

# Создание СОМ-компонент "среднего слоя" в среде Microsoft Visual FoxPro

Михаил Дроздов

<http://vfpdev.narod.ru>

Группа компаний ИВС, Пермь

<http://www.ics.perm.ru>

# Содержание

- Возможности VFP в создании приложений по обработке данных
- Функциональные возможности уровня VFP-базы данных
- Схема сетевого обмена данными в традиционном VFP-приложении
- Многослойные архитектурные решения при создании приложений по обработке данных
  - Схема обращения клиента к серверной компоненте через RPC (Remote Procedure Call)
  - Схема обращения клиента к серверу через SOAP (Simple Object Access Protocol) протокол
- Шаги создания COM-компоненты в среде VFP
  - Код примера-теста VFP-COM-компоненты
  - Интерфейсы созданной нами VFP-COM-компоненты
- Шаги регистрации COM-компоненты в «Службе компонент» сервера
  - Создание COM+ приложения
  - Настройка прав доступа к COM+ приложению
  - Компоненты поддержки VFP-COM на стороне сервера
  - Регистрация VFP-COM-объекта в COM+ приложении
  - Проверка работы VFP-COM-объекта в рамках COM+ приложения
  - Экспорт VFP-COM-компонент приложения для установки на стороне клиентов
- Публикация COM-компоненты как Web Services
  - Публикация с использованием WDSL Generator из SOAP 3.0 Toolkit
  - Завершение публикации и проверка работоспособности нашего Web Service
- Перечень возможностей COM+ приложений, выполняющихся в «Службе компонент» сервера
  - Завершение COM+ приложения из VFP-кода
  - Настройки Активизации COM+ компоненты (Just-in-time activation)
    - Получение ObjectContext
    - Использование ObjectContext (явная схема)
    - Использование ObjectContext (неявная схема VFP 7.0 и выше)
    - Схема «работы» Stateless объекта
    - Некоторые сравнительные характеристики Statefull и Stateless объектов
  - Использование общих в рамках приложения данных
    - Пример кода, показывающего использование «Shared Property Manager» из «Службы Компонент»
    - Примеры кода, показывающего проверку роли клиента и контроль за флагами состояния в «Службе Компонент»
- Пример функционального разбиения VFP-приложения в трёхслойной архитектуре
- Список литературы

# Возможности VFP в создании приложений по обработке данных

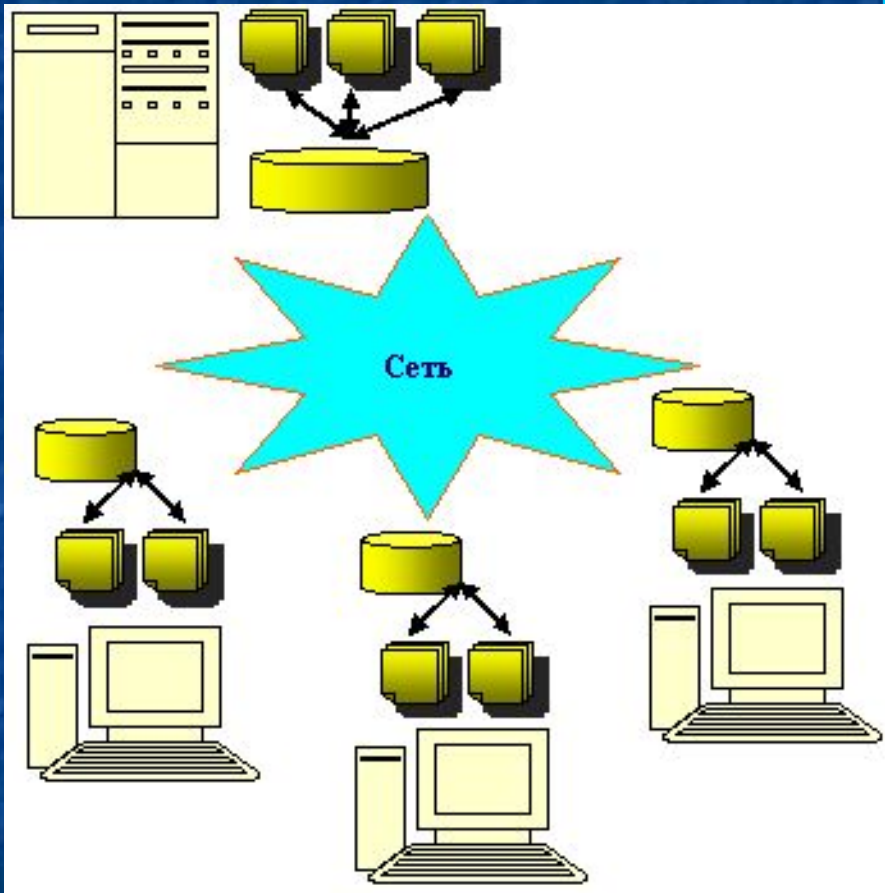
- VFP имеет «свою базу данных» с поддержкой SQL и ODBC/OLEDB драйверов к ней, что позволяет «внешним приложениям» пользоваться данными из VFP-базы данных через стандартные интерфейсы, в частности, через ADODB... Также VFP-база данных доступна в среде программирования MS VS 2003 .NET (и выше). Доступ осуществляется с использованием библиотеки MS Framework .NET (1.1 и выше).
- Специализированный на обработку данных объектно-ориентированный язык программирования позволяет минимальными усилиями создавать прикладные задачи, связанные с обработкой данных.
- VFP имеет также «объектную модель» визуальных объектов, включающую наследование и обеспечивающих создание «интерфейса пользователя» (т.е. клиентских приложений), причём все визуальные объекты легко интегрируются с источниками данных путём простого механизма присоединения к таблицам/полям данных.
- Средства программирования в VFP позволяют создавать COM-компоненты, которые могут быть использованы как VFP-приложениями, так и «внешними приложениями», допускающими использование OLE Automation объектов. Кроме того, создаваемые VFP-COM-компоненты совместимы с MTS (OS NT 4), с «Службой Компонент» (OS NT 5), также как легко могут быть опубликованы как «Web Services» для работы из-под MS IIS 5.0 (и выше)

# Функциональные возможности уровня VFP-базы данных

■ Средствами VFP-базы данных вы можете:

- хранить определения для соединений с «внешними источниками» данных
- создать часто используемые представления данных как к таблицам базы данных, так и к внешним источникам
- реализовать «общие функции» обработки данных на «хранимых процедурах»
- определить первичные ключи таблиц и установить постоянные межтабличные отношения
- определить значения по умолчанию для полей (в последних версиях ключи с автоматическим приращением значений), а также маски ввода и формат отображения данных
- осуществить контроль целостности данных
- выполнить проверку корректности данных в полях и/или в записи
- осуществить транзакционность изменений в нескольких таблицах
- организовать «следственные изменения» в логически связанных таблицах
- хранить описания полей/таблиц и «подвязанные» к полям классы, осуществляющие отображение данных на стороне клиента
- в последних версиях программируемая событийная модель позволяет программировать «административный уровень» обслуживания базы данных
- Нужно иметь ввиду и имеющуюся здесь ограничения (не все команды/функции VFP-среды поддержаны см. раздел "Unsupported Visual FoxPro Commands and Functions" в документации), в частности: не следует пытаться использовать классы (хотя формально ограничений вроде как нет) это сделает недоступным использованию VFP-базы данных через OLEDB

# Схема сетевого обмена данными в традиционном VFP-приложении



- Если источником служит VFP-база данных, то сервер используется чисто как файл-сервер.

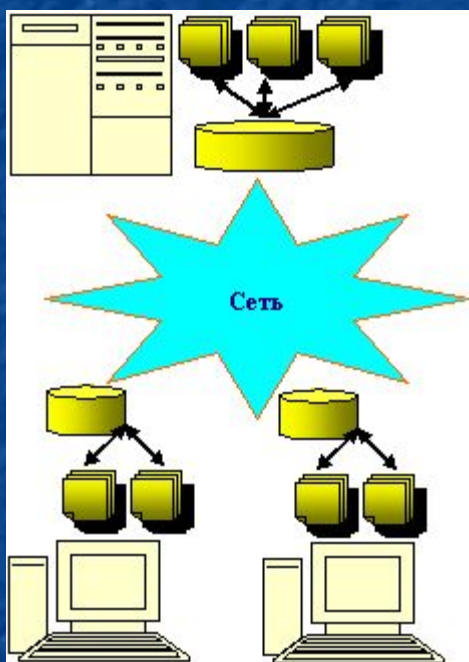
- Выполнение кода происходит только и только на машинах клиентов...

- ... при этом, все требуемые для работы данные по сети доставляются каждому из клиентов.

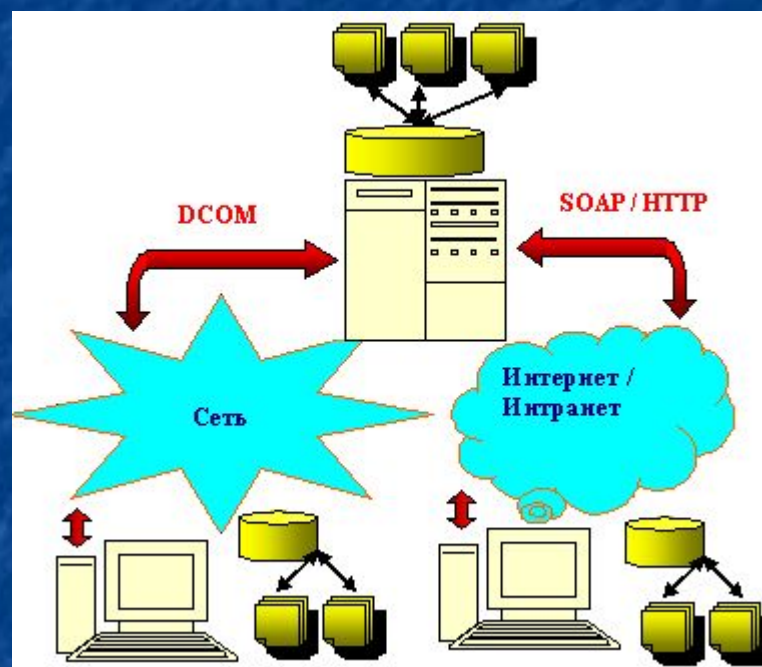
- Управление сетевым трафиком достаточно туманно... и скрыто во внутренних механизмах работы с данными среды VFP

# Многослойные архитектурные решения при создании приложений по обработке данных

Двухслойная архитектура:

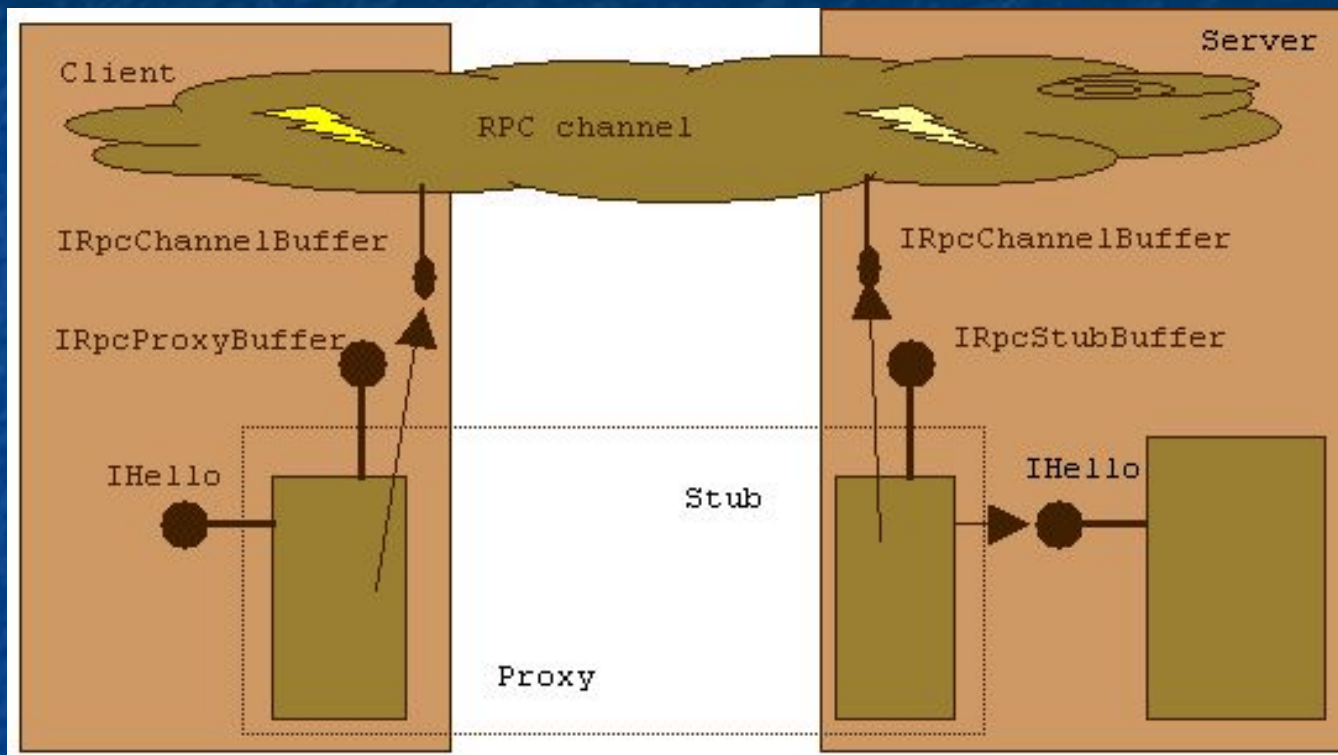


Трёх и более слоёная архитектура:



Основное отличие в том, что в двухслойной архитектуре любой клиент имеет «прямой доступ к базе данных», в то время как в трёхслойной – нет... И «доступ к данным осуществляется только и только через слой компонент» на сервере. Как следствие, управление сетевым трафиком полностью ложится на плечи разработчика.

## Схема обращения клиента к серверной компоненте через RPC (Remote Procedure Call)



Обращение клиента к серверу в локальной сети осуществляется через стандартизованный механизм удалённого вызова (RPC), который предполагает наличие посредников [заместителя (proxy) + заглушки (stub)], обеспечивающих вызов. На стороне клиента в качестве посредника используется Proxy, а на стороне сервера Stub... Так что именно за ними скрыты все тонкости/детали организации собственно самого вызова.

# Схема обращения клиента к серверу через SOAP (Simple Object Access Protocol) протокол

<http://msdn.microsoft.com/webservices/building/soaptk/>

Для передачи SOAP сообщений может быть использован любой коммуникационный протокол

**SOAP отправитель**

**SOAP получатель**

```
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  <soap:Header> <!-- необязательный -->
    <!-- здесь данные заголовка... -->
  </soap:Header>
  <soap:Body>
    <!-- здесь любая информация как содержание сообщения... -->
  </soap:Body>
</soap:Envelope>
```

Структура SOAP сообщения

Пример использования SOAP через HTTP-протокол:

```
POST /path/bank.asmx HTTP/1.1
Content-Type: text/xml
SOAPAction: "urn:banking:transfer"
Content-Length: nnnn

<soap:Envelope...
```

Запрос клиента к серверу (Request)

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<soap:Envelope...
```

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml
Content-Length: nnnn

<soap:Envelope...
```

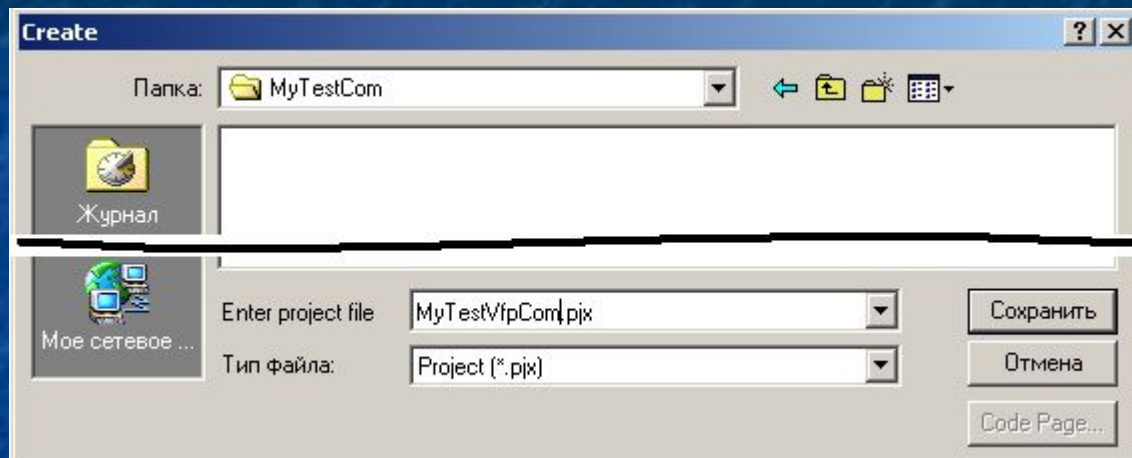
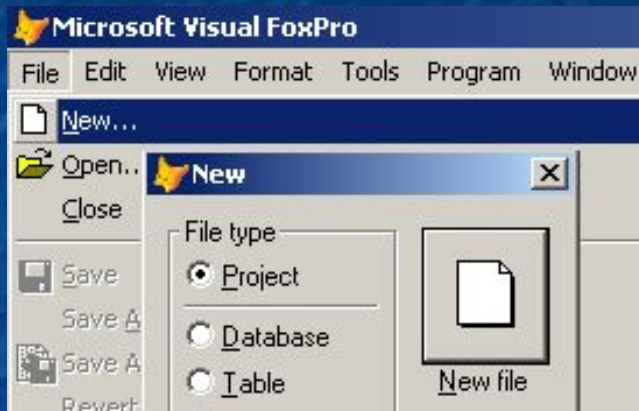
Ответ сервера клиенту (Response)



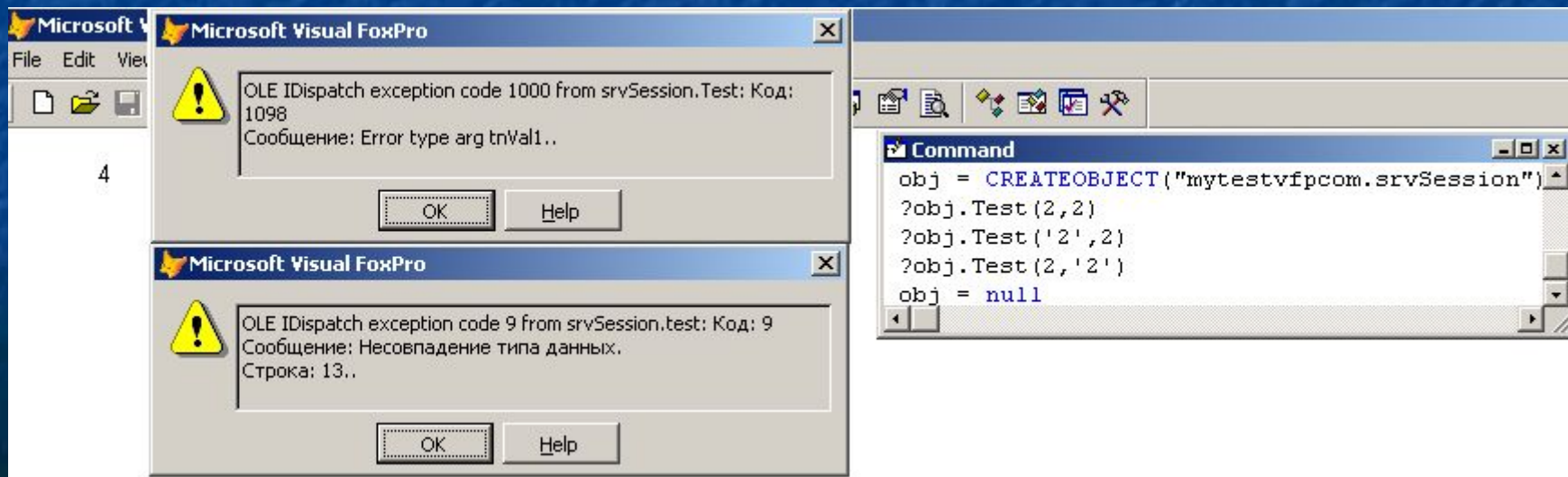
# Шаги создания COM-компоненты в среде VFP

Чтобы создать VFP-COM-компоненту, нужно проделать следующее:

- Создать новый VFP-проект:



- Поместить в него код, показанный на следующем слайде... (вы можете создавать OLEPUBLIC классы как в PRG-файлах, так и в VCX, проект может содержать более чем один класс)
- ... и откомпилировать полученное.
- Чтобы убедиться, что компонента работает, выполните из окна команд следующее:

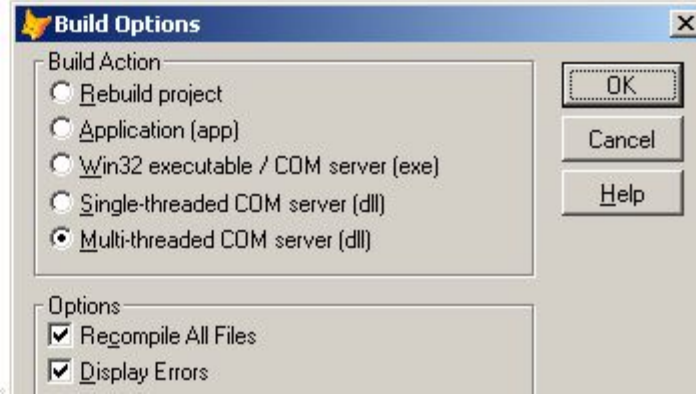
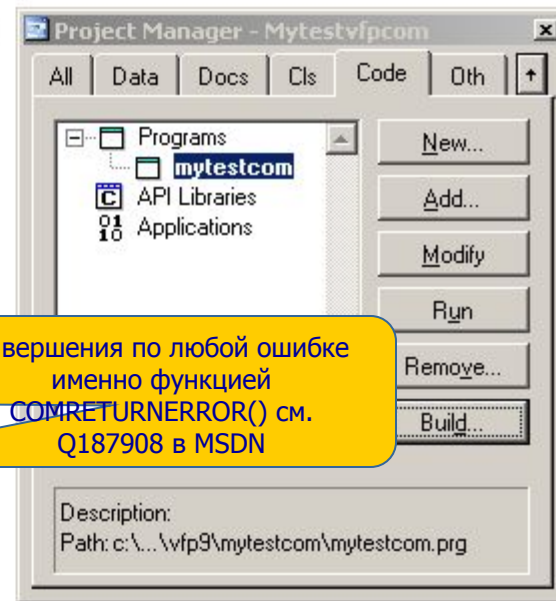


# Код примера-теста VFP-COM-компоненты

```
#DEFINE CRLF      CHR(13)+CHR(10)
#DEFINE C_ERRCODE_LOC  "Код: "
#DEFINE C_ERRMSG_LOC   "Сообщение: "
#DEFINE C_ERRLINE_LOC  "Строка: "
DEFINE CLASS srvSession as Session OLEPUBLIC
    Name = "srvSession"
    DataSession = 2
    FUNCTION Test(tnVal1 as Integer, tnVal2 as Integer) as Integer
        IF VARTYPE(tnVal1) # 'N'
            this.ShowErr(1098, 'Test', 'Error type arg tnVal1')
            RETURN 0
        ENDIF
        RETURN tnVal1*tnVal2
    ENDFUNC
    PROCEDURE Error(nError, cMethod, nLine)
        LOCAL ARRAY laErrorArray[7]
        =AERROR(laErrorArray)
        LOCAL lcMsg as String
        lcMsg = laErrorArray[2] ;
            + CRLF + C_ERRLINE_LOC + LTRIM(STR(nLine))
        this.ShowErr(nError, cMethod, lcMsg)
        NODEFAULT
    ENDPROC
    HIDDEN PROCEDURE ShowErr(tnCode, tcMethod, tcErrMsg)
        LOCAL lcErrMsg as String
        lcErrMsg = C_ERRCODE_LOC + LTRIM(STR(tnCode));
            + CRLF + C_ERRMSG_LOC + tcErrMsg
        IF !INLIST(Application.StartMode, 0, 4)
            COMRETURNERROR(IIF(!EMPTY(tcMethod), This.Name+'.'+tcMethod, This.Name+'.Error'),
                ,CRLF + lcErrMsg)
        ELSE
            MESSAGEBOX(lcErrMsg, 16, _SCREEN.Caption)
            RETURN TO MASTER
        ENDIF
    ENDPROC
    PROCEDURE Init
        IF SYS(2339) # '0'
            = SYS(2339, 0) && see Q258736 in MSDN
        ENDIF
        IF CPCURRENT() <> 1251
            = SYS(2300, 1251, 1)
        ENDIF
    ENDPROC
ENDCLASS
```

Именно Session (с DataSession=2) и OLEPUBLIC

Завершения по любой ошибке именно функцией COMRETURNERROR() см. Q187908 в MSDN



# Интерфейсы созданной нами VFP-COM-компоненты

The screenshot displays the Microsoft Visual FoxPro environment. The **Object Browser** window is open, showing the project structure for **mytestvfpcom**. Under **Interfaces (1)**, the **Isrvsession** interface is selected. The **Members of 'Isrvsession'** pane shows the **Test** method. The **Command** window contains the following code:

```
obj = CREATEOBJECT("mytestvfpcom.srvSession")
?obj.Test(2,2)
?obj.Test('2',2)
?obj.Test(2,'2')
obj = null
```

At the bottom of the Object Browser, the following text is displayed:

Method **Test**(**tnVal1** As Long, **tnVal2** As Long) As Long  
Member of [mytestvfpcom.Isrvsession](#)

Ready.

Обратите внимание: нет ничего лишнего...

# Шаги регистрации COM-компоненты в «Службе компонент» сервера (создание COM+ приложения)

00 Server

Программы  
Документы  
Настройка  
Найти

Microsoft .NET Framework SDK v1.1  
Microsoft Office  
Администрирование  
Стандартные  
Microsoft SQL Server  
Microsoft Visual FoxPro

Диспетчер служб Интернета  
Просмотр событий  
Службы компонентов  
Управление компьютером  
Настройка приложений COM+ и управление ими

Службы компонентов

Вас приветствует Мастер установки приложений COM

Установка или создание нового приложения  
Выберите, что следует сделать: установить готовое приложение или создать новое.

Корень консоли  
Службы компонентов  
Компьютеры  
My Computer  
Координатор распределен  
Приложение  
.NET  
Analy

Создать  
Приложение

Создать пустое приложение.

Вас приветствует Мастер установки приложений COM

Создание нового приложения  
Укажите имя нового приложения.

Введите имя нового приложения:  
AppTestVfp

Способ активизации

- Библиотечное приложение  
Компоненты запускаются процессом создателя объекта.
- Серверное приложение  
Компоненты запускаются выделенным серверным процессом.

Запускаемого на сервере

< Назад    Далее >    Отмена

Обратите внимание: для выполнения этих действий, вам потребуются права «локального администратора» этого сервера.

# Настройка прав доступа к COM+ приложению

Для обеспечения прав доступа к компонентам вашего COM+ приложения следует определить пользователя, от имени которого будет запускаться ваше приложение. Такой пользователь должен обладать неограниченными правами на файлы и ресурсы, используемые COM-компонентами приложения. На мой взгляд лучший вариант следующий:

- Создать «локального пользователя», например: AppRunner и придать ему права «локального администратора».
- Именно такого «локального пользователя» использовать в показанных ниже диалогах.

The image shows a composite screenshot of Windows XP installation and configuration windows. On the left, the 'Удостоверение приложения' (Application Authentication) dialog is visible, with the 'Указанный пользователь' (Specified user) option selected. The user field contains 'HOMESERVER\Michael'. In the center, the 'Службы компонентов' (Component Services) console shows the tree structure expanded to 'AppTestVfp'. A context menu is open over 'AppTestVfp', with 'Свойства' (Properties) selected. On the right, the 'Свойства: AppTestVfp' (Properties: AppTestVfp) dialog is shown, with the 'Указанный пользователь' (Specified user) option selected. The user field contains 'HOMESERVER\Michael'. Below this, a 'Выбор: Пользователь или Группа' (Select User or Group) dialog is open, showing a search for 'HOMESERVER' and a list of users including 'HOMESERVER' and 'WORKGROUP'. Three yellow callout boxes provide instructions: one points to the 'Свойства' menu item, another points to the user selection dialog, and a third points to the user field in the 'Свойства' dialog.

Вас приветствует Мастер установки приложения

Выбор: Пользователь или Группа

Удостоверение приложения  
Выберите учетную запись для запуска приложения

Учетная запись  
Приложение будет связано с указанной учетной записью. Если приложение будет выполняться от нескольких приложений, это будет означать, что приложение будет выполняться от имени выбранной записи.

Текущий (вошедший в систему) пользователь

Указанный пользователь:

Пользователь: HOMESERVER\Michael

Пароль: xxxxxxx

Подтверждение: xxxxxxx

Службы компонентов

Переустановите здесь «локального пользователя», обладающего правами «локального администратора», если вы получили проблемы ограниченных прав

Экспорт...  
Запустить  
Завершить работу  
Вид  
Новое окно отсюда  
Удалить  
Свойства  
Справка

Свойства: AppTestVfp

Очереди  
Дополнительно  
Безопасность  
Удостоверение

Выполняться от имени выбранной записи.

Текущий (вошедший в систему) пользователь

Указанный пользователь:

Пользователь: HOMESERVER\Michael

Пароль:

Подтверждение:

Выбор: Пользователь или Группа

Искать в: HOMESERVER

Имя  
HOMESERVER  
WORKGROUP

Все  
Прошедшие проверку  
АНОНИМНЫЙ ВХОД  
ПАКЕТНЫЕ ФАЙЛЫ  
СОЗДАТЕЛЬ-ВЛАДЕЛЕЦ  
ГРУППА-СОЗДАТЕЛЬ

Имя:

Введите здесь «локального пользователя», обладающего правами «локального администратора»

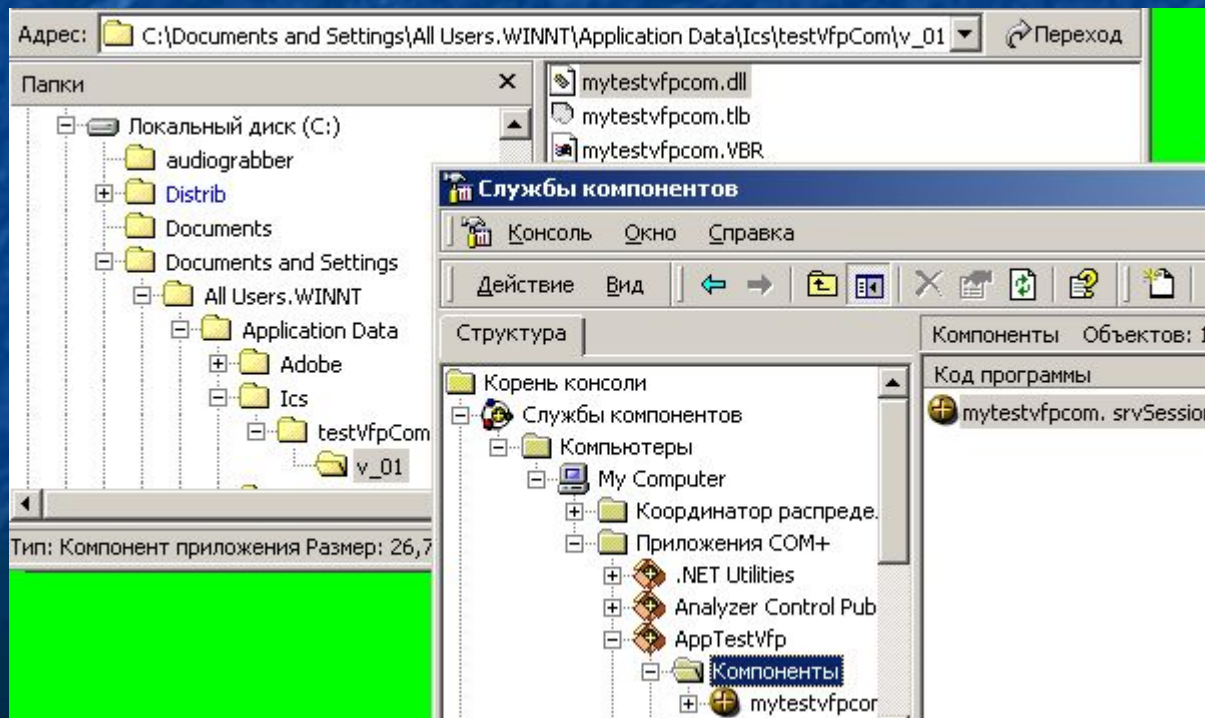
# Компоненты поддержки VFP-COM на стороне сервера

Прежде чем пытаться зарегистрировать вашу VFP-COM-компоненту, обратите внимание на то, что должно быть установлено на сервере:

- Для успешной регистрации VFP-COM-компоненты на сервере, требуется установка следующих библиотек (см. в C:\Program Files\Common Files\Microsoft Shared\VFP\): vfp<N>t.dll [,vfp<N>r.dll], vfp<N> renu.dll [,vfp<N>rrus.dll], [%windir%\system32\GDIPlus.dll, MSVCR70.dll, здесь <N> - номер версии
- Если вы используете какие-либо дополнительные VFP-средства, то полный список файлов VFP-runtime можно найти на <http://fox.wikis.com/> точнее, в зависимости от версий это:
  - в MSDN: Q190869 INFO: Visual FoxPro 6.0 Required Run-Time Files – VFP 6
  - <http://fox.wikis.com/wc.dll?Wiki~VFP7RuntimeFiles~VFP> – VFP 7
  - <http://fox.wikis.com/wc.dll?Wiki~VFP8RuntimeFiles~VFP> – VFP 8
  - <http://fox.wikis.com/wc.dll?Wiki~VFP9RuntimeFiles~VFP> – VFP 9
- Если вы используете только VFP OLEDB, то установка VFP-runtime вообще говоря не требуется, однако необходимо установить vfpoledb.dll (см. в C:\Program Files\Common Files\System\Ole DB). Последнюю версию инсталляционного пакета можно скачать с <http://msdn.microsoft.com/vfoxpro/downloads/updates/default.aspx>
- В VFP 9 появилась функция SYS(3101 [, nCodePage]) для установки требуемой «кодированной страницы», однако, если у вас установлены VFP-runtime версии младше 9, то здесь имеются проблемы (т.е. того, что возвращает функция CPCURRENT()), как исправить положение дел в зависимости от версий описано здесь:
  - [http://vfpdev.narod.ru/docs/mtskom\\_r.html#cpsolve7](http://vfpdev.narod.ru/docs/mtskom_r.html#cpsolve7) – для vfp6t.dll, vfp7t.dll, vfp8t.dll
  - [http://vfpdev.narod.ru/docs/spcall\\_r.html#cpsolve](http://vfpdev.narod.ru/docs/spcall_r.html#cpsolve) – для vfpoledb.dll (8.0.0.3006, 8.0.0.3117)
- Для создания пакета установки в VFP 6.0 можно воспользоваться помощником (меню: tools/wizards/setup), а в VFP 7-9 приложением InstallShield Express
- Установки следует производить только обладая правами «локального администратора» сервера.

# Регистрация VFP-COM-объекта в COM+ приложении

- Собственно сама регистрация созданной в VFP многопоточной COM dll-ки тривиальна: достаточно из-под «Проводника Windows» зацепив мышкой «перетащить и отпустить» её в папку «Компоненты» нашего пока ещё пустого COM+ приложения с названием AppTestVfp.
- На слайде ниже показан результат этого действия.
- Обратите внимание на то: где именно на сервере расположена наша mytestvfpcom.dll, созданная как VFP-COM-компонента.

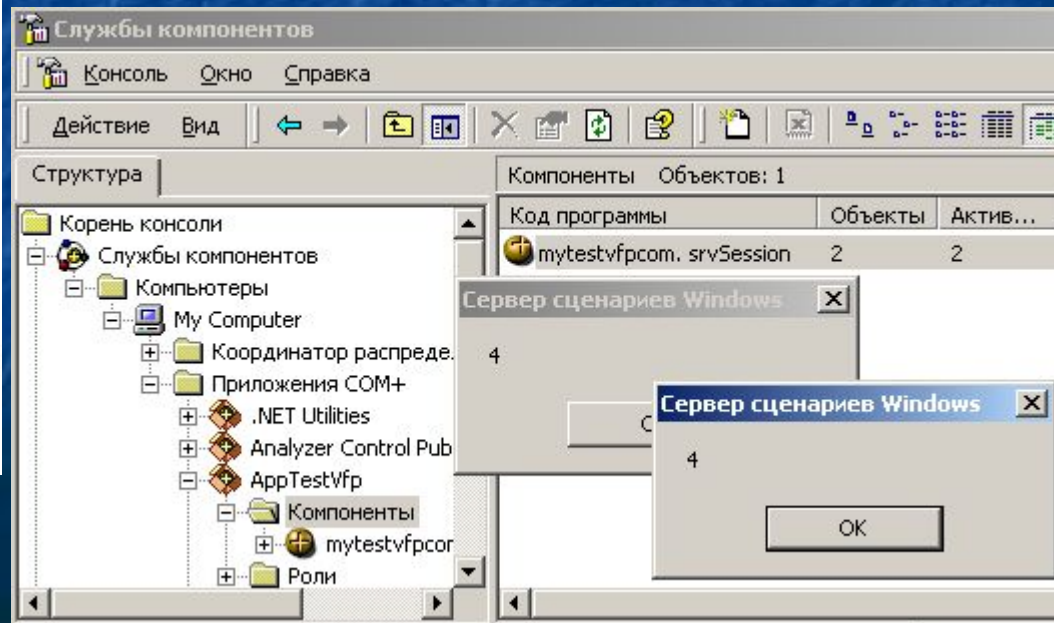


# Проверка работы VFP-COM-объекта в рамках COM+ приложения

```
mytestvfpcom.js
////////////////////////////////////
// file: mytestvfpcom.js
var obj = null;
var sProgID = "myTestVfpCom.srvSession";
var bOk = false;
try
{
    obj = new ActiveXObject(sProgID);
    var nRetVal = obj.test(2,2);
    WScript.Echo(nRetVal);
    bOk = true;
}
catch(err)
{
    WScript.Echo("Ошибка в '" + sProgID + "'"
        + "\nКод: " + hex(err.number)
        + "\nСообщение: " + err.description);
}
obj = null;

function hex(nmb)
{
    if (nmb > 0)
        return nmb.toString(16);
    else
        return (nmb + 0x100000000).toString(16);
}
// the end of file: mytestvfpcom.js
////////////////////////////////////
```

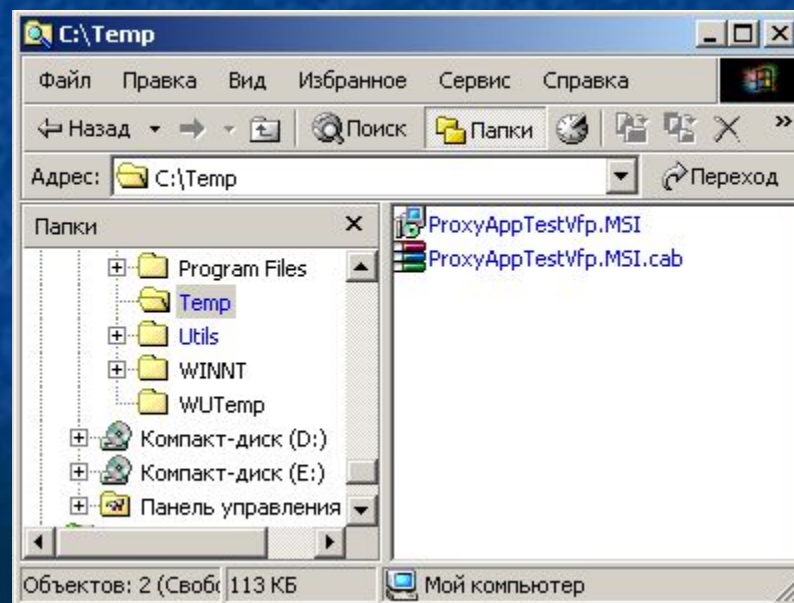
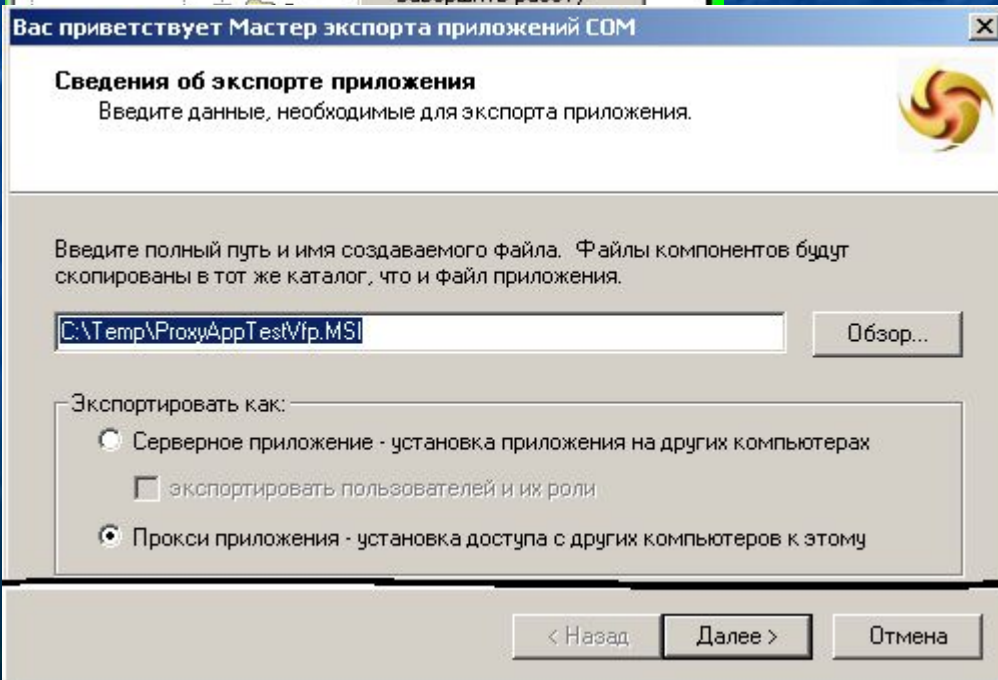
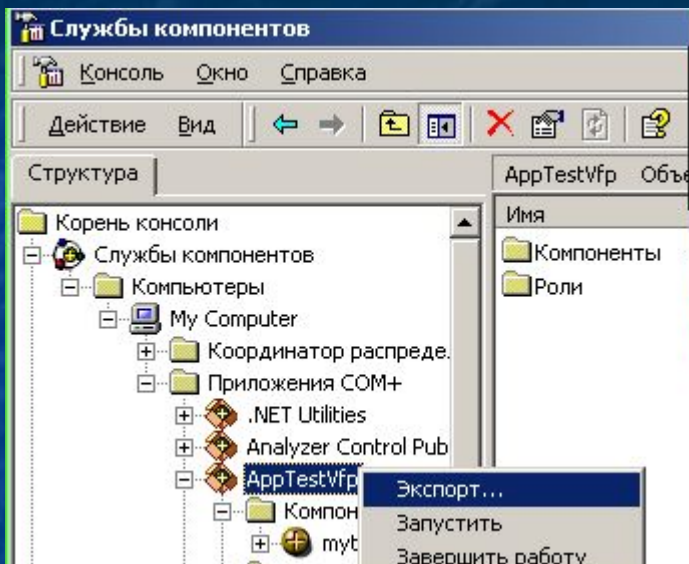
- Здесь слева приведён JavaScript-код обращения к нашей тестовой VFP-COM-компоненте.
- Если этот код выполнить непосредственно на сервере, то при успешном создании экземпляра нашего объекта, в то время пока диалог показа результата ещё не закрыт, в окне «Службы компонент» мы можем наблюдать «активность» нашей тестовой компоненты:





# Экспорт VFP-COM-компонент приложения для установки на стороне клиентов

- Наконец, последним шагом является получение инсталляционных файлов нашей VFP-COM-компоненты, чтобы установить последние на всех потенциальных клиентах.
- Благодаря «Службе компонент» этот шаг существенно облегчен, однако предполагается, что:
  - На клиентах установлен как минимум Windows Installer 2.0
  - Также как и VFP-runtime, включая vfp<N>t.dll
- Также очевидно, что это шаг следует делать только в том случае, если вашими клиентами являются VFP (или другие) приложения, работающие в локальной сети через DCOM, в противном случае (при работе через SOAP и/или через HTTP) необходимость в этом шаге отпадает.



# Публикация COM-компоненты как Web Services с использованием WSDL Generator из SOAP Toolkit 3.0

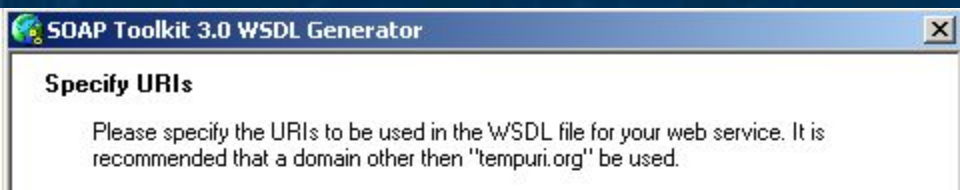
Путь и файл не должны содержать русские буквы, только ANCI символы

The image shows a sequence of four dialog boxes from the SOAP Toolkit 3.0 WSDL Generator application:

- Dialog 1:** "Select the COM .dll file to analyze." The user has entered "mytestvfpcom" as the service name and "\lcs\testVfpCom\y\_01\mytestvfpcom.dll" as the local path.
- Dialog 2:** "Select the services you would like to expose." The user has selected "mytestvfpcom", "srvSession", and "Test".
- Dialog 3:** "Specify the location for the new WSDL and WSML files." The user has selected "UTF-8" for the character set and "c:\inetpub\WebServices\mytestvfpcom" for the storage location.
- Dialog 4:** "SOAP listener information." The user has specified the listener URI as "http://localhost/webservices/mytestvfpcom/" and selected "ISAPI" as the listener type.

The application interface includes standard navigation buttons: "About", "Cancel", "< Back", "Next >", and "Finish".

# Завершение публикации и проверка работоспособности нашего Web Service



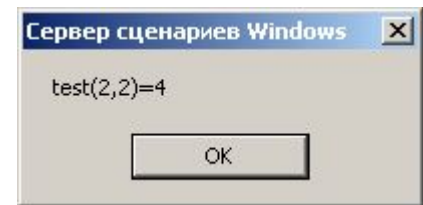
WSDL Namespace URI:

Schema Type Definition Namespace:

Message Namespace URI:

Base SOAP Action URI (will have me

```
mytestvfpcomClient.js
////////////////////////////////////
// file: mytestvfpcomClient.js
//
var oSOAPClient = null;
var sProgID = "MSOSOAP.SoapClient30";
try
{
    oSOAPClient = new ActiveXObject(sProgID);
    // See: <service ... <port ... /></service> in your WSDL-file
    oSOAPClient.MSSoapInit("http://localhost/WebServices/mytestvfpcom/mytestvfpcom.WSDL",
        "mytestvfpcom", "srvSessionSoapPort");
    var nRetVal = oSOAPClient.test(2,2);
    WScript.Echo("test(2,2)=" + nRetVal);
}
catch(err)
{
    WScript.Echo("Error in '" + sProgID + "'"
        + "\nCode: " + hex(err.number)
        + "\nDescription: " + err.description);
}
oSOAPClient = null;
function hex(nmb)
{
    if (nmb > 0)
        return nmb.toString(16);
    else
        return (nmb + 0x100000000).toString(16);
}
//
// the end of file mytestvfpcomClient.js
////////////////////////////////////
```



## Перечень возможностей СОМ+ приложений, выполняющихся в «Службе компонент» сервера

- При работе СОМ-компонент на стороне сервера возникает целая серия общих для этого случая задач, таких как: вынесение отдельных задач в отдельные потоки для увеличения производительности системы в целом, контроль доступа к объектам, и т.п.
- СОМ-компонентами, работающими в рамках «Службы компонент» предоставляется целая серия дополнительных возможностей, к ним в частности относятся:
  - активизация на время выполнения (just-in-time activation)
  - общие и совместно используемые наборы свойств (shared properties)
  - ролевая система безопасности (role-based security)
  - поддержка транзакционности изменений (transactions)
  - поддержка асинхронных очередей сообщений (queued components)
  - обеспечение пула готовых к немедленному выполнению объектов (object pooling)
  - и др.

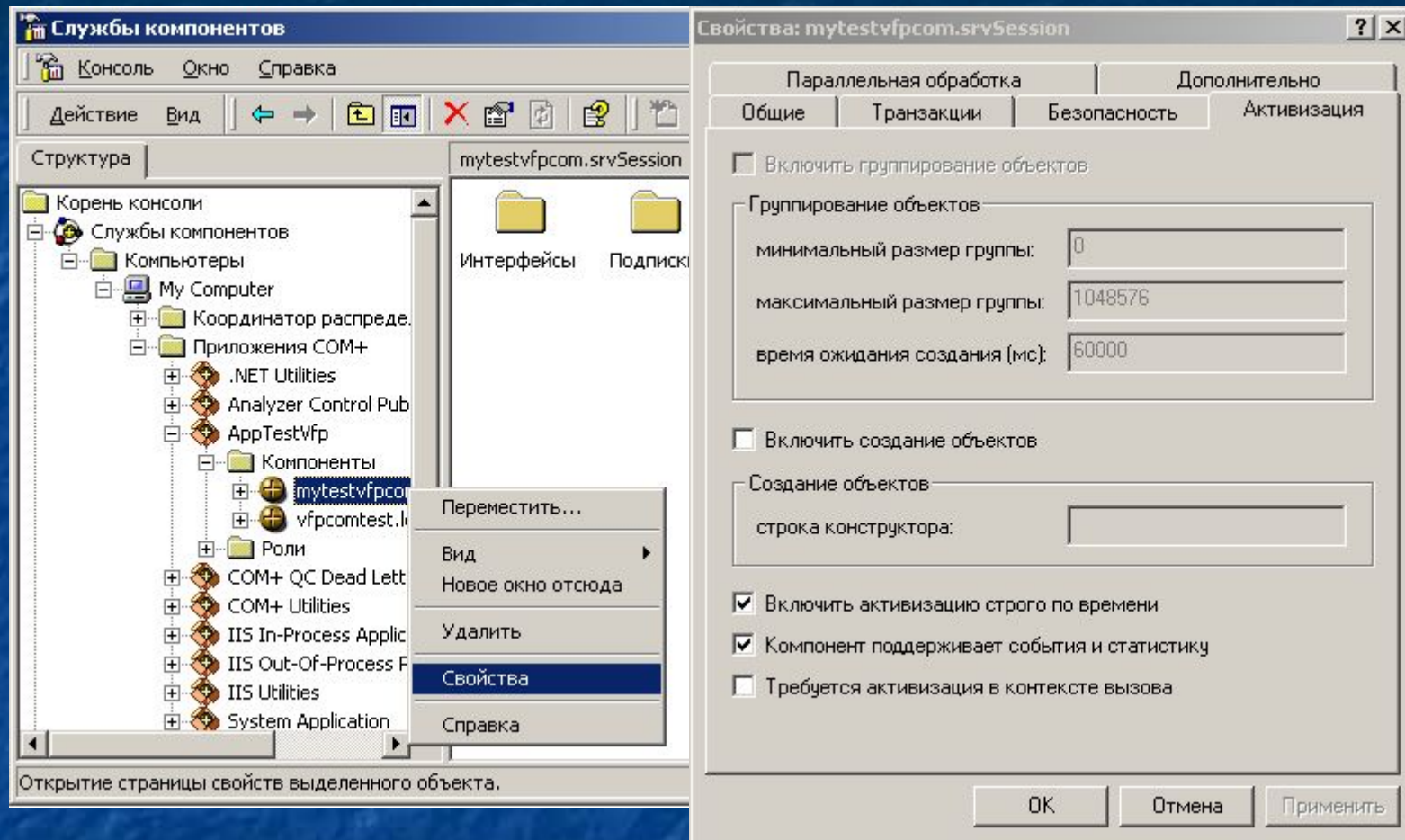
Ниже мы рассмотрим лишь некоторые из них...

# Завершение COM+ приложения из VFP-кода

```
Object: projhook Procedure: BeforeBuild View Parent Code
LPARAMETERS cOutputName, nBuildAction, lRebuildAll, lShowErrors, lBuildNewGuids
#DEFINE MTS_CATALOG "MTSAdmin.Catalog.1"
#DEFINE MSG_MTSCHECK_LOC "Shutting down MTS servers...."
LOCAL oCatalog, oPackages, oUtil, i, j, oComps
LOCAL oProject, lnServers, laProgIds, lcSaveExact
this.lBuildNewGuids = lBuildNewGuids
oProject = _VFP.ActiveProject
lnServers = oProject.Servers.Count
DIMENSION this.aServerInfo[1]
STORE "" TO this.aServerInfo
IF lnServers = 0 OR nBuildAction # 4
    RETURN
ENDIF
WAIT WINDOW MSG_MTSCHECK_LOC NOWAIT
DIMENSION laProgIds[lnServers,3]
FOR i = 1 TO lnServers
    laProgIds[m,i,1] = oProject.servers[m,i].progID
    laProgIds[m,i,2] = oProject.servers[m,i].CLSID
    laProgIds[m,i,3] = this.GetLocalServer(laProgIds[m,i,2])
ENDFOR
ACOPY(laProgIds, this.aServerInfo)
oCatalog = CreateObject(MTS_CATALOG)
oPackages = oCatalog.GetCollection("Packages")
oUtil = oPackages.GetUtilInterface
oPackages.Populate()
lcSaveExact = SET("EXACT")
SET EXACT ON
FOR i = 0 TO oPackages.Count - 1
    oComps = oPackages.GetCollection("ComponentsInPackage",;
        oPackages.Item(m,i).Key)
    oComps.Populate()
    FOR j = 0 TO oComps.Count-1
        IF ASCAN(laProgIds,oComps.Item(m,j).Value("ProgID")) # 0
            oUtil.ShutdownPackage(oPackages.Item(m,i).Value("ID"))
            EXIT
        ENDIF
    ENDFOR
ENDFOR
ENDFOR
WAIT CLEAR
```

- Код приведённый здесь и помещённый в событие BeforeBuild класса ProjectHook, который вы должны подключить к вашему проекту, обеспечит вам автоматическое завершение COM+ приложения при перекомпиляции вашего проекта. Т.е. перед подменой прежней dll-библиотеки на новую. Это видимо первое, с чем вам придётся столкнуться на этом пути... См. [1] в «Списке литературы».

# Настройки Активизации COM+ компоненты (Just-in-time activation)



- По умолчанию поддержка JIT активизации компоненты включена
- Однако, она действительно необходима только в случае поддержки транзакций для ваших компонент

# ПолучениеObjectContext

```
#DEFINE MTXAPPSRV      "MTxAS.AppServer.1"
#DEFINE DATA_SESSION 2
DEFINE CLASS baseSrvCls As Session
    name = "baseSrvCls"
    DataSession = DATA_SESSION
    HIDDEN oMtx && as COMSVCSLib.IMTxAS
    HIDDEN oCtx && as COMSVCSLib.ObjectContext
    this.oMtx = NULL
    this.oCtx = NULL
    PROTECTED FUNCTION GetContextObject
        WITH this
            .oMtx = CreateObject(MTXAPPSRV)
            IF VARTYPE(.oMtx) # 'O'
                .DoError(1098, .Name+'.GetContextObject', LINENO();
                    , "Can not create '"+MTXAPPSRV+"' object.")
                RETURN .F.
            ENDIF
            IF !INLIST(Application.StartMode, 0, 4)
                *
                *-- Get ObjectContext
                .oCtx = .oMtx.GetObjectContext()
                IF VARTYPE(.oCtx) # 'O'
                    .DoError(1098, .Name+'.GetContextObject', LINENO();
                        , "Can not GetObjectContext().")
                    RETURN .F.
                ENDIF
            ENDIF
        ENDWITH
    ENDFUNC
    PROTECTED FUNCTION EndContextObject(tbSuccessful)
        WITH this
            IF VARTYPE(.oCtx) = 'O'
                IF tbSuccessful
                    .oCtx.SetComplete()
                ELSE
                    .oCtx.SetAbort()
                ENDIF
                .oCtx = NULL
            ENDIF
        ENDWITH
    ENDFUNC
```

# ИспользованиеObjectContext (явная схема)

```
FUNCTION AnyTopLevelClientMethod(ListParameters)
#IF STATELESS
    *
    *-- Get Context obbject
    IF !this.GetContextObject()
        RETURN .F.
    ENDIF
#ENDIF
    LOCAL lbRetVal as Boolean
    *
    *-- Code...
    * ...
    *lbRetVal = ...
#IF STATELESS
    this.EndContextObject(lbRetVal)
#ENDIF
    ENDFUNC
    PROCEDURE Error(nError, cMethod, nLine)
        RETURN this.OnError(nError, cMethod, nLine)
    ENDPROC
    HIDDEN PROCEDURE DoError(nError, cMethod, nLine, lcContext)
        *
        *-- Code ...
        *
        *lcErrMsg = ...
        *
        *-- See Q262038
        IF NOT ("init" $ LOWER(cMethod))
            IF !INLIST(Application.StartMode, 0, 4)
                LOCAL lcMethod as String
                lcMethod = IIF(!EMPTY(cMethod), cMethod, This.Name+'.Error')
#IF STATELESS
                this.EndContextObject(.F.)
#ENDIF
                COMRETURNERROR(lcMethod, lcErrMsg)
            ELSE
                MessageBox(lcErrMsg, 16, _SCREEN.Caption)
                RETURN TO MASTER
            ENDIF
        ENDIF
    ENDPROC
```

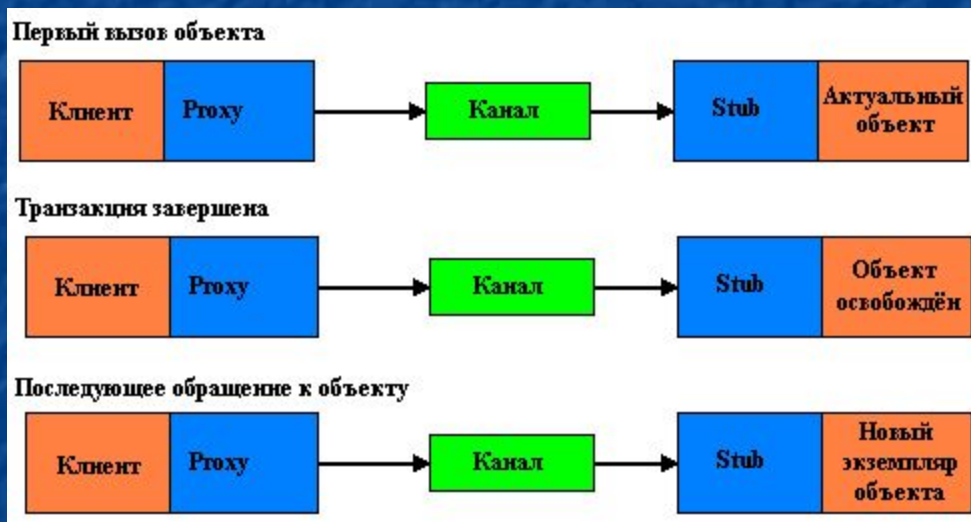


# ИспользованиеObjectContext (неявная схема VFP 7.0 и выше)

```
DEFINE CLASS SrvCls AS Session OLEPUBLIC
    PROTECTED oCtx
    this.oCtx = NULL
    IMPLEMENTS ObjectControl EXCLUDE IN "c:\winnt\system32\comsvcs.dll"
    PROTECTED PROCEDURE ObjectControl_Activate() AS VOID;
        HELPSTRING "Called when this object is Activated"
        This.GetObjectContext()
    ENDPROC
    PROTECTED PROCEDURE ObjectControl_Deactivate() AS VOID;
        HELPSTRING "Called when this object is Deactivated"
        This.oCtx = NULL
    ENDPROC
    PROTECTED PROCEDURE ObjectControl_CanBePooled() AS LOGICAL ;
        HELPSTRING "Called when deactivated to see if this object can be pooled."
        RETURN .F.
    ENDPROC
    PROTECTED FUNCTION GetObjectContext()
        *
        *-- See Q193295 in MSDN
        IF VARTYPE(This.oCtx) # 'O'
            LOCAL loMtx
            loMtx = CREATEOBJECT(MTXAPPSRV)
            IF VARTYPE(loMtx) = 'O'
                This.oCtx = loMtx.GetObjectContext()
            ENDIF
        ENDIF
        RETURN (VARTYPE(This.oCtx) = 'O')
    ENDFUNC
    PROTECTED FUNCTION SetComplete(tbResult)
        IF VARTYPE(This.oCtx) = 'O'
            IF tbResult
                This.oCtx.SetComplete()
            ELSE
                This.oCtx.SetAbort()
            ENDIF
            RETURN .T.
        ELSE
            RETURN .F.
        ENDIF
    ENDFUNC
    PROCEDURE Init()
```

# Схема «работы» Stateless объекта

```
LOCAL loRefToServerObject  
loRefToServerObject = CreateObject("MyLib.MyClass")  
loRefToServerObject.MyAnyMethod1(...)  
loRefToServerObject.MyAnyMethod2(...)
```



См. [3] в «Списке литературы».

# Некоторые сравнительные характеристики Statefull и Stateless объектов

Характеристика	Statefull	Stateless
Время жизни	Между серверными экземплярами объектов и клиентами установлено взаимно-однозначное соответствие. Каждый серверный объект создаётся конкретным клиентом и каждый из них существует так долго, сколько в нём имеется необходимость этого конкретного клиента.	Один и тот же экземпляр серверного объекта может обслуживать сразу нескольких клиентов и может быть уничтожен сервером, как только необходимость в нём исчезает.
Владение объектом	Поскольку каждый экземпляр объекта имеет уникальную информацию в виде значений своих свойств, то он может быть использован только одним конкретным клиентом-владельцем и никем другим.	Т.к. любой объект не имеет свойств, то не содержит никакой уникальной информации от клиента, и может быть использован сервером повторно, реализуя т. н. концепцию «пула объектов» (object pulling) [к сожалению, в VFP пока не поддержан]
Ресурсы	Для $n$ одновременно работающих клиентов с $m$ объектами объём требуемых на сервере ресурсов – $n*m$	В аналогичных условиях объём требуемых на сервере ресурсов $\leq n*m$

COM+ допускает использование как одного таки и другого вариантов.

# Использование общих в рамках приложения данных

- В многопоточном варианте использования компонент (multi-threaded dll) существует проблема доступа к общим ресурсам для компонент выполняющихся в разных потоках одного и того же процесса. Чтобы в этих условиях обеспечить корректность данных (не потерять изменения и не получать непредсказуемые значения), в VFP реализовано ряд решений:
  - Чтобы «инкапсулировать» работу с таблицами данных в рамках экземпляра класса, в качестве базового следует использовать класс Session с DataSession=2 (см. в MSDN Q193953 PRB: COM.Dll In MTS 'Sharing' Data sessions Between Instances)
  - Относительно использования глобальных переменных в нужно иметь ввиду следующее:
    - во-первых, - для глобальных переменных в каждом потоке VFP образует собственный apartment, где содержит локальную копию данных (thread-local storage). Другими словами переменные, объявленные как PUBLIC, в действительности глобальны только по отношению к потоку.
    - во-вторых, - участки соответствующего кода можно поместить в критические секции, применяя VFP-функцию SYS(2336 [, nAction])
    - наконец, - можно использовать «Share Properties Manager» «Службы компонент», где применены semaphores и locks в этих целях.

# Пример кода, показывающего использование «Shared Property Manager» из «Службы Компонент»

```
#DEFINE MTX_CLASS "MTXAS.APPSERVER.1"
#DEFINE MTX_SHAREDPROPGRPMGR "MTxSpm.SharedPropertyGroupManager.1"
PROCEDURE GetCount (lReset)
    LOCAL oCount
    LOCAL oMTX, oContext
    LOCAL nIsolationMode, nReleaseMode, lExists
    oMTX = CREATEOBJECT(MTX_CLASS)
    oContext = oMTX.GetObjectContext()
    oSGM = oContext.CreateInstance(MTX_SHAREDPROPGRPMGR)
    nIsolationMode = 0
    nReleaseMode = 1
    *
    * Get group reference in which property is contained
    oSG = oSGM.CreatePropertyGroup("CounterGroup", nIsolationMode,
        nReleaseMode, @lExists)
    *
    * Get object reference to shared property
    oCount = oSG.CreateProperty("nCount", @lExists)
    *
    * Check if property already exists otherwise reset
    IF lReset OR !lExists
        oCount.Value = 1
    ELSE
        oCount.Value = oCount.Value + 1
    ENDIF
    RETURN oCount.Value
ENDPROC
```

См. [1] в «Списке литературы».

## Примеры кода, показывающего проверку роли клиента и контроль за флагами состояния в «Службе Компонент»

Пример кода в VFP 7.0 и выше показывающий использование интерфейса IContextState из comsvcs.dll для контроля за флагами состояния (consistent flag and a done flag) в COM+ приложении в случае использования поддержки транзакций:

```
LOCAL oMTX, oContext, oContextState
LOCAL lTxnState, lGetTxnState, lDone, lGetDone
lGetDone = .F.    && initialize setting
lGetTxnState = 0 && initialize setting

oMTX = CREATEOBJECT("MTXAS.APPSERVER.1")
oContext = oMTX.GetObjectContext()
oContextState = GetInterface(oContext,"IContextState")

* Handle activation setting (Doneness)
* Values: .T. - Deactivate, .F. - Leave activated
lDone = .T.
oContextState.SetDeactivateOnReturn(lDone)
oContextState.GetDeactivateOnReturn(@lGetDone)

* Handle transaction setting (Consistency)
* Values: 0 - commit, 1 - abort
lTxnState = 1
oContextState.SetMyTransactionVote(lTxnState)
oContextState.GetMyTransactionVote(@lGetTxnState)
```

Пример кода, показывающего проверку роли клиента в COM+ приложении в случае использования ролевой системы безопасности:

```
PROCEDURE GetRole (tcRole)
    LOCAL oMTX,oContext,lSecurity,cRole,lHasRole
    IF EMPTY(tcRole)
        RETURN "No Role"
    ENDIF
    oMtx = CREATEOBJECT(MTX_CLASS)
    oContext = oMtx.GetObjectContext()
    IF oContext.IsSecurityEnabled
        THIS.SkipError=.T.
        lHasRole = oContext.IsCallerInRole(tcRole)
        THIS.SkipError=.F.
        DO CASE
            CASE THIS.HadError
                THIS.HadError = .F.
                cRole="Bad Role"
            CASE lHasRole
                cRole="Yep"
            OTHERWISE
                cRole="Nope"
            ENDCASE
    ELSE
        cRole="No Security"
    ENDIF
    oContext.SetComplete()
    RETURN cRole
ENDPROC
```

Чтобы дополнительные возможности на уровне компонент были поддержаны, необходима соответствующая настройка в окне свойств компоненты в «Службе компонент».

# Пример функционального разбиения VFP-приложения в трёхслойной архитектуре

Объекты визуального интерфейса, отображающие данные

Данные, полученные/отправляемые на/с сервера

Классы-посредники с одной стороны, обеспечивающие такие операции как:

- выполнить регистрацию пользователя
- получить/обновить данные согласно критериям, введённым пользователем
- сохранить/отказаться от изменений

с другой, использующие интерфейсы серверных объектов, для реализации перечисленных операций

Обработчик ошибок, включая ведение журнала ошибок

Получение/уничтожение ObjectContext

Получение общих параметров, используя SPM  
включая их загрузку из постоянного хранилища  
(в моём случае из реестра)

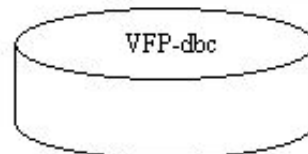
Регистрация/контроль доступа пользователя

Фабрика объектов DataEnvironment

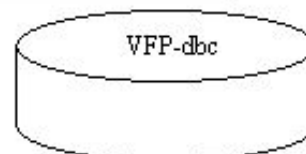
Ряд общих функций обслуживания данных:

- сериализация dbf в строку и десериализация
- получение свойств данных из dbc
- переустановка пути к базе данных у курсоров
- установка параметров в PDS
- перенос изменений произведённых пользователем в базу данных,  
(в частности обслуживание редактируемых параметризованных View)

Стандартные для VFP-dbc данные:  
таблицы, индексы, отношения,  
представления, хранимые процедуры,  
и т.п.



Прикладные данные



Административные данные

Слой  
Представлений

Слой  
Средний

Слой  
Данных

# Список литературы

1. "Microsoft Transaction Server for Visual FoxPro Developers" by Randy Brown, October 1998, MSDN Visual FoxPro 6.0 Technical Articles
2. "Understanding COM+ with VFP, Part 1-3" by Craig Berntson in MSDN (this article is reproduced from the May 2001 issue of FoxTalk)
3. "COM+ and Windows 2000: Ten Tips and Tricks for Maximizing COM+ Performance" by David S. Platt in MSDN
4. Как "работает" multi-threaded COM компонента под MS MTS/Component Services? М. Дроздов ([http://vfpdev.narod.ru/docs/mtscom\\_r.html](http://vfpdev.narod.ru/docs/mtscom_r.html) - содержит код ряда примеров, позволяющих уяснить некоторые детали работы VFP-COM-компонент)