

## Глава 3

# Microsoft Dynamics AX и .NET

### В этой главе

- Введение
- Использование сторонних сборок
- Написание управляемого кода
- Поддержка горячей замены для сборки на сервере

## Введение

Сложные системы, такие как **Microsoft Dynamics AX 2012**, зачастую развертываются в гетерогенных средах, где сосуществует несколько разрозненных систем. При этом нередко такие системы хранят исторические данные, которые могут понадобиться для работы Microsoft Dynamics AX, либо они предлагают функциональность, которая является жизненно важной для деятельности организации.

Microsoft Dynamics AX 2012 предлагает несколько способов интеграции с другими системами. К примеру, вашей организации может понадобиться собирать информацию из старых файлов Microsoft Excel. Для этого вы можете написать простой дополнительный компонент в Microsoft Visual Studio и легко интегрировать его с Microsoft Dynamics AX. Или же у вашей организации может быть унаследованная система, которая физически расположена на значительном удалении, из-за чего может понадобиться обеспечить поддержку отказоустойчивости при отправке ей информации об отгрузках. В этом случае вы могли бы организовать очередь сообщений для осуществления передачи данных. Вы могли бы использовать Microsoft .NET Framework для взаимодействия с очередью сообщений из Microsoft Dynamics AX.

В этой главе освещаются некоторые способы интеграции Microsoft Dynamics AX с другими системами, которые доступны вам за счет возможности использования управляемого кода из кода X++. Один из способов заключается в непосредственном использовании управляемого кода в коде X++; другой заключается в том, чтобы с помощью Visual Studio

создать новую или расширить существующую бизнес-логику, реализованную в управляемом коде. Для помощи в этом взаимодействии Microsoft Dynamics AX предоставляет компоненты управляемого кода с классами, представляющими артефакты X++ (так называемыми *проху*-классами). Благодаря этому вы можете писать управляемый код, который использует функциональность, предоставляемую этими *проху*-классами, в привычном и строго-типизированном виде.

В обоих случаях доступ к функциональности обеспечивает .NET Framework, и эта функциональность используется в Microsoft Dynamics AX.



**Примечание.** Вы также можете обеспечить доступ других систем к функциональности Microsoft Dynamics AX за счет использования сервисов. Более детальную информацию вы можете найти в главе 12.

## Использование сторонних сборок

Иногда вы можете реализовать функциональность, которую хотите обеспечить, за счет использования управляемого компонента (сборки .NET), приобретенного у сторонней компании-разработчика. Использование этих динамически подключаемых библиотек (DLLs) может – и зачастую является – более эффективным с точки зрения затрат, нежели самостоятельная разработка. Эти компоненты поставляются в виде управляемых сборок (файлов .dll) вместе с PDB-файлами, содержащими символьные данные, которые вы можете использовать при отладке, и сопутствующими XML-файлами, содержащими документацию, которая используется для работы IntelliSense в Visual Studio. Обычно эти сборки поставляются вместе с программой установки, которая, как правило, устанавливает их в глобальный кэш сборок (global assembly cache, GAC) на том компьютере, где будет использоваться соответствующая функциональность. Это может быть как клиентский компьютер, так и сервер – или оба сразу. В GAC могут быть установлены только сборки со строгими именами (strong names).

### Использование сборок со строгими именами

Всегда предпочтительнее использовать DLL со строгим именем, т.е. имеющую электронную подпись издателя, независимо от того, размещена ли

сборка в GAC или нет. Это правило одинаково применимо к сборкам, устанавливаемым как на уровне клиента, так и на уровне сервера. Строгое имя состоит из удостоверения сборки, включающего ее простое текстовое имя, номер версии и сведения о языке и региональных параметрах (если они имеются), а также открытого ключа и цифровой подписи. Если у двух сборок совпадают строгие имена, то предполагается, что такие сборки идентичны.

Строгие имена удовлетворяют следующим требованиям.

- **Уникальность строгого имени гарантируется использованием уникальных пар ключей.** Никто не сможет создать такое же имя сборки, поскольку имя сборки, созданной с использованием одного закрытого ключа, отличается от имени сборки, созданной с использованием другого закрытого ключа.
- **Защищают развитие версий сборки.** Строгое имя гарантирует, что никто другой не сможет создать следующую версию данной сборки. Пользователи могут быть уверены, что загружаемая ими версия сборки и версия, с которой было разработано приложение, произведены одним и тем же издателем.
- **Обеспечивают надежный контроль целостности.** Успешный результат при проверке безопасности платформы .NET Framework гарантирует, что содержимое сборки не было изменено после ее формирования. Однако стоит отметить, что строгие имена сами по себе не подразумевают такой же уровень доверия, который обеспечивается, например, при использовании цифровой подписи и сертификатов.

При использовании ссылки на сборку со строгим именем можно пользоваться определенными преимуществами, например, отслеживанием версий и защитой имен. Если же затем сборка со строгим именем ссылается на сборку с простым именем (которая не имеет указанных преимуществ), то преимущества использования сборки со строгим именем теряются, и опять становятся возможными конфликты DLL-библиотек. Таким образом, сборки со строгими именами могут ссылаться только на другие сборки со строгими именами.

Если сборка, которую вы используете, не обладает строгим именем и, как следствие, не установлена в GAC, вы можете вручную скопировать ее (и все другие сборки, от которых она зависит) в каталог, где .NET Framework сможет найти эту сборку, когда потребуется загрузить ее для

выполнения. Хорошей практикой является размещение сборки в том же каталоге, где находится исполняемый файл программы, который в конечном итоге загрузит ее (иными словами, в каталоге на стороне клиента или сервера, в котором находится приложение). Также вы можете разместить сборку в каталоге Client\Bin (даже если она используется исключительно на сервере), чтобы Windows-клиент мог найти ее и использовать для работы IntelliSense.

## Ссылки в Microsoft Dynamics AX на DLL-файлы управляемых сборок

В Microsoft Dynamics AX 2012 нет встроенного механизма массового развертывания или установки того или иного DLL-файла на клиентские или серверные компьютеры, потому что процесс установки у каждой сторонней DLL-библиотеки свой. Вам необходимо самостоятельно установить эти библиотеки на компьютеры с помощью скрипта установки, предоставленного издателем, или же вручную разместить сборки в соответствующих каталогах.

После установки сборки на клиентский или серверный компьютеры вам необходимо добавить ссылку на нее в Microsoft Dynamics AX, чтобы вы могли использовать эту сборку в коде X++. Это делается за счет добавления сборки в узел *References* в AOT. Щелкните правой кнопкой мыши по узлу *References* и выберите пункт меню *Добавить ссылку*. При этом появится диалоговое окно, представленное на рис. 3-1.

В верхней панели диалогового окна показаны сборки, установленные в GAC. Если ваша сборка установлена в GAC, щелкните **«Выбрать»** для добавления ссылки на нее в узел *References*. Если сборка находится в каталоге Client\Bin либо Server\Bin щелкните **Обзор**. Появится диалоговое окно открытия файла, где вы сможете выбрать вашу сборку. После того как вы выберете сборку, она появится на нижней панели, и на нее будет добавлена ссылка, когда вы нажмете ОК.

## Использование в коде X++ типов, реализованных в сборке

После добавления ссылки на сборку вы можете начать использовать ее в коде X++. Если вы установили сборку в каталог Client\Bin, для помощи в редактировании кода вам будут доступны возможности IntelliSense. Теперь вы можете использовать функции поддержки управляемого кода в X++ для создания экземпляров открытых управляемых классов, вызова их методов и т.д. Более детальную информацию вы можете найти в главе 4.

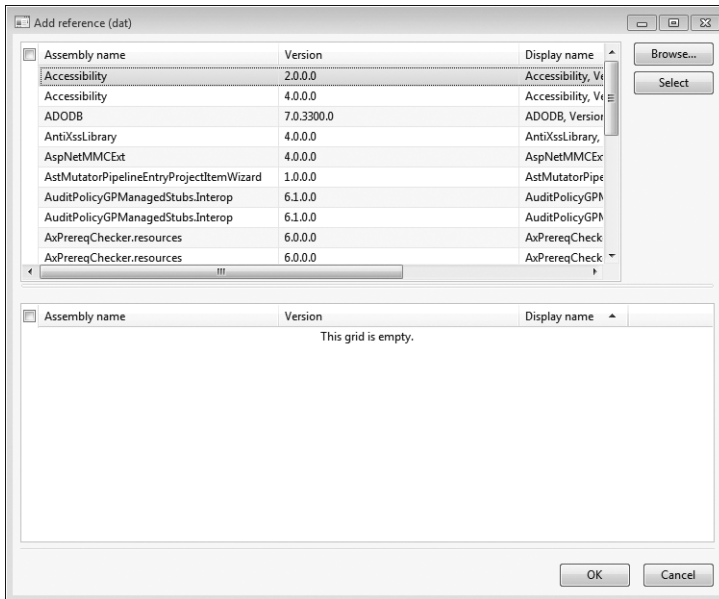


Рис. 3–1. Добавление ссылок на сторонние сборки

Стоит учесть, что есть определенные ограничения, касающиеся того, что вам доступно при вызове управляемого кода из X++. Одно из этих ограничений связано с отсутствием простого способа использования типов-дженериков (или вызова методов-дженериков). Происхождение этого ограничения связано с тем, как работает интерпретатор X++. Любой управляемый объект представляется экземпляром типа *ClrObject*, и у этого подхода есть некоторые интересные проявления. Например, рассмотрим следующий фрагмент кода:

```
static void TestClr(Args _args)
{
    if (System.Int32::Parse("0"))
    {
        print "Do not expect to get here";
    }
    pause;
}
```

Очевидно, вы не ожидаете, что код в блоке *if* будет выполнен, потому что результат вызова управляемого кода – 0, что интерпретируется как

ложь. Однако в действительности код печатает строковый литерал, потому что возвращаемое значение в вызове является экземпляром *ClrObject*, не равным null (иными словами, *истина*). Вы можете решить подобные проблемы за счет сохранения результатов в переменных перед тем, как будете их использовать: оператор присваивания корректно распакует значение, как показано в следующем примере.

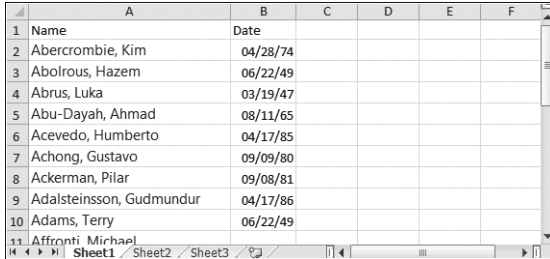
```
static void TestClr(Args _args)
{
    int i = System.Int32::Parse("0");
    if (i)
    {
        print "Do not expect to get here";
    }
    pause;
}
```

## Написание управляемого кода

Иногда существующие компоненты не могут удовлетворить ваши требования, и вам приходится засучить рукава и написать немного кода на C# или VB.NET. Microsoft Dynamics AX предоставляет для этого все условия: возможности интеграции между Microsoft Dynamics AX и Visual Studio позволяют вам обращаться с артефактами X++ (классами, таблицами и перечислениями) как с управляемыми классами, которые ведут себя так, как ожидает разработчик, пишущий управляемый код. Business Connector Microsoft Dynamics AX управляет взаимодействием между этими двумя средами. В общих чертах, вы можете создать проект в Visual Studio так же, как вы обычно это делаете, и затем добавить его в узел *Projects* (проекты Visual Studio) в AOT. В этом разделе мы подробно рассмотрим, как это сделать. В нашем примере будет показано, как написать управляемый код на C# (также можно было бы использовать VB.NET), который читает содержимое листа Excel и вставляет полученные данные в таблицу Microsoft Dynamics AX. Данный пример выбран не столько для реализации какой-то полезной функциональности, сколько для иллюстрации описанных в этой главе идей.

Процесс достаточно прост: вы создаете код в **Visual Studio**, а затем добавляете решение в Application Explorer (это – просто название для AOT в Visual Studio). Затем функционал Microsoft Dynamics AX становится доступен для использования в коде C#, что иллюстрирует возможности

проху-классов. Предположим, что файл Excel содержит названия клиентов и даты, когда они были зарегистрированы в качестве клиентов в вашей организации, как показано на рис. 3-2.



	A	B	C	D	E	F
1	Name	Date				
2	Abercrombie, Kim	04/28/74				
3	Abolrous, Hazem	06/22/49				
4	Abrus, Luka	03/19/47				
5	Abu-Dayah, Ahmad	08/11/65				
6	Acevedo, Humberto	04/17/85				
7	Achong, Gustavo	09/09/80				
8	Ackerman, Pilar	09/08/81				
9	Adalsteinsson, Gudmundur	04/17/86				
10	Adams, Terry	06/22/49				
11	Affronti, Michael					

Рис. 3-2. Лист Excel, содержащий список клиентов

Также предположим, что вы создали в AOT таблицу (скажем, *CustomersFromExcel*), в которой в конечном итоге должны оказаться данные, необходимые для дальнейшей обработки. В коде X++ читать данные из файлов Excel можно несколькими способами: один из них заключается в использовании модели OLE-автоматизации Excel; другой основан на работе с документами Office Open XML посредством классов работы с XML. Однако поскольку содержимое файлов Excel очень легко читать с помощью ADO.NET, именно на этом способе вы решили остановиться. Вы запускаете Visual Studio, создаете библиотеку классов C#, называющуюся *ReadFromExcel*, и затем пишете следующий код:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Contoso
{
    using System.Data;
    using System.Data.OleDb;
    public class ExcelReader
    {
        static public void ReadDataFromExcel(string filename)
        {
            string connectionString;
            OleDbDataAdapter adapter;
            connectionString = @"Provider=Microsoft.ACE.OLEDB.12.0;"
                + "Data Source=" + filename + ";"
                + "Extended Properties='Excel 12.0 Xml';"
```

```
+ "HDR=YES"; // Since sheet has row with column titles
adapter = new OleDbDataAdapter(
"SELECT * FROM [sheet1$]",
connectionString);
DataSet ds = new DataSet();
// Get the data from the spreadsheet:
adapter.Fill(ds, "Customers");
DataTable table = ds.Tables["Customers"];
foreach (DataRow row in table.Rows)
{
    string name = row["Name"] as string;
    DateTime d = (DateTime)row["Date"];
}
}
}
```

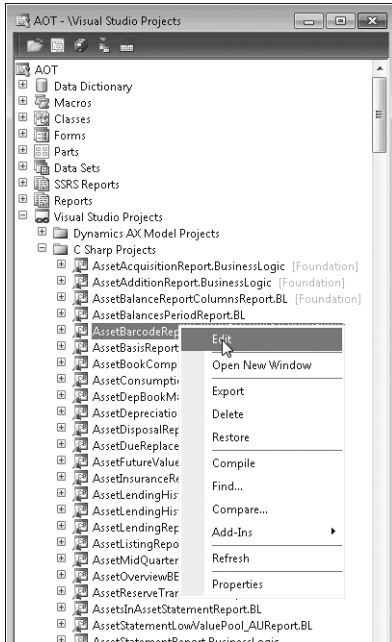
Метод *ReadDataFromExcel* читает данные из файла Excel, название которого передано в качестве параметра, но пока он ничего не делает с этими данными. Вам все еще нужно установить соединение с системой Microsoft Dynamics AX, чтобы сохранить значения в таблице. Есть несколько способов решения этой задачи, но в данном случае вы просто будете использовать таблицу Microsoft Dynamics AX из кода C# за счет возможностей проху-классов.

В качестве первого шага нужно превратить проект Visual Studio (тот, который содержит код) в одного из «граждан» Microsoft Dynamics AX. Сделайте это, выбрав пункт меню Add ReadFromExcel to AOT в контекстном меню проекта Visual Studio. Когда добавление будет завершено, проект окажется сохраненным в AOT и сможет использовать всю функциональность, которая доступна для узлов AOT. Проект может храниться в разных слоях, его можно будет импортировать и экспортировать и т.д. Проект хранится в виде отдельной сущности, и вы можете открыть Visual Studio для изменения проекта, выбрав из контекстного меню пункт Правка, как показано на рис. 3-3.



**Совет.** Вы можете определить, что проект был добавлен в AOT, по маленькой иконке Microsoft Dynamics AX в левом нижнем углу иконки проекта Visual Studio.





**Рис. 3-3.** Контекстное меню проектов Visual Studio, которые хранятся в AOT

По завершении этого шага вы можете использовать версию AOT, доступную в **Application Explorer в Visual Studio**, для **выбора таблицы**, которую вы будете использовать в коде C# (см. рис. 3-4). Если окно Application Explorer еще не открыто, вы можете открыть его, выбрав пункт Application Explorer в меню View.

Вы можете создать представление таблицы на **C#, перетащив узел таблицы** из Application Explorer в проект. После того, как вы это сделаете, то увидите узел в проекте, представляющий таблицу. Те объекты, которые вы таким образом перетащите в проект, станут доступными для использования в коде C#, как если бы они были написаны на C#. Это происходит потому, что «за кулисами» операция перетаскивания создает гроху-класс для таблицы; этот класс отвечает за всю дополнительную работу, необходимую для взаимодействия с системой Microsoft Dynamics AX, при этом предоставляя разработчику высококачественный управляемый интерфейс. Теперь вы можете продолжить разработку, добавляя в код C# недостающие фрагменты для записи данных в таблицу. Измените код так, как показано в следующем примере.

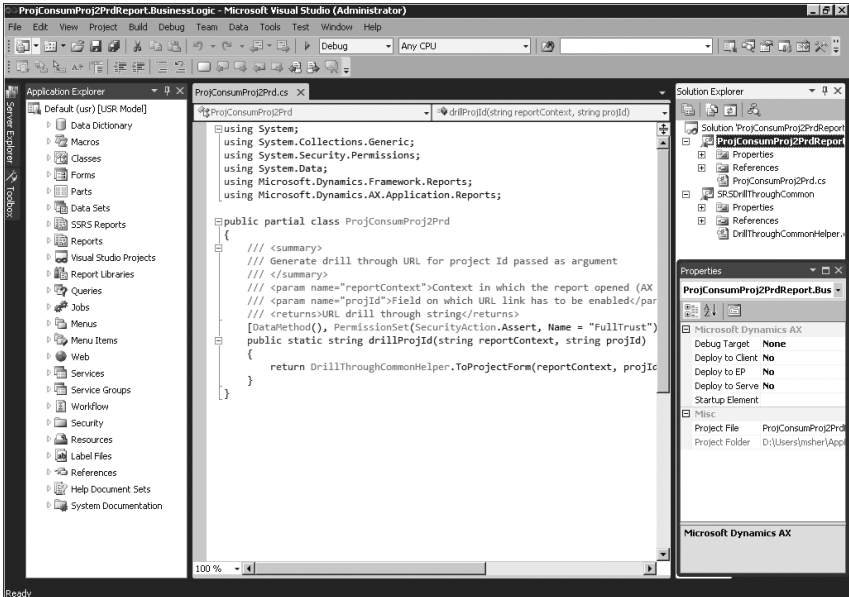


Рис. 3-4. Application Explorer с открытым проектом Microsoft Dynamics AX

```

DataTable table = ds.Tables["Customers"];
var customers = new ReadFromExcel.CustomersFromExcel();
foreach (DataRow row in table.Rows)
{
    string name = row["Name"] as string;
    DateTime d = (DateTime)row["Date"];
    customers.Name = name;
    customers.Date = d;
    customers.Write();
}

```



**Примечание.** Таблица Microsoft Dynamics AX представляется так же, как любой другой тип в C#. Она поддерживает IntelliSense, и во время редактирования кода вам будут доступны комментарии XML-документации к методам, добавленные в коде X++.

Данные будут вставляться в таблицу CustomersFromExcel по мере чтения из таблицы ADO.NET, которая представляет содержимое листа Excel. Однако прежде чем клиент или сервер смогут использовать этот код, вы

должны выполнить процедуру его развертывания. Это можно сделать, заполнив свойства в окне Properties для проекта Microsoft Dynamics AX в Visual Studio. В данном случае код будет выполняться на клиента, поэтому установите свойство *Deploy to Client* в *Yes*. Тут, впрочем, есть один тонкий момент: вы не можете проводить развертывание сборки на клиента, когда клиент запущен, поэтому вам нужно закрыть все клиенты Microsoft Dynamics AX перед развертыванием.

Чтобы выполнить развертывание, щелкните правой кнопкой по проекту Visual Studio и выберите из контекстного меню пункт *Deploy*. Если все пройдет гладко, в строке состояния появится сообщение *Deploy Succeeded*.



**Примечание.** Вам не нужно создавать ссылку на вашу сборку, поскольку ссылка неявно добавляется на проекты, которые вы добавляете в АОТ. Вам нужно создавать ссылки лишь на те сборки, которые не создаются в рамках добавленного в АОТ проекта.

Как только вы провели развертывание сборки, вы можете использовать ее в коде X++. В следующем примере показан небольшой фрагмент кода в виде задания на X++:

```
static void ReadCustomers(Args _args)
{
    ttsBegin;
    Contoso.ExcelReader::ReadDataFromExcel(@"c:\Test\customers.xlsx");
    ttsCommit;
}
```

При выполнении это задание вызывает управляемый код и вставляет записи в базу данных Microsoft Dynamics AX.

## Отладка управляемого кода

Чтобы облегчить процесс развертывания после сборки, свойства проекта Visual Studio позволяют вам задать, что должно происходить, когда вы запускаете проект Microsoft Dynamics AX. Для этого служат свойства *Debug Target* и *Startup Element*. Вы можете указать название элемента, который нужно выполнить, – обычно это класс с подходящим методом *main* либо задание (*job*). Когда вы запустите проект в Visual Studio, среда разработки запустит новый экземпляр клиента и выполнит этот класс или задание. Код X++ затем вызовет код C#, где установлены точки останова. Более детальную информацию вы можете найти в разделе «Debugging Managed

Code in Microsoft Dynamics AX» по адресу <http://msdn.microsoft.com/en-us/library/gg889265.aspx>.

Альтернативным способом является подключение отладчика Visual Studio к выполняющемуся клиенту Microsoft Dynamics AX (пункт Attach To Process в меню Debug в Visual Studio). После подключения вы можете установить точки останова и использовать все прочие возможности отладчика, как при обычной отладке. Если сервер приложений (AOS) выполняется на вашем собственном компьютере, вы также можете подключиться и к нему, но для этого у вас должны быть права локального администратора.



**Важно.** Не выполняйте отладку в рабочей среде.

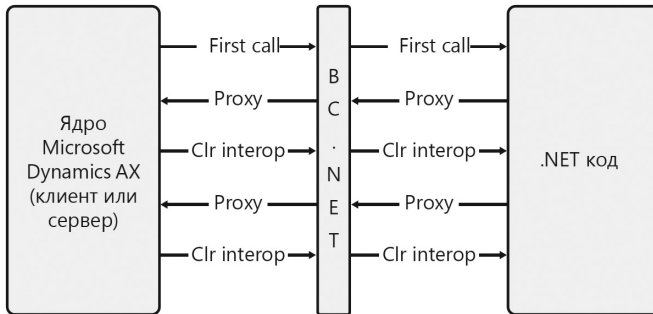
## Проку-классы

Как вы видите, управляемый код очень просто «научить» работать с Microsoft Dynamics AX за счет проку-классов, которые генерируются за кулисами для представления таблиц, перечислений и классов Microsoft Dynamics AX. Обычным делом в процессе разработки являются итерации, когда эти элементы приложения разрабатываются в Microsoft Dynamics AX, затем пишется код C#, который с ними работает, потом эти элементы изменяются в Microsoft Dynamics AX, снова пишется код C#... Этот процесс протекает без каких-либо затруднений за счет того, что Visual Studio генерирует проку-классы во время сборки проекта, поэтому они всегда синхронизированы с соответствующими артефактами в AOT; иными словами, проку-классы никогда не устаревают. В этом отношении проку-классы для объектов приложения Microsoft Dynamics AX отличаются от проку-классов Visual Studio для веб-сервисов. Предполагается, что программный интерфейс (API) этих проку-классов стабилен, поэтому не происходит обращения к серверу, на котором развернуты веб-сервисы, при каждой сборке проекта.

Проку-классы генерируются не только для тех элементов, которые вы переносите в узел *Project*, как описано выше. Например, когда генерируется проку-класс для класса, также будут сгенерированы проку-классы для всех его родительских классов, для всех артефактов, являющихся частью параметров любого метода, и т.д.

Чтобы увидеть, что представляют из себя проку-классы, наведите курсор на любое название проку-класса в редакторе кода, такое как *Customers-FromExcel* в данном примере, щелкните правой кнопкой мыши и выбери-

те пункт Go To Definition (или же просто нажмите F12). Все проху-классы хранятся в каталоге проекта Obj\Debug. Если вы посмотрите внимательно, то увидите, что они используют Business Connector Microsoft Dynamics AX для выполнения работы по взаимодействию с системой Microsoft Dynamics AX. Для поддержки такого сценария работы Business Connector был полностью переписан по сравнению с предыдущей версией; в прежних версиях Business Connector непременно создавал новую сессию, в рамках которой происходило все взаимодействие. В случае с новой версией Business Connector это не так (по крайней мере, когда он используется так, как здесь продемонстрировано). Вот почему транзакция, которая была начата в показанном ранее задании, остается открытой, когда в таблицу вставляются записи. На самом деле управляемому коду доступны все аспекты функционирования пользовательской сессии. Это ключевое отличие между реализацией бизнес-логики в управляемом коде и вызовами из управляемого кода уже реализованной бизнес-логики. Когда вы реализуете бизнес-логику, управляемый код становится расширением для кода на X++, а это означает, что работа Microsoft Dynamics AX и управляемого кода может пересекаться в рамках согласующегося окружения. Когда же вы вызываете уже реализованную бизнес-логику, вам лучше воспользоваться инфраструктурой сервисов, предоставляемой Microsoft Dynamics AX, и затем вызывать сервис из своего приложения. Это дает существенные преимущества в плане масштабируемости и гибкости развертывания. На рис. 3-5 показаны отношения Business Connector с Microsoft Dynamics AX и кодом .NET-приложения.



**Рис. 3-5.** Взаимодействие между Microsoft Dynamics AX и .NET-кодом через Business Connector

Чтобы продемонстрировать новую роль **Business Connector**, следующий пример открывает форму в клиенте, вызвавшем этот код:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace OpenFormInClient
{
    public class OpenFormClass
    {
        public void DoOpenForm(string formName)
        {
            Args a = new Args();
            a.name = formName;
            var fr = new FormRun(a);
            fr.run();
            fr.detach();
        }
    }
}
```

А в этом примере показано задание, вызывающее управляемый код для открытия формы *CustTable*:

```
static void OpenFormFromDotNet(Args _args)
{
    OpenFormInClient.OpenFormClass opener;
    opener = new OpenFormInClient.OpenFormClass();
    opener.DoOpenForm("CustTable");
}
```



**Примечание.** Класс *FormRun* в приведенном примере является классом ядра. Поскольку в Application Explorer представлены только классы приложения, вы не можете создать для него проху-класс с помощью простого перетаскивания мышью, как было описано ранее. Вместо этого перетащите любой класс из Application Explorer в проект Visual Studio и затем установите свойство имени файла класса в *Class.<kernelclassname>.axproxy*. В данном примере имя было бы *Class.FormRun.axproxy*.

Это было бы невозможно с более ранними версиями **Business Connector**, потому что они, по сути, были клиентами без графического пользовательского интерфейса. Теперь Business Connector фактически является

частью клиентского (или серверного) приложения и потому может делать все то же, что могут они. В Microsoft Dynamics AX 2012 R2 вы все еще можете использовать Business Connector как самостоятельного клиента, однако, такое его использование не рекомендуется, поскольку эту функциональность теперь лучше реализовывать с помощью сервисов (см. главу 12). Business Connector, включенный в поставку Microsoft Dynamics AX, построен на базе .NET Framework 3.5. Это означает, что проще будет создавать бизнес-логику с использованием именно этой версии .NET; если по каким-либо причинам вы не можете использовать эту версию, то нужно будет изменить файл App.config. Если вы используете программу на базе .NET Framework 4.0 и вам нужно использовать Business Connector (через проху-классы, как описано выше), вам обычно нужно будет добавлять такой блок XML в конфигурационный файл:

App.config для вашего приложения:

```
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

## Поддержка горячей замены для сборок на сервере

В предыдущем разделе было рассказано, как реализовывать бизнес-логику в управляемом коде. Для простоты был рассмотрен пример с кодом, выполняющимся на клиенте. В этом разделе описывается управляемый код, выполняющийся на сервере.

Вы указываете, что код должен выполняться на сервере, устанавливая свойство проекта *Deploy to Server* в Yes, как представлено на рис. 3-6.

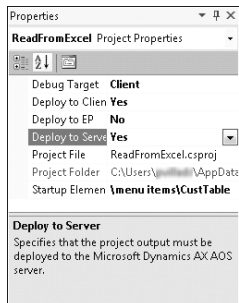
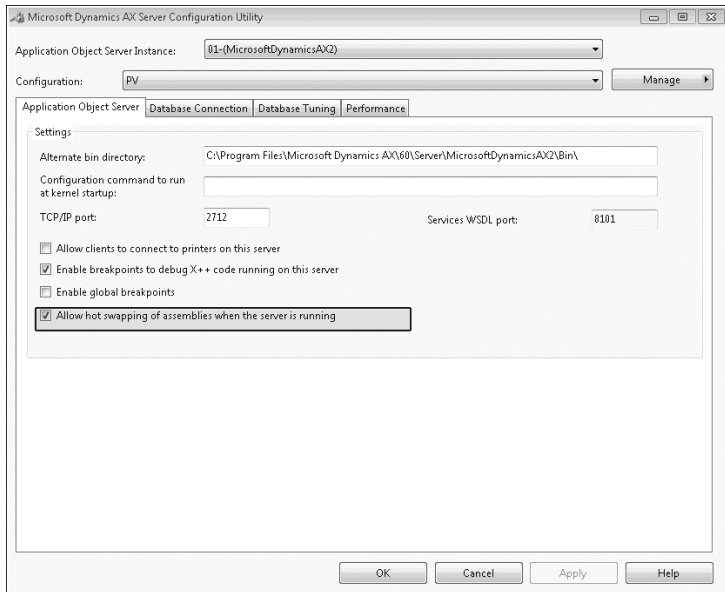


Рис. 3-6. Список свойств, где свойство Deploy to Server установлено в Yes

Когда вы устанавливаете это свойство, как показано на рис. 3-6, развертывание сборки происходит в каталог на сервере. Если сервер уже работал некоторое время, то обычно он уже загрузил сборки в текущий домен приложений. Если бы **Visual Studio** нужно было провести развертывание новой версии существующей сборки, то этот процесс закончился бы неудачно, поскольку сборка была бы уже загружена в текущий домен приложений. Во избежание таких ситуаций у сервера есть опция запуска нового домена приложений, в котором он будет выполнять код из новой сборки. Когда к серверу подключится новый клиент, будет выполняться обновленный код в новом домене приложений, в то время как уже подключенные клиенты продолжают использовать старую версию.

Для использования функции горячей замены вам нужно включить ее в конфигурационной утилите сервера **Microsoft Dynamics AX**, **взведя флажок «Разрешить горячую замену сборок во время выполнения сервера» (Allow Hot Swapping of Assemblies When The Server Is Running)**, как показано на рис. 3-7. Для запуска конфигурационной утилиты сервера **Microsoft Dynamics AX**, выберите пункт меню **Пуск, Администрирование, Microsoft Dynamics AX 2012 Server Configuration**.



**Рис. 3-7.** Разрешение горячей замены в конфигурационной утилите сервера **Microsoft Dynamics AX**





**Примечание.** Пример в предыдущем разделе показывал, как выполнять и отлаживать управляемый код на клиенте, что является безопасным, поскольку код выполняется лишь на компьютере разработчика. Вы также можете отлаживать код, выполняемый на сервере (запустив **Visual Studio** под привилегированным пользователем и подключившись к серверному процессу, как описано в разделе «Отладка управляемого кода»). Однако никогда не следует этого делать на рабочем сервере, потому что любые точки останова, которые при этом встретятся, приведут к остановке выполнения всего управляемого кода, по сути, заблокировав работу пользователей, которые подключены к этому серверу, и выполнение процессов обработки данных. Еще одна причина, по которой не следует использовать функцию горячей замены в рабочей среде, заключается в том, что вызовы кода в другом домене приложений требуют дополнительных накладных расходов и снижают общую производительность. Эта возможность предназначена только для сценариев разработки, где производительность приложения не имеет особого значения.