = 255;
                lvItem.pszText = m_szItemID.GetBuffer();
                lvItem.mask = LVIF_TEXT;
                m_pValueView->DeleteAllItems();
                int i =m_pValueView->InsertItem(&lvItem);

                m_pValueView->SetItem(0,1,LVIF_TEXT,
CToStringConverter::GetVTypeString(pvValues[0].vt),0,0,
0,0,0);

m_pValueView->SetItem(0,3,LVIF_TEXT,
CToStringConverter::GetFTString(pftTimeStamps[0]),0,0,0,0,0);

m_pValueView->SetItem(0,2,LVIF_TEXT,
CString(pvValues[0]),0,0,0,0,0);

m_pValueView->SetItem(0,4,LVIF_TEXT,
CToStringConverter::GetQualityString(pwQualities[0]),0,0,0,0,0);
return S_OK;
          }

  STDMETHODIMP CDataCallback::raw_OnReadComplete (
        /*[in]*/ unsigned long dwTransid,
        /*[in]*/ unsigned long hGroup,
        /*[in]*/ HRESULT hrMasterquality,
        /*[in]*/ HRESULT hrMastererror,
        /*[in]*/ unsigned long dwCount,
        /*[in]*/ unsigned long * phClientItems,
        /*[in]*/ VARIANT * pvValues,
        /*[in]*/ unsigned short * pwQualities,
        /*[in]*/ struct _FILETIME * pftTimeStamps,
              /*[in]*/ HRESULT * pErrors ) {return S_OK;}

  STDMETHODIMP CDataCallback::raw_OnWriteComplete (
        /*[in]*/ unsigned long dwTransid,
        /*[in]*/ unsigned long hGroup,
        /*[in]*/ HRESULT hrMastererr,
        /*[in]*/ unsigned long dwCount,
        /*[in]*/ unsigned long * pClienthandles,
          /*[in]*/ HRESULT * pErrors ) {return S_OK;}

  STDMETHODIMP CDataCallback::raw_OnCancelComplete (
        /*[in]*/ unsigned long dwTransid,
          /*[in]*/ unsigned long hGroup ) {return S_OK;}
```

Теперь можно приступать к операции асинхронного обмена данными.
Для этого вынесем на форму кнопку и дадим ей название «Подписаться», как представлено на рисунке

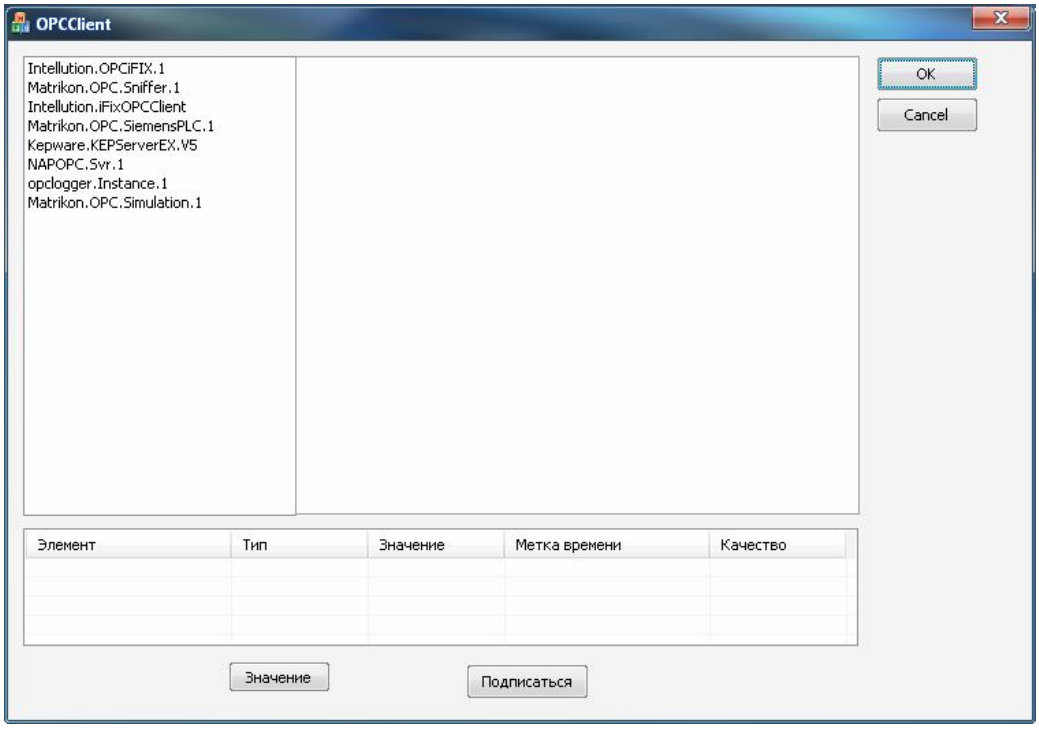


Рисунок 14. – Добавление кнопки «Подписаться»

При нажатии на эту кнопку будет происходить подписка на выбранный элемент данных и при каждом его изменении сервер будет посылать его клиенту в метод DataCalback::OnDataChange.

Обработчик нажатия на эту кнопку будет иметь вид

```
void COPCClientDlg::OnBnClickedAdvise()
{
HRESULT hRes;
LPWSTR szItemID = GetCurrentItemID();
if (NULL == szItemID) return;

long bActive = 1;
DWORD dwUpdateRate =0;
IOPCItemMgt *pItemMgt = NULL;
unsigned hClientGroup = 1;

if (m_dwCookie!=0)
     {
hRes = ((IConnectionPoint*)m_pDataCallback)->Unadvise(
m_dwCookie);
          m_pDataCallback->Release();
          m_dwCookie = 0;
     }

if (m_hGroup!=0)
{    hRes = m_pOPCServer->RemoveGroup(m_hGroup,1);
     m_hGroup = 0;
}
```

```
else
hRes = m_pOPCServer-
>AddGroup(OLESTR("MyGroup"),bActive,dwUpdateRate,hClientGroup,NULL
,NULL,0,

     &m_hGroup,&dwUpdateRate,(GUID*)&__uuidof(IOPCItemMgt),
                                        (IUnknown**)&pItemMgt);

IConnectionPointContainer *pCPC;
IID IID_CPC =__uuidof(IConnectionPointContainer);
hRes = pItemMgt->QueryInterface(IID_CPC,(void**)&pCPC);

hRes = pCPC->FindConnectionPoint(__uuidof(IOPCDataCallback),
(IConnectionPoint**)&m_pDataCallback);
pCPC->Release();


if (NULL == m_pSink)
     m_pSink = new CDataCallback(&m_valueView,szItemID);
else
     m_pSink->SetItemID(szItemID);


hRes = ((IConnectionPoint*)m_pDataCallback)->Advise((IUnknown
*)m_pSink,&m_dwCookie);

if (FAILED(hRes))
{
     LPWSTR lpError;
     m_pOPCServer->GetErrorString(hRes,2,&lpError);
     MessageBox(lpError,L"Ошибка",MB_OK);
}

//Добавляем элементы в группу
DWORD          dwCount  = 1;
tagOPCITEMDEF*    pItems   =
(tagOPCITEMDEF*)CoTaskMemAlloc(dwCount*sizeof(tagOPCITEMDEF));
tagOPCITEMRESULT* pResults = NULL;
HRESULT*       pErrors  = NULL;

pItems[0].szItemID             = szItemID;
pItems[0].szAccessPath        = NULL;
pItems[0].bActive             = TRUE;
pItems[0].hClient             = 0;
pItems[0].vtRequestedDataType = VT_EMPTY;
pItems[0].dwBlobSize          = 0;
pItems[0].pBlob               = NULL;

hRes = pItemMgt->AddItems(1, pItems, &pResults, &pErrors);
if (FAILED(hRes))
{
     LPWSTR lpMsg = NULL;
```

```
     m_pOPCServer->GetErrorString(hRes,2,&lpMsg);
     MessageBox(lpMsg,L"Ошибка", MB_OK);
}
else
{
CoTaskMemFree(pResults);
CoTaskMemFree(pErrors);
}
// Уменьшает кол-во ссылок на интерфейс
pItemMgt->Release();

//Освобождаем выделенную память
CoTaskMemFree(pItems);
}
```

Теперь с каждым изменение элемента данных его значения и параметры будут посылаться клиенту в метод OnDataChange.

Для корректной работы необходимо раскомментировать участок в функции синхронного чтения, который отменяет подписку на события сервера.

*Последние штрихи*

Напоследок необходимо сделать еще несколько действий.

Во-первых, наполнить функцию OnServerChange(), которая каждый раз при изменении выбранного сервера будет отменять подписку на данные и созданные в сервере группы

```
void COPCClientDlg::OnServerChange()
{
     if (m_dwCookie != 0)
       {
            m_pDataCallback->Unadvise(m_dwCookie);
            m_dwCookie = 0;
        }
        if (m_hGroup != 0)
        {
            m_pOPCServer->RemoveGroup(m_hGroup, 1);
            m_hGroup = 0;
        }
}
```

Во-вторых, инициализировать переменные в конструкторе класса диалога

```
COPCClientDlg::COPCClientDlg(CWnd* pParent /*=NULL*/)
     : CDialog(COPCClientDlg::IDD, pParent)
     , m_fItemValue(0)
{
     m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
     m_dwCookie = 0;
```

```
     m_pSink = NULL;
     m_hGroup = 0;
     m_pOPCServer = NULL;
}
```

В-третьих, обработать сообщение закрытие диалога, в котором необходимо освободить все ссылки и удалить объекты.

```
void COPCClientDlg::OnDestroy()
{
CDialog::OnDestroy();

OnServerChange();
m_pOPCServer->Release();
delete m_pSink;
}
```

## Замечания по поводу версии OPC DA 3.0

Здесь к разработанному приложению будет добавлена поддержка интерфейса IOPCBrowse.

Первое, что необходимо сделать – это изменить функцию ConnectAndBrowseServer, которая теперь будет запрашивать интерфейс IOPCBrowse, а если сервер его не поддерживает, т.е. соответствует лишь спецификации 2.0, то интерфейс IOPCBrowseServerAddressSpace

```
int COPCClientDlg::ConnectAndBrowseServer(const GUID *pGuid)
{
if (m_pOPCServer!= NULL)
     {
          OnServerChange();
          m_pOPCServer->Release();
     }

IID IID_IOPCSERVER=__uuidof(IOPCServer);
//пробуем подключиться как к локальному dll серверу
HRESULT hRes=CoCreateInstance(*pGuid,NULL,CLSCTX_INPROC_SERVER,
IID_IOPCSERVER,(void**)&m_pOPCServer);
if (hRes!=S_OK)
{
     //пробуем подключиться как к локальному exe серверу
     hRes=CoCreateInstance(*pGuid,NULL,CLSCTX_LOCAL_SERVER,
IID_IOPCSERVER,(void**)&m_pOPCServer);
     if (hRes!=S_OK)
     {
     MessageBox(L"Не удалось подключиться к серверу",L"Ошибка
подключения",MB_OK);
    return -1;
     }
}
//Подключение установлено
IID IID_IOPCBrowse = __uuidof(IOPCBrowse);
IOPCBrowse *pBrowse;
hRes = m_pOPCServer-
>QueryInterface(IID_IOPCBrowse,(void**)&pBrowse);

if (S_OK == hRes)
{
     DisplayChildren(TVI_ROOT,L"",pBrowse);
     pBrowse->Release();}
else
{
IID IID_IOPCBrowseServerAddressSpace=
__uuidof(IOPCBrowseServerAddressSpace);
IOPCBrowseServerAddressSpace* pBrowse;
hRes= m_pOPCServer-
>QueryInterface(IID_IOPCBrowseServerAddressSpace,
```

```
(void**)&pBrowse);
if (hRes!=S_OK)
{
    MessageBox(L"Не удалось получить
IID_IOPCBrowseServerAddressSpace",
L"Ошибка просмотра",MB_OK);
    return -1;
}
     DisplayChildren(TVI_ROOT,pBrowse);
pBrowse->Release();
}
return 0;
}
```

Далее необходимо добавить перегруженный вариант функции DisplayChildren

```
void COPCClientDlg::DisplayChildren(HTREEITEM hParent,
LPWSTR szItemName, IOPCBrowse* pBrowse)
{
//Это должен быть массив ID свойств, но т.к. мы запрашиваем все, то
//ограничимся переменной dwPropID, которая будет игнорироваться
//сервером
DWORD dwPropID = 0;

long bMoreElements=0;

LPWSTR szContinuationPoint = NULL;
LPWSTR szVendorFilter = L"";
LPWSTR szElementNameFilter = L"";

DWORD dwCount;

tagOPCBROWSEELEMENT *OPCItemElements = NULL;
tagOPCBROWSEELEMENT *OPCBranchElements = NULL;

HRESULT hRes = pBrowse->Browse(szItemName, &szContinuationPoint, 0,
                               OPC_BROWSE_FILTER_ITEMS, L"", L"",
                               true,true ,0,&dwPropID,&bMoreElements,
                               &dwCount,&OPCItemElements);

HTREEITEM hItem;
TVINSERTSTRUCT tvInsert;
ZeroMemory(&tvInsert, sizeof(tvInsert));
tvInsert.hParent = hParent;
tvInsert.item.cchTextMax = 255;
tvInsert.item.mask = TVIF_TEXT;

for (int i = 0; i < dwCount; i++)
{
     tvInsert.item.pszText = OPCItemElements[i].szName;
```

```
     hItem = m_treeOPCServerBrowse.InsertItem(&tvInsert);
     m_treeOPCServerBrowse.SetItemData(hItem,
(DWORD_PTR)OPCItemElements[i].szItemID);
}

pBrowse->Browse(szItemName, &szContinuationPoint, 0,
OPC_BROWSE_FILTER_BRANCHES, L"", L"",
true,true,0,&dwPropID, &bMoreElements,
&dwCount,&OPCBranchElements );

for (int i = 0; i < dwCount; i++)
{
 tvInsert.item.pszText = OPCBranchElements [i].szName;
 hItem = m_treeOPCServerBrowse.InsertItem(&tvInsert);
 DisplayChildren(hItem, OPCBranchElements [i].szName, pBrowse); }}
```

Результат работы приведен на рисунке 15

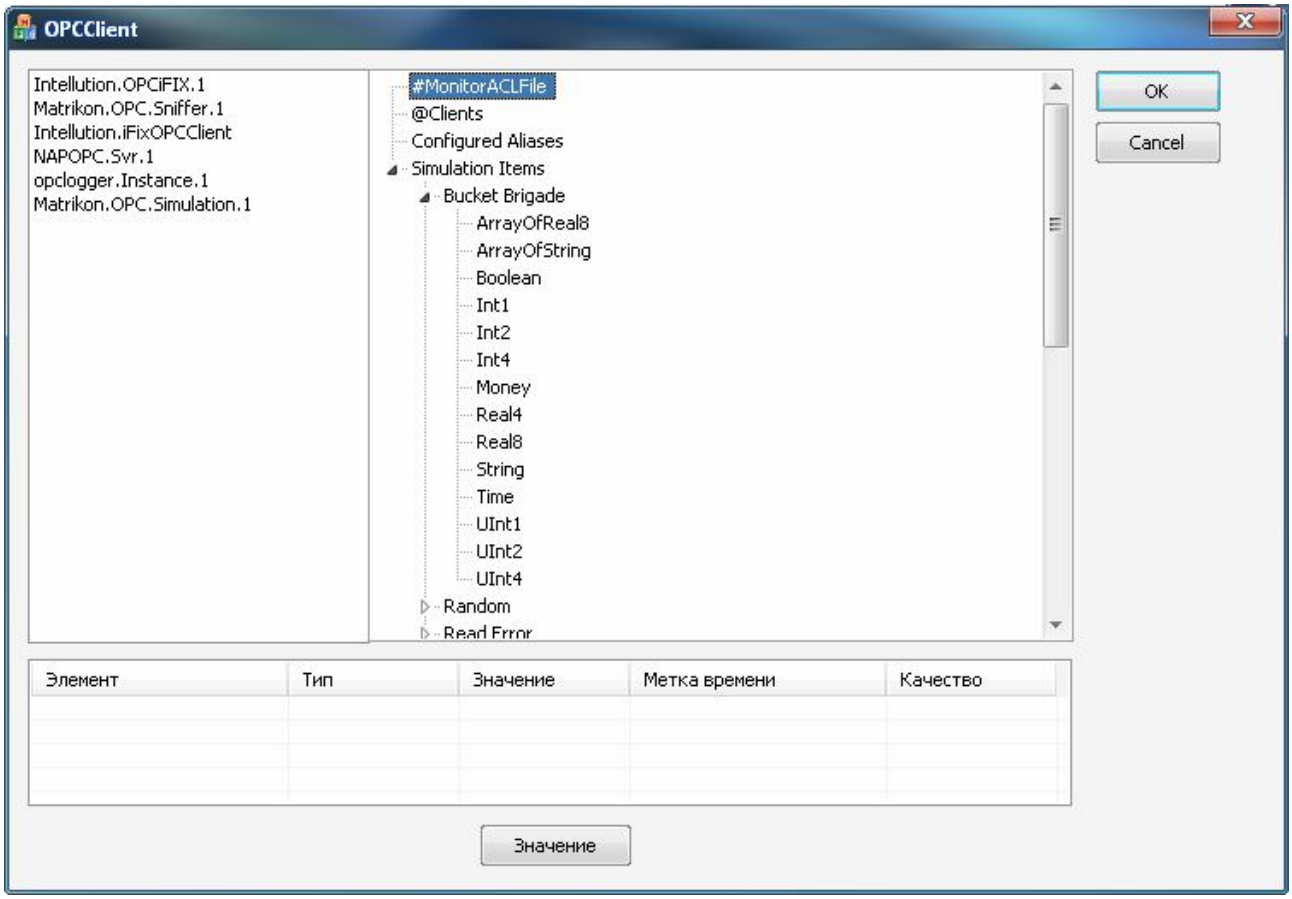



Рисунок 15. – Просмотр содержимого OPC DA 3.0 сервера посредством IOPCBrowse

## Вариант C#/.NET

При разработке OPC клиентов средствами .NET не используются SDK предлагаемые OPC Foundation. Хоть такой подход и несколько усложняет разработку, однако не ставит разработчика в зависимость от методов сторонних SDK и не требует дополнительных инсталляций, т.е. для того, чтобы реализовать примеры, указанные здесь, необходимо лишь наличие стандартных библиотек opcEnum.exe и opcproxy.dll, как и в случае использования C++.

## Предисловие. Взаимодействие COM и .NET

Прежде чем приступить к написанию OPC клиента с использованием .NET необходимо сделать некоторые замечания по поводу используемых в .NET типах данных.

В предыдущей части главы, когда рассматривалась разработка OPC клиента посредством MFC, работа с COM осуществлялась напрямую, поэтому мы программировали в стиле COM.

Но в отличие от MFC .NET является полноценной платформой программирования, а не просто библиотекой классов для облегчения жизни разработчиков. Поэтому она подразумевает использование своих типов данных и ничего не знает о типах COM. Доступ к компонентам COM осуществляется из .NET при помощи оболочки RCW (runtime callable wrapper).

Эта оболочка преобразует интерфейсы COM в интерфейсы, совместимые с платформой .NET Framework.

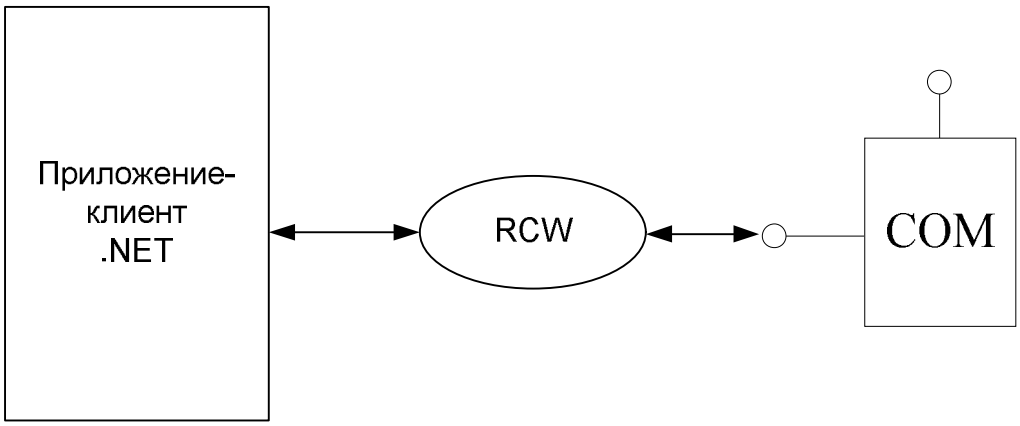



Рисунок 16. – Принцип взаимодействия клиентов .NET с объектами COM

У большинства типов COM есть соответствующие аналоги в .NET. Список соответствия типов COM и .NET, вместе с псевдонимами C# приведены в Таблице 1

Таблица 1. Соответствие типов данных COM и .NET

| № | Тип данных COM | Тип данных .NET | Псевдоним C# |
|---|---|---|---|
| 1 | char, boolean, small | System.SByte | sbyte |
| 2 | wchar_t, short | System.Int16 | short |
| 3 | long, int | System.Int32 | int |
| 4 | hyper | System.Int64 | long |
| 5 | unsigned short | System.UInt16 | ushort |
| 6 | unsigned long, unsigned int | System.UInt32 | uint |
| 7 | unsigned hyper | System.UInt64 | ulong |
| 8 | single | System.Single | float |
| 9 | double | System.Double | double |
| 10 | VARIANT_BOOL | - | bool |
| 11 | HRESULT | System.Int32 | int |
| 12 | BSTR | System.String | string |
| 13 | LPSTR, char * | System.String | string |
| 14 | LPWSTR, wchar_t* | System.String | string |
| 15 | VARIANT | System.Object | object |
| 16 | DECIMAL | System.Decimal | - |
| 17 | DATE | System.DateTime | - |
| 18 | GUID | System.Guid | - |
| 19 | CURRENCY | System.Decimal | |
| 20 | IUnknown | System.Object | object |
| 21 | IDispatch | System.Object | object |

Все методы, реализованные в интерфейсах COM также транслируются в .NET, например, если в интерфейсе реализован метод, который описан в IDL как

```
HRESULT SomeMethod([in] LPWSTR msgText)
```

то модуль RCW представит этот метод клиенту .NET как

```
void SomeMethod(string msgText)
```

С некоторыми отрицательными моментами этого мы столкнемся при реализации функций чтения из OPC сервера, когда придется подправлять библиотеку, сгенерированную по библиотеке типов.

Еще одним отличием является то, что в модели COM для контроля выполнения метода используется возвращенное значение HRESULT, а .NET все методы объявлены как **void**. В случае .NET функции контроля выполнения метода исполняет механизм исключений, т.е. необходимо использовать **try** .. **catch** .. **finally**.

Кроме всего прочего нам понадобится использовать класс **Marshal** для обмена между управляемой и неуправляемой областями памяти. Для этого необходимо включить адресное пространство

```
using System.Runtime.InteropServices;
```

в создаваемый проект.

*Шаг 0. Подготовка диалога*

Первое, что сделаем - создадим приложение Windows Forms и назовем его OPCClient.

Разместим два элемента управления: ListView и TreeView. В первом мы будем отображать список доступных серверов, а в дереве будем показывать его структуру адресного пространства.

Дадим им имена, соответственно m_listOPCServers и m_treeOPCServerBrowse.

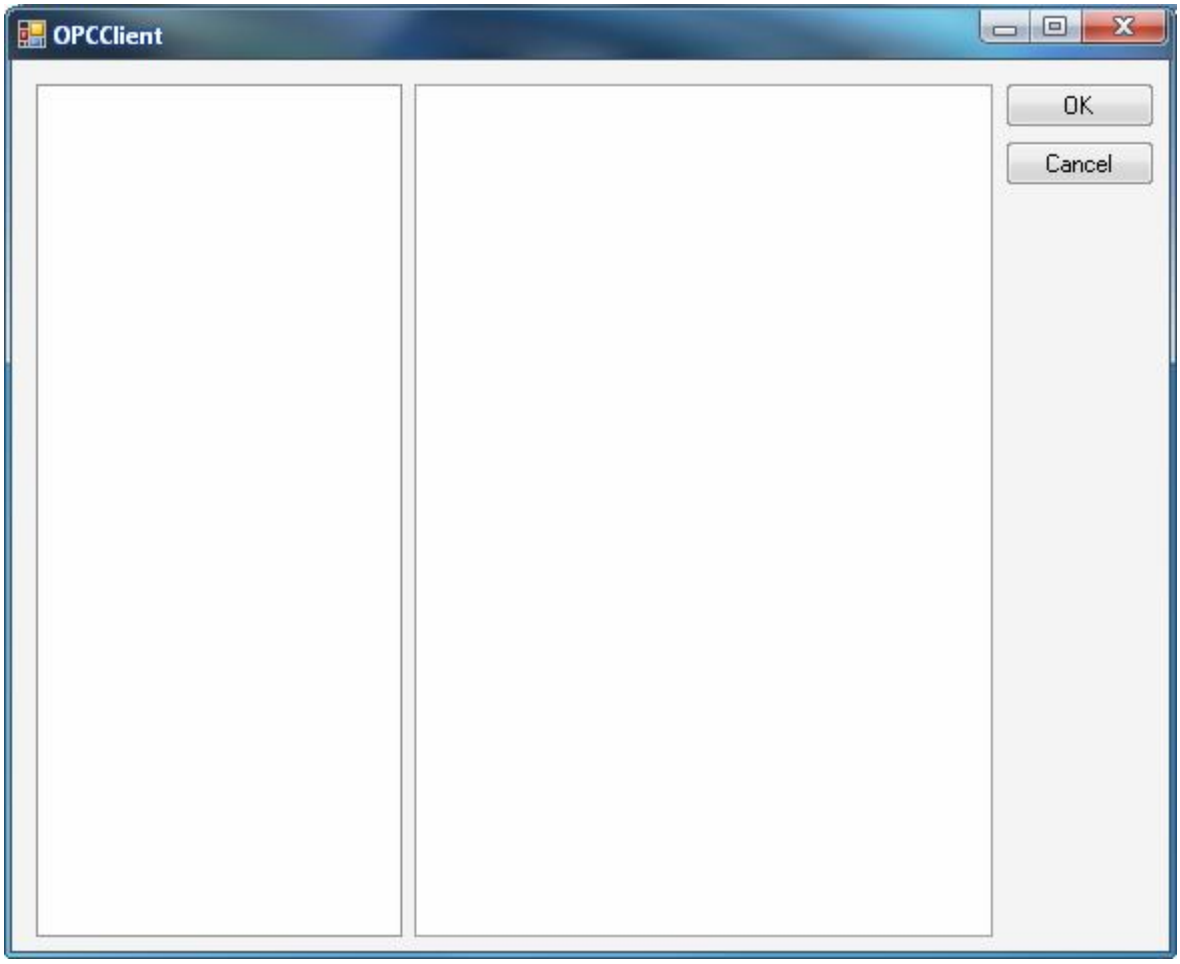



Рисунок 16. – Подготовка
диалога (формы) к разработке

*Шаг 1. Просмотр установленных на локальной машине OPC-серверов*

Для решения задачи просмотра списка установленных OPC-серверов сообществом OPC Foundation была выпущена специализированная утилита OPCEnum. Эта утилита находится в свободном пользовании и, потратив немного времени на поиск, ее можно без проблем найти в Интернете. Кроме того, большинство SCADA систем, являясь OPC-клиентами по умолчанию, при установке инсталлируют указанную утилиту.

OPCEnum содержит в себе библиотеку типов (type library), описывающую инкапсулированные интерфейсы

Любую информацию об OPCEnum можно получить, просмотрев соответствующие ветки системного реестра, или же воспользовавшись утилитой OleView (см. рисунок 17)..

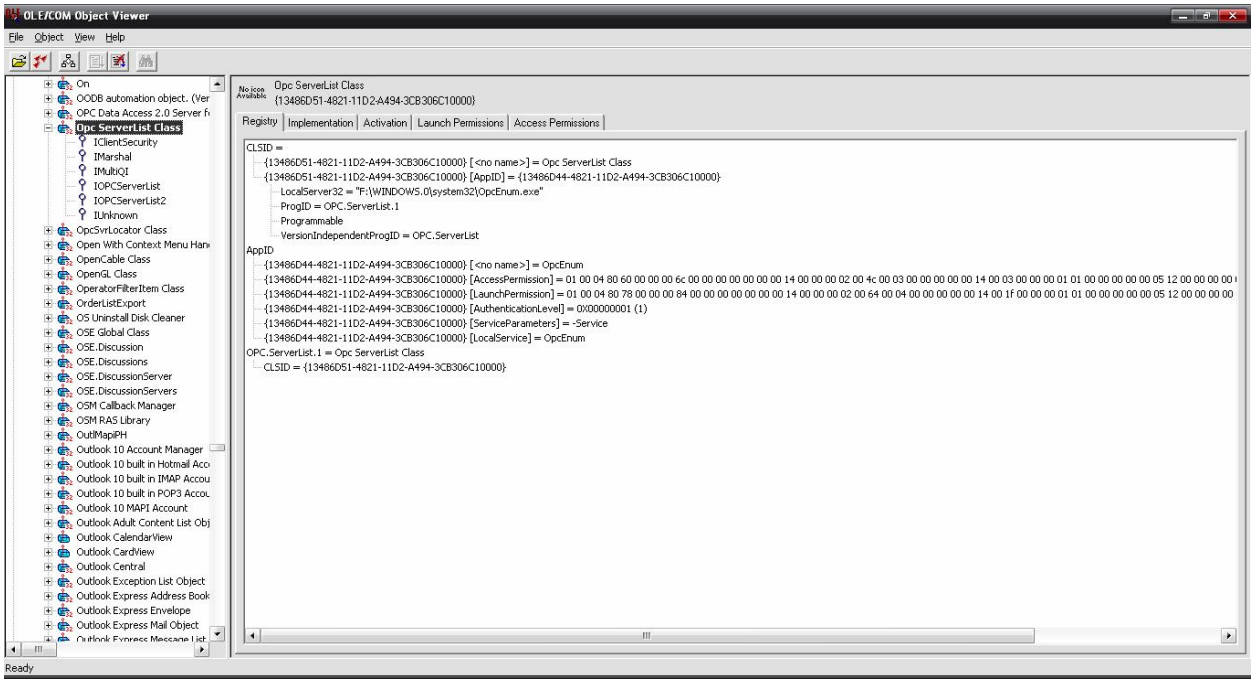



Рисунок 17. – Объект OPC_ServerList утилиты OPCEnum

Для удобства COM-компоненты могут быть сгруппированы в группу с индивидуальным идентификатором, именно так обстоит дело с серверами OPC. Зарегистрированные группы и их идентификаторы также могут быть просмотрены посредством утилиты OLEView(см. рисунок 18)

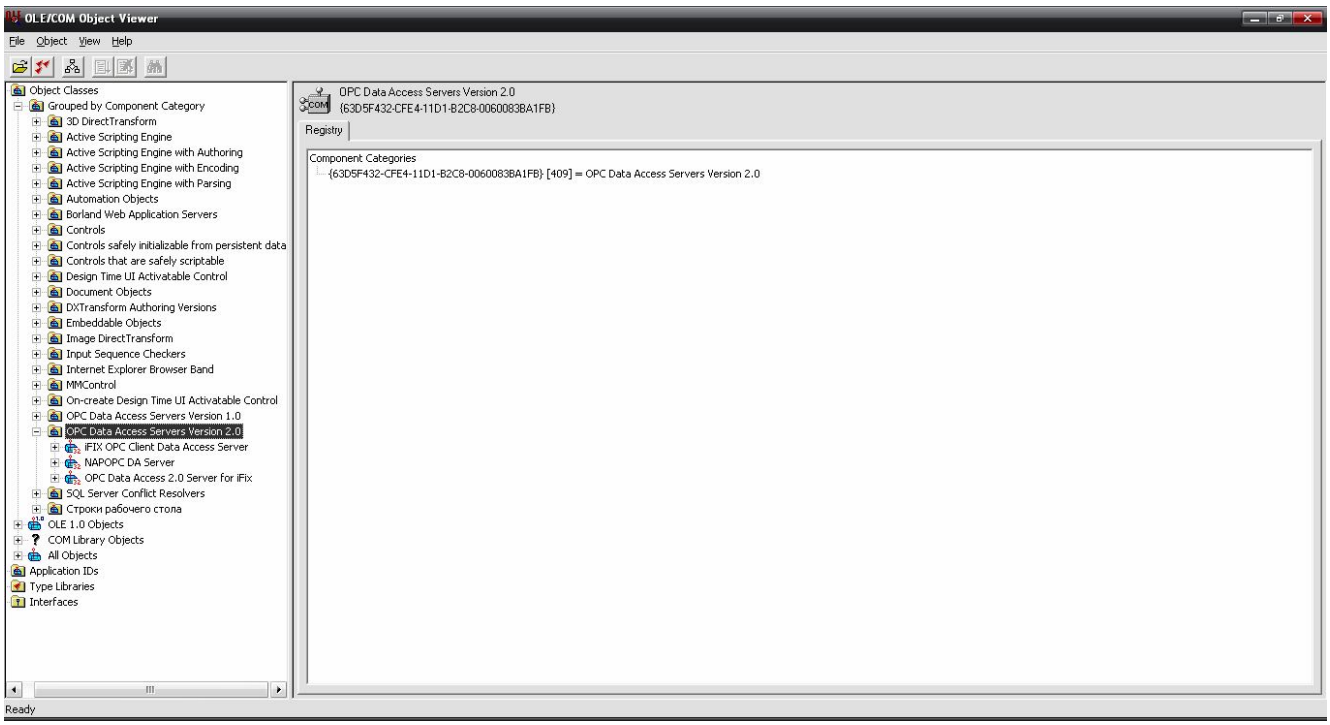



Рисунок 18. – Группа компонентов OPC DA 2.0

Первое что необходимо – это импортировать объявления классов из библиотеки OpcEnum. Т.к. этот файл уже является зарегистрированной библиотекой типов, то его сразу можно включить в проект. Для этого необходимо добавить ссылку на него, Project->Add Reference->COM->OPCEnum 1.1 Type Library.

Если все выполнено верно, то появится возможность использования адресного пространства OPCEnumLib, которое мы можем включить в свой проект обычным образом (выделено курсивом).

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using OpcEnumLib;
```

Теперь доступна работа с OPCEnum. Добавим функцию ShowRegisteredServers, которая будет выводить список зарегистрированных на локальном узле серверов, каждый раз при загрузке формы. Функция будет добавлять в список установленные OPC сервера и возвращать их общее количество.

```
private int ShowRegisteredServers()
```

```
{
//Создаем объект списка OPC серверов
OpcServerList pServerList = new OpcServerList();
// Идентификатор категории OPC DA 2.0
Guid clsidcat = new Guid("{63D5F432-CFE4-11D1-B2C8-
0060083BA1FB}");
//перечислитель, в котором будут храниться GUID серверов
IOPCEnumGUID * pIOPCEnumGuid;
//запрос по группе серверов спецификации OPC DA 2.0
pServerList.EnumClassesOfCategories(1, ref clsidcat, 0,
ref clsidcat, out pIOPCEnumGuid);

string pszProgID; // буфер для записи ProgID серверов
string pszUserType; // буфер для записи описания серверов
Guid guid = new Guid();
int nServerCnt = 0;
uint iRetSvr; // количество серверов, предоставленных запросом

 // получение идентификаторов серверов
pIOPCEnumGuid.Next(1, out guid, out iRetSvr);
while (iRetSvr != 0)
   {
      nServerCnt++;
      pServerList.GetClassDetails(ref guid, out pszProgID,
out pszUserType, out pszVerIndProgID);
      ListViewItem lvItem = new ListViewItem();
      lvItem.Tag = (object)guid;
      lvItem.Text = pszProgID;
      m_listOPCServers.Items.Add(lvItem);
      pIOPCEnumGuid.Next(1, out guid, out iRetSvr);
    }
return nServerCnt;
 }
```

Теперь добавим обработчик события загрузки текущей формы(Load) и поместим туда вызов функции ShowRegisteredServers.

```
protected void OnLoadForm(object sended, EventArgs e)
 {
    if (0 == ShowRegisteredServers())
       {
         MessageBox.Show("Нет установленных серверов");
         } }
```

Результат работы программы приведен на рисунке

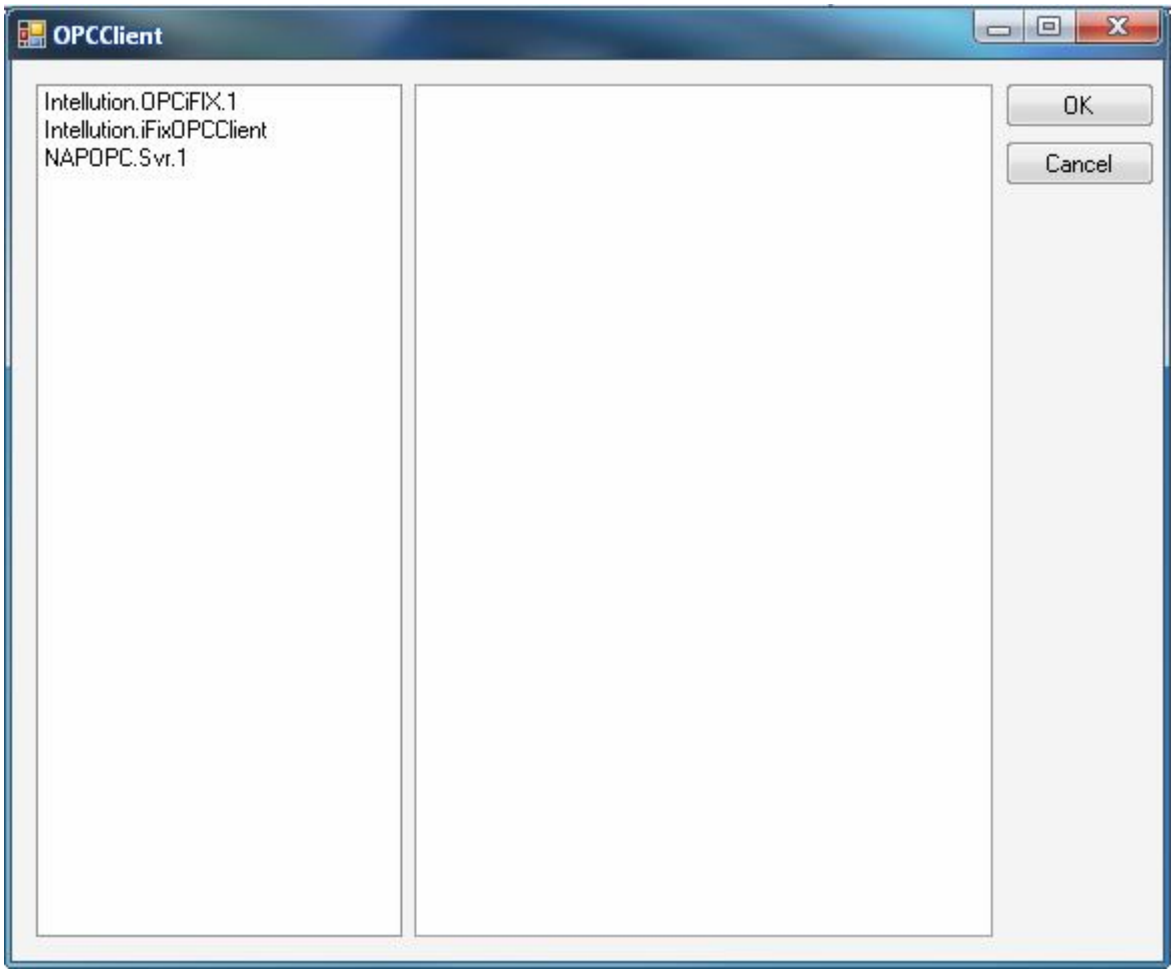


Рисунок 19. – Результат получения списка зарегистрированных серверов

*Шаг 2. Просмотр содержимого OPC сервера*

Реализуем теперь обработчик, который при выделении соответствующего элемента списка будет показывать адресное пространство соответствующего сервера. Важно помнить, что мы имеем дело с обменом данными между приложениями реального времени, поэтому сервер, к которому происходит подключение, должен быть запущен.

Обычно клиенты имеют возможность сами запустить выбранный сервер, однако это не всегда приводит к необходимому результату, т.к. вполне может быть, что сервер должен быть запущен со специфическими параметрами и т.п. В рамках этого клиента мы не будем реализовывать запуск сервера, чтобы не загромождать код.

Первым делом необходимо создать типы данных .NET ссылающиеся на типы данных COM для работы с OPC сервером. Для этого необходимо найти директорию, где расположена библиотека opcproxy.dll, обычно это

```
c:\Windows\system32\opcproxy.dll
```

и импортировать из нее типы данных COM, которые преобразуются в данные .NET.
Для этого необходимо в командной строке набрать команду

```
tlbImp.exe c:\\Windows\system32\opcproxy.dll /out:c:\opcproxy.dll
```

Команду необходимо выполнять из директории, в которой располагается утилита tlbImp или же прописать путь к утилите в переменной окружения PATH, тогда эту команду можно будет вызывать с любого места.
По указанному адресу (в нашем случае это «c:\») появится dll библиотека с импортированными типами данных в нашем случае «opcenum.dll», которые необходимо включить в проект.

Теперь можно добавлять вновь сгенерированную библиотеку в проект в качестве ссылки Object->Add Reference->Browse-> c:\opcproxy.dll.

Станет доступным адресное пространство opcproxy, которое нужно также включить в проект.

```
using opcproxy;
```

Спецификация OPC DA 2.0 и выше регламентирует необязательный интерфейс IOPCBrowseServerAddressSpace.

Не смотря на то, что спецификация не требует его обязательной реализации, у большинства серверов он реализован, что дает возможность клиенту просмотреть его внутреннее строение. Именно этот интерфейс мы и будем использовать.

Далее будем предполагать, что сервер имеет иерархическое адресное пространство (в большей части серверов так и есть).

В сервере с иерархическим адресным пространством существует два типа узлов:

Ветви (OPC_BRANCH) – элементы, у которых есть дочерние элементы

Листья (OPC_LEAF) - элементы у которых нет дочерних элементов

Так ветвями является элемент самого сервера (Server) и элементы групп (Group) в сервере, а листьями являются теги (Item).

Таким образом, необходимо разработать алгоритм, который бы рекурсивно проходил по элементам сервера и разворачивал их в дерево. Такой алгоритм у нас будет реализовывать функция DisplayChildren.

Функция в качестве аргументов принимает ссылку на родительский элемент в дереве и интерфейс IOPCBrowseServerAddressSpace спозиционированный на родительский элемент в сервере.

Тело DisplayChildren приведено ниже

```
private void DisplayChildren(TreeNode ParentNode,
IOPCBrowseServerAddressSpace pParent)
```

```
{
uint cnt;
string strName;
string szItemID;
IEnumString pEnum;
// Вначале выводим все листья на данном уровне
pParent.BrowseOPCItemIDs(tagOPCBROWSETYPE.OPC_LEAF, "", 1, 0, out
pEnum);
pEnum.RemoteNext(1, out strName, out cnt);
while (cnt != 0)
 {
 ParentNode.Nodes.Add(strName);
//получает полный идентификатор тега
 pParent.GetItemID(strName, out szItemID);
 tvNode.Tag = (object)szItemID;
 pEnum.RemoteNext(1, out strName, out cnt);
  }

// Получаем ветви на данном уровне
 pParent.BrowseOPCItemIDs(tagOPCBROWSETYPE.OPC_BRANCH, "", 1, 0,
out pEnum);
 pEnum.RemoteNext(1, out strName, out cnt);
 while (cnt != 0)
    {
  TreeNode parentNode = ParentNode.Nodes.Add(strName);

Parent.ChangeBrowsePosition(tagOPCBROWSEDIRECTION.OPC_BROWSE_DOWN,
strName);
  DisplayChildren(parentNode, pParent); //Рекурсивно вызываем
метод
//После того как все дочерние узлы показаны постепенно поднимаемся
//вверх
  Parent.ChangeBrowsePosition(tagOPCBROWSEDIRECTION.OPC_BROWSE_UP,
strName);
  pEnum.RemoteNext(1, out strName, out cnt);  } }
```

Следующим шагом добавим функцию, которая будет выполнять подключение к серверу и непосредственно вызывать экспорт адресного пространства.
Добавим переменную

```
private object m_pIfaceObject
```

в которой будем хранить интерфейс IOPCServer и через которую впоследствии получать все остальные интерфейсы OPC.
А также функцию

```
private void OnServerChange()
{
}
```

тело которой будет пустым и заполнится позже, когда будут рассмотрены операции чтения данных

Функция принимает в качестве аргумента идентификатор текущего сервера

```
private void ConnectAndBrowseServer(Guid guid)
        {
try
{
 Type typeOfServer = Type.GetTypeFromCLSID(guid);
 m_pIfaceObj = Activator.CreateInstance(typeOfServer);

 if (m_pIfaceObj is IOPCBrowseServerAddressSpace)
  {
IOPCBrowseServerAddressSpace pBrowse =
(IOPCBrowseServerAddressSpace)m_pIfaceObj;
   DisplayChildren(null, pBrowse);
   }
 }
catch (ApplicationException ex)
 {
string msg;
   //Получаем HRESULT, соответствующий сгенерированному исключению
   int hRes = Marshal.GetHRForException(ex);
   //Запрашиваем у сервера текст ошибки
   pServer.GetErrorString(hRes, 2, out msg);
   //Показываем сообщение ошибки
   MessageBox.Show(msg, "Ошибка");
  }
}
```

Теперь необходимо добавить события изменения выделенного узла (SelectedIndexChanged) списка, который будет развертывать адресное пространство сервера

```
private void OPCListClick(object sender, EventArgs e)
{
ListView.SelectedIndexCollection selCol=
m_listOPCServers.SelectedIndices;
if (selCol.Count == 0)  return;
OnServerChange();
Guid guid = (Guid)m_listOPCServers.SelectedItems[0].Tag;
m_treeOPCServerBrowse.Nodes.Clear();
BrowseServer(guid);
         }
```

На рисунке приведен результат работы клиента с серверами

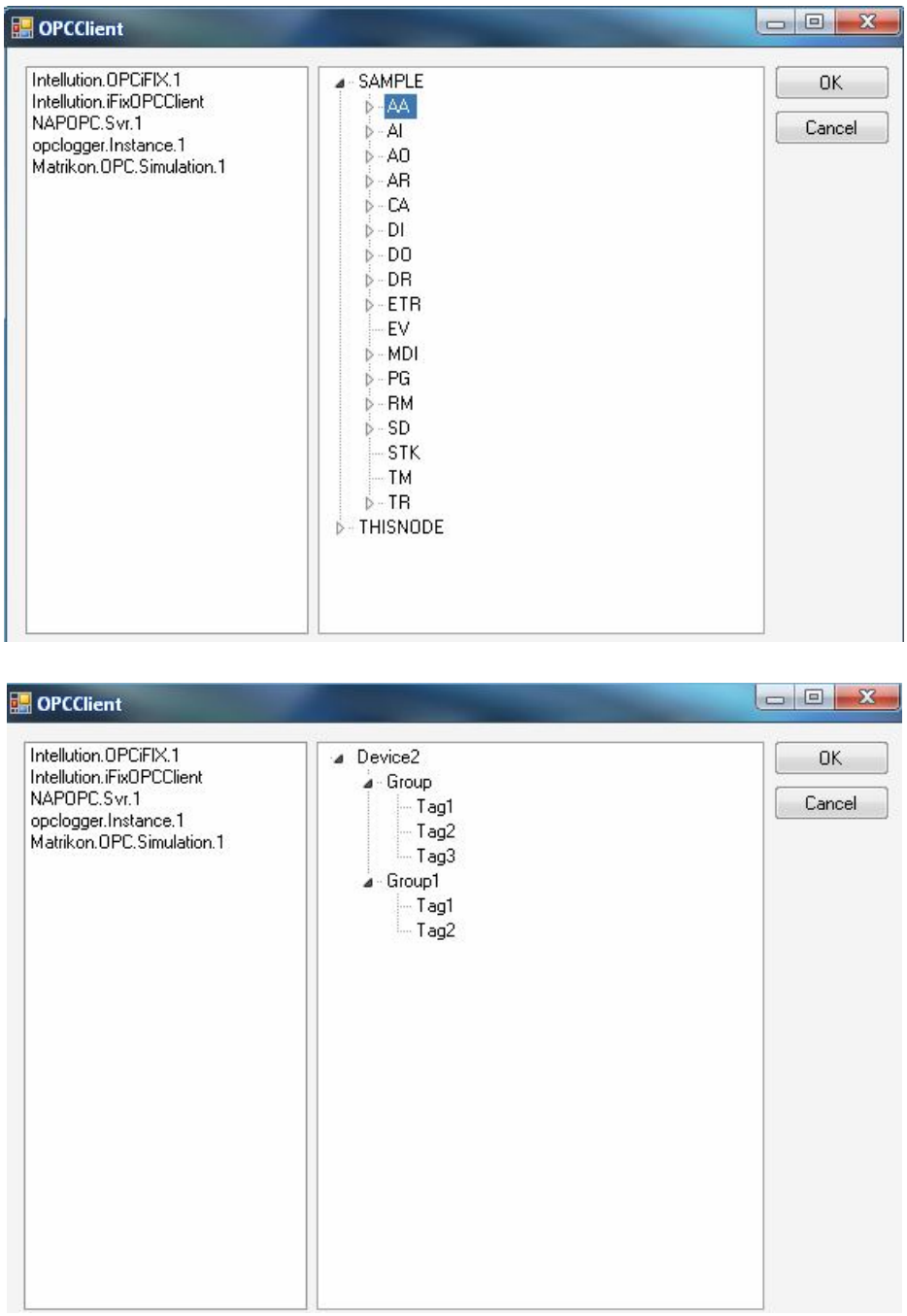



Рисунок 20. – Результат просмотра адресного пространства различных OPC DA серверов

*Шаг 3.1. Синхронное чтение данных из OPC сервера*

В качестве подготовки нашего приложения, добавим элемент ListView, который назовем m_valueView, установим свойство View = Details и добавим пять столбцов с подписями «Элемент», «Тип», «Значение», «Метка времени» и «Качество», куда мы будем выводить соответствующие данные. А также кнопку, которую подпишем «Значение», которую назовем ValueBtn (см. рисунок 21).

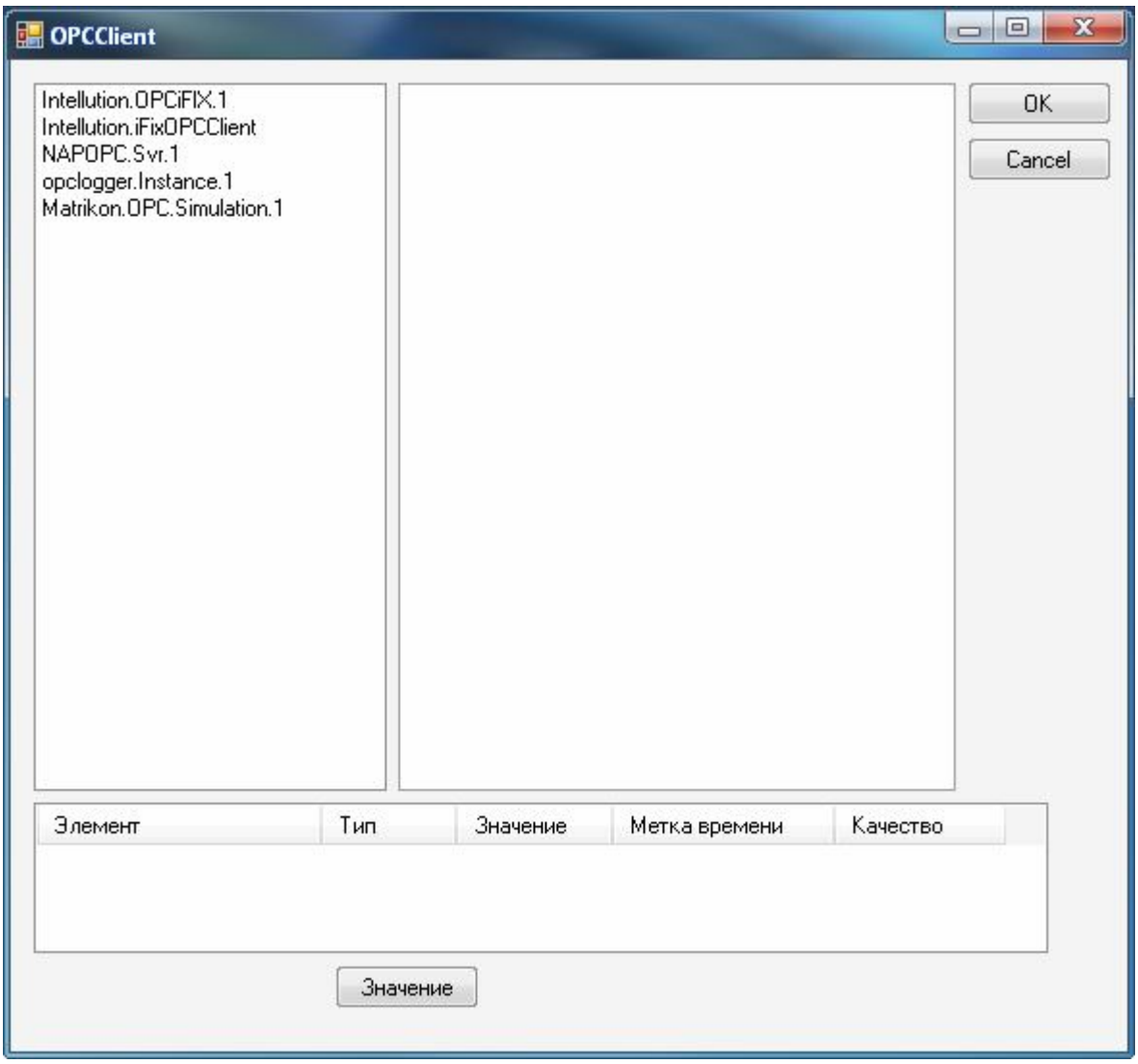


Рисунок 21. – Добавление ListView для вывода свойств считанного элемента

Создадим переменную, в которой будем хранить описатель созданной группы

```
private uint m_hGroup; // Описатель группы
```

и класс со статическими методами для получения текстовой информации о текущем типе элемента данных, качестве и временной метке.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using opcprox;

namespace OPCClient
{
    class ToStringConverter
    {
```

```
        static public string GetQualityString(ushort usQuality)
        {
            switch (usQuality)
            {
                case 0x00: return "Bad";
                case 0x04: return "Config Error";
                case 0x08: return "Not Connected";
                case 0x0C: return "Device Failure";
                case 0x10: return "Sensor Failure";
                case 0x14: return "Last Known";
                case 0x18: return "Comm Failure";
                case 0x1C: return "Out of Service";
                case 0x20: return "Initializing";
                case 0x40: return "Uncertain";
                case 0x44: return "Last Usable";
                case 0x50: return "Sensor Calibration";
                case 0x54: return "EGU Exceeded";
                case 0x58: return "Sub Normal";
                case 0xC0: return "Good";
                case 0xD8: return "Local Override";
                default: return "Unknown";
            }
        }

        static public string GetVTString(ushort vt)
        {
            return ((VarEnum)vt).ToString();
        }

        static public string GetFTSting(_FILETIME ft)
        {
            long lFT = (((long)ft.dwHighDateTime) << 32) +
ft.dwLowDateTime;
            DateTime dt = DateTime.FromFileTime(lFT);
            return dt.ToString();
        }
    }
}
```

Подготовка завершена можно начинать реализацию необходимых функций.

Добавим обработчик нажатия на кнопку «Значение». В листинге определенные функции выделены подчеркиванием, это связано с тем, что, вероятнее всего, сразу они не будут верно функционировать и будут подчеркиваться как содержащие ошибку. Причиной этому является не совсем корректная генерация прототипов методов интерфейсов по данным библиотеки типов.

После листинга приведены действия, которые необходимо выполнить, чтобы ликвидировать эту проблему.

```
private void ValueButton_Click(object sender, EventArgs e)
{
//Получаем текущий выбранный элемент данных
if (null == m_treeOPCServerBrowse.SelectedNode) return;
string itemID = (string) m_treeOPCServerBrowse.SelectedNode.Tag;

// Запрашиваем интерфейс IOPCServer
IOPCServer pServer = (IOPCServer)m_pIfaceObj;

uint updateRate=1000; // время опроса создаваемой группы
int bActive = 1;     // активность группы - активна
uint hClientGroup = 1; // клиентский описатель группы

object iFaceObj; // сюда вернем интерфейс к группе
Guid riid = typeof(IOPCItemMgt).GUID;

int TimeBias=0; // не используем смещения по времени
float DeadBand=0; // не используем зону нечувствительности

  try
  {
     /*
Этот участок на данном этапе должен быть закомментирован. Его
необходимо будет использовать тогда, когда добавим асинхронную
операцию чтения по подписке
if (m_dwCookie != 0)
      {
          m_pDataCallback.Unadvise(m_dwCookie);
          m_dwCookie = 0;
      }
*/
      if (m_hGroup != 0) // Если группа была создана
      {
          pServer.RemoveGroup(m_hGroup, 1); // удалим ее
          m_hSyncGroup = 0;
      }

//Создаем группу и получаем интерфейс IOPCItemMgt для управления
ее состоянием

pServer.AddGroup("MyGroup", bActive, updateRate, hClientGroup, ref
TimeBias, ref DeadBand, 2, out m_hSyncGroup, out updateRate, ref
riid, out iFaceObj);
 IOPCItemMgt pItemMgt = (IOPCItemMgt)iFaceObj;


//Создаем список элементов для добавления в группу, размером 1
элемент
uint dwCount  = 1;

//Создаем описатель добавляемого элемента
tagOPCITEMDEF pItems = new tagOPCITEMDEF();
```

```
pItems.szItemID = itemID;
pItems.szAccessPath = null;
pItems.bActive = 1;
pItems.hClient = 1;
pItems.vtRequestedDataType = (ushort)VarEnum.VT_EMPTY;
pItems.dwBlobSize = 0;
pItems.pBlob = IntPtr.Zero;

// В эти две переменные будут записаны массивы ошибок и
результатов выполнения
IntPtr iptrErrors = IntPtr.Zero;
IntPtr iptrResults = IntPtr.Zero;

// Добавляем элемент данных в группу
pItemMgt.AddItems(dwCount,pItems, out iptrResults, out
iptrErrors);

// Переносим результаты и ошибки из неуправляемой памяти в
управляемую
tagOPCITEMRESULT pResults =
(tagOPCITEMRESULT)Marshal.PtrToStructure(iptrResults,
typeof(tagOPCITEMRESULT));
int[] hRes = new int[1];
Marshal.Copy(iptrErrors, hRes, 0, 1);

//Генерируем исключение в случае ошибки в HRESULT
Marshal.ThrowExceptionForHR(hRes[0]);

// Получаем интерфейс IOPCSyncIO для операций синхронного чтения
IOPCSyncIO pSyncIO = (IOPCSyncIO)iFaceObj;

// В эту переменную будут записаны результаты чтения
IntPtr iptrItemState = IntPtr.Zero;

iptrErrors = IntPtr.Zero;

//Читаем данные из сервера
pSyncIO.Read(tagOPCDATASOURCE.OPC_DS_DEVICE, 1, ref
pResults.hServer, out iptrItemState, out iptrErrors);

// Переносим результаты и ошибки из неуправляемой памяти в
управляемую
tagOPCITEMSTATE pItemState =
(tagOPCITEMSTATE)Marshal.PtrToStructure(iptrItemState,
typeof(tagOPCITEMSTATE));

Marshal.Copy(iptrErrors, hRes, 0, 1);

//Генерируем исключение в случае ошибки в HRESULT
Marshal.ThrowExceptionForHR(hRes[0]);
```

```
//Выводим полученные данные
ListViewItem lvItem = new ListViewItem();
ListViewItem.ListViewSubItem[] lvSubItem = new
ListViewItem.ListViewSubItem[3];
lvItem.Text = itemID;

lvItem.SubItems.Add(ToStringConverter.GetVTString(pResults.vtCanon
icalDataType));
lvItem.SubItems.Add(pItemState.vDataValue.ToString());
lvItem.SubItems.Add(ToStringConverter.GetFTSting(pItemState.ftTime
Stamp));
lvItem.SubItems.Add(ToStringConverter.GetQualityString(pItemState.
wQuality));

m_valueView.Items.Clear();
m_valueView.Items.Add(lvItem);
  }
  catch (System.Exception ex)
  {
      string msg;
    //Получаем HRESULT соответствующий сгененрированному
исключению
      int hRes = Marshal.GetHRForException(ex);
    //Запрашиваем у сервера текст ошибки, соответствующий текущему
HRESULT
      pServer.GetErrorString(hRes, 2, out msg);
    //Показываем сообщение ошибки
      MessageBox.Show(msg, "Ошибка");
  }
  }
```

Теперь разберем проблему с подчеркнутыми функциями, у нас их две

pItemMgt.AddItems();

pSyncIO.Read();

которые подчеркнуты и Visual Studio как содержащие ошибки (если не подчеркнуты, то последующее описание можно не читать). Проблема заключается в том, что искомом IDL файле описывающем интерфейсы эти функции объявлены как

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

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)
```

И как уже упоминалось .NET не оперирует такими типами данных и требует преобразовать их в свои, поэтому RCW преобразует эти объявления функции в прототипы вида

```
void AddItems(uint dwCount, ref opcprox.tagOPCITEMDEF pItemArray,
System.IntPtr ppAddResults,  System.IntPtr ppErrors)

void Read(opcprox.tagOPCDATASOURCE dwSource, uint dwCount, ref
uint phServer, System.IntPtr ppItemValues, System.IntPtr ppErrors)
```

т.е., заранее не зная, размеры массивов структур оно преобразовало двумерные массивы в тип IntPtr.

Как известно, если переменная передается в функцию в C# и при этом она подразумевается лишь как выходная и не используется внутри метода никаким образом кроме как запись в нее данных, то перед ней в прототипе метода и при вызове должно стоять ключевое слово out. Но компилятор idl не добавил этого в прототип и поэтому компилятор C# будет требовать выделения памяти под переменные, которые передаются в качестве параметров ppAddResults, ppItemValues, ppErrors.

Присвоение этим переменным значения IntPtr.Zero будет генерировать ошибку «Заглушке передан нулевой указатель», выделение под них памяти в управляемой куче посредством с помощью new будет вызывать ошибку памяти, а использование Marshal.AllocComTaskMem(), хоть и не генерирует ошибок, не приводит ни к какому результату, т.к. память должна выделяться на стороне сервера и сервер должен заполнять эти значения, т.е. эти параметры должны передаваться с параметром out.

Поэтому необходимо переделать сгенерированную библиотеку вручную, что, впрочем, совсем не сложно.

Корректировка библиотеки происходит в 3 шага: деассемблирование полученной библиотеки, изменения, обратная сборка библиотеки.

Первый шаг осуществляется посредством утилиты ildasm.exe, которая вероятнее всего находится в папке

c:\Program Files\Microsoft SDKs\Windows\v7.0A\bin\

Теперь необходимо в командной строке выполнить следующую команду

Ildasm.exe c:\opcprox.dll  /out:c:\opcprox.il

как приведено на рисунке

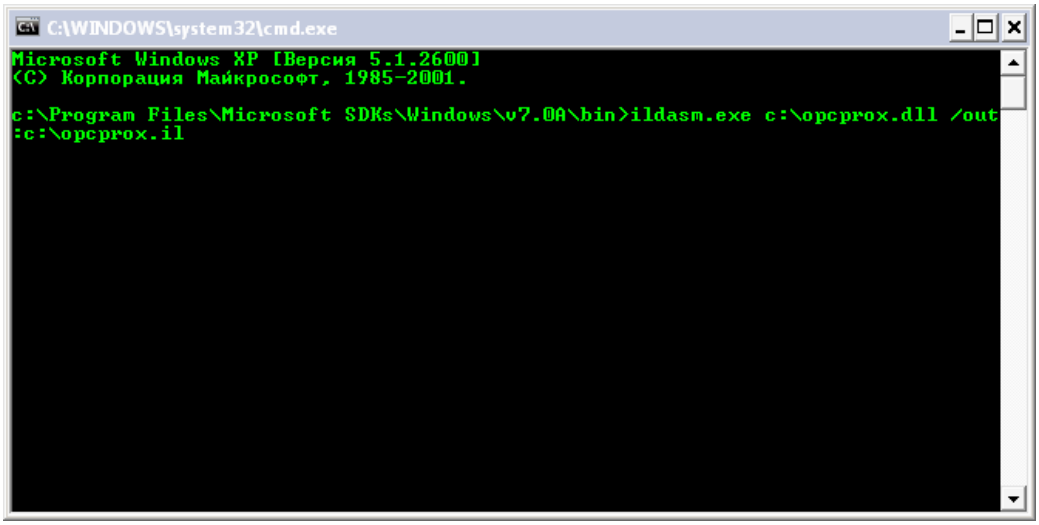



Рисунок 22. – Деассемблирование библиотеки opcprox.dll

Напомним, что c:\opcprox.dll – это библиотека, которую мы сгенерировали утилитой tlbImp из библиотеки типов opcproxy.dll и затем включили в качестве ссылки в наш проект.

Если все проделано верно, то на указанному пути появится текстовый файл opcprox.il. Его необходимо открыть для редактирования. Вначале начнем с редактирования метода AddItems() интерфейса IOPCItemMgt для этого необходимо найти место, где представлено следующее

```
.method public hidebysig newslot abstract virtual
          instance void  AddItems([in] uint32 dwCount,
                                  [in] valuetype
opcprox.tagOPCITEMDEF& pItemArray,
                                  [out] native int ppAddResults,
                                  [out] native int ppErrors)
 runtime managed internalcall
  {
  } // end of method IOPCItemMgt::AddItems
```

Нас интересует два аргумента native int ppAddResults и native int ppError.

Как видете конструкция вида [out] native int в результате транслируется в System.IntPtr, а нам необходимо, чтоб в результате получилось out System.IntPtr, для этого необходимо добавить знак ссылки & перед переменными, чтоб объявления приняли вид

```
.method public hidebysig newslot abstract virtual
          instance void  AddItems([in] uint32 dwCount,
                                  [in] valuetype
opcprox.tagOPCITEMDEF& pItemArray,
                                  [out] native int& ppAddResults,
                                  [out] native int& ppErrors)
 runtime managed internalcall
  {
```

```
  } // end of method IOPCItemMgt::AddItems
```

Аналогично необходимо поступить и с функцией IOPCSyncIO::Read, чтоб ее объявление приняло вид

```
instance void  Read([in] valuetype opcprox.tagOPCDATASOURCE
dwSource,
                                [in] uint32 dwCount,
                                [in] uint32& phServer,
                                [out] native int& ppItemValues,
                                [out] native int& ppErrors) runtime
managed internalcall
  {    } // end of method IOPCSyncIO::Read
```

На этом второй шаг окончен, осталось собрать библиотеку вновь. Для этого предназначена другая утилита – ilasm.exe, которое вероятнее всего находится в папке

c:\windows\Microsoft.NET\Framework\v2.0.50727

необходимо выполнить следующую команда

ilasm /dll c:\opcprox.il

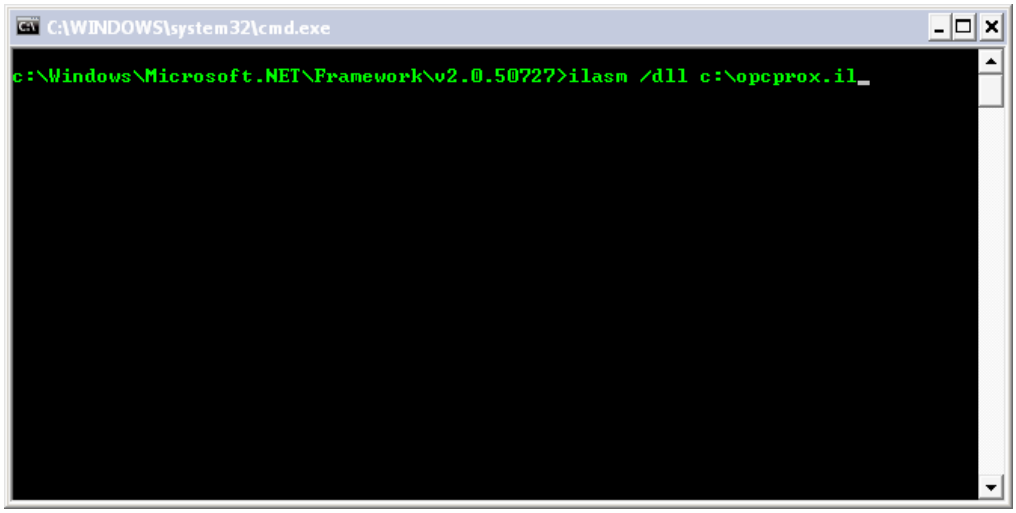



Рисунок 23. – Обратная сборка библиотеки opcprox.dll

В результате библиотека opcprox пересобирется и связанные с этим ошибки в проекте исчезнут.
Результат коммуникаци с сервером приведен на рисунке

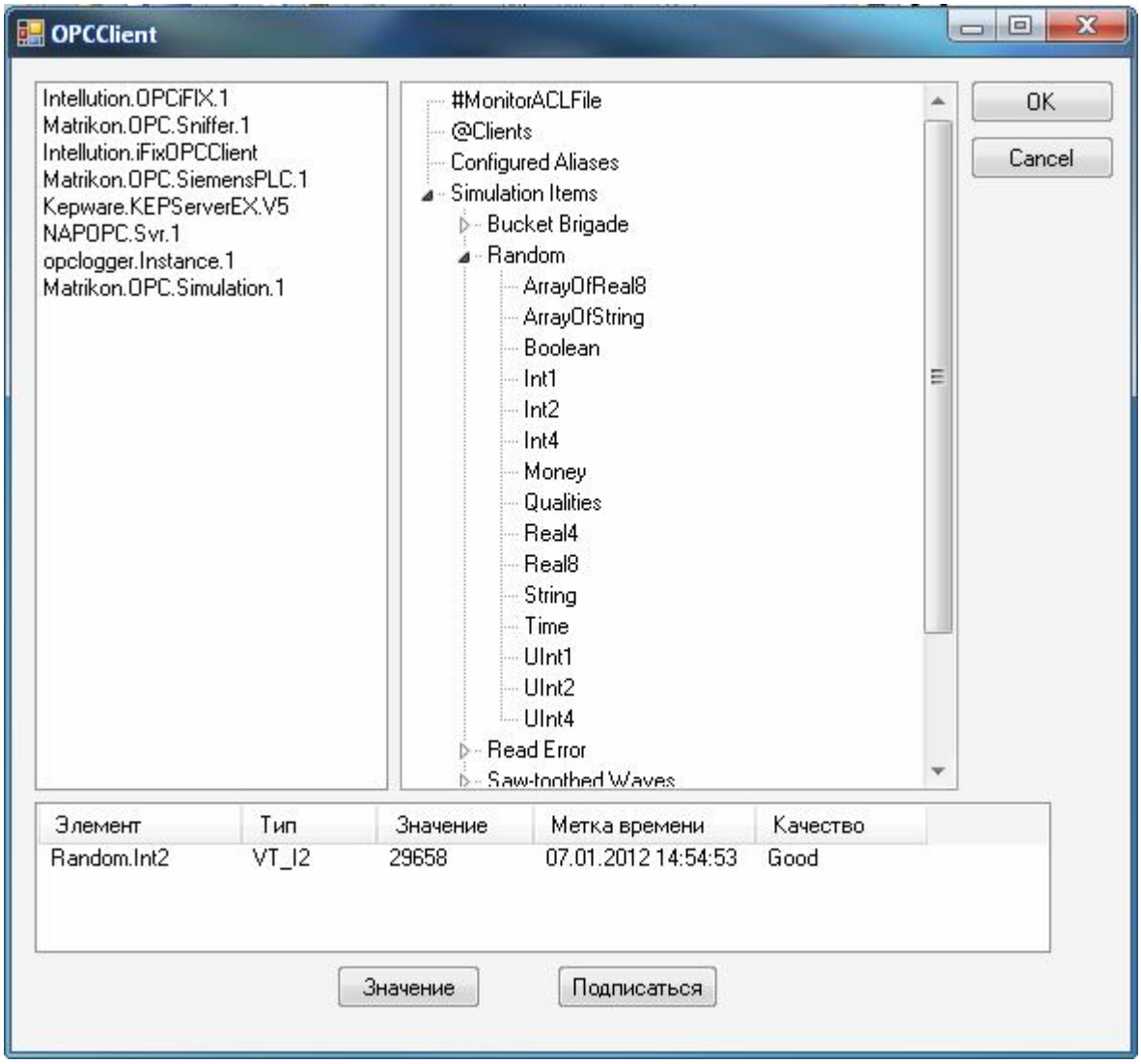



Рисунок24. - Результат синхронного чтения данных с сервера

*Шаг 3.2. Асинхронное чтение данных из сервера*

Добавим необходимы переменные

```
public DataCallback m_pSink; //Объект, который будет принимать вызовы сервера
public IConnectionPoint m_pDataCallback; // Точка подключения к событиям сервера
int m_dwCookie; // Описатель подписки к событиям сервера
```

Все методы класса DataCallback будут выполняться в рабочем потоке. Т.к. нам необходимо будет в этом потоке выводить данные в элемент управления ListView нам необходимо создать делегата в нашей Форме и сделать его ответственным за взаимодействие с элементом m_valueView. Для того, чтобы это сделать нужно в класс формы вставить код

```
delegate void SetItemCallback(ListViewItem lvItem);
 public void SetItemValue(ListViewItem lvItem)
   {
      if (m_valueView.InvokeRequired)
```

```
        {
            SetItemCallback d = new SetItemCallback(SetItemValue);
            Invoke(d, new object[] { lvItem });
        }
        else
        {
            m_valueView.Items.Clear();
            m_valueView.Items.Add(lvItem);
        }
    }
```

Теперь рабочий поток будет не на прямую работать с элементом управления, а делегировать эти действия основному потоку, через функцию SetItemValue.
Реализация класса DataCallback имеет вид

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using opcprox;
using System.Runtime.InteropServices;
using System.Windows.Forms;
using System.Threading;
using System.Runtime.InteropServices.ComTypes;

namespace OPCClient
{
  //Наследуем класс от интерфейса IOPCDataCallback
public class DataCallback : IOPCDataCallback
  {
    //конструктор
public DataCallback(OPCClientDlg dlg, string szItemID)
  {
     m_Dlg = dlg; // Передаем ссылку на нашу форму в качестве параметра
     SetItemID(szItemID); // и идентификатор текущего элемента данных
        }

  public void SetItemID(string szItemID)
  {
    m_szItemID = szItemID;
  }
   //========Перегрузка методов интерфейса  IOPCDataCallback ============
 public void OnCancelComplete(uint dwTransid, uint hGroup) { }

public void OnDataChange(uint dwTransid, uint hGroup, int hrMasterquality, int hrMastererror,
uint dwCount, ref uint phClientItems, ref object pvValues, ref ushort pwQualities, ref
_FILETIME pftTimeStamps, ref int pErrors)
{
//Получаем тип элемента данных
```

```
IntPtr iptrValues = Marshal.AllocCoTaskMem(2);
Marshal.GetNativeVariantForObject(pvValues, iptrValues);
byte [] vt = new byte[2];
Marshal.Copy(iptrValues,vt, 0, 2);
Marshal.FreeCoTaskMem(iptrValues);
ushort usVt =(ushort) (vt[0] + vt[1]*255);

ListViewItem lvItem = new ListViewItem();
ListViewItem.ListViewSubItem[] lvSubItem =
new ListViewItem.ListViewSubItem[3];

lvItem.Text = m_szItemID;

lvItem.SubItems.Add(ToStringConverter.GetVTString(usVt));
lvItem.SubItems.Add(pvValues.ToString());
lvItem.SubItems.Add(ToStringConverter.GetFTSting(pftTimeStamps));
lvItem.SubItems.Add(ToStringConverter.GetQualityString(pwQualities));

m_Dlg.SetItemValue(lvItem); //делегируем вызов форме
 }

public void OnReadComplete(uint dwTransid, uint hGroup, int hrMasterquality, int
hrMastererror, uint dwCount, ref uint phClientItems, ref object pvValues, ref ushort pwQualities,
ref _FILETIME pftTimeStamps, ref int pErrors)
  {
   }

public void OnWriteComplete(uint dwTransid, uint hGroup, int hrMastererr, uint dwCount, ref
uint pClienthandles, ref int pErrors) { }

//================Переменные=========================
private OPCClientDlg m_Dlg;
private string m_szItemID;
      }
   }
```

Теперь можно приступать к операции асинхронного обмена данными. Для этого вынесем на форму кнопку и дадим ей название «Подписаться»

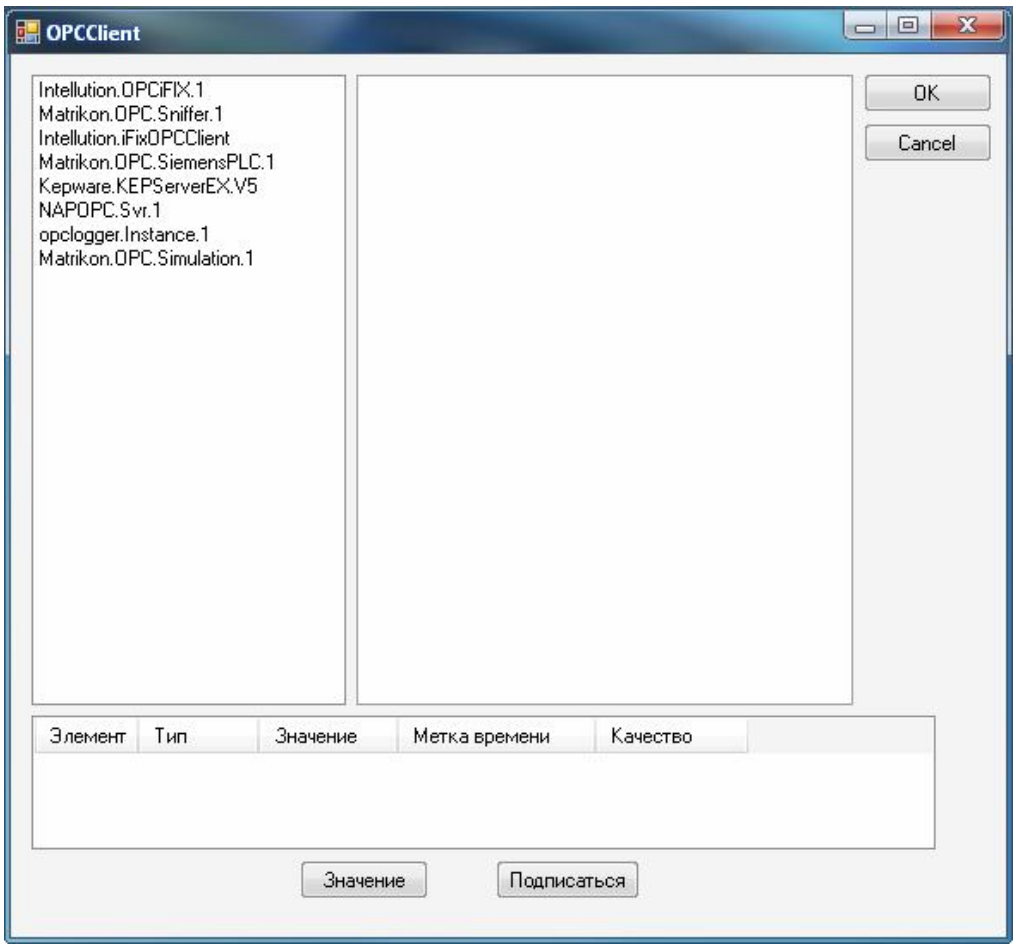



Рисунок 25. – Добавление кнопки «Подписаться»

При нажатии на эту кнопку будет происходить подписка на выбранный элемент данных и при каждом его изменении сервер производить метод DataCalback.OnDataChange.

Обработчик нажатия на эту кнопку будет иметь вид

```
private void AdviseButton_Click(object sender, EventArgs e)
{

//Процесс создания группы и добавления элементов данных аналогичен синхронному чтению

if (null  == m_treeOPCServerBrowse.SelectedNode) return;
string szItemID = (string) m_treeOPCServerBrowse.SelectedNode.Tag;

IOPCServer pServer = (IOPCServer)m_pIfaceObj;

 uint updateRate=1000; //частота опроса группы

object iFaceObj = null;
Guid riid = typeof(IOPCItemMgt).GUID;
int TimeBias=0;
float DeadBand=0;
int bActive = 1;
```

```
uint hClientGroup = 1;
  try
  {
 //Если ранее была активирована подписка, то отменить ее
if (m_dwCookie != 0)
      m_pDataCallback.Unadvise(m_dwCookie);
 if (m_hGroup != 0)
 {
    pServer.RemoveGroup(m_hGroup, 1);
    m_hASyncGroup = 0;
 }

pServer.AddGroup("MyGroup", bActive, updateRate, hClientGroup, ref TimeBias, ref DeadBand,
2, out m_hASyncGroup, out updateRate, ref riid, out iFaceObj);
IOPCItemMgt pItemMgt = (IOPCItemMgt)iFaceObj;

uint dwCount  = 1;
IConnectionPointContainer pCPC;
pCPC = (IConnectionPointContainer)iFaceObj;

riid = typeof(IOPCDataCallback).GUID;
pCPC.FindConnectionPoint(ref riid, out m_pDataCallback);


tagOPCITEMDEF pItems = new tagOPCITEMDEF();

//Создаем объект обработчика событий сервера или если он уже был создан изменяем
идентификатор элемента данных на текущий
if (null == m_pSink)
   m_pSink = new DataCallback(this, szItemID);
else
   m_pSink.SetItemID(szItemID);

//Подписываемся на события сервера
m_pDataCallback.Advise(m_pSink, out m_dwCookie);

pItems.szItemID = szItemID;
pItems.szAccessPath = null;
pItems.bActive = 1;
pItems.hClient = 1;
pItems.vtRequestedDataType = (ushort)VarEnum.VT_EMPTY;
pItems.dwBlobSize = 0;
pItems.pBlob = IntPtr.Zero;

IntPtr iptrErrors = IntPtr.Zero;
IntPtr iptrResults = IntPtr.Zero;

pItemMgt.AddItems(dwCount,pItems, out iptrResults, out iptrErrors);

int[] hRes = new int[1];
Marshal.Copy(iptrErrors, hRes, 0, 1);
```

```
Marshal.ThrowExceptionForHR(hRes[0]);

tagOPCITEMRESULT pResults =
(tagOPCITEMRESULT)Marshal.PtrToStructure(iptrResults, typeof(tagOPCITEMRESULT));

Marshal.FreeCoTaskMem(iptrResults);
Marshal.FreeCoTaskMem(iptrErrors);
 }
  catch (ApplicationException ex)
  {
     string msg;
     int hRes = Marshal.GetHRForException(ex);
     pServer.GetErrorString(hRes, 2, out msg);
     MessageBox.Show(msg, "Ошибка");
  }
  }
```

Теперь с каждым изменение элемента данных его значения и параметры будут посылаться клиенту в метод OnDataChange.

Для корректной работы теперь необходимо раскомментировать участок в функции синхронного чтения, который отменяет подписку на события сервера. Результат подписки приведен на рисунке

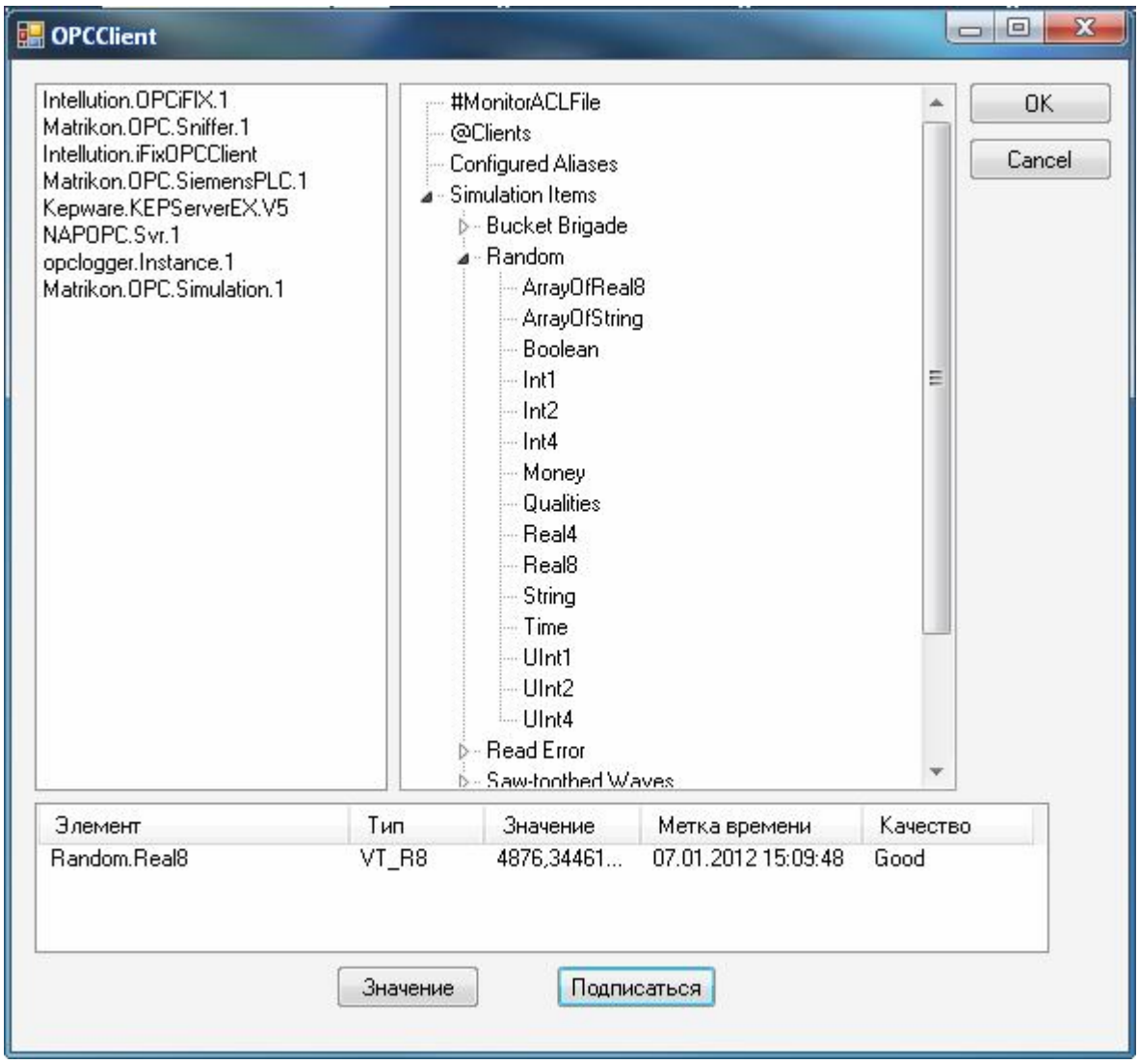



Рисунок *26. – Результат подписки на изменения данных сервера*

*Последние штрихи*

Напоследок необходимо сделать еще несколько действий.

Во-первых, наполнить функцию OnServerChange(), которая каждый раз при изменении выбранного сервера будет отменять подписку на данные и созданные в сервере группы

```
private void OnServerChange()
{
  if (m_dwCookie != 0)
    {
      m_pDataCallback.Unadvise(m_dwCookie);
      m_dwCookie = 0;
        }
  if (m_hGroup != 0)
    {
      IOPCServer pServer = (IOPCServer)m_pIfaceObj;
      pServer.RemoveGroup(m_hGroup, 1);
      m_hGroup = 0;
```

```
    }
  }
```

Во-вторых, обработчик завершения приложения, который, по большому счету, должен выполнять те же действия, что и при смене сервера, поэтому может быть представлен как

```
private void OPCClientDlg_FormClosing(object sender, FormClosingEventArgs e)
    {
     try
     {
        if (m_dwCookie != 0)
        {
           m_pDataCallback.Unadvise(m_dwCookie);
        }

        if (m_hGroup != 0)
        {
     IOPCServer pServer = (IOPCServer)m_pIfaceObj;

           pServer.RemoveGroup(m_hGroup, 1);
        }
     }
     catch (ApplicationException ex)
     {
        int hRes = Marshal.GetHRForException(ex);
        string msg;
        pServer.GetErrorString(hRes, 2, out msg);
        MessageBox.Show(msg, "Ошибка");
     }
    }
```

## Замечания по поводу версии OPC DA 3.0

Здесь к разработанному приложению будет добавлена поддержка интерфейса IOPCBrowse.

Первое, что необходимо сделать – это изменить функцию BrowseServer, которая теперь будет запрашивать интерфейс IOPCBrowse, а если сервер его не поддерживает, т.е. соответствует лишь спецификации 2.0, то интерфейс IOPCBrowseServerAddressSpace

```
  private void ConnectAndBrowseServer(Guid guid)
        {
            try
            {
                Type typeOfServer = Type.GetTypeFromCLSID(guid);
```

```
                m_pIfaceObj =
Activator.CreateInstance(typeOfServer);
                if (m_pIfaceObj is IOPCBrowse)
                    {
                        IOPCBrowse pBrowse =
(IOPCBrowse)m_pIfaceObj;
                        DisplayChildren(null, "", pBrowse);
                    }

                  else
                   if (m_pIfaceObj is
IOPCBrowseServerAddressSpace)
                  {
                      IOPCBrowseServerAddressSpace pBrowse =
(IOPCBrowseServerAddressSpace)m_pIfaceObj;
                      DisplayChildren(null, pBrowse);
                }

            }
            catch (ApplicationException ex)
            {
               MessageBox.Show(ex.Message, "Ошибка");
            }

        }
```

Далее необходимо добавить перегруженный вариант функции DisplayChildren

```
 private void DisplayChildren(TreeNode ParentNode,
string szItemName,IOPCBrowse pBrowse)
{
uint uiProp = 0;
int bMoreElements;
string ContinuationPoint = "";
uint dwCount;
IntPtr pElements = IntPtr.Zero;

pBrowse.Browse(szItemName, ref ContinuationPoint, 0,
tagOPCBROWSEFILTER.OPC_BROWSE_FILTER_ITEMS, "", "", 1,1,0,
uiProp, out bMoreElements, out dwCount, out pElements);

tagOPCBROWSEELEMENT[] OPCItemElements = new
tagOPCBROWSEELEMENT[dwCount];
int sz = Marshal.SizeOf(typeof(tagOPCBROWSEELEMENT));
TreeNode tvNode;
for (int i = 0; i < dwCount; i++)
{
 OPCItemElements[i] (tagOPCBROWSEELEMENT)Marshal.PtrToStructure
(pElements + i * sz, typeof(tagOPCBROWSEELEMENT));
if (null == ParentNode)
```

```
 tvNode = m_treeOPCServerBrowse.Nodes.Add(OPCItemElements[i].szName);
else
 tvNode = ParentNode.Nodes.Add(OPCItemElements[i].szName);

tvNode.Tag = OPCItemElements[i].szItemID;
  }

pBrowse.Browse(szItemName, ref ContinuationPoint, 0,
tagOPCBROWSEFILTER.OPC_BROWSE_FILTER_BRANCHES, "", "",
1,1,0,uiProp,out bMoreElements, out dwCount, out pElements);

tagOPCBROWSEELEMENT[] OPCBranchElements = new
tagOPCBROWSEELEMENT[dwCount];

for (int i = 0; i < dwCount; i++)
{
 OPCBranchElements[i]=(tagOPCBROWSEELEMENT)Marshal.PtrToStructure(
pBrowseElements + i * sz, typeof(tagOPCBROWSEELEMENT));

if (null == ParentNode)
 tvNode = m_treeOPCServerBrowse.Nodes.Add(OPCBranchElements[i].szName);
else
 tvNode = ParentNode.Nodes.Add(OPCBranchElements[i].szName);

DisplayChildren(tvNode, OPCBranchElements[i].szName, pBrowse);
  }
 }
```

С вызовом функции Browse могут возникнуть проблемы, описанные в предыдущем разделе, которые решаются редактированием библиотеки opcprox.dll утилитами ilDAsm и ilDAsm.

Результат работы приведен на рисунке 27

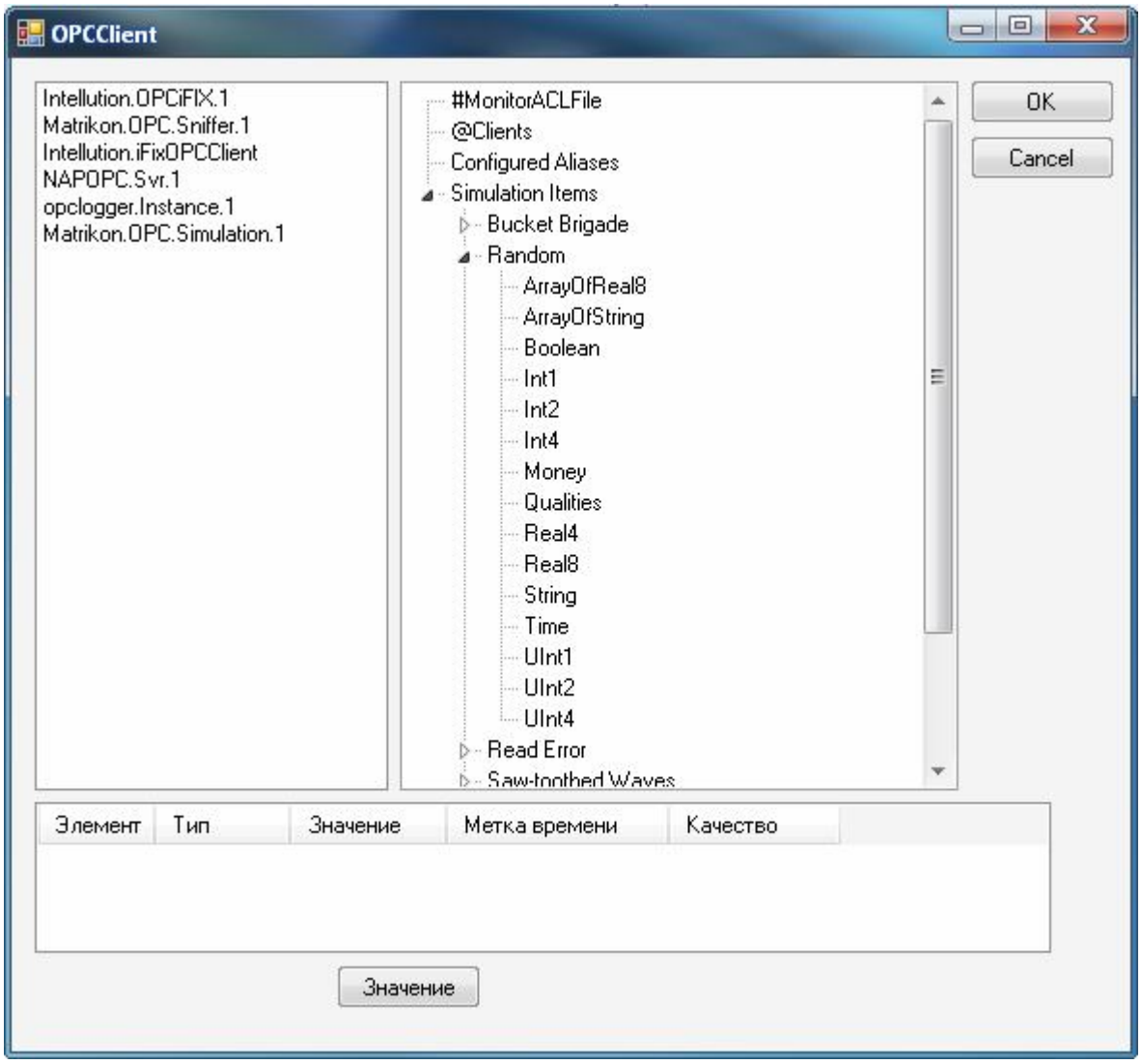


Рисунок 27. – Результат просмотра адресного пространства OPC DA 3.0 сервера посредством IOPCBrowse