Igor Popov
Maxim Chikinev

EFFECTIVE ArcObjects

Guide to Developing ArcGIS Extensions

УДК 681.3.06
ББК 32.973.2
П 57

Попов И. В., Чикинев М. А. Эффективное использование ArcObjects. Методическое руководство. — Новосибирск: Издательство СО РАН, 2003. — 160 с.

Эта книга детально исследует способы обработки атрибутивной и пространственной информации. Она включает ряд задач, начиная с обычного копирования данных из одного слоя в другой, вплоть до использования теоретико-множественных и топологических операций (разности, объединения, пересечения, упрощения). Она показывает, как выполнить обработку составных геометрий, исследовать пространственное отношение между объектами, поддерживать разные системы координат, измерять геометрии и использовать графику карты. Мы также уделили внимание смежным темам, таким как хранение параметров в файлах и реестре и работа с другими приложениями с использованием COM объектов. Книга предназначена для широкого круга специалистов в области ГИС, программистов, пользователей программного обеспечения ESRI.

Утверждено к печати техническим советом ООО "Дата Ист"

ISBN 5-7692-0618-7

Оглавление

Предисловие

В течение последних двух лет мы наблюдали рождение новой настольной ГИС от ESRI Inc. Несложно убедиться, что ArcView 8 предлагает не только иной интерфейс пользователя, но и другую идеологию, пересматривающую почти каждую часть ArcView 3.x, включая даже терминологию. Эти изменения неизбежно повлияли на разработку приложений и скриптов. Трудно недооценить те нововведения, с которыми столкнулись ГИС-программисты. Главное отличие заключается в том, что если предыдущие версии приложений настраивались с использованием Avenue, то теперь, если мы хотим придать ArcGIS некоторую дополнительную функциональность, нам приходится изучать ArcObjects.

Предлагаемая вашему вниманию книга посвящена, в основном, проблемам конвертирования кодов, написанных на языке Avenue, в коды, использующие ArcObjects. Мы столкнулись с этими проблемами в начале 2001 года и с тех пор занимаемся конвертированием приложений и скриптов из Avenue в ArcObjects. Надеемся, что эта книга, являясь обобщением нашего реального опыта, поможет нашим читателям при решении однотипных проблем.

Почему необходимо конвертировать приложения?

Первый вопрос, который обычно возникает у ГИС-программиста: "Насколько необходимо конвертировать уже написанное и отлаженное приложение"? Несомненно, конвертирование является трудоемким и дорогим процессом, поэтому необходимо вначале уяснить себе его цели и преимущества. Имеются, по крайней мере, две причины, почему необходимо этим заниматься.

Первой важной причиной является преимущество ArcView 8. Эта версия не только поддерживает базовую функциональность ArcView 3.x, но и включает массу улучшений. Например, ArcView 8 является не только самостоятельной настольной ГИС, но и входной точкой в ArcGIS — интегрированное семейство программных продуктов фирмы ESRI Inc., включая ArcSDE и ArcIMS. Поэтому, если вы хотите расширить ваши возможности обработки пространственно-привязанных данных, то вам не остается другого выбора, кроме обновления программного обеспечения. Кроме того, намного удобнее работать с новым пользовательским интерфейсом для MS Windows, в то время как ArcView 3.x предлагает многоплатформенную поддержку, вынуждая как пользователя, так и разработчика придерживаться определенных интерфейсных ограничений.

Фактически, функциональность новой версии имеет только один недостаток по сравнению со старой: для нее пока существует слишком мало приложений, за исключением созданных в ESRI. Необходимо также подчеркнуть, что в будущем линия продуктов ArcView 3.x вряд ли будет подвергнута серьезным революционным изменениям, тогда как ArcGIS является основой внедрения новых технологий. Например, ArcView 8 позволяет пользователю хранить пространственные данные в "geodatabase" — новом формате данных, ориентированном на использование коммерческих баз данных и расширяющем идеологию покрытий.

По всей видимости, единственным серьезным препятствием для специалиста, желающего сменить свое ГИС-приложение, является тот факт, что ArcView 8 требует больше памяти и более мощного процессора для успешной работы по сравнению с ArcView 3.x.

Второй причиной, очевидно, является преимущество ArcObjects. Поскольку ArcView нового поколения является Windows-приложением, для него не было необходимости создавать специальный, независимый от платформы язык написания скриптов.

В настоящее время во всех продуктах ESRI Inc. поддерживается создание скриптов на Visual Basic for Applications (VBA), который хорошо известен как основной инструмент настройки приложений семейства Microsoft Office (о том, как использовать VBA, рассказывается в главе 1). Для его поддержки ArcMap и ArcCatalog содержат встроенный VBA-редактор для создания макросов (скриптов). Отсутствие поддержки ГИС-программирования в VBA компенсируется наличием библиотеки COM-объектов, называемой ArcObjects. Эта библиотека является не только способом поддержки автономных операций ГИС, но она используется также для построения самого программного средства ArcView 8. Вот почему становится более естественным работать с ArcObjects — вы получаете доступ к объекту, который реально ЯВЛЯЕТСЯ частью структуры ArcView, а не некоторым объектом посредником.

Вы также можете создавать автономные приложения, которые используют ArcObjects; при этом они остаются независимыми от приложений ArcView. Обычно эта возможность не часто используется, но некоторые приложения, требующие особой функциональности, могут создаваться таким путем.

Еще одно преимущество связано с технологией COM, которая составляет основу ArcObjects. COM — это технология программирования, созданная Microsoft для того, чтобы программисты могли эффективно использовать бинарный код. Подробнее об этом можно прочитать в MSDN или в ArcObjects Developer Help. Здесь мы упомянем только пару главных фактов, связанных с COM.

Именно COM позволяет разработчикам писать приложения с помощью Visual Basic, Visual C++, Delphi и других COM-совместимых языков программирования вместо некоторого самостоятельного языка написания скриптов. Многоязыковая поддержка упрощает разработку интерфейса, поскольку VBA и все вышеупомянутые IDE (Integrated Development Environments — Интегрированные Среды Разработки) позволяют создавать новые формы

с использованием визуального программирования. Особо отметим, что профессиональные IDE включают намного большее количество утилит и инструментов, предназначенных для исследования, отладки и проектирования кода, чем предоставляет стандартный скрипт-редактор Avenue.

Именно COM поддерживает создание персональных элементов для ArcView, начиная с фильтров диалогов и кончая классами объектов (feature class), а также рабочими пространствами (workspace). ArcObjects представляет собой некий большой каркас, позволяющий конструировать ваши собственные предметно-ориентированные компоненты на основе стандартных компонентов. Вы можете разрабатывать новые легко подключаемые объекты, выполняющие сложную обработку геоданных. Как упоминалось выше, это расширяемая библиотека — можно добавлять новые объекты и использовать их совместно с ArcObjects. Имеется также возможность использовать множество COM-объектов сторонних производителей, такие как объекты Microsoft Office.

С одной стороны, изучение того, как создавать COM-объекты, — трудоемкая работа, но с другой стороны, эта сложная технология обычно скрыта внутри различных интегрированных сред разработки. Например, Visual Basic 6 рассматривает все классы, созданные программистом, как COM-объекты; при этом программист может ничего не знать об этой технологии. Visual C++ является более сложным языком, но и он включает некоторые средства (ATL — Библиотека Активных Шаблонов), упрощающие процесс создания COM-объектов. ArcView 8 предлагает также набор COM-утилит, облегчающих процесс программирования и проектирования.

Что необходимо для осуществления процесса конвертирования?

Как отмечалось в предыдущем разделе, все, что вам необходимо для конвертирования скриптов, — это ArcView 8 со встроенным

VBA-редактором. В общем случае не имеет значения, какую версию ArcView 8 вы используете, поскольку эта книга затрагивает наиболее важные и в то же время наиболее устойчивые составляющие ArcObjects. Если же вы собираетесь создать новое приложение, у вас появится выбор. Вам нужно будет выбрать некоторую IDE (и язык, разумеется) для создания COM-объектов, представляющих приложение (о создании приложения будет рассказано в следующей главе). Мы обычно используем языки программирования Visual Basic и Visual C++. Первый язык более удобен для создания интерфейса пользователя, и в то же время он позволяет избежать некоторых традиционных, присущих Visual C++ проблем, таких как утечка памяти. Второй позволяет создавать более мощные и быстрые приложения с поддержкой контейнеров, многопоточности и т. д. Очевидно, что для исследования оригинальных приложений и скриптов, просмотра исходных кодов и подготовки тестовых данных для новых конвертированных скриптов лучше всего иметь на машине ArcView 3.x. Если вы не обладаете достаточным опытом чтения кодов, созданных в среде Avenue, или если вы работаете с огромным количеством исходных кодов, использование текстовых редакторов, способных выделять синтаксические конструкции, может значительно облегчить работу.

Но гораздо более важным является следующее требование: на вашем компьютере обязательно ДОЛЖЕН быть установлен ArcMap. Дело в том, что вы не сможете инсталлировать ArcObjects без использования ArcView 8. Впрочем, это не является большой проблемой, поскольку главная часть программ на ArcObjects представляют собой приложения и скрипты, работающие только с ArcMap или ArcCatalog.

Итак, если вы всерьез решили заняться проблемой конвертирования ваших скриптов из ArcView 3.x в среду ArcGIS, то эта книга написана специально для вас.

Глава 1

Настройка интерфейса ArcView

Эта глава описывает средства, с помощью которых пользователь может настроить ArcView. Мы поговорим об активизации приложения, добавлении и настройке панелей инструментов, разработке скриптов (макросов) и о сохранении всех этих изменений. Работая с приложениями, вам не потребуется решать все эти задачи, однако создание скрипта на основе документа может включать их все. Можно также настроить настольную среду на основе имеющихся приложений, чтобы сделать ее более удобной.

Сохранение и распространение ваших скриптов и настроек

Как только новый документ создан, появляется возможность сохранить его со всеми скриптами и настройками. Кроме того, вы обязательно столкнетесь с напоминаниями и требованиями сохранения во время процесса настройки приложения под ваши нужды.

ArcMap предлагает несколько способов хранения скриптов и настроек и распространения их среди пользователей вашей организации. Простейшим является сохранение скриптов внутри ArcMap MXD документа. После этого можно передать этот документ для использования заложенной в нем функциональности другим пользователям. Однако это не лучший вариант, поскольку пользователи в ходе работы обычно создают множество доку-

ментов, и они хотят иметь в каждом документе один и тот же набор функций. В этом случае используются шаблоны — MXT файлы. Шаблон — это обычный документ, но он используется в качестве основы для других документов. Поэтому любой документ, созданный на основе шаблона, будет использовать его функциональность и настройки. Просто сохраните ваш документ как шаблон, и все его скрипты станут доступны во всех основанных на нем документах. Третья возможность распространения скриптов состоит в добавлении их в системный шаблон Normal.mxt. После этого, "добавленные" скрипты будут доступны в любом документе ArcMap. Это хороший способ распространения некоторых основных скриптов в пределах организации. Для реализации этой возможности необходимо добавить нужные модули и формы в проект Normal в окне Project Explorer среды VBA. Наконец, вы можете распределить код по отдельным модулям и в случае необходимости включать их в соответствующий документ.

Добавление приложения, панели инструментов и инструмента/команды

Приложения

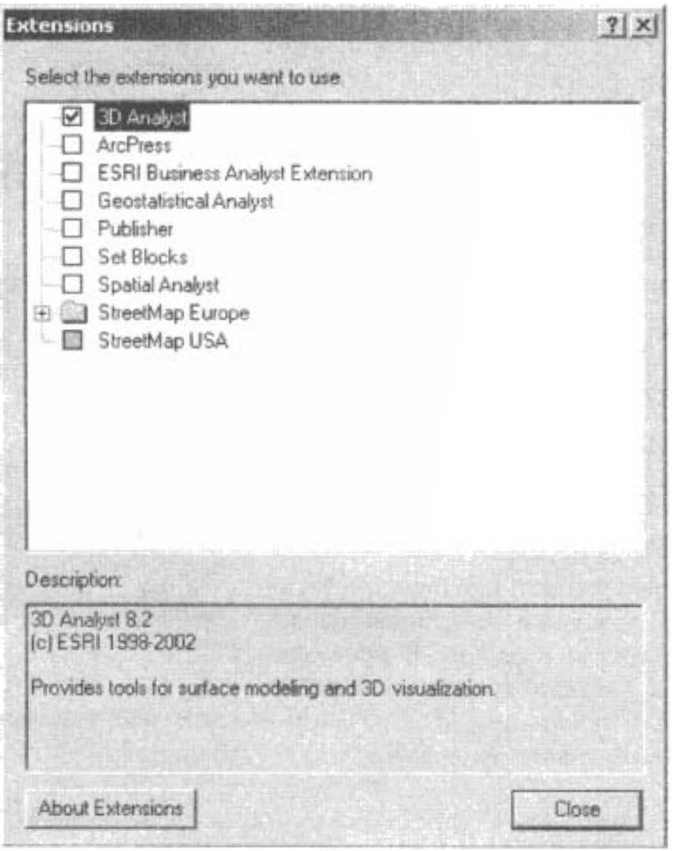
Перед созданием персональных приложений для выполнения ваших задач можно попробовать использовать некоторые приложения ESRI, такие как 3D Analyst или ArcPress, или приложения других производителей. Для этого необходимо активизировать эти приложения при условии, что они инсталлированы на ваш компьютер (можно также использовать некоторые приложения, включенные в ядро пакета версии 8.x) и у вас есть лицензия на их использование. Некоторые приложения, такие как Street Map, способны автоматически активизироваться, но в общем случае пользователь должен сделать это самостоятельно. Для того чтобы использовать корректно инсталлированное приложение, нуж-

но включить его в диалоговом окне Extensions того приложения, в котором оно будет использоваться. Этот диалог можно открыть, выбрав пункт Extensions... в меню Tools.

Этот пункт доступен либо в ArcMap либо в ArcCatalog, однако эти диалоги содержат различные списки приложений; например, невозможно активизировать приложение ArcMap в диалоге приложении ArcCatalog и наоборот.

Этот диалог можно использовать для получения краткого описания приложения и для активации/деактивации каждого доступного приложения с помощью соответствующих флажков. После активации выбранное приложение готово к работе. В приведенном на рисунке примере мы активизировали приложение 3D Analyst. Внизу окна видно краткое описание выбранного приложения.

Как и в ArcView 3.x, управлять работой приложения можно с помощью пунктов меню, элементов управления панели инструментов (команд, действующих подобно выпадающим спискам или окнам редактирования), команд и инструментов. Главное отличие новой версии — это то, что меню обычно помещаются на панелях инструментов. Следует избегать изменения главного меню, удаления некоторых из его пунктов и т. д. Вот почему мы не будем больше обсуждать такие изменения, хотя ArcView 8, как и предыдущая версия, позволяет добавлять новые пункты главного меню для своих целей. Поскольку элементы управления панели инструментов ведут себя подобно любому инструменту или команде, мы также уделим им меньше внимания.

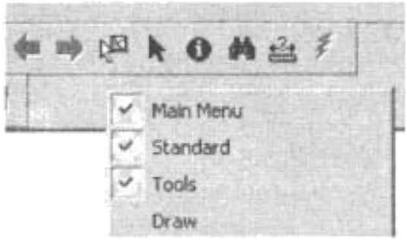


Панели инструментов

Есть, по крайней мере, два способа добавить требуемую панель инструментов. Можно щелкнуть правой кнопкой мыши в верхней части окна приложения, где размещаются все остальные инструментальные панели, и выбрать из списка нужную панель.

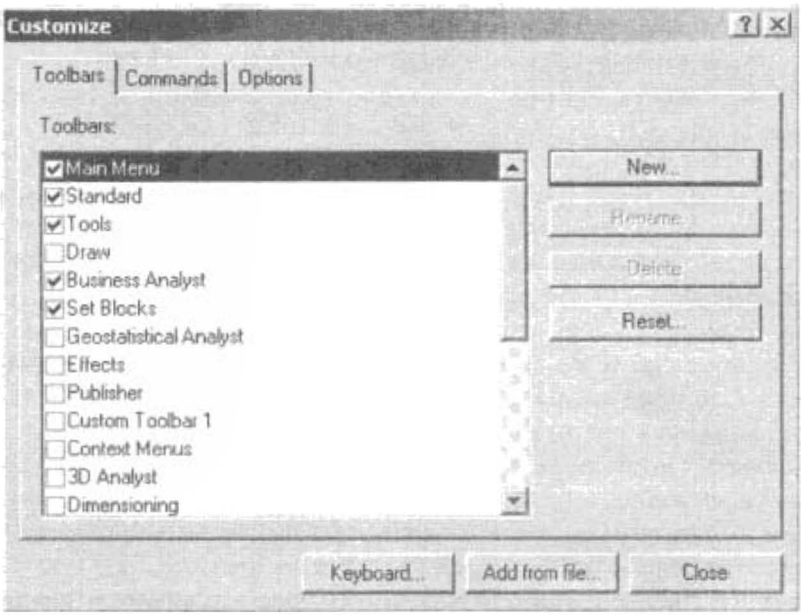
Это меню доступно также через пункт Toolbars меню View. Вторым способом является выбор пункта Customize...

вышеупомянутого подменю (он также доступен из Tools → Customize...).



Диалог Customize, который появится при выборе второго варианта, позволяет не только выбирать нужную панель инструментов, но и редактировать имя панели, создавать и удалять новые панели инструментов (эти операции поддерживаются в диалоге при выбранной закладке Toolbars). Вторая закладка позволяет подбирать нужные команды и инструменты. Третья закладка служит для задания некоторых свойств и не является предметом рассмотрения данного обзора.

Возвращаясь к теме раздела, давайте исследуем закладку Toolbars. Во-первых, можно использовать список панелей так же,



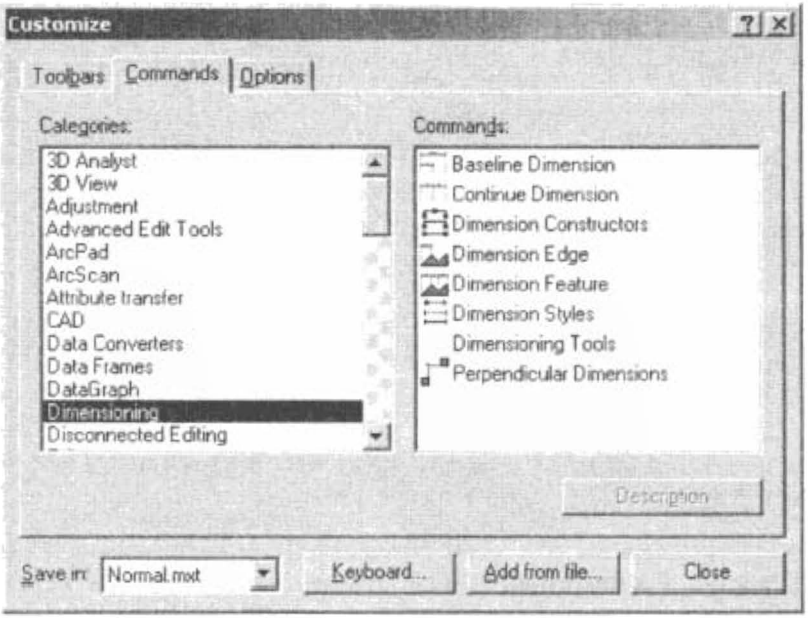
как и соответствующие пункты меню. С другой стороны, некоторые из этих панелей инструментов, такие как упоминаемая ниже панель Context Menus, служат интерфейсом для других задач настройки. Во-вторых, как уже упоминалось, можно создать свою собственную панель инструментов. Только созданные пользователем панели инструментов могут быть переименованы и удалены. Для создания новой панели инструментов нажмите кнопку New... и затем введите имя создаваемой панели в открывшемся диалоге. Второй параметр этого диалога определяет способ сохранения ваших настроек, как было описано выше. После этого можно переместить вновь созданную или любую существующую панель инструментов в походящее место и закрепить ее там.

При необходимости операции переименования и удаления панели инструментов могут быть выполнены с помощью кнопок Rename и Delete. Операция восстановления (Reset) применима только к включенным панелям инструментов, и она приводит их в исходное состояние, после того как пользователь проведет различные модификации над панелями. Нужно только выбрать шаблон, который восстанавливает все сделанные ранее установки.

Инструменты и команды

Чтобы заполнить настроенные под ваши нужды панели инструментов командами или добавить некоторые команды и инструменты в другие панели инструментов, следует выбрать закладку Commands.

Все команды и инструменты разделены на категории (обычно имя категории как-то связано с именем панели инструментов или приложения), поэтому, чтобы найти нужный инструмент, следует выбрать соответствующую категорию. Затем инструмент переносится на любую панель методом "перетаскивания-и-отпускания". Когда диалог открыт, вы также можете перемещать любые инструменты и команды из одной панели инструментов в другую или удалять любые из них, переместив выбранную команду или ин-



струмент за пределы панели. Этот же метод можно использовать для помещения команд в меню или удаления их из меню. Наконец, имеется средство для получения описания команды/инструмента, представленное кнопкой Description.

Обратите внимание, что существует возможность выбора файла, в котором следует сохранить все изменения интерфейса, посредством выпадающего списка внизу диалога при выбранной закладке Commands.

Меню

В общем смысле меню можно рассматривать либо как команды (если их видно в списке Commands), либо как некоторый элемент управления, привязанный к панели инструментов (если оно появляется только на панели инструментов при включении этой пане-

ли). Так же, как и в случае команд, вы можете применять к ним обычный подход "перетащить-и-отпустить".

Создание нового меню представляет собой несколько более сложный процесс. Для этого следует найти команду New Menu в категории New Menu. Затем поместите ее подобно любой другой команде или инструменту на панель инструментов или на строку меню либо в любое существующее меню.

Также можно добавить любые команды и меню в контекстное меню. Для этого нужно активизировать панель инструментов Context Menus на закладке Toolbars диалога Customize. Эта панель инструментов содержит список всех контекстных меню приложения, поэтому с ее помощью любой пункт меню или команду можно перетащить в требуемое контекстное меню.

Команды диалога Customize

В завершение, давайте рассмотрим кнопки внизу диалога Customize. Кнопка Keyboard... позволяет добавлять клавиши быстрого доступа к командам и инструментам. Кнопка Add from file... позволяет добавить команды и инструменты, которые не являются зарегистрированными инструментами ArcView 8. Следует подчеркнуть, что инструменты и команды хранятся в библиотеках типов (обычно это DLL файл). Таким образом, указанный способ позволяет загружать эти библиотеки, а также извлекать из них команды и инструменты. Эта команда очень полезна, если возникает проблема с автоматической регистрацией приложения.

Редактирование активных команд, инструментов и меню

Если диалог Customize открыт, можно также редактировать команды интерфейса, щелкая правой клавишей мыши на имени команды.

Всплывающее меню позволяет удалить команду с ее текущей позиции в меню, изменить ее вид или имя, объявить ее началом новой группы команд (разделяемых вертикальной или горизонтальной линией) и выбрать тип отображения (пиктограмму, текстовое имя или и то, и другое). Также существует возможность отмены изменений, связанных с этой командой, посредством пункта меню Reset. Следует подчеркнуть, что не все команды можно восстановить. Не удастся применить эту опцию к командам, представляющим макросы (скрипты), или к элементам управления панели инструментов. Все другие атрибуты элемента интерфейса, такие как всплывающие подсказки при наведении курсора мыши на элемент управления, могут быть изменены только программным путем.

Очевидно, намного важнее знать, как создавать персональные скрипты и приложения, чем понимать, как осуществлять настройку интерфейса вручную. Поэтому далее мы рассмотрим встроенный редактор VBA скриптов среды ArcView и все связанные с ним процессы.

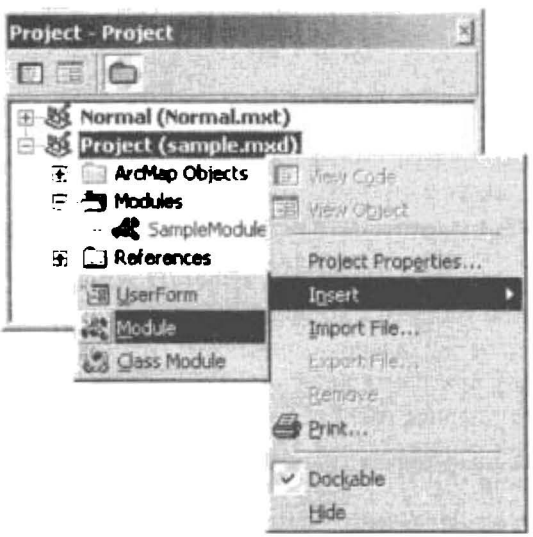
Простой VBA скрипт

Давайте посмотрим, как создать простую команду в среде VBA. Например, вам требуется команда, которая будет перемещать первый слой карты на заданную позицию. Прежде всего, вам нужно открыть среду программирования VBA. Для этого просто нажмите комбинацию клавиш Alt+F11 во время сеанса работы с ArcMap или выберите опцию Visual Basic Editor в меню Tools->Macros. VBA редактор ArcMap (ArcCatalog) представляет собой стандартную среду Microsoft VBA. Поэтому, если у вас есть опыт

программирования на VBA в среде MS Office, не составит особых проблем начать разработку и ГИС-приложения.

Среда VBA, в отличие от редактора Avenue, содержит много полезных инструментов для разработчиков. Наиболее важными усовершенствованиями являются редактор форм, браузер объектов, инструменты отладки, а также окно отслеживания значений переменных, окно свойств и возможность использования в вашей программе любой зарегистрированной библиотеки COM-объектов. Среда разработки имеет свой собственный диалог настройки, очень похожий на аналог в ArcMap. Он помогает настроить VBA интерфейс в соответствии с вашими требованиями и предпочтениями.

Лучшим способом изучения любой среды программирования является ее пошаговое изучение на примере. Первый шаг состоит в открытии редактора VBA, а затем окна Project Explorer. Оно позволяет легко управлять компонентами проекта. Нажмите клавиши Ctrl+R, если оно не открыто. Затем щелкните правой кнопкой мыши на пункте проекта в окне Project Explorer и добавьте в проект новый модуль.



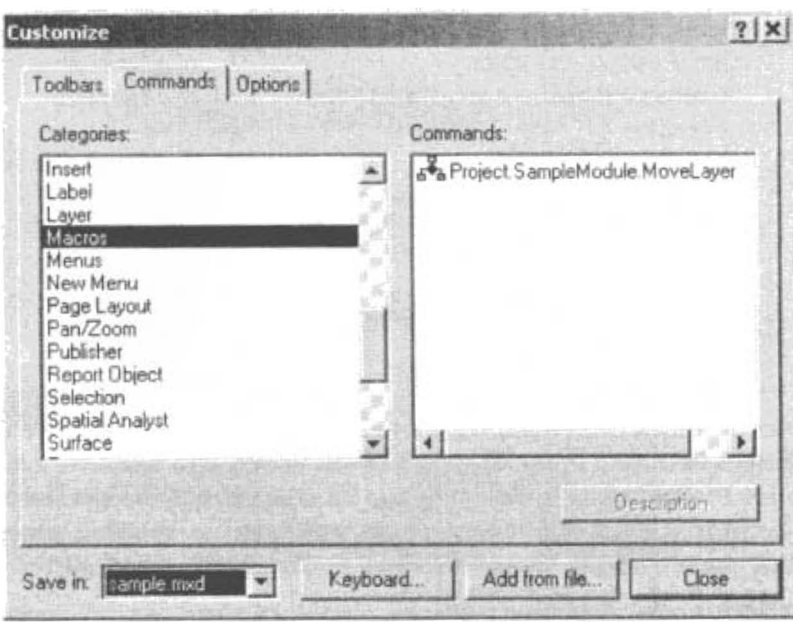
Сделайте двойной щелчок мышью по пункту модуля, и окно кода модуля станет текущим. В отличие от Avenue, в VBA код нужно записывать в процедуры. Имеется два типа процедур: процедура Sub и процедура-функция Function (используйте процедуру-функцию, если хотите, чтобы она возвращала значение). В нашем примере будем использовать Sub-процедуру. Введите следующий код в открытое окно модуля:

```
Sub MoveLayer()
  Dim pMxDoc As IMxDocument
  Set pMxDoc = ThisDocument     ' Get the reference to current
    ' document

  Dim pMap As IMap
  Set pMap = pMxDoc.FocusMap  ' Get reference to the active
    ' map

  Randomize     ' Initialize the random-number generator
  Dim iPosition As Integer
  iPosition = Int(pMap.LayerCount * Rnd)        ' New random
    ' position
  pMap.MoveLayer pMap.Layer(0), iPosition     ' Move first layer to
    ' the new position

  pMxDoc.ActiveView.Refresh          ' Refresh the view
  pMxDoc.UpdateContents              ' Update table of contents
End Sub
```

Этот код просто выбирает первый слой карты и перемещает его на произвольную позицию. Для запуска процедуры нажмите клавишу F5 или щелкните кнопку Run на стандартной панели инструментов. Перед этим не забудьте добавить, по крайней мере, один слой в карту, иначе появится сообщение о динамической ошибке.

Запуск кода непосредственно из среды VBA будет несколько неудобным для пользователей вашего скрипта. Давайте рассмотрим, как связать скрипт с кнопкой панели инструментов ArcMap. Для этого можно использовать тот же самый диалог Customize. Выберите категорию Macros на закладке Commands диалога и перетащите вашу процедуру из списка команд на любую панель инструментов ArcMap (ее можно создать перед этим шагом). Если не удается найти нужную процедуру в списке команд, проверьте установку в выпадающем списке Save in:. В нем должно быть выбрано имя документа, в котором был создан скрипт.

Подобным образом можно создать сколько угодно скриптов и сделать их доступными через кнопки на панели инструментов. Это простейший способ внести в ArcMap и ArcCatalog ваши собственные функции.

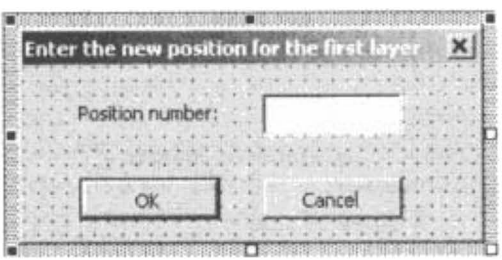


Формы VBA

Достаточно часто требуется получить некоторые параметры от пользователя перед запуском скрипта. В этом случае обычно используются формы. Давайте продолжим работу с текущим примером. Сейчас мы собираемся добавить форму, в которой будет выбираться новая позиция первого слоя на карте.

Щелкните правой кнопкой мыши по пункту проекта в Project Explorer и вставьте новую форму пользователя (UserForm). Конструктор форм VBA позволяет быстро создавать простые диалоги с обычными элементами управления. Однако вы можете использовать в форме любой дополнительный зарегистрированный элемент управления. Это свойство существенно расширяет возможности вашего интерфейса.

Создайте следующую форму с помощью конструктора форм и окна Properties.



Задайте следующие имена для элементов:

Форма	-	PositionDlg
Кнопка OK	-	btnOK
Кнопка Cancel	-	btnCancel
Текстовое окно	-	tbPosition

Затем следует добавить процедуры обработки событий для кнопок и формы. Сделайте двойной щелчок по каждой кнопке и введите следующий код:

```
Public m_bCanceled As Boolean    ' Stores form close method

Private Sub btnCancel_Click()
 m_bCanceled = True              ' User pressed cancel
 Hide
End Sub

Private Sub btnOK_Click()

 If (Not IsNumeric(tbPosition.Text)) Then
  MsgBox "Please, enter a number between 1 and layer count", _
   vbExclamation, "Wrong value"
  Exit Sub
 End If

 m_bCanceled = False      ' Accept user settings
 Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, _
 CloseMode As Integer)
 m_bCanceled = True       ' Form was closed
End Sub
```

В заключение измените код модуля нашего примера:

```
Sub MoveLayer()
 Dim pMxDoc As IMxDocument
 Set pMxDoc = ThisDocument
 Dim pMap As IMap
 Set pMap = pMxDoc.FocusMap
```

```
Dim pPosDlg As New PositionDlg ' Create form instance
pPosDlg.Show vbModal   ' Show the form as modal dialog

Dim iPosition As Integer   ' Stores new layer position

If (pPosDlg.m_bCanceled = False) Then            ' OK button
  ' was pressed
  iPosition = CInt(pPosDlg.tbPosition.Text) - 1   ' Get position
  pMap.MoveLayer pMap.Layer(0), iPosition
  pMxDoc.ActiveView.Refresh
  pMxDoc.UpdateContents
End If

End Sub
```

Теперь можно переключиться в ArcMap и посмотреть форму в работе.

Отладка в VBA

VBA имеет мощный набор инструментов отладки, которые можно использовать в ходе разработки приложения. Например, вам нужно проконтролировать значение переменной перед ее использованием. Давайте воспользуемся вышеприведенным примером и посмотрим на значение переменной *iPosition* перед переходом к следующему оператору, включающему ее. Простейшим способом

```
  Dim iPosition As Integer

  If (pPosDlg.m_bCanceled = False) Then
    iPosition = CInt(pPosDlg.tbPosition.Text) - 1
    pMap.MoveLayer pMap.Layer(0), iPosition
    pMxDoc.ActiveView.Refresh
    pMxDoc.UpdateContents
  End If
End Sub
```

является установка точки прерывания на соответствующей строке кода. Переместите курсор на соответствующую позицию и нажмите F9 (или выберите пункт Toggle Breakpoint из меню Debug).

Затем запустите программу и введите значение в диалоге. Щелкните кнопку OK, и программа завершит выполнение. Имеется несколько возможностей для наблюдения за значениями переменной:

- Если поместить курсор мыши на имя переменной, то ее значение отобразится в маленьком окошке как подсказка.
- Можно открыть специальное окно для отслеживания значений переменных (Debug -> Add Watch...)
- Наконец, можно использовать окно Immediate. В этом окне вы можете выполнять выражения и запрашивать значения переменных с помощью символа ?. Например, если вы введете ? iPosition и нажмете клавишу Enter, то получите значение переменной *iPosition*.

Для пошагового исполнения скрипта используйте клавишу F8 (или Shift+F8). Этот набор инструментов достаточен для быстрого выявления и исправления ошибок в коде.

Глава 2

Скелет приложения

В некоторых случаях вы, возможно, предпочтете создать макрос, но в общем случае все важные и достаточно сложные программы лучше разрабатывать как приложения. Как и в ArcView 3.2, приложение можно включить и выключить при необходимости, кроме того, переносимость приложения намного лучше. Однако реализовать приложение в среде ArcView 8 гораздо сложнее. Вот почему мы проанализируем эту проблему на примере исходных кодов приложения XTools, его панели инструментов, меню и команд. Кроме того, мы рассмотрим задачу создания инструмента и способа регистрации этих компонентов для их интеграции в ArcView.

Создание нового приложения

Давайте создадим новое приложение, используя MS Visual Basic, и выберем тип нового проекта как ActiveX DLL. Вы увидите новый класс с именем Class1, установленным по умолчанию (вам следует переименовать класс, если создаете профессиональное приложение). В приложении XTools мы назвали его XExtension. Кроме того, измените имя проекта на более содержательное. После этого мы написали следующий код.

Прежде всего, мы установили опцию Explicit во избежание различных проблем, связанных с объявлением переменных и неправильным написанием их имен.

```
Option Explicit

Implements IExtension
Implements IExtensionConfig

Private m_pApp As IApplication
Private m_extState As esriExtensionState


Private Property Get IExtension_Name() As String

  IExtension_Name = "XTools"

End Property

Private Sub IExtension_Startup(
  ByRef initializationData As Variant)

  Set m_pApp = initializationData
  Set m_pDocEvents = m_pApp.Document

End Sub

Private Sub IExtension_Shutdown()

  Set m_pDocEvents = Nothing
  Set m_pApp = Nothing

End Sub

Private Property Get IExtensionConfig_Description() As String

  IExtensionConfig_Description = "XTools contains useful vector
    spatial analysis, shape conversion and table management tools"
```

```
End Property

Private Property Get IExtensionConfig_ProductName() As String

  IExtensionConfig_ProductName = "XTools"

End Property

Private Property Let IExtensionConfig_State(
  ByVal state As esriCore.esriExtensionState)

  m_extState = state

End Property

Private Property Get IExtensionConfig_State()
  As esriCore.esriExtensionState

  IExtensionConfig_State = m_extState

End Property

...
```

Программирование, которое требуется при создании нового приложения, ограничивается реализацией интерфейсов **IExtension** и **IExtensionConfig**. Первый интерфейс определяет ваш класс как приложение ESRI, а второй делает ваше приложение видимым в диалоге Extensions, который обсуждался в главе 1. Дополнительно были объявлены несколько закрытых членов класса для хранения важных ссылок. Естественно, мы не будем здесь обсуждать все особенности алгоритма за исключением тех частей кода, которые необходимы для создания приложения.

Свойство **IExtension_Name** позволяет ArcMap (или ArcCatalog) определить имя вашего приложения. Мы установили его равным XTools, поскольку это подлинное приложение XTools.

Очевидно, должен существовать некоторый способ уведомления, что приложение запущено или остановлено, а также способ передачи параметров инициализации приложения. Для выполнения этой задачи в ArcObjects для вышеупомянутого интерфейса имеются две процедуры **IExtension_Startup (ByRef initializationData As Variant)** и **IExtension_Shutdown ()**. Первый метод позволяет получить ссылку на приложение как параметр инициализации. Обычно обе эти процедуры не очень популярны, так как имеется другой способ проверить, активно приложение или нет. Мы рассмотрим его при создании команды.

Поскольку интерфейс IExtensionConfig поддерживает взаимодействие с диалогом Extension, то он может быть использован для обеспечения диалога различными данными.

Свойство **IExtensionConfig_Description** помогает задать описание приложения, которое можно наблюдать в диалоге (см. соответствующее описание в Главе 1).

Свойство **IExtensionConfig_ProductName** определяет имя приложения в списке диалога. Лучше всего задать его аналогичным имени, возвращаемому свойством IExtension_Name.

Свойство **IExtensionConfig_State** служит для управления состоянием приложения (т. е., является ли оно активным в данный момент или нет). Этот метод позволяет программисту получить состояние приложения вне этого объекта или задать его в коде.

Затем после компиляции вы получите работоспособную библиотеку dll. Однако остались нерешенными две проблемы: приложение не зарегистрировано (мы решим эту проблему позже) и оно пока не имеет пользовательского интерфейса. Давайте приступим к решению второй проблемы — создадим новую панель инструментов, поскольку она является главным контейнером для элементов управления пользовательского интерфейса.

Создание новой панели инструментов

Создайте новый класс в Visual Basic для панели инструментов и как-нибудь назовите его (мы назвали наш класс в XTools — XToolsToolBar). Затем добавьте примерно такой код:

```
Option Explicit

Implements IToolBarDef

Private Property Get IToolBarDef_Caption() As String

 IToolBarDef_Caption = "XTools"

End Property

Private Sub IToolBarDef_GetItemInfo( _
 ByVal pos As Long, ByVal itemDef As esriCore.IItemDef)

 Select Case pos
  Case 0
   itemDef.ID = "XTools.XMenu"
   itemDef.Group = False
  Case 1
   itemDef.ID = "XTools.Defaults"
   itemDef.Group = True
  Case 2
   itemDef.ID = "XTools.CreateShapefile"
   itemDef.Group = True
  Case 3
   itemDef.ID = "XTools.EraseFeatures"
   itemDef.Group = True
  Case 4
   itemDef.ID = "XTools.IdentityLayer"
   itemDef.Group = False
```

```
...

  Case 19
    itemDef.ID = "XTools.Coordinates"
    itemDef.Group = False
  Case 20
    itemDef.ID = "XTools.HelpCommand"
    itemDef.Group = True
  End Select

End Sub

Private Property Get IToolBarDef_ItemCount() As Long

  IToolBarDef_ItemCount = 21

End Property

Private Property Get IToolBarDef_Name() As String

  IToolBarDef_Name = "XToolsToolbar"

End Property
```

Очевидно, что класс панели инструментов является индивидуальной реализацией интерфейса **IToolBarDef**. Как и для класса приложения, в нем имеются два свойства, определяющих имя панели инструментов и ее заголовок: **IToolBarDef_Name** и **IToolBarDef_Caption**.

Другая часть интерфейса представляет список элементов, поскольку это панель инструментов, которая знает все о размещении этих компонентов. Свойство **IToolBarDef_ItemCount** определяет число элементов (команд, инструментов, пунктов меню и элементов панели инструментов), а свойство

IToolBarDef_GetItemInfo определяет, какая команда размещается в данной позиции. Позиция задается параметром pos, а результат передается как объект **itemDef** — реализация интерфейса IItemDef. Этому объекту для работы необходимо одно установленное свойство — идентификатор элемента ID (**ID**), ссылающийся на некоторый элемент графического интерфейса. Этот ID совпадает с Program ID соответствующего класса. Другое свойство (**Group**) определяет, начинает ли этот элемент новую группу или нет (группы разделяются вертикальными линиями).

Операция заполнения панели инструментов является обычно итерационным процессом, поскольку команды и инструменты разрабатываются последовательно. Хотя подобный код можно скомпилировать прямо сейчас, он абсолютно бесполезен без компонентов панели.

Создание нового меню

Новое меню является классом, реализующим интерфейс IMenuDef. Взгляните на исходный код главного меню XTools:

```
Option Explicit

Implements IRootLevelMenu
Implements IMenuDef

Private Property Get IMenuDef_Caption() As String

  IMenuDef_Caption = "XTools"

End Property

Private Sub IMenuDef_GetItemInfo( _
  ByVal pos As Long, ByVal itemDef As esriCore.IItemDef)

  Select Case pos
```

```
  Case 0
    itemDef.ID = "XTools.Defaults"
    itemDef.Group = False
  Case 1
    itemDef.ID = "XTools.CreateShapefile"
    itemDef.Group = True
  Case 2
    itemDef.ID = "XTools.LayerOperationsMenu"
    itemDef.Group = True
  Case 3
    itemDef.ID = "XTools.FeatureConvertionsMenu"
    itemDef.Group = True
  Case 4
    itemDef.ID = "XTools.TableOperationsMenu"
    itemDef.Group = True
  Case 5
    itemDef.ID = "XTools.CalcSize"
    itemDef.Group = True
  Case 6
    itemDef.ID = "XTools.HelpCommand"
    itemDef.Group = True
  End Select

End Sub

Private Property Get IMenuDef_ItemCount() As Long

  IMenuDef_ItemCount = 7

End Property

Private Property Get IMenuDef_Name() As String

  IMenuDef_Name = "XToolsMenu"

End Property
```

Интерфейс IMenuDef имеет много общего с интерфейсом IToolBarDef. Здесь вы можете также увидеть реализацию свойств **IMenuDef_Name** и **IMenuDef_Caption**, определяющих в ArcView имя меню и его заголовок.

Свойство **IMenuDef_ItemCount** и метод **IMenuDef_GetItemInfo** также аналогичны соответствующей паре вышеупомянутого интерфейса. Свойство задает также число пунктов меню, а метод распределяет пункты меню по позициям, используя тот же интерфейс ItemDef, что и в предыдущем случае. Как вы могли заметить, почти половина всех пунктов являются меню (это интуитивно понятно из названий их идентификаторов). Именно так создаются выпадающие меню: одно меню помещается как пункт в другое меню. Как было показано в предыдущем разделе, меню можно поместить на панель инструментов.

Создание новой команды

Процесс создания новой команды также требует создания нового класса. Мы рассмотрим эту задачу на примере команды Polygon2Polyline (трансформация полигонов в полилинии). Давайте добавим следующий код в ваш новый класс.

```
Option Explicit

Implements ICommand

Private m_ResourceForm As ResourceForm
Private m_pApp As IApplication
Dim m_pBitmap As IPictureDisp
Private Sub Class_Initialize()

  Set m_ResourceForm = New ResourceForm
  Set m_pBitmap = _
    m_ResourceForm.Polygon2PolylineImage.Picture
```

```
End Sub

Private Property Get ICommand_Bitmap() _
  As esriCore OLE_HANDLE

  ICommand_Bitmap = m_pBitmap

End Property

Private Property Get ICommand_Caption() As String

  ICommand_Caption = "Convert Polygons to Polylines"

End Property

Private Property Get ICommand_Category() As String

  ICommand_Category = "XTools"

End Property

Private Property Get ICommand_Checked() As Boolean

  ICommand_Checked = False

End Property

Private Property Get ICommand_Enabled() As Boolean
  On Error GoTo ErrorHandler

  If m_pApp Is Nothing Then

    ICommand_Enabled = False
    Exit Property

  End If
```

```
Dim pUID As New UID
pUID.Value = "XTools.XExtension"

Dim pExtConf As IExtensionConfig
Set pExtConf = m_pApp.FindExtensionByCLSID(pUID)

If (pExtConf Is Nothing) Then

  ICommand_Enabled = False
  Exit Property

ElseIf (pExtConf.state <> esriESEnabled) Then

  ICommand_Enabled = False
  Exit Property

End If

Dim pMxDocument As IMxDocument
Set pMxDocument = m_pApp.Document
Dim pMap As IMap
Set pMap = pMxDocument.FocusMap
Dim pEnumLayer As IEnumLayer
Set pEnumLayer = pMap.Layers
pEnumLayer.Reset

Dim pFLayer As IFeatureLayer
Dim pLayer As ILayer
Set pLayer = pEnumLayer.Next
Dim isPolygonal As Boolean
isPolygonal = False

Do While (Not pLayer Is Nothing) And (Not isPolygonal)

  If TypeOf pLayer Is IFeatureLayer Then
```

```
    Set pFLayer = pLayer

    If Not pFLayer.FeatureClass Is Nothing Then

      If (pFLayer.FeatureClass.ShapeType = _
        esriGeometryPolygon) Then

        isPolygonal = True

        Exit Do

      End If

    End If

  End If

  Set pLayer = pEnumLayer.Next

 Loop

 ICommand_Enabled = isPolygonal

 Exit Property
ErrorHandler:
 ICommand_Enabled = False
End Property

Private Property Get ICommand_HelpContextID() As Long

End Property

Private Property Get ICommand_HelpFile() As String

End Property
```

```
Private Property Get ICommand_Message() As String

  ICommand_Message = "Convert Polygons to Polylines"

End Property

Private Property Get ICommand_Name() As String

  ICommand_Name = "XTools_Polygon2Polyline"

End Property

Private Sub ICommand_OnClick()

  Dim pFeatureTypeConverter As FeatureTypeConverter
  Set pFeatureTypeConverter = New FeatureTypeConverter
  Set pFeatureTypeConverter.m_pApp = m_pApp
  pFeatureTypeConverter.Polygon2Polyline

End Sub

Private Sub ICommand_OnCreate(ByVal hook As Object)

  Set m_pApp = hook

End Sub

Private Property Get ICommand_Tooltip() As String

  ICommand_Tooltip = "Convert Polygons to Polylines"

End Property
```

Этот класс реализует интерфейс **ICommand**, являющийся основой для каждой команды и инструмента. Он обеспечивает ArcView следующей информацией.

Свойство **ICommand_Bitmap** возвращает дескриптор растрового изображения. Имеются различные способы получения дескриптора требуемого изображения, простейшим из которых является создание новой формы (мы назовем ее Resource — формой ресурсов) и помещение в нее либо набора элементов управления Image (одного для каждой картинки), либо одного элемента управления ImageList, также заполненного растровыми картинками. Поместите ваши изображения в элементы управления Image или ImageList и затем используйте свойство Picture или получите нужный элемент из коллекции ListImages, соответственно. Кроме того, вы можете использовать встроенную функцию LoadImage для загрузки изображения из файла. Последний вариант кажется наиболее простым, однако он усложняет поддержку переносимости, поскольку не загружает изображения в проект. В приложении XTools мы использовали набор отдельных объектов Image, что видно из приведенного кода.

Свойства ICommand_Caption, ICommand_Category и **ICommand_Name** определяют заголовок команды в списке Commands диалога Customize, категорию команд в списке Category этого диалога (см. Главу 1) и имя команды (обычно включает категорию и заголовок команды).

Свойство **ICommand_Tooltip** задает строку, появляющуюся в виде подсказки для команды при наведении курсора мыши на название команды.

Свойство **ICommand_Message** задает строку сообщения, которое выводится в строке статуса приложения при прохождении курсора по именам команд.

Свойства **ICommand_HelpContextID** и **ICommand_HelpFile** помогают связать команду с некоторой частью файла помощи для поддержания эффективной работы с командой.

Свойство **ICommand_Checked** представляет собой флаг, описывающий состояние команды (выбрана/не выбрана). Поскольку данная команда не нуждается в этом свойстве, ему присвоено значение False.

Свойство **ICommand_Enabled** также является флагом, задающим доступность команды. Для правильной работы этой команды необходимы два условия: приложение подключено, и текущая карта содержит, по крайней мере, один полигональный слой. Вот почему оба критерия проверяются в коде.

Процедура **ICommand_OnCreate** является очень важной частью интерфейса, поскольку здесь команда получает ссылку на объект приложения, который широко используется при программировании с использованием ArcObjects.

Процедура **ICommand_OnClick** инкапсулирует все, для чего мы начали создание команды, — код, который будет выполняться при выборе команды. В некоторых случаях достаточно поместить сюда весь код, в других случаях следует применить несколько методов другого объекта для упрощения кода класса, как было сделано в данном случае.

Создание нового инструмента

Создание нового инструмента очень похоже на создание новой команды. Прежде всего, вы должны создать новый класс, реализующий оба интерфейса ITool и ICommand. С точки зрения реализации ICommand, между инструментом и командой нет различий. Давайте рассмотрим код типичного инструмента.

```
Option Explicit

Implements ICommand
Implements ITool

Private m_ResourceForm As ResourceForm
Private m_pApp As IApplication
Dim m_pBitmap As IPictureDisp
Dim m_pCursor As IPictureDisp
...

Private Sub Class_Initialize()

  Set m_ResourceForm = New ResourceForm
  Set m_pBitmap = m_ResourceForm.ShCommandImage.Picture
  Set m_pCursor = m_ResourceForm.ShCursorImage.Picture
  ...

End Sub

Private Property Get ICommand_Bitmap() _
  As esriCore.OLE_HANDLE

  ICommand_Bitmap = m_pBitmap

End Property

Private Property Get ICommand_Caption() As String

  ICommand_Caption = "Shape Shift"

End Property

Private Property Get ICommand_Category() As String

  ICommand_Category = "Shape Shift"
```

```
End Property

Private Property Get ICommand_Checked() As Boolean

 ICommand_Checked = False

End Property

Private Property Get ICommand_Enabled() As Boolean
 On Error GoTo ErrorHandler

 ...

 ICommand_Enabled = True

 Exit Property
ErrorHandler:
 ICommand_Enabled = False
End Property


Private Property Get ICommand_HelpContextID() As Long

End Property


Private Property Get ICommand_HelpFile() As String

End Property

Private Property Get ICommand_Message() As String

 ICommand_Message = "Shifts features of a Shapefile
   in the current active Layer according to a user line drawn"
```

```
End Property

Private Property Get ICommand_Name() As String
  ICommand_Name = "Shapeshift_ShCommand"
End Property

Private Sub ICommand_OnClick()
End Sub

Private Sub ICommand_OnCreate(ByVal hook As Object)
  Set m_pApp = hook
End Sub

Private Property Get ICommand_Tooltip() As String
  ICommand_Tooltip = "Shape shift"
End Property
Private Property Get ITool_Cursor() As esriCore.OLE_HANDLE
  ITool_Cursor = m_pCursor
End Property
Private Function ITool_Deactivate() As Boolean
```

```
 ITool_Deactivate = True
 ...

End Function

Private Function ITool_OnContextMenu( _
 ByVal X As Long, ByVal Y As Long) As Boolean

End Function

Private Sub ITool_OnDblClick()

End Sub

Private Sub ITool_OnKeyDown( _
 ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnKeyUp( _
 ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnMouseDown(ByVal Button As Long, _
 ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

...

End Sub

Private Sub ITool_OnMouseMove(ByVal Button As Long, _
 ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

 ...
```

```
End Sub

Private Sub ITool_OnMouseUp(ByVal Button As Long, _
  ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

  ...

End Sub

Private Sub ITool_Refresh( _
  ByVal hDC As esriCore.OLE_HANDLE)

End Sub
```

Это инструмент, который мы создали в рамках одного из наших проектов. Как вы могли заметить, для работы ему необходимы внешние данные — идентификатор курсора. Мы использовали элемент управления Image для хранения содержимого файла курсора (.cur файл можно создать в редакторе Visual C++). Результат доступен в виде свойства **ITool_Cursor**.

Флаг **ITool_Deactivate** определяет, можно ли деактивировать этот инструмент. Этот инструмент можно деактивировать в любом случае, однако вы можете выполнить некоторый специальный анализ. Этот метод вызывается каждый раз, когда пользователь выбирает другой инструмент (но не команду), в то время как ваш инструмент — активный.

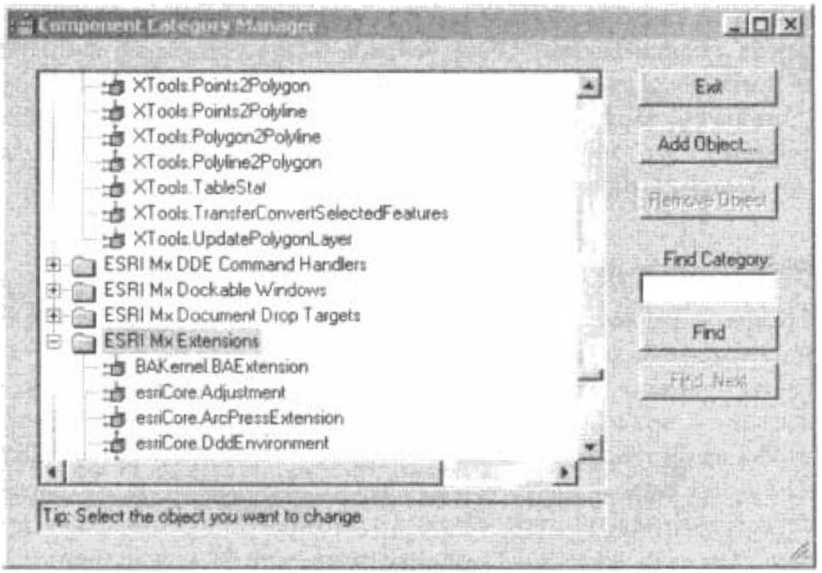
Все методы интерфейса ITool предназначены для обработки различных событий: **ITool_OnContextMenu** — для обработки вызова контекстного меню во время работы инструмента; **ITool_OnDblClick** — для обработки двойного щелчка мышью; **ITool_OnKeyDown** — для обработки события нажатия клавиши; **ITool_OnKeyUp** — для обработки события отжатия клавиши; **ITool_OnMouseDown** — для обработки события нажатия кнопки мыши; **ITool_OnMouseMove** — для обработки события переме-

щения мыши; **ITool_OnMouseUp** — для обработки события отжатия кнопки мыши; **ITool_Refresh** — для обработки события обновления экрана. Вы должны просто выбрать требуемый набор обработчиков событий и написать для них необходимый код. Разумеется, эти методы будут вызываться только тогда, когда инструмент выбран.

Регистрация приложения

Теперь у нас есть скелет нового приложения с меню, панелью инструментов, командой и инструментом. Если вам этого достаточно, можно создать замечательное, но невидимое для ArcView 8 приложение. Хотя данная версия ArcView не требует, чтобы ваши приложения располагались в некоторой конкретной папке, как было раньше, тем не менее, для ее работы необходима регистрация приложения. Очевидно, можно проигнорировать эту проблему, вручную добавляя элементы панели инструментов с помощью диалога Customize ArcView (это подробно описано в Главе 1), но этот подход не годится для профессиональной установки приложения, так как он не позволяет добавлять новую панель инструментов и само приложение в ArcView. Имеются два способа регистрации с помощью Visual Basic. Первый способ заключается в использовании инструмента Component Category Manager, расположенного в папке <ArcGIS installation folder>\bin.

Вы должны найти необходимые категории, такие как ESRI Mx Extensions для приложений ArcMap, ESRI Mx Commands для команд и инструментов ArcMap, ESRI Mx CommandBars для панелей инструментов ArcMap или ESRI Mx Tool Menu Command для меню. Затем нажмите кнопку Add Object..., выберите нужную скомпилированную библиотеку и отметьте те классы в списке, которые вы хотите добавить в эту категорию компонентов. С этим инструментом очень легко работать, однако его главным недостатком является невозможность автоматизации процедуры регистрации. Вы должны повторить эту процедуру на каждом компьютере, на котором устанавливаете вашу программу.



Для выполнения более гибкой регистрации вы должны создать новый файл регистрации (.reg). Этот файл должен содержать информацию о категориях, используемых вашими COM-классами. Давайте рассмотрим файл XTools.reg.

```
REGEDIT4

; This Registry Script enters CoClasses into their appropriate
  Component Category
; Use this script during installation of the components


; CoClass: XTools.XExtension
; CLSID: {1C32B81C-93F0-471C-8DB8-A9D32AC31E4D}
; Component Category: ESRI Mx Extensions
  [HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\
  {1C32B81C-93F0-471C-8DB8-A9D32AC31E4D}\Implemented
  Categories\{B56A7C45-83D4-11D2-A2E9-080009B6F22B}]
```

```
; CoClass: XTools.XToolsToolbar
; CLSID: {821410E8-15C8-4FAC-A86E-67BED4549384}
; Component Category: ESRI Mx CommandBars
  [HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\
  {821410E8-15C8-4FAC-A86E-67BED4549384}\Implemented
  Categories\{B56A7C4A-83D4-11D2-A2E9-080009B6F22B}]

...

; CoClass: XTools.Polygon2Polyline
; CLSID: {A3EB629E-593F-4866-8144-A03D3E7FB139}
; Component Category: ESRI Mx Commands
  [HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\
  {A3EB629E-593F-4866-8144-A03D3E7FB139}\Implemented
  Categories\{B56A7C42-83D4-11D2-A2E9-080009B6F22B}]

...

; CoClass: XTools.XMenu
; CLSID: {75F35734-3A6D-481A-935B-7B69FBCADB04}
; Component Category: ESRI Mx Tool Menu Command
  [HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\
  {75F35734-3A6D-481A-935B-7B69FBCADB04}\Implemented
  Categories\{6B1DF87B-DEC8-49B0-884F-345FE2EB1274}]
```

Этот файл включает назначения категорий для каждой команды, инструмента, панели инструмента, меню и приложения. Каждое из этих назначений включает некоторую категорию компонентов, чей ID можно найти в справочной системе ArcObjects. Здесь вы видите ID четырех упомянутых категорий: ESRI Mx Extensions ({B56A7C45-83D4-11D2-A2E9-080009B6F22B}), ESRI Mx CommandBars ({B56A7C4A-83D4-11D2-A2E9-080009B6F22B}), ESRI

Mx Commands ({B56A7C42-83D4-11D2-A2E9-080009B6F22B}) и ESRI Mx Tool Menu Commands ({6B1DF87B-DEC8-49B0-884F-345FE2EB1274}).

Весь этот код помещает ID категорий в список реализуемых категорий компонента с данным CLSID. Очевидно, вы должны заменить приведенные здесь CLSID своими собственными.

Вы можете запустить этот файл вручную с тем, чтобы добавить все эти изменения в системный реестр, либо его можно добавить в мастер инсталляции для автоматического запуска в конце процедуры инсталляции. Этот способ обеспечивает лучшую переносимость и помогает вам создать реальный программный продукт.

Теперь вы можете приступить к созданию нового приложения, которое обладает всеми теми же возможностями, что и те, которые мы видели в диалогах Extensions и Customize приложений ArcMap или ArcCatalog.

Глава 3

Наиболее важные фрагменты кода

В данной главе приведены фрагменты исходных текстов программ, которые мы считаем важными и полезными для понимания технологии разработки приложений в среде ArcObjects. Мы постарались снабдить приведенные части кода максимально подробными комментариями. Некоторые примеры посвящены какой-то одной конкретной теме, в то время как другие охватывают смежные или каким-то образом связанные темы.

Работа с источниками данных

Способность использовать новую модель данных ArcObjects является краеугольным камнем разработки любой программы для ArcGIS. Диапазон источников данных становится все шире, и сами данные становятся все более сложными. В среде ArcGIS компания ESRI представила новую обобщенную объектную модель базы геоданных, состоящую из пространственных наборов данных, а также соответствующие "физические средства их хранения": персональную базу геоданных, основанную на технологии Microsoft Access, и распределенную базу геоданных на основе крупного сервера реляционных баз данных. Эта концепция использует последние достижения современных технологий в области СУБД, объектно-ориентированного программировании и ГИС. Более подробную информацию о новой модели баз геоданных вы можете найти в справочной системе разработчика ArcObjects и в некоторых популярных книгах, изданных компанией ESRI. Тем не менее, для

большинства практических приложений вам вряд ли потребуется создание и редактирование достаточно сложных баз геоданных. На практике, и программист, и конечный пользователь, манипулируют одной (максимум двумя) структурами и/или форматами.

Ключевым свойством модели данных ArcObjects является универсальность доступа к географическим данным из различных источников: шейп-файлов, покрытий или баз геоданных. Теперь вы можете манипулировать данными с помощью специального набора объектов доступа к ним. При этом доступ осуществляется с помощью специального подмножества модели ArcObjects, почти независимо от типа данных. Поскольку создание и открытие данных являются часто повторяемыми процедурами в ходе разработки приложений, настоятельно рекомендуем подойти к этой проблеме вдумчиво и серьезно. Универсализация доступа существенно упростит разработку ваших приложений и позволит избежать множества довольно типичных ошибок. Целью данного раздела является показать несколько примеров конкретного использования ArcObjects. Для этого в качестве основы мы будем использовать примеры исходных кодов XTools.

XTools поддерживает только один формат выходных пространственных данных для результатов всех пространственных операций. Этот формат представляет собой ESRI шейп-файл. Мы выбрали этот формат потому, что он очень популярен, широко распространен и использовался в предыдущих версиях XTools. Естественно, шейп-файлы имеют множество ограничений, а ESRI предоставляет и более универсальные модели хранения сложных пространственных данных. Однако, шейп-файлы остаются простейшим, гибким и хорошо известным способом хранения, переноса и обработки ваших данных. Кроме того, вы можете легко преобразовать шейп-файл в любой нужный вам формат, используя набор инструментов и мастеров ArcCatalog или ArcToolbox.

В целях унификации доступа к данным большинство инструментов XTools вызывают специальную функцию для создания пусто-

го выходного класса пространственных объектов перед запуском процесса. Эта функция создает новый шейп-файл, принимая в качестве параметров путь, набор полей и пространственную привязку (систему координат). В результате своей работы она возвращает указатель на интерфейс класса пространственных объектов нового набора данных. Имея этот указатель, можно заполнить новый шейп-файл необходимыми данными.

Исследуем функцию более подробно. Как упоминалось выше, в качестве первого параметра она принимает строку, содержащую полный путь к новому шейп-файлу. Вторым параметром (необязательным) является тип геометрии объектов, которые вы хотели бы сохранить в шейп-файле. Вы можете обратиться к справочной системе разработчика ArcObjects за полным списком доступных констант для этого параметра. Например, *esriGeometryPoint* или *esriGeometryPolygon*. Третий (также необязательный) параметр представляет собой требуемую пространственную привязку. Вы должны правильно ее задать перед вызовом функции. Если ваше приложение работает с внешними (исходными) данными, пространственная привязка может быть извлечена из исходного слоя. Например:

```
Dim pClone As IClone
Dim pSR As ISpatialReference
Dim pGeoDS As IGeoDataset
Set pGeoDS = pInputLayer
Set pClone = pGeoDS.SpatialReference
Set pSR = pClone.Clone
```

Затем вы должны передать в функцию подготовленный набор полей. Эти поля будут добавлены в новый класс пространственных объектов. Вы можете опустить этот параметр, в этом случае вы получите класс пространственных объектов с двумя обязательными полями: FID, Shape и одним специальным полем Id, зарезервированным нами для идентификации. Для подготовки необходимого набора полей вы можете скопировать их также из исходного слоя:

```
' Declare and setup field collection

Dim pNewFields As IFieldsEdit
Set pNewFields = New Fields

Dim pClone As IClone
Dim pNewField As IField
Set pClone = pFields.Field(i)
' get field by index from the source layer field collection
Set pNewField = pClone.Clone
pNewFields.AddField pNewField
```

или создать новые поля:

```
Dim pNewField As IFieldEdit
Set pNewField = New Field
With pNewField
   .Name = "My_field"
   .Type = esriFieldTypeString
   .Editable = True
 End With

pNewFields.AddField pNewField
```

Теперь мы можем перейти к содержимому функции. Хорошим стилем программирования является организация обработки ошибок внутри каждой функции. Поэтому мы начнем с инструкции обработки ошибок:

```
Function CreateShapefile(sShapefilePath As String, _
                         geomType As esriGeometryType, _
                         pSR As ISpatialReference, _
                         Optional pFields As IFields) _
                         As IFeatureClass

On Error GoTo ErrorHandler
```

Ядром описываемой нами функции является метод CreateFeatureClass интерфейса IFeatureDataset (или IFeatureWorkspace). Мы должны только подготовить соответствующие параметры и запустить эту функцию. Единственной целью нашей функции CreateShapefile является уменьшение количества повторяющегося кода каждый раз, когда вам необходимо создать новый шейп-файл. Итак, сначала проверяем обязательные параметры.

```
If pSR Is Nothing Then
  Set pSR = New UnknownCoordinateSystem ' default spatial
    reference if nothing was passed
End If

Set CreateShapefile = Nothing
```

Далее, нам нужно отделить имя нового шейп-файла от пути к его директории (Workspace). Мы создали свой собственный специальный объект для таких файловых операций и назвали его FileInfo (см. главу 4). Он поможет нам разделить строку полного пути на соответствующие компоненты. Мы просто устанавливаем свойство FullPath объекта и затем запрашиваем имя директории и файла, соответственно.

```
Dim pFileInfo As FileInfo
Set pFileInfo = New FileInfo
pFileInfo.FullPath = sShapefilePath ' first function parameter
```

Один только путь к директории недостаточен для выполнения операций с набором данных в модели ArcObjects. Нам нужно открыть соответствующее рабочее пространство и начать работу с классами пространственных объектов в нем или вначале запросить набор классов объектов. За более подробным описанием объектной модели базы геоданных ArcObjects обращайтесь к соответствующим книгам издательства ESRI и документации.

В простейшем случае мы просто должны создать объект фабрики рабочего пространства шейп-файлов (shapefile workspace factory) и открыть необходимое рабочее пространство с помощью пути к директории, извлекаемого из объекта FileInfo.

```
Dim pFeatureWorkspace As IFeatureWorkspace
Dim pWSF As IWorkspaceFactory
Set pWSF = New ShapefileWorkspaceFactory
Set pFeatureWorkspace = pWSF.OpenFromFile(
  pFileInfo.ParentDirPath, 0)
```

Следующей операцией является подготовка соответствующего набора полей для нового шейп-файла в виде коллекции. Не забудьте подготовить корректное поле геометрии и добавить его в коллекцию, поскольку по умолчанию коллекция полей (*pFields*), передаваемая как параметр, либо пуста, либо не содержит поля геометрии. Нам следует также задать простую коллекцию полей, если пользователь не задает ее сам.

```
Dim pFieldsEdit As IFieldsEdit

If (pFields Is Nothing) Or IsMissing(pFields) Then
  Set pFieldsEdit = New Fields
Else
  Set pFieldsEdit = pFields
End If
```

Начнем с поля геометрии. Сначала мы должны задать ее тип и пространственную привязку:

```
Dim pGeomDef As IGeometryDef
Dim pGeomDefEdit As IGeometryDefEdit
Set pGeomDef = New GeometryDef
Set pGeomDefEdit = pGeomDef
With pGeomDefEdit
  .GeometryType = geomType ' the second parameter
```

```
  .GridCount = 1
  .GridSize(0) = 10
  .HasM = False
  .HasZ = False
  Set .SpatialReference = pSR ' the third parameter or unknown
End With
```

Затем добавляем поле геометрии в коллекцию:

```
Dim pField As IField
Dim pFieldEdit As IFieldEdit
Set pField = New Field
Set pFieldEdit = pField

pFieldEdit.Name = "SHAPE"
pFieldEdit.Type = esriFieldTypeGeometry
Set pFieldEdit.GeometryDef = pGeomDef
pFieldsEdit.AddField pField
```

Для проверки корректности находим имя поля с геометрией, которое мы должны передать в метод интерфейса ArcObjects:

```
Set pFields = pFieldsEdit
Dim sShapeFld As String
Dim j As Integer
For j = 0 To pFields.FieldCount – 1
  If pFields.Field(j).Type = esriFieldTypeGeometry Then
    sShapeFld = pFields.Field(j).Name
  End If
Next
```

Последней операцией является определение типа пространственных объектов для нового класса. Примечание: XTools использует только простые типы пространственных объектов (см. документацию по ArcObjects).

```
Set pCLSID = Nothing
Set pCLSID = New UID
pCLSID.Value = "esricore Feature" ' simple feature
```

В заключение мы вызываем соответствующий метод полученного ранее рабочего пространства для создания самостоятельного класса простанственных объектов в виде шейп-файла. В результате работы нашей функции будет возвращаться ссылка на IFeatureClass новый класс пространственных объектов. Обратите внимание на обработчик ошибок в последних строках функции. Он возвращает пустую ссылку в случае сбоя.

```
    Set CreateShapefile = pFeatWorkspace.CreateFeatureClass(
      pFileInfo.FileName, _
      pFields, pCLSID, Nothing, esriFTSimple, sShapeFld, "")

    Exit Function
ErrorHandler:
    Set CreateShapefile = Nothing
End Function
```

Конечно, описанная выше функция не является идеальной. Вы можете модифицировать ее или даже полностью преобразовать с целью сделать ее более универсальной. Например, вы можете добавить возможность создания произвольного класса объектов, а не только шейп-файлов. Для этого вы можете передать в нее специальный объект FeatureClassName.

Следующей задачей является обсуждение примера создания таблицы. Здесь мы снова используем исходные коды XTools, поскольку две функции этого приложения (MultiSummarize и Table Statistics) в результате своей работы создают таблицы DBF. Создание таблиц является достаточно простым процессом. Для этого вам не нужно никакого поля геометрии и, соответственно, пространственной привязки. Вам нужно только открыть соответствующее рабочее пространство и вызвать метод CreateTable для

его объекта. В данном случае мы с самого начала решили разделить определение пути и имени таблицы и передать их в функцию как отдельные параметры.

```
Public Function CreateDBF( _
  sTableName As String, sTableFolder As String, _
  Optional pFields As IFields) As ITable

On Error GoTo ErrorHandler
```

Далее мы открываем рабочее пространство таким же образом, как это делали ранее. Обратите особое внимание на дополнительную проверку существования папки. Для этой цели вы можете использовать объект FileSystemObject.

```
Dim pFWS As IFeatureWorkspace
Dim pWorkspaceFactory As IWorkspaceFactory
Set pWorkspaceFactory = New ShapefileWorkspaceFactory

Dim fso As Object ' file system object
Set fso = CreateObject("Scripting.FileSystemObject")

If Not fso.FolderExists(sTableFolder) Then
  Set CreateDBF = Nothing
  Exit Function
End If

Set pFWS = pWorkspaceFactory.OpenFromFile(sTableFolder, 0)
```

Кроме того, мы должны также проверить наличие информации о полях. Нет полей — нет и таблицы.

```
If pFields Is Nothing Then
  Set CreateDBF = Nothing
  Exit Function
End If
```

И, наконец, вызываем функцию:

```
  Set CreateDBF = pFWS.CreateTable(
    sTableName, pFields, Nothing, Nothing, "")

  Exit Function
ErrorHandler:
  CreateDBF = Nothing
End Function
```

Итак, мы обсудили вопросы создания источников данных в ArcObjects на примере приложения XTools. Обратите внимание на то, что вам часто потребуется использовать некоторый существующий класс пространственных объектов или таблицу. Интересной особенностью XTools является то, что в нем не требуется открывать какой-либо источник данных напрямую, поскольку для получения данных все инструменты используют слои и таблицы из текущей карты. XTools непосредственно открывает шейп-файл в единственном случае — когда его необходимо удалить! Немного странно, но это именно так.

Во-первых, давайте посмотрим, как открываются шейп-файлы и затем как XTools используют функцию OpenShapefile для своих внутренних целей. Для открытия шейп-файла нужен только полный путь к нему.

```
Function OpenShapeFile(sPath As String) As IFeatureLayer
  On Error GoTo ErrorHandler
  Set OpenShapeFile = Nothing
```

Чтобы разделить полный путь к файлу на составляющие, мы снова используем объект FileInfo. Для открытия, а также для создания шейп-файла вначале мы должны задать соответствующее рабочее пространство.

```
Dim sFileName As String
Dim sFolderPath As String

Dim pFI As New FileInfo
pFI.FullPath = sPath
sFileName = pFI.FileName
sFolderPath = pFI.ParentDirPath

Dim pWorkspaceFactory As IWorkspaceFactory
Set pWorkspaceFactory = New ShapefileWorkspaceFactory
Dim pWorkspace As IFeatureWorkspace
Set pWorkspace = pWorkspaceFactory.OpenFromFile( _
  sFolderPath, 0)
```

На последнем шаге вызывается метод OpenFeatureClass интерфейса IFeatureWorkspace.

```
  Set OpenShapeFile = pWorkspace.OpenFeatureClass( _
    sFileName)

  Exit Function
ErrorHandler:
  Set OpenShapeFile = Nothing
End Function
```

Это очень простой, но не универсальный метод. Если вам нужно открыть классы пространственных объектов из различных источников данных (шейп-файлов, покрытий, баз геоданных), вам следует использовать специальный метод Open интерфейса IName. Для этого вы можете подготовить соответствующий объект FeatureClassName, а затем открыть соответствующий класс объектов через интерфейс IName. За дополнительной информацией обратитесь к примерам справочной системы по ArcObjects.

В заключение мы приводим фрагмент кода, в котором приложение XTools использует функцию OpenShapefile для удаления ненужного набора данных. Эта задача имеет место при перезаписи старого шейп-файла новым. В этом случае файл должен быть сначала удален, а потом создан заново.

```
Dim pDataset As IDataset
If pGxBrowseDialog.ReplacingObject Then
  Set pDataset = OpenShapeFile(sLayerFullName)
  pDataset.Delete ' removes shapefile from disk
End If
```

Таким образом, мы обсудили основные вопросы, связанные с тем, как XTools обращается с источниками данных. Дополнительные сведения о модели доступа к данным ArcObjects вы можете почерпнуть из справочной системы разработчика ArcObjects, диаграмм объектной модели или книг от ESRI: "Modeling Our World", "Exploring ArcObjects" и других.

Как наладить связь с другими приложениями?

Практически ни одно приложение не работает само по себе. Взаимодействие вашего приложения с другими, скорее всего, будет осуществляться постоянно. Особенно актуально это при работе в среде ArcObjects. Взаимодействие между приложениями осуществляется на уровне обмена данными, запросами, при формировании выходных документов и т. п. Иногда программист вдруг осознает, что лучше использовать функциональность существующего программного обеспечения вместо своей собственной (повторной) реализации той же самой функциональности. Например, если вы собираетесь формировать отчеты в вашей программе, то для этих целей лучше использовать специальные приложения, такие как Crystal Reports. Вы можете сами привести многочисленные примеры и преимущества использования внешних сервисных программ для ваших приложений. Единственным

затруднением здесь является организация взаимодействия между вашей программой и внешним приложением.

Методология Microsoft COM значительно упрощает проблему взаимодействия приложений с помощью технологий ActiveX компонент и серверов автоматизации. Следовательно, если вы используете COM-совместимый язык программирования, то можете легко "подключить" другие объекты ActiveX к вашему приложению или организовать взаимодействие с внешним сервером автоматизации.

Когда вы разрабатываете приложение для ArcMap или ArcCatalog в форме скрипта или приложения, вы всегда должны использовать некоторый COM-совместимый язык и среду разработки (VBA, Visual Basic, Visual C++ или Delphi). Это требование обусловлено тем, что ArcObjects основан на COM-технологии. С другой стороны, это означает, что вы можете использовать любые другие объекты ActiveX или средства серверов автоматизации в своих программах, что значительно расширяет их возможности.

Предположим, вам нужно выполнить некоторые статистические расчеты, которые не поддерживаются стандартными средствами ArcMap. В этом случае наилучшим решением будет использование специальной статистической программы, например Microsoft Excel. В этом случае возникает проблема переноса данных из ArcMap в таблицу Microsoft Excel. Здесь вам поможет концепция сервера автоматизации. Она позволяет легко манипулировать приложением Excel из вашего приложения или скрипта. Посмотрим, как это реализуется. Следующий фрагмент кода был взят из команды Export to Excel приложения XTools.

Шаг первый. Прежде всего, мы должны получить ссылку на текущий документ ArcMap:

```
Dim pDoc As IMxDocument
Set pDoc = m_pApp.Document
```

Здесь m_pApp представляет собой ссылку на приложение, получаемую через параметр метода OnCreate для команды ArcMap (мы говорили о нем ранее). ArcMap вызывает его автоматически.

Мы использовали специальную диалоговую форму для экспорта выбранной таблицы и ее полей. Первый комбинированный список позволяет вам выбрать таблицу, а связанный с ним второй список показывает набор полей выбранной таблицы. Таблица может представлять собой либо таблицу атрибутов слоя, либо автономную таблицу, ассоциированную с текущей картой.

Шаг второй. Итак, создадим форму и установим соответствующие надписи:

```
Dim pDlg As FieldsForm
Set pDlg = New FieldsForm
pDlg.Caption = "Export data to MS Excel"
pDlg.TableLbl = "Select source table:"
pDlg.FieldsLbl = "Select fields to export:"
Set pDlg.m_pApp = m_pApp
```

Заметьте также, что в объект формы передается ссылка на приложение, чтобы иметь в диалоге возможность доступа к данным карты.

Затем мы должны заполнить комбинированный список формы именами таблиц, связанных с картой. Для этого мы просто просмотрим коллекцию векторных слоев и автономных таблиц карты. Во-первых, извлечем названия слоев:

```
Dim pLayerCol As IEnumLayer ' special layer enumerator

If (pDoc.FocusMap.LayerCount > 0) Then
   Set pLayerCol = pDoc.FocusMap.Layers
   pLayerCol.Reset
End If
```

```
Dim pFL As IFeatureLayer
Dim pL As ILayer

If (pDoc.FocusMap.LayerCount > 0) Then
   Set pL = pLayerCol.Next

   Do While (Not pL Is Nothing)
     If (TypeOf pL Is IFeatureLayer) Then

       Set pFL = pL
       If (Not pFL.FeatureClass Is Nothing) Then
         pDlg.ResponseCombo.AddItem pL.Name
       End If

     End If
     Set pL = pLayerCol.Next
   Loop

End If
```

Обратите внимание на то, что мы должны контролировать число слоев в карте (если нет слоев, то и перебирать нечего), а также корректность ссылок на данные в слое (FeatureClasses). Иначе может произойти сбой программы когда, например, встретится "пустой слой". "Пустой слой" означает, что он ссылается на несуществующие данные. Кроме того, мы должны вначале извлекать каждый слой с помощью интерфейса ILayer. Это обусловлено тем, что ваша карта может содержать много типов слоев.

Далее мы переходим к автономным таблицам карты, которые не доступны через коллекцию слоев. Извлечь их можно при помощи другого интерфейса — IStandaloneTableCollection. При этом мы можем получить имена доступных таблиц через интерфейс IDataset, поскольку каждая автономная таблица является одним из видов набора данных (см. описание объектной модели ArcObjects).

```
Dim pTabCol As IStandaloneTableCollection
Set pTabCol = pDoc.FocusMap
Dim i As Integer

Dim pDS As IDataset
For i = 0 To pTabCol.StandaloneTableCount - 1

  If (Not pTabCol.StandaloneTable(i).Table Is Nothing) Then
    Set pDS = pTabCol.StandaloneTable(i)
    pDlg.ResponseCombo.AddItem pDS.Name
  End If

Next i
```

Итак, комбинированный список почти готов. Необходимо только выбрать его первый элемент текущим, предварительно проверив содержимое списка на случай отсутствия элементов.

```
If (pDlg.ResponseCombo.ListCount > 0) Then
  pDlg.ResponseCombo.ListIndex = 0
Else
  MsgBox "There must be at least one feature layer
    or standalone table to proceed!", _
    vbExclamation, "XTools"
  Exit Sub
End If
```

Шаг третий. С помощью следующей процедуры заполняем связанный список именами полей текущей таблицы. Это происходит каждый раз, когда пользователь делает свой выбор в комбинированном списке.

```
pDlg.ResetState
```

Затем мы открываем диалоговое окно для пользователя и анализируем результат: был ли диалог закрыт по Cancel или нет, какая таблица и поля были выбраны и т. д.

```
pDlg.Show vbModal

If (pDlg.Canceled) Then
   Set pDlg = Nothing
   Exit Sub
End If

If (pDlg.FList.Count = 0) Then
   MsgBox "No fields to export", vbExclamation, "XTools"
   Set pDlg = Nothing
   Exit Sub
End If

If (pDlg.SelectedTable Is Nothing) Then
   MsgBox "Cannot find selected table. Try another one ",
    vbExclamation, "XTools"
   Set pDlg = Nothing
   Exit Sub
End If
```

В результате мы получаем ссылку на интерфейс ITable выбранной таблицы и заполненную коллекцию имен полей.

```
Dim pTable As ITable
Set pTable = pDlg.SelectedTable
```

Необходимо учитывать еще одно обстоятельство — возможность экспорта только выделенных записей. Иногда бывает очень полезно извлекать только те записи таблицы, которые выбрал пользователь. Для этой цели мы используем интерфейс ITableSelection. Затем для непосредственного извлечения строк таблицы мы будем использовать специальный объект-курсор, обеспечивающий быстрый перебор записей. Перед тем, как получить курсор, мы создаем пустой объект-фильтр для выбора всех записей. А затем используем небольшой трюк. В действительности, здесь мы имеем две различные ситуации. Первая имеет место

тогда, когда необходимо экспортировать все записи, а вторая — когда используются только выбранные записи. Мы можем легко преодолеть эту двойственность, используя одну и ту же переменную курсора для каждого случая. Нам нужно только установить ее в соответствии с рассматриваемой ситуацией. В случае использования только выделенных записей мы просто вызываем метод Search из интерфейса ISelectionSet вместо ITable. Ниже приведен код, демонстрирующий этот прием:

```
Dim pTableSelection As ITableSelection
Set pTableSelection = pTable

Dim pQueryFilter As IQueryFilter
Set pQueryFilter = New QueryFilter

Dim pCursor As ICursor
If (pTableSelection.SelectionSet.Count > 0) Then
   pTableSelection.SelectionSet.Search pQueryFilter, True, _
    pCursor
Else
   Set pCursor = pTable.Search(pQueryFilter, True)
End If
```

Теперь исходные данные подготовлены, и мы переходим к ключевой части раздела: использованию внешнего приложения. Мы объявляем объектную переменную и создаем объект с помощью специальной функции CreateObject(). Помните, что каждый COM-сервер имеет свой собственный ProgID, хранящийся в реестре. Поэтому мы передаем ProgID сервера Excel в функцию CreateObjects() и получаем новый экземпляр приложения Excel. Не забудьте проверить ссылку на созданный объект (без проверки возможны сбои приложения при создании) и обеспечить отображение приложения на экране, если это для него необходимо (иначе операции будут происходить "за кадром").

```
Dim xlApp As Object
Set xlApp = CreateObject("Excel.Application")
If (xlApp Is Nothing) Then
   MsgBox "There is no Excel Application on your machine.", _
    vbCritical, "XTools"
   Exit Sub
End If
xlApp.Visible = True
```

Приведенный код обеспечивает нам запуск своего экземпляра приложения Microsoft Excel, и затем мы можем управлять его работой с помощью объектной ссылки *xlApp*. Прежде всего, нам необходимо создать новую рабочую книгу Excel и пустую рабочую таблицу в ней. Для этого объявите две ссылки и создайте новые объекты с помощью свойств и методов *xlApp*.

```
Dim xlWBook As Object
Set xlWBook = xlApp.Workbooks.Add
Dim xlSheet As Object
Set xlSheet = xlWBook.ActiveSheet
```

Теперь давайте посмотрим, как можно управлять внешним приложением, используя методы и свойства объекта Sheet. Нашей первоочередной задачей является экспорт имен столбцов таблицы. При необходимости мы можем форматировать ячейки рабочего листа, в которые осуществляется экспорт, — устанавливать их шрифт, цвет фона и т. д. Итак, возьмем список выбранных полей из пользовательского диалога и заполним ячейки листа Excel их названиями.

```
Dim fld As Variant
Dim iFCount As Integer
iFCount = 0

If (Not xlSheet Is Nothing) Then
```

```
    For Each fld In pDlg.FList
      xlSheet.Cells(2, 1 + iFCount).Font.Bold = True
      xlSheet.Cells(2, 1 + iFCount).Value = CStr(fld)
      iFCount = iFCount + 1
    Next

End If
```

Наиболее важным шагом является перенос каждой записи таблицы на рабочий лист MS Excel. Для этого мы организуем цикл перебора записей с помощью описанного выше объекта-курсора. Чтобы получить доступ к значениям атрибутов записей, мы используем специальный интерфейс IRow. Для каждой записи имеется один и тот же набор полей, подлежащих переносу. Каждое поле содержит некоторое значение, которое нужно экспортировать. Итак, мы создаем второй цикл для передачи значений полей каждой записи в Excel.

Таким образом, задача экспорта решена предельно просто. Однако давайте внесем в наш код небольшое усовершенствование. При передаче некоторого значения в ячейку Excel программа пытается автоматически отформатировать ее. Если вы пересылаете число в виде строки в Excel, то строка будет преобразована в числовой формат. Например, строка "00002345,56000" будет преобразована в число 2345,56. Иногда бывает необходимо предотвратить такие автоматические преобразования данных. Поэтому нам следует задать соответствующий формат для ячейки MS Excel перед тем, как копировать в нее какое-либо значение.

Свойство *NumberFormat* объекта-ячейки может нам в этом помочь. За дополнительной информацией о форматах ячеек обращайтесь к справочной системе по MS Excel VBA. Итак, вот наш цикл:

```
Dim iRCount As Integer
iRCount = 0

Dim pRow As IRow
Set pRow = pCursor.NextRow

Do While (Not pRow Is Nothing)
  iFCount = 0
  For Each fld In pDlg.FList
    If (pRow.Fields.Field(pRow.Fields.FindField( _
      CStr(fld))).Type = _ esriFieldTypeString) Then
      xlSheet.Cells(4 + iRCount, 1 + iFCount).NumberFormat = "@"
    End If
    xlSheet.Cells(4 + iRCount, 1 + iFCount).Value = _
      pRow.Value(pRow.Fields.FindField(CStr(fld)))

    iFCount = iFCount + 1
  Next
  iRCount = iRCount + 1

  Set pRow = pCursor.NextRow
Loop
```

Наконец-то мы все сделали. Последний, уже не обязательный шаг, это удалить используемые объекты путем установки их объектных переменных в *Nothing*:

```
  Set pCursor = Nothing
  Set pRow = Nothing
  Set pDlg = Nothing

  Set xlSheet = Nothing
  Set xlWBook = Nothing
  Set xlApp = Nothing
```

Как перенести объекты из одного векторного слоя в другой?

Одной из часто встречающихся задач, возникающих при работе с картой, является перенос объектов из одного слоя в другой в соответствии с некоторым критерием (обычно, выбором пользователя) и последующее выполнение некоторых операций над переносимыми пространственными данными. Эту задачу можно решить многими способами. Один из наиболее простых предполагает использование интерфейса IExportOperation. Для экспорта данных в этом интерфейсе предусмотрен метод ExportFeatureClass, позволяющий легко автоматизировать процесс. Следует помнить, что использование данного метода сопровождается некоторыми неудобствами с точки зрения конечного пользователя. Прежде всего, при при его использовании отображается специальный диалог хода процесса, который не может быть заменен никаким другим. Метод также не допускает какую-либо дополнительную обработку данных. И, наконец, он всегда создает новый класс пространственных объектов. Последнее свойство является полезным для нас только в случае отсутствия запрашиваемого класса объектов.

Таким образом, простота решения не всегда подразумевает его эффективность. Поэтому мы будем использовать другой способ, а именно перенос данных "объект за объектом".

Решение задачи способом "объект за объектом" мы рассмотрим на примере исходного кода команды Transfer/Convert Selected Features из приложения XTools. Для простоты изложения мы будем опускать некоторые части кода в очевидных случаях или в случаях, уже затронутых ранее, таких как, например, поддержка форм и получение параметров функций. Поэтому давайте сосредоточим наше внимание на работе с заранее выбранными слоями (*pInputLayer* и *pOutputLayer* для входного и выходного слоев соответственно).

Первым шагом является определение системы координат. Пространственная привязка является очень важным элементом при работе с геометрией объектов различных пространственных классов, поскольку они могут иметь различные пространственные привязки. Поэтому нам нужно получить пространственную привязку целевого класса пространственных объектов для перепроектирования в нее всех геометрий объектов исходного класса. Простейшим способом решения этой задачи является запрос интерфейса IGeoDataset на объекте FeatureLayer с последующим вызовом метода SpatialReference.

```
...
Dim pSR As ISpatialReference
Dim pGDataSet As IGeoDataset

Set pGDataSet = pOutputLayer
Set pSR = pGDataSet.SpatialReference
```

Поскольку рассматриваемая нами команда XTools осуществляет перенос только выбранных объектов, нам необходимо получить доступ к текущей выборке. Для этого класс FeatureLayer поддерживает интерфейс IFeatureSelection, который мы и будем использовать для работы. Для получения доступа к выбранным объектам и для выполнения операций над ними почти как с классом объектов целиком мы будем запрашивать интерфейс ISelectionSet.

```
Dim pInputFeatSel As IFeatureSelection
Set pInputFeatSel = pInputLayer
Dim pInputSelSet As ISelectionSet
Set pInputSelSet = pInputFeatSel.SelectionSet
```

Так как мы имеем дело с подмножеством объектов, мы должны знать их количество для контроля правильности и инициализации индикатора диалога хода выполнения процесса. Вышеупомянутый интерфейс ISelectionSet помогает получить эту информацию.

```
Dim nTotalIn As Long
nTotalIn = pInputSelSet.Count

If nTotalIn <= 0 Then

  MsgBox "There are no selected records in the " & _
    sInputLayerName & " layer.", _
    vbExclamation, "XTools"
  TransferConvertSelectedFeatures = False
  Exit Function

End If
```

Перед тем, как приступить непосредственно к обработке данных, мы должны выполнить некоторые подготовительные операции. В приведенном ниже листинге мы создаем буфер IFeatureBuffer для копирования данных из исходного слоя в целевой, а также пустой по умолчанию фильтр IQueryFilter для определения критерия запросов. Дополнительно, чтобы начать отсчет значений идентификаторов новых объектов, нам необходимо получить индекс поля-идентификатора объектов (ID) в целевом слое и число пространственных объектов самого целевого слоя.

```
Dim pFeatureBuffer As IFeatureBuffer
Set pFeatureBuffer = pOutputLayer.FeatureClass.
  CreateFeatureBuffer
Dim pInsertFCursor As IFeatureCursor
Dim pSearchFCursor As IFeatureCursor
Dim pQF As IQueryFilter
Set pQF = New QueryFilter

Dim pFeature As IFeature
Dim pSubCalcSize As SubCalcSize

Dim pGeometry As IGeometry
Dim pPointCollection As IPointCollection
```

```
Dim pOutPointCollection As IPointCollection
Dim pTopologicalOperator As ITopologicalOperator

Dim i As Long
Dim nIdFieldIndex As Long
nIdFieldIndex = pOutputLayer.FeatureClass.Fields.FindField("ID")
nCounter = pOutputLayer.FeatureClass.FeatureCount(pQF) – 1
```

Далее нам нужно создать два новых объекта-курсора: один для получения выбранных объектов исходного слоя и еще один для вставки новых объектов в целевой класс. Для ускорения процессов выборки и вставки записей оба курсора будут многократно использовать фиксированные блоки памяти для перебираемых объектов. Это означает, что для курсора создается один объект, заполняемый новыми свойствами каждый раз, когда курсор выполняет итерации. Этот момент очень важен для оптимизации приложения, так как существует и другой тип курсора, который при каждой итерации создает новый объект и, следовательно, непроизводительно расходует память. Получив нужные курсоры, мы запрашиваем первый объект из выбранного набора в исходном слое.

```
Set pInsertFCursor = pOutputLayer.FeatureClass.Insert(True)
pInputSelSet.Search pQF, True, pSearchFCursor
Set pFeature = pSearchFCursor.NextFeature
```

Для отображения диалога хода выполнения процесса и возможности отмены циклической операции мы создадим объект, реализующий интерфейс IProgressDialog. Инициализация объекта включает в себя установку механизма слежения за состоянием кнопки Cancel, различных параметров визуализации диалога и параметров индикатора хода выполнения процесса, таких как интервал индикатора процесса и размер шага итерации. Интерфейс IStepProgressor обеспечивает анимацию индикатора выполнения, тогда как интерфейс ITrackCancel отслеживает нажатие кнопки отмены и, кроме того, используется для уведомления программы об отмене операции пользователем.

```
Dim pTrackCancel As ITrackCancel
Set pTrackCancel = New CancelTracker
Dim pPDlgFact As IProgressDialogFactory
Set pPDlgFact = New ProgressDialogFactory
Dim pStepPro As IStepProgressor
Dim bContinue As Boolean

Dim pPDlg As IProgressDialog2
Set pPDlg = pPDlgFact.Create(pTrackCancel, m_pApp.hWnd)
pPDlg.CancelEnabled = True
pPDlg.Description = "Transfer / Convert " & nTotalIn & " features
  from " & pInputLayer.Name & " to " & pOutputLayer.Name
pPDlg.Title = "XTools"
pPDlg.Animation = esriDownloadFile

Set pStepPro = pPDlg
pStepPro.MinRange = 0
pStepPro.MaxRange = nTotalIn
pStepPro.StepValue = 1
pStepPro.Message = "Processing..."
```

Мы разделим весь код, отвечающий за перенос объектов, на части в соответствии с геометрическими типами переносимых данных. Это важно, поскольку различные типы геометрии требуют различной обработки. В нашем случае предполагается, что типы геометрии объектов в исходном и целевом слоях совпадают.

```
If pInputLayer.FeatureClass.ShapeType = _
  pOutputLayer. FeatureClass.ShapeType Then

    Do While Not (pFeature Is Nothing)
```

Поскольку геометрия объектов будет переноситься в новые объекты, а курсор использует одну и ту же область памяти, то она (геометрия) должна быть клонирована. Кроме того, мы должны спроектировать геометрию объектов исходного слоя в систему

координат целевого класса объектов во избежание получения искажения данных (помните, ранее мы извлекли пространственную привязку целевого слоя?). После этого геометрия объекта может быть помещена в FeatureBuffer.

```
Set pGeometry = pFeature.ShapeCopy
pGeometry.Project pSR
Set pFeatureBuffer.Shape = pGeometry
```

И, наконец, мы будем копировать или создавать необходимые атрибуты объектов. В данном примере мы будем задавать новое значение только полю ID для объекта, если только в целевом классе пространственных объектов имеется такое поле (иначе индексу nIdFieldIndex присваивается значение –1).

```
nCounter = nCounter + 1
If nIdFieldIndex >= 0 Then
  pFeatureBuffer.Value(nIdFieldIndex) = nCounter
End If
```

Новый объект подготовлен, и мы помещаем его в целевой класс пространственных объектов. Мы также контролируем нажатие пользователем кнопки Cancel; сам механизм отслеживания был уже запущен ранее. Перед следующей итерацией цикла нужно извлечь новый объект из исходного класса объектов.

```
pInsertFCursor.InsertFeature pFeatureBuffer
bContinue = pTrackCancel.Continue

If Not bContinue Then Exit Do

Set pFeature = pSearchFCursor.NextFeature

Loop
```

Мы рассмотрели вариант программы, когда типы геометрии исходного и целевого слоя совпадают. Другая ситуация описывает-

ся ниже. Здесь мы будем выполнять преобразование полигона в полилинию.

```
ElseIf (pInputLayer.FeatureClass.ShapeType =
  esriGeometry Polygon) And
  (pOutputLayer.FeatureClass.ShapeType =
    esriGeometry Polyline) Then

  Do While Not (pFeature Is Nothing)
```

Первоначальная обработка геометрии в этом случае та же самая, что и описанная выше. Но в данной ситуации она недостаточна, поскольку геометрические типы объектов различаются. Для выполнения преобразования одного типа геометрии в другой будет использован интерфейс ITopologicalOperator.

```
  Set pGeometry = pFeature.ShapeCopy
  pGeometry.Project pSR
  Set pTopologicalOperator = pGeometry
```

Полилиния извлекается как свойство "граница" полигона из интерфейса ITopologicalOperator.

```
  Set pFeatureBuffer.Shape = pTopologicalOperator.Boundary
```

Наконец, копируем необходимые атрибуты. И снова будем копировать только значение ID, если имеется соответствующее поле.

```
  nCounter = nCounter + 1

  If nIdFieldIndex >= 0 Then
    pFeatureBuffer.Value(nIdFieldIndex) = nCounter
  End If
```

Новый объект сохраняется тем же, ранее продемонстрированным способом.

```
    pInsertFCursor.InsertFeature pFeatureBuffer
    bContinue = pTrackCancel.Continue

    If Not bContinue Then Exit Do

    Set pFeature = pSearchFCursor.NextFeature

  Loop
```

Следующий фрагмент кода имеет отношение к преобразованию полигонального или полилинейного слоя в многоточечный слой.

```
ElseIf ((pInputLayer.FeatureClass.ShapeType = _
  esriGeometryPolygon) Or (pInputLayer.FeatureClass.ShapeType = _
  esriGeometryPolyline)) And _
  (pOutputLayer.FeatureClass.ShapeType = _
  esriGeometryMultipoint) Then

  Do While Not (pFeature Is Nothing)
```

Извлечение и проектирование геометрии реализуются абсолютно аналогично.

```
    Set pGeometry = pFeature.ShapeCopy
    pGeometry.Project pSR
```

Далее мы создадим новую геометрию типа multipoint как геометрию объектов целевого слоя и поместим исходную геометрию в нее как коллекцию точек с помощью интерфейса IPointCollection. Результирующая коллекция точек помещается в IFeatureBuffer.

```
    Set pPointCollection = New Multipoint
    pPointCollection.SetPointCollection pGeometry
    Set pFeatureBuffer.Shape = pPointCollection
```

Здесь мы опускаем оставшуюся часть кода для помещения нового объекта в целевой класс, поскольку она не отличается от вышеприведенных фрагментов.

```
...

Loop
```

Следующий вариант описывает экспорт объектов слоя с неточечными типами геометрии в точечный слой.

```
ElseIf (pOutputLayer.FeatureClass.ShapeType = _
  esriGeometryPoint) Then

  Do While Not (pFeature Is Nothing)
```

Получение геометрии и ее проектирование остаются неизменными, и здесь мы приводим код только для наглядности.

```
    Set pGeometry = pFeature.ShapeCopy
    pGeometry.Project pSR
```

Для преобразования текущей геометрии в коллекцию точек мы запрашиваем у нее интерфейс IPointCollection.

```
    Set pPointCollection = pGeometry
```

Для расщепления неточечной геометрии (представленной в виде коллекции точек) на точки, организуем цикл для перебора всех точек коллекции и будем помещать каждую отдельную точку в целевой класс объектов как новый отдельный объект.

```
    For i = 0 To pPointCollection.PointCount – 1

      Set pFeatureBuffer.Shape = pPointCollection.Point(i)

      nCounter = nCounter + 1

      If nIdFieldIndex >= 0 Then
        pFeatureBuffer.Value(nIdFieldIndex) = nCounter
```

```
    End If

    pInsertFCursor.InsertFeature pFeatureBuffer

  Next i
```

В конце цикла осуществляем проверку нажатия пользователем кнопки Cancel и получаем следующий объект исходного слоя.

```
    bContinue = pTrackCancel.Continue

    If Not bContinue Then Exit Do

    Set pFeature = pSearchFCursor.NextFeature

  Loop

End If
```

Закончив все операции, мы закрываем и уничтожаем диалог хода выполнения процесса обработки. Затем можно также освободить объекты классов FeatureBuffer и FeatureCursor, поскольку желательно, чтобы они были явно удалены (иногда они блокируют доступ к данным).

```
pPDlg.HideDialog
Set pTrackCancel = Nothing
Set pStepPro = Nothing
Set pPDlg = Nothing
Set pFeatureBuffer = Nothing
Set pInsertFCursor = Nothing
```

В заключение программа рассчитает площадь и периметр полигональных и полилинейных объектов с помощью созданного вспомогательного класса SubCalcSize приложения XTools.

```
If (pOutputLayer.FeatureClass.ShapeType = _
 esriGeometryPolygon) Or (pOutputLayer.FeatureClass.ShapeType =
 esriGeometryPolyline) Then
 Set pSubCalcSize = New SubCalcSize
 Set pSubCalcSize.m_pApp = m_pApp
 pSubCalcSize.CalcSize pOutputLayer
End If
```

В конце мы выполняем обновление карты.

```
pDoc.ActiveView.Refresh
```

Вот и все. Мы перенесли выбранные объекты исходного слоя в другой, выполнив необходимые преобразования геометрии.

Как работать с составными геометриями?

Работая с различными данными, вы часто будете сталкиваться с проблемой обработки составной геометрии объектов. Выполняя некоторые общие операции, вам не нужно уделять особого внимания этой проблеме, однако в некоторых случаях, скажем, таких как подсчет количества зданий, подлежащих сносу при строительстве автомагистралей, требуется анализ составных геометрических объектов. Вы уже сталкивались с подобным процессом в одной из частей этой книги, связанной с переносом объектов, когда неточечные пространственные объекты преобразовывались в точечные с использованием интерфейса IPointCollection. Продемонстрированный подход хорошо работает только когда требуется представить геометрию в виде набора формообразующих точек, поэтому в других случаях нужен более продуманный и универсальный метод. Рассмотрим проблему на примере работы команды Convert multipart shapes to single parts из приложения XTools.

Эта команда позволяет пользователю разделить каждый объект заданного входного слоя с составной геометрией на серию объектов с простой односвязной геометрией, сохраняющихся в новом

слое. Наиболее сложной частью этой функции является та, которая отвечает за разбиение полигонов, содержащих внутренние кольца, на полигоны, состоящие из односвязных областей. Вот почему этот пример полезен также для понимания механизма работы со сложными полигональными объектами. Вновь предупреждаем, что в некоторых тривиальных случаях мы будем опускать некоторые части кода.

Прежде всего, объявим несколько необходимых для дальнейшей работы переменных и проинициализируем часть из них. Назначение этих переменных достаточно очевидно, и мы демонстрируем их только с целью улучшения читаемости кода. В конце блока мы подготавливаем массив pFieldsCorrespondence для установки соответствия между полями исходного и результирующего классов.

```
...
Dim pClone As IClone
Dim pFields As IFields
Set pFields = pInputFC.Fields
Dim pCopyField As IField
Dim i As Long
Dim j As Long
Dim k As Long
Dim pFieldsCorrespondence() As Long
ReDim pFieldsCorrespondence(pFields.FieldCount) As Long
```

Затем мы создадим коллекцию полей для результирующего класса объектов и обеспечим возможность их редактирования и создадим другую коллекцию для хранения соответствия между новыми и старыми именами полей, которая является очень важной в случае, если имеет место изменение некоторых имен.

```
Dim pFieldEdit As IFieldEdit
Dim pNewFields As IFields
Dim pNewFieldsEdit As IFieldsEdit
```

```
Set pNewFields = New Fields
Set pNewFieldsEdit = pNewFields
```

Чтобы информировать пользователя о ходе процесса копирования и создания полей, мы воспользуемся строкой состояния ArcMap, доступ к которой обеспечивают интерфейс IStatusBar и соответствующее свойство объекта-приложения — IApplication.StatusBar. Со строкой состояния работать намного проще, чем с диалогом хода выполнения процесса (ProgressDialog), однако она не дает возможности прервать процесс, поэтому следует использовать ее для некоторых быстро выполняющихся операций или для тестирования вашего приложения. Иногда целесообразно использовать строку состояния совместно с диалогом хода выполнения процесса. В идеале, диалог хода выполнения процесса показывает общий ход выполнения задачи, тогда как строка состояния отражает текущее состояние ее подзадач.

Зададим сопутствующую надпись и диапазон значений для индикатора строки состояния и двинемся дальше.

```
Dim pStatusBar As IStatusBar
Set pStatusBar = m_pApp.StatusBar
pStatusBar.ShowProgressBar "Copying fields....", 0, _
  pFields.FieldCount, 1, True
pStatusBar.ProgressBar.Position = 0
```

После создания коллекции полей мы должны заполнить ее необходимыми экземплярами. Первым является поле-идентификатор (ID). Поскольку коллекция полей исходного слоя уже могла содержать это поле, нужно выполнить предварительную проверку. Если проверка даст отрицательный результат, то будет создано новое поле с именем ID и тем же псевдонимом, хранящее данные целого типа и являющееся редактируемым. Наконец, оно будет добавлено в пустую пока коллекцию полей, и индикатор хода процесса в строке состояния будет сдвинут на один шаг вправо.

```
If (pFields.FindField("ID") = – 1) Then

  Set pFieldEdit = New Field

  With pFieldEdit
    .Name = "ID"
    .AliasName = "ID"
    .Type = esriFieldTypeInteger
    Editable = True
  End With

  pNewFieldsEdit.AddField pFieldEdit
  pStatusBar.StepProgressBar

End If
```

Затем будут скопированы (клонированы) все другие поля исходного класса пространственных объектов, за исключением полей OID и Geometry, поскольку они создаются в результирующем классе пространственных объектов с помощью функции CreateShapefile (поле OID создается автоматически в ArcObject, поскольку это служебное поле, в то время как поле Geometry требует некоторой заботы программиста). Для поддержки источников данных, отличных от шейп-файлов, была разработана специальная функция ToDBFFieldName. Она генерирует подходящее имя поля для нового шейп-файла на основе имени входного поля и уже созданной коллекции полей. Этот подход позволяет нам обрезать длинные имена полей, используемые в других форматах, и подгонять их под 10-символьное ограничение на длину поля, принятое в формате шейп-файла. Однако при этом мы должны использовать некоторый способ поддержания соответствия между исходными и результирующими полями (массив pFieldsCorrespondence) и возвращения индекса результирующего поля по индексу исходного поля. Следует отметить, что на этой стадии вы вряд ли сможете найти универсальную формулу соответствия между двумя наборами полей. Для большинства задач

вы сможете сделать это после того, как результирующий набор данных будет создан. Но в нашем случае копируются все поля, и мы с самого начала знаем, что в начале набора полей точно существуют два поля (поле OID и поле геометрии).

Существенным моментом также является изменение статуса доступа для редактирования у нового клонированного поля, поскольку некоторые наборы данных содержат огромное количество нередактируемых полей, простое копирование которых будет приводить к аварийному завершению подпрограммы каждый раз, когда данные помещаются в это поле.

В конце тела цикла мы делаем следующий шаг индикатора хода процесса в строке состояния. После окончания копирования полей индикатор хода процесса должен быть скрыт. И снова эта операция выполняется намного проще, чем при использовании специального диалога хода выполнения процесса.

```
For i = 0 To pFields.FieldCount – 1

  pFieldsCorrespondence(i) = –1

  If ((pFields.Field(i).Type <> esriFieldTypeOID) And _
        (pFields.Field(i).Type <> esriFieldTypeGeometry)) Then

    Set pClone = pFields.Field(i)
    Set pCopyField = pClone.Clone
    Set pFieldEdit = pCopyField
    pFieldEdit.Name = ToDBFFieldName(
      pCopyField, pNewFields)
    pFieldEdit.Editable = True
    pNewFieldsEdit.AddField pCopyField

    pFieldsCorrespondence(i) =
      pNewFieldsEdit.FindField(pCopyField.Name) + 2
    pStatusBar.StepProgressBar
```

```
    End If

  Next i

  pStatusBar.HideProgressBar
```

В данном случае создание результирующего класса пространственных объектов требует наличия их пространственной привязки (системы координат), которая клонируется наряду с коллекцией полей. Создание нового класса пространственных объектов обеспечивается функцией CreateShapefile, упомянутой выше.

```
Dim pOutputFC As IFeatureClass
Dim pGDataSet As IGeoDataset
Set pGDataSet = pInputLayer
Set pClone = pGDataSet.SpatialReference
Set pOutputFC = CreateShapefile(pConversionForm.NewLayer
  FullPath, pInputLayer.FeatureClass.ShapeType, Clone.Clone,
  pNewFieldsEdit)

If pOutputFC Is Nothing Then

  MsgBox "Cannot create shapefile here.", vbExclamation,
  "XTools"
  Multipart2SinglePart = False
  Exit Function

End If
```

Поскольку рассматриваемая команда может обрабатывать либо все объекты, либо только выбранные (если такие имеются), то мы должны позаботиться о разборе случаев. Вспомним, что класс FeatureLayer поддерживает нужный нам интерфейс IFeatureSelection, которым мы и воспользуемся. Для того чтобы получить доступ к

выбранному набору объектов и обращаться с ним почти так же, как с классом объектов целиком, мы запросим у IFeatureSelection интерфейса свойство SelectionSet.

```
Dim pInputFeatSel As IFeatureSelection
Set pInputFeatSel = pInputLayer
Dim pInputSelSet As ISelectionSet
Set pInputSelSet = pInputFeatSel.SelectionSet
```

Перед тем, как приступить к непосредственно обработке геометрии объектов, нужно выполнить некоторые подготовительные операции. Необходимо объявить переменные, используемые для доступа к исходным объектам, и часть из них инициализировать. Ниже мы создаем новый, по умолчанию выбирающий все записи, объект-фильтр (IQueryFilter), который используют для установки условий отбора в запросах, и новый IFeatureCursor для получения объектов исходного слоя.

```
Dim pQF As IQueryFilter
Set pQF = New QueryFilter
Dim pInputFeature As IFeature
Dim pInputFCursor As IFeatureCursor
Dim pGeometryCollection As IGeometryCollection
Set pInputFCursor = pInputFC.Search(pQF, True)
```

Как и в случае с копированием полей, рассчитаем количество обрабатываемых объектов для инициализации индикатора диалога хода выполнения процесса.

```
Dim nTotalRec As Long
nTotalRec = pInputSelSet.Count

If nTotalRec = 0 Then
  nTotalRec = pInputFC.FeatureCount(pQF)
End If
```

Затем мы создадим сам диалог хода выполнения процесса. Эта процедура уже обсуждалась в предыдущем разделе.

```
Dim pTrackCancel As ITrackCancel
Set pTrackCancel = New CancelTracker

Dim pPDlgFact As IProgressDialogFactory
Set pPDlgFact = New ProgressDialogFactory

Dim pStepPro As IStepProgressor
Dim bContinue As Boolean

Dim pPDlg As IProgressDialog2
Set pPDlg = pPDlgFact.Create(pTrackCancel, m_pApp.hWnd)
pPDlg.CancelEnabled = True
pPDlg.Description = "Converting " & CStr(nTotalRec) & _
                    " multipart features to single parts...."
pPDlg.Title = "XTools"
pPDlg.Animation = esriDownloadFile

Set pStepPro = pPDlg
pStepPro.MinRange = 0
pStepPro.MaxRange = nTotalRec
pStepPro.StepValue = 1
pStepPro.Message = "Processing..."
```

Теперь нам необходимо выполнить серию шагов по инициализации процесса обработки объектов. Эти шаги включают в себя объявление переменных, необходимых для обработки объектов, инициализацию курсоров для получения объектов исходного слоя и вставки новых в результирующий слой. Нам потребуется также индекс поля ID, поскольку мы заполняем это поле информацией о порядковом номере объекта. Интересно еще раз отметить незначительное, но весьма удобное в применении различие в обработке только выбранных объектов и объектов всего класса с помощью одной переменной курсора.

```
Dim pGeometry As IGeometry
Dim pPolygon As IPolygon2
Dim pOGeometryCollection As IGeometryCollection
Dim pInsertFCursor As IFeatureCursor
Set pInsertFCursor = pOutputFC.Insert(True)
Dim pFeatBuff As IFeatureBuffer
Set pFeatBuff = pOutputFC.CreateFeatureBuffer
Dim isBySelection As Boolean
isBySelection = pInputSelSet.Count > 0

Dim nOutputIDFieldIndex As Long
nOutputIDFieldIndex = pOutputFC.Fields.FindField("ID")

Dim isMultipart As Boolean
isMultipart = False

Dim nGeometryNumber As Long
nGeometryNumber = 0

If isBySelection Then
  pInputSelSet.Search pQF, True, pInputFCursor
Else
  Set pInputFCursor = pInputFC.Search(pQF, True)
End If

Dim nCorrFieldIndex As Long
```

Наконец, мы можем начать непосредственно процесс разбиения геометрии объектов.

```
Set pInputFeature = pInputFCursor.NextFeature

Do While (Not pInputFeature Is Nothing)
```

Поскольку мы будем использовать геометрию текущего входного объекта в качестве источника для наших новых простых геометрий, лучше всего получить ее копию (клона). Необходимо под-

черкнуть, что и на этот раз мы воспользуемся курсором, использующим одну область памяти при итерациях; следовательно, на каждой итерации поле геометрии будет изменяться.

Любая геометрия может быть представлена как коллекция других, составляющих ее, геометрий. Поэтому с самого начала мы запрашиваем у геометрии интерфейс IGeometryCollection. Если текущая коллекция содержит более одного объекта (случай составной геометрии), то мы устанавливаем флаг, который в конце процедуры поможет понять, имело ли место какое-либо разбиение геометрии объекта.

```
Set pGeometryCollection = pInputFeature.ShapeCopy

If pGeometryCollection.GeometryCount > 1 Then

 isMultipart = True

End If
```

Как уже отмечалось, имеются некоторые отличия в процессе разбиения объектов полигональных слоев и объектов слоев иного типа, связанные с наличием внутренних колец у полигонов. И внешние, и внутренние кольца полигона хранятся в его коллекции геометрий в виде отдельных элементов, но в нашем случае нужно отделить только внешние кольца и оставить внутренние внутри соответствующих полигонов, иначе мы потеряем "полости" внутри полигонов. Поэтому мы рассмотрим два случая. Первый случай — это обычное разбиение неполигональных объектов.

```
If pInputFC.ShapeType <> esriGeometryPolygon Then
```

Работая с неполигональной геометрией, мы просто организуем просмотр всей коллекции геометрий, представляющей геометрию текущего объекта, и сохраняем каждый элемент коллекции со всеми его атрибутами как новый отдельный объект.

```
For j = 0 To pGeometryCollection.GeometryCount – 1

 Set pGeometry = pGeometryCollection.Geometry(j)
```

Новая геометрия будет создана заново по образцу исходной геометрии. Мы вынуждены действовать именно таким образом вместо простого сохранения текущей pGeometry, потому что элементы коллекции геометрий обычно имеют иные типы геометрии в отличие от исходной (напр., Polyline может быть сконструирована из объектов сегментов Line), поэтому прямое присваивание приведет к несоответствию типов.

```
If pInputFC.ShapeType = esriGeometryMultipoint Then

 Set pOGeometryCollection = New Multipoint

ElseIf pInputFC.ShapeType = esriGeometryPolyline Then

 Set pOGeometryCollection = New Polyline

End If
```

Затем мы добавляем текущее значение переменной pGeometry в новую IGeometryCollection и сохраняем результирующую геометрию в IFeatureBuffer как новый объект.

```
pOGeometryCollection.AddGeometry pGeometry
Set pFeatBuff.Shape = pOGeometryCollection
```

После фиксации геометрии мы копируем атрибуты полей, используя установленное соответствие между полями. Помните, что массив, хранящий соответствия между полями, возвращает отрицательное число для полей, не найденных в итоговом классе объектов.

```
For i = 0 To pInputFC.Fields.FieldCount – 1

 nCorrFieldIndex = pFieldsCorrespondence(i)

 If nCorrFieldIndex >= 0 Then
  pFeatBuff.Value(
    nCorrFieldIndex) = pInputFeature. Value(i)
 End If

Next i
```

На следующем шаге задается новое значение поля ID, производится вставка объекта в результирующий класс и увеличивается счетчик, определяющий идентификатор следующего объекта.

```
pFeatBuff.Value(nOutputIDFieldIndex) = nGeometryNumber
pInsertFCursor.InsertFeature pFeatBuff
nGeometryNumber = nGeometryNumber + 1

Next j
```

В случае, если обрабатывается полигональный слой, делаем следующим образом:

```
Else
```

Поскольку мы знаем, что исходные объекты являются полигонами, запросим интерфейс IPolygon для исходной геометрии. Объявим также два массива для хранения внешних и внутренних колец.

```
Set pPolygon = pGeometryCollection
Dim pExtRings() As IRing
Dim pIntRings() As IRing
```

Чтобы задать основу разбиения геометрии, мы извлекаем все внешние кольца с помощью метода IPolygon.QueryExteriorRingsEx.

Одновременно в приведенном ниже блоке кода запрашивается количество таких колец.

```
Dim nExtRingsCount As Long
nExtRingsCount = pPolygon.ExteriorRingCount
ReDim pExtRings(nExtRingsCount)
pPolygon.QueryExteriorRingsEx nExtRingsCount, _
 pExtRings(0)
```

Массив, хранящий все внешние кольца текущего полигона, является основой для разделения геометрии на односвязные области. Каждое внешнее кольцо с полным набором своих внутренних колец образует новую геометрию.

```
For j = 0 To nExtRingsCount – 1
```

Создадим новую геометрию как новый объект класса Polygon и запросим у него интерфейс IGeometryCollection для размещения в нем всех необходимых колец, начиная с текущего внешнего кольца исходной геометрии.

```
Set pOGeometryCollection = New Polygon
pOGeometryCollection.AddGeometry pExtRings(j)
```

Затем дополним коллекцию соответствующими внутренними кольцами. Интерфейс IPolygon позволяет нам извлекать все внутренние кольца, находящиеся внутри данного внешнего кольца (а это именно то, что нам нужно!). Поэтому инициализируем для них новый массив.

```
Dim nIntRingsCount As Long
nIntRingsCount = pPolygon.InteriorRingCount(pExtRings(j))
ReDim pIntRings(nIntRingsCount)
pPolygon.QueryInteriorRingsEx pExtRings(j), _
 nIntRingsCount, pIntRings(0)
```

Поскольку все эти внутренние кольца находятся внутри текущего внешнего кольца, поместим их в новую коллекцию без какой-либо дополнительной сортировки и обработки.

```
For k = 0 To nIntRingsCount – 1

 pOGeometryCollection.AddGeometry pIntRings(k)

Next k
```

Наконец, наша геометрия подготовлена к сохранению в новом пространственном объекте через соответствующий интерфейс IFeatureBuffer.

```
Set pFeatBuff.Shape = pOGeometryCollection
```

Процесс копирования атрибутов уже обсуждался ранее, поэтому здесь мы оставляем его без комментариев.

```
For i = 0 To pInputFC.Fields.FieldCount – 1

 nCorrFieldIndex = pFieldsCorrespondence(i)

 If nCorrFieldIndex >= 0 Then
  pFeatBuff.Value(nCorrFieldIndex) =
   pInput Feature.Value(i)
 End If

Next i

pFeatBuff.Value(nOutputIDFieldIndex) = nGeometryNumber
```

Далее мы сохраняем новый полигональный объект, теперь уже с простой геометрией, но со всеми его внутренними кольцами.

```
    pInsertFCursor.InsertFeature pFeatBuff
    nGeometryNumber = nGeometryNumber + 1

  Next j

End If
```

Теперь надо еще проверить, не пытался ли пользователь отменить операцию. Если он не отменял операцию, то следующий объект будет извлечен с помощью курсора из исходного набора объектов.

```
bContinue = pTrackCancel.Continue

  If Not bContinue Then Exit Do

  Set pInputFeature = pInputFCursor.NextFeature

Loop
```

После завершения всех преобразований мы закрываем и уничтожаем диалог хода выполнения процесса. Затем мы уничтожаем также ссылки на объекты, реализующие интерфейсы IFeatureBuffer и IFeatureCursor, поскольку их объектные переменные лучше очистить явно (иначе это может привести к ошибкам, связанным с совместным доступом к данным).

```
pPDlg.HideDialog
Set pTrackCancel = Nothing
Set pStepPro = Nothing
Set pPDlg = Nothing
Set pInputFCursor = Nothing
Set pInsertFCursor = Nothing
Set pFeatBuff = Nothing
```

Во избежание простого дублирования данных важно также уведомить пользователя в том случае, если входной слой содержит только простые геометрии — их просто нет смысла преобразовывать.

```
If Not isMultipart Then

  MsgBox sLayerName & " has no multipart shapes.",
    vbExclamation, "XTools"
  Multipart2SinglePart = False
  Exit Function

End If
```

Затем мы переходим к последней операции — автоматическому подсчету площадей, периметров и длин. Вначале осуществляется проверка того, что установлены требуемые параметры в реестре, в противном случае выдается сообщение для пользователя.

```
If getstring(HKEY_CURRENT_USER, "Software\DataEast\
  XTools", "CalculateArea") = ""
   Then

  MsgBox "Please, set XTools Defaults", vbExclamation, "XTools"

End If
```

Для добавления нового класса пространственных объектов на карту нам необходимо создать новый слой и связать этим с классом объектов. Для того чтобы задать имя слоя в соответствии с именем файла класса объектов, воспользуемся свойством BrowseName интерфейса IDataset, запрашиваемого для этого источника данных.

```
Dim pNewFLayer As IFeatureLayer
Set pNewFLayer = New FeatureLayer
Set pNewFLayer.FeatureClass = pOutputFC
```

```
Dim pDataset As IDataset
Set pDataset = pNewFLayer
pNewFLayer.Name = pDataset.BrowseName
```

Получив слой, мы рассчитываем площадь/периметр новых объектов и добавляем слой на карту.

```
Dim sAreaCalculate As String
sAreaCalculate = getstring(HKEY_CURRENT_USER, _
  "Software\DataEast\XTools", "CalculateArea")

If (sAreaCalculate = "Yes") Then

 Dim pSubCalcSize As SubCalcSize
 Set pSubCalcSize = New SubCalcSize
 Set pSubCalcSize.m_pApp = m_pApp
 pSubCalcSize.CalcSize pNewFLayer

End If

pMap.AddLayer pNewFLayer

pDoc.ActiveView.Refresh
```

Вот и все. Мы разделили каждый объект с составной геометрией на объекты с геометрией, имеющей простую структуру.

Как работать с графическими объектами карты?

В ArcMap для работы с графическими элементами карты используются специальные графические слои, размещаемые поверх остальных слоев с данными. Библиотека ArcObjects обеспечивает программисту доступ к ним, а также позволяет экспортировать находящиеся в них графические примитивы. Полезно иметь инструмент, позволяющий экспортировать нарисованные пользователем объекты в шейп-файлы или сохранять некоторые другие

графические примитивы, получающиеся в результате работы программы. Например, команда Convert Polylines To Polygon Tool приложения XTools создает набор точек пересечения полилиний в графическом слое карты. Вы можете также использовать подобную технологию для собственных целей. Имеется масса других примеров, когда нецелесообразно создавать новый слой с данными и заполнять его объектами, такими как простые графические символы в определенных местах карты, и который не будет эксплуатироваться в будущем, а будет использоваться только для поддержки "общения" с пользователем.

Приложение XTools содержит для описанных выше целей пример процедуры для работы с графикой — команду Convert Graphics To Shape. Мы подробно рассмотрим ее код, поскольку данный инструмент предлагает гибкий способ экспорта графических элементов.

В нашем описании мы будем опускать те части кода, которые выполняют очевидные операции. На этот раз начнем рассмотрение с кода, ответственного за инициализацию диалоговой формы. Особенностью данного фрагмента программы является способ подсчета числа различных типов геометрий.

В самом начале мы объявляем и инициализируем наиболее важные переменные. Как вы можете заметить, ссылка на новый интерфейс IGraphicsContainer устанавливается с помощью свойства GraphicsContainer текущего вида документа (ActiveView). Этот интерфейс поможет нам в работе с графическими элементами, так как он обладает всей функциональностью контейнера объектов. Следует вновь подчеркнуть, что карта может содержать более одного графического слоя, поэтому не все операции с графическими объектами можно выполнять таким способом.

```
Dim pGraphicsContainer As IGraphicsContainer
Set pGraphicsContainer = pDoc.ActiveView.GraphicsContainer
pGraphicsContainer.Reset
Dim pElement As IElement
```

```
Dim pGeometry As IGeometry

Dim nPointCount As Long
Dim nPolylineCount As Long
Dim nPolygonCount As Long
nPointCount = 0
nPolylineCount = 0
nPolygonCount = 0
```

Далее мы извлекаем первый элемент из контейнера и проверяем его ссылку на пустоту, поскольку хороший стиль программирования требует информирования пользователя о пустом контейнере. Очевидно, форма не будет выводиться на экран, если контейнер графических объектов пуст.

```
Set pElement = pGraphicsContainer.Next

If (pElement Is Nothing) Then

  MsgBox "There are no graphics elements in the active view.", _
    vbExclamation, "XTools"
  Exit Sub

End If
```

Если в карте имеется хотя бы один графический элемент, то производится подсчет количества элементов, имеющих полигональный, полилинейный и точечный тип геометрии для предоставления пользователю соответствующей информации. Все, что мы должны сделать, это организовать перебор всех элементов контейнера, извлечение их геометрии и проверку ее типа. Соответствующие переменные увеличивают свое значение в цикле в соответствии с типом геометрии.

```
Do While Not pElement Is Nothing
  Set pGeometry = pElement.Geometry
```

```
    If pGeometry.GeometryType = esriGeometryPolygon Then
      nPolygonCount = nPolygonCount + 1

    ElseIf pGeometry.GeometryType = esriGeometryPolyline Then
      nPolylineCount = nPolylineCount + 1

    ElseIf pGeometry GeometryType = esriGeometryPoint Then
      nPointCount = nPointCount + 1
    End If

    Set pElement = pGraphicsContainer.Next
  Loop
```

Далее мы опустим строки кода, начиная с проверки на существование хотя бы одного подходящего элемента, до начала обработки текущего выбранного элемента из списка в диалоге формы.

Понятно, что после того, как пользователь выбрал типы геометрии для экспорта, нам следует спросить у пользователя, в каком файле сохранить соответствующие данные. С этой целью мы решили использовать стандартный для ArcGIS диалог GxDialog, поскольку он очень удобен и мы можем легко изменять набор поддерживаемых типов файлов благодаря гибкому механизму фильтров источников данных. Вы можете даже написать свой собственный фильтр для требуемого типа файла, реализуя простой интерфейс IGxObjectFilter.

Если вам просто необходим уже существующий фильтр, скажем, только для шейп-файлов, то вам нужно воспользоваться свойством ObjectFilter интерфейса IGxDialog. Но если необходимо иметь более одного фильтра, вы должны использовать другой интерфейс — IGxObjectFilterCollection. Хотя в данном случае мы имеем дело только с одним типом файлов и единственным фильтром, будет продемонстрирован второй подход. Иногда полезно придерживаться именно этого подхода, особенно если вы

планируете расширить набор поддерживаемых наборов данных (фильтров).

Итак, мы создаем новый объект класса GxDialog и устанавливаем параметры интерфейса IGxDialog, влияющие на его поведение и способ отображения.

```
Set pGxBrowseDialog = New GxDialog
pGxBrowseDialog.AllowMultiSelect = False
pGxBrowseDialog.ButtonCaption = "Save"
```

Далее мы запрашиваем интерфейс IGxObjectFilterCollection объекта GxDialog, чтобы установить поддержку фильтров, как упоминалось выше. Кроме того, мы создаем новый экземпляр класса GxFilterShapefiles для данных из шейп-файлов. Так как этот единственный фильтр является также фильтром, используемым по умолчанию, мы укажем это его свойство с помощью второго параметра метода AddFilter интерфейса IGxObjectFilterCollection.

```
Dim pGxFilter As IGxObjectFilter
Dim pGxFilterCol As IGxObjectFilterCollection

Set pGxFilterCol = pGxBrowseDialog
pGxFilterCol.RemoveAllFilters
Set pGxFilter = New GxFilterShapefiles
pGxFilterCol.AddFilter pGxFilter, True
```

Затем мы анализируем текущий выбранный элемент списка формы и выясняем тип геометрии, обрабатываемой в данный момент, и соответственно изменяем заголовок диалога GxDialog. Кроме того, параллельно сохраняем информацию о типе геометрии объектов слоя в более удобной форме (переменная nLayerType была объявлена как esriGeometryType) для будущих ссылок на нее.

```
If pGraphics2ShapeForm.GeometryList.Item(i) = _
  CStr(nPolygonCount) & " graphic polygons." Then
```

```
    pGxBrowseDialog.Title = "Name new Polygon layer"
    nLayerType = esriGeometryPolygon
  ElseIf pGraphics2ShapeForm.GeometryList.Item(i) = _
    CStr(nPolylineCount) & " graphic polylines." Then
    pGxBrowseDialog.Title = "Name new Polyline layer"
      nLayerType = esriGeometryPolyline
  ElseIf pGraphics2ShapeForm GeometryList.Item(i) = _
    CStr(nPointCount) & " graphic points." Then
    pGxBrowseDialog.Title = "Name new Point layer"
    nLayerType = esriGeometryPoint
  End If
```

После процедуры инициализации мы выводим диалоговое окно, используя приложение ArcMap как родительское окно (это соотношение можно установить с помощью параметра hWnd). Важно правильно задать родительское окно для диалога с тем, чтобы избежать различных сбоев в работе модального диалога, таких, например, как его внезапное исчезновение с экрана.

Метод DoModalSave помогает пользователю специфицировать имя нового файла и возвращает значение типа Boolean, указывающее, было ли выбрано некоторое имя файла или нет. Если была нажата кнопка Cancel или диалог был закрыт, этот метод возвращает значение False. Вот почему происходит выход из текущей процедуры, если возвращается значение False.

```
  If pGxBrowseDialog.DoModalSave(
    m_pApp.hWnd) = False Then Exit Sub
```

Полный путь к новому файлу данных может быть получен достаточно очевидно: с помощью двух составляющих: пути к папке (свойство FinalLocation.FullName) и имени файла (свойство Name).

```
  If (Right( _
    pGxBrowseDialog.FinalLocation.FullName, 1) = "\") Then
```

```
  sNewLayerFullName = pGxBrowseDialog.FinalLocation.
    FullName &
    pGxBrowseDialog.Name
Else
  sNewLayerFullName = pGxBrowseDialog.FinalLocation.
    FullName & "\" &      pGxBrowseDialog.Name
End If
```

Класс GxDialog может также облегчить вам работу с файлами, уведомляя вас о том, должен ли быть заменен текущий выбранный объект или создан новый, т. е. существует ли уже некоторый файл с заданным именем. Мы создали простую процедуру RemoveAccordingToShpPath, которая просматривает слои с векторными данными в пределах документа и удаляет слой, у которого путь к данным совпал с выбранным пользователем. Этот шаг необходим, если вы хотите, чтобы ваши данные в карте остались корректными

После того, как старый слой удален из карты, вы можете открыть его класс пространственных объектов как набор данных (IDataset) и удалить его с помощью метода Delete. Этот подход применим почти к каждому источнику данных, который может быть описан как отдельный набор данных. Поэтому вы также можете применить его к таблице, TIN или даже всему рабочему пространству.

```
If pGxBrowseDialog.ReplacingObject Then
  RemoveAccordingToShpPath sNewLayerFullName, m_pApp

  Set pDataset = OpenShapeFile( _
    sNewLayerFullName). FeatureClass
  pDataset.Delete
End If
```

Далее мы опускаем описание процесса создания нового класса объектов, поскольку он был детально описан в предыдущих разделах.

```
...
Set pOutputFC = CreateShapefile(...)
...
```

Единственным отличием между кодом следующего фрагмента и кодом описанных в предыдущих частях фрагментов является добавление нового поля в класс объектов после его создания вместо формирования набора полей до создания класса. Метод AddField интерфейса IFeatureClass позволяет выполнить такую операцию. Обратите внимание, что операция добавления коллекции полей при создании класса объектов эффективнее, чем операция добавления полей с помощью AddField, поскольку уже существующий класс требует внутренней перестройки каждый раз, когда в него добавляется новое поле, тогда как объект рабочего пространства создает класс объектов один раз для всего набора полей.

Более того, вам будет полезно узнать, что существуют некоторые другие интерфейсы, поддерживающие добавление полей в текущий набор данных. Фактически все классы и таблицы наследуют этот метод из интерфейса IClass. Поэтому вы можете использовать интерфейсы IClass, IObjectClass и ITable, если ваша задача не допускает использования интерфейса IFeatureClass, например, при использовании объектов таблиц.

```
If (pGraphics2ShapeForm.AddField = vbChecked) Then

  Dim pTxtFld As IField
  Dim pETxtFld As IFieldEdit
  Set pTxtFld = New Field
  Set pETxtFld = pTxtFld

  With pETxtFld
    .Name = "Text"
    .AliasName = "Text"
    .Editable = True
    .Type = esriFieldTypeString
```

```
    End With

    pOutputFC.AddField pETxtFld

  End If
```

Следующим интересным моментом является изменение курсора мыши. Иногда очень важно информировать пользователя о выполняющихся в программе расчетах с помощью вывода на экран курсора мыши в форме часов. Для этого в ArcObjects имеются специальные средства: интерфейс IMouseCursor и соответствующий класс MouseCursor. Как только вы создали новый MouseCursor, он самостоятельно связывает себя с текущей информацией о курсоре, поэтому вы легко можете задать свой собственный курсор (за дополнительной информацией обращайтесь к справочной системе ArcObjects).

```
  Dim pMouseCursor As IMouseCursor
...

  Set pMouseCursor = New MouseCursor
  pMouseCursor.SetCursor 2
```

Следующий шаг некоторым образом повторяет начальный фрагмент функции. Организуется просмотр всей коллекции геометрий элементов графики и помещение подходящих (в соответствии с типом геометрии) элементов в созданный класс объектов.

```
  Set pElement = pGraphicsContainer.Next

  Do While Not pElement Is Nothing

    Set pGeometry = pElement.Geometry
```

Здесь мы пользуемся информацией, о ранее полученном типе геометрии. К сожалению, другого способа получения элементов фиксированного геометрического типа не существует.

```
If (pGeometry.GeometryType = nLayerType) Then
```

Если текущий элемент содержит требуемый тип геометрии, то его геометрия клонируется, проектируется и сохраняется в буфере для записи.

```
Set pClone = pGeometry
Set pGeometry = pClone.Clone
pGeometry.Project pSR

Set pFeatBuf.Shape = pGeometry
pFeatBuf.Value(nIdIndex) = nCounter
```

Если пользователь выбрал в диалоге опцию создания дополнительного текстового поля, то мы проверяем, является ли текущий элемент графики текстом или не является. Если является, то текст, который он содержит, вставляется в это поле. При этом используется специальный интерфейс ITextElement, описывающий поведение текстовых элементов. Таким способом мы добиваемся, чтобы текст метки сохранялся параллельно с точкой ее привязки.

```
If (bHaveTxtFld) Then

  If (TypeOf pElement Is ITextElement) Then
    Dim pTxtElem As ITextElement
    Set pTxtElem = pElement
    pFeatBuf.Value(nTextField) = pTxtElem.Text
  Else
    pFeatBuf.Value(nTextField) = ""
  End If

End If
```

Остальной код, ответственный за перебор элементов, выглядит вполне обычно, и мы оставляем его без комментариев.

```
    pInsertFCursor.InsertFeature pFeatBuf
    nCounter = nCounter + 1

  End If

  Set pElement = pGraphicsContainer.Next

Loop
```

Напоминаем, что часто бывает необходимо уничтожить объекты буферов и курсоров после выполнения обработки. Эта ошибка может стоить вам потери нескольких часов на отладку вследствие возникающих проблем с доступом к данным (верно только для случая обращения к тем же данным в рамках этой же функции). Visual Basic сам удалит ссылки, когда они выйдут за пределы области видимости соответствующих переменных.

Помните также о необходимости восстановления обычного вида курсора мыши, так как это сильно влияет на отношение пользователя к вашей программе.

```
Set pFeatBuf = Nothing
Set pInsertFCursor = Nothing

  ..

pMouseCursor.SetCursor 0

...
```

Вот таким образом мы использовали графический слой карты в нашем приложении.

Как выполнять пространственные преобразования и проверять пространственные отношения между объектами?

При работе с геометриями пространственных объектов любой пользователь сталкивался с необходимостью выполнения ряда

топологических операций. Операции пересечения, объединения, разности и др. лежат в основе анализа пространственных перекрытий объектов, демографического анализа, создания сложных пространственных элементов и т. д. Операция Simplify позволяет приводить геометрию объектов к топологически корректному виду. Особенно часто приходится применять операцию Simplify при различных преобразованиях и анализах геометрических форм. Ниже мы рассмотрим эту операцию наряду с другими.

Помимо топологических операций, помогающих конструировать новые формы из уже имеющихся, во многих задачах требуется также анализ взаимного расположения геометрий, т. е. определение некоторых пространственных соотношений. Например, правильнее будет проверить наличие пересечения между двумя геометриями перед непосредственным нахождением пересечения между ними во избежание появления ошибок, связанных с пустой геометрией.

Выполнение всех топологических операций обеспечивается интерфейсами ITopologicalOperator и ITopologicalOperator2 на объектах геометрии. Мы часто использовали эти интерфейсы при создании расширения XTools, однако наиболее важные в этом отношении части кода находятся внутри трех команд приложения: Erase Features, Identity и Update Polygon Layer. Мы не будем рассматривать код последней команды, поскольку она по своей природе является комбинацией двух первых команд.

Займемся изучением команды Erase Features. Поскольку многие вопросы, связанные с созданием нового слоя, добавлением нового поля и т. п., уже были обсуждены выше, перейдем непосредственно к предмету нашего исследования, оставляя кое-где некоторые части программы в виде псевдокода для наглядности.

Итак, данная команда получает на вход два слоя, один из которых (мы назвали его "отсекающий слой" — pEraseLayer) служит для удаления объектов из другого слоя (pInputLayer — исходного

слоя, по нашей терминологии). Третий, новый слой создается для сохранения результатов операции Erase и структурно базируется в основном на исходном слое.

```
...
Set pInputLayer = {a layer to produce erase on}
Set pEraseLayer = {polygonal layer to make erase with}
...
Set pOutputFC = CreateShapefile(...)
...
```

Первый шаг состоит в объявлении переменных, необходимых для выполнения топологических операций. Мы объявляем две переменные, как ссылки на интерфейсы ITopologicalOperator и ITopologicalOperator2. На первый взгляд кажется излишним иметь ссылки на два практически одинаковых интерфейса, поскольку ITopologicalOperator2 наследует ITopologicalOperator. Но имеется одно фундаментальное различие: класс Point, который не поддерживает интерфейс ITopologicalOperator2, тогда как классы других типов геометрии используют этот интерфейс для более корректного проведения операции Simplify.

```
Dim pTopologicalOperator As ITopologicalOperator
Dim pTO2 As ITopologicalOperator2
Dim pGeometryCollection As IGeometryCollection
Dim pOutGeometryCollection As IGeometryCollection

...
```

Для того чтобы провести отсечение геометрий объектов исходного слоя, перекрывающихся с геометриями объектов "отсекающего" слоя, мы создадим топологическое объединение геометрий отсекающих объектов. Можно было бы воспользоваться методом ITopologicalOperator.Union, однако он способен объединить только две геометрии и неэффективен при объединении большого числа геометрий. Поэтому мы будем использовать метод

ConstructUnion, поскольку он поддерживает объединение большого числа геометрий за один прием. Единственным недостатком такого подхода является невозможность отслеживания хода выполнения процесса объединения средствами Visual Basic. Фактически, на этом языке почти невозможно запустить отдельный поток в программе для операции объединения геометрий, поэтому данный метод, в случае большого числа объектов, может приводить к остановке вашего приложения на длительное время. Поэтому иногда, с целью повышения производительности и обеспечения дружественности интерфейса пользователя, следует писать приложения в среде Visual C++.

Возвращаясь к предмету нашего изучения, сконструируем новый объект EnumFeatureGeometry, который реализует интерфейс IEnumGeometry , необходимый для работы с коллекцией геометрий. Для преобразования геометрий выделенного набора объектов слоя в перечень геометрий нам нужно запросить интерфейс IEnumGeometryBind.

```
Dim pEnumGeometry As IEnumGeometry
Set pEnumGeometry = New EnumFeatureGeometry
Dim pEnumGeometryBind As IEnumGeometryBind
Set pEnumGeometryBind = pEnumGeometry
```

Затем мы воспользуемся этим интерфейсом для создания коллекции геометрий, используя либо текущий набор выбранных объектов слоя, либо специальный запрос к классу пространственных объектов в зависимости от ситуации. В обоих случаях будет применяться метод BindGeometrySource. В качестве первого параметра он принимает фильтр запроса, а в качестве второго параметра — набор выделенных объектов или весь класс пространственных объектов. Таким образом, этот метод поможет нам создать коллекцию геометрий объектов независимо от способа отбора самих объектов.

```
If isEraseSelection Then
  pEnumGeometryBind.BindGeometrySource Nothing, _
    pEraseSelSet
Else
  pEnumGeometryBind.BindGeometrySource pQF, _
    pEraseLayer.FeatureClass
End If
```

Наконец, мы можем создать новый полигон, как объединение всей коллекции геометрий "отсекающих" объектов.

```
Set pTopologicalOperator = New Polygon
pTopologicalOperator.ConstructUnion pEnumGeometry
```

Теперь, когда у нас есть единый пространственный "критерий отсечения" в виде одного полигона, мы можем начать обработку. Прежде всего мы должны запросить другой интерфейс у нашего полигона: IRelationalOperator, который поможет нам установить, в каких пространственных отношениях находятся этот полигон и геометрии объектов исходного слоя.

```
Set pRelationalOperator = pTopologicalOperator
```

Поскольку мы собираемся анализировать пространственные отношения между созданным объединением геометрий и геометриями исходного слоя, то нам нужно сделать такие геометрии топологически корректными. Это также довольно продолжительная по времени операция, но она помогает предотвратить сбои, связанные с обработкой сложных, неправильно построенных геометрий. Если вы хотите получить дополнительную информацию об операции Simplify, прочитайте соответствующий раздел в справочной системе разработчика ArcObjects.

Необходимо особо подчеркнуть важность предварительной установки значения свойства ITopologicalOperator2.IsKnownSimple в False. Это свойство указывает, что данная геометрия корректна,

т. е. даже некорректные геометрии с установленным значением этого свойства в True не будут подвергаться операции топологического корректирования. Поскольку этот случай вполне обычен, вы должны каждый раз вручную изменять это свойство, когда хотите упростить некоторую геометрию.

```
...
Set pTO2 = pTopologicalOperator
pTO2.IsKnownSimple = False

If Not pTopologicalOperator.IsSimple Then

  pTopologicalOperator.Simplify

End If
```

Еще надо не забыть, что мы должны спроектировать данные в одну систему координат. Для этой цели использование объединения геометрии объектов в виде одного полигона предпочтительнее, поскольку для него нужно выполнить только одну операцию проектирования, в отличие от необходимости проектирования каждого "отсекающего" объекта по отдельности.

```
Set pGeometry = pTopologicalOperator
pGeometry.Project pSR
```

Процесс создания курсора для перебора записей исходного слоя достаточно очевиден и приводится здесь без комментариев.

```
If isInputSelection Then

  pInputSelSet.Search pQF, True, pFeatureCursor

Else

  Set pFeatureCursor = pInputLayer.Search(pQF, True)
```

```
End If

Set pInsertCursor = pOutputFC.Insert(True)
```

Без комментариев остается и начало цикла перебора объектов.

```
Dim j As Long
Dim nUpperLimit As Long

Set pFeature = pFeatureCursor.NextFeature

Do While (Not pFeature Is Nothing)

  Set pGeometry = pFeature.ShapeCopy
```

Сразу же сделаем запрос интерфейса ITopologicalOperator у геометрии исходного объекта (помните, что мы должны от нее что-то отсечь).

```
Set pTopologicalOperator = pGeometry
```

Поскольку точечные объекты не нуждаются в операции топологической коррекции и не поддерживают интерфейс ITopological Operator2, следует избегать применения к ним следующей процедуры. Для других типов геометрии мы проводим коррекцию с той же целью, что и для построенного выше полигона.

```
If pGeometry.GeometryType <> esriGeometryPoint Then

  Set pTO2 = pTopologicalOperator
  pTO2.IsKnownSimple = False

  If Not pTopologicalOperator.IsSimple Then

    pTopologicalOperator.Simplify
```

```
    End If

  End If
```

Перебор геометрий объектов включает в себя также проверку на наличие по крайней мере одного объекта для отсечения. Иначе нужно выдавать сообщение пользователю о том, что в результате операции никаких преобразований не произошло. Поэтому в цикле организуется проверка на пересечение между геометрией текущего объекта и "отсекающим" полигоном. При этом проверяются любые типы пересечений и используется метод ITopologicalOperator.Disjoint, возвращающий значение True при отсутствии любых пересечений, т. е. геометрии пространственно разъединены

```
  If (Not isErase) Then

    If (Not pRelationalOperator.Disjoint(pGeometry)) Then

      isErase = True

    End If

  End If
```

Далее происходит собственно "отсечение". Для повышения производительности геометрия исходного объекта проверяется на то, что она не содержится целиком внутри "отсекающего" полигона, поскольку такие объекты "обречены" на удаление. Эта проверка также выполняет собственно всю работу по отсечению в случае, если исходный слой имеет точечную геометрию

```
  If Not pRelationalOperator.Contains(pGeometry) Then
```

Геометрия исходного объекта проверяется также на то, что она не является пустой, поскольку в противном случае могут возникнуть проблемы с применением топологических операций.

```
If Not pGeometry.IsEmpty Then
```

В заключение вновь организуется проверка на то, что мы работаем с неточечной геометрией, поскольку метод IRelational Operator.Contains уже ответил нам на вопрос, должен ли быть отмечен текущий точечный объект или нет. Объекты с другими типами геометрии могут иметь пересечения с "отсекающим" полигоном, и поэтому надо находить геометрическую разность (кстати, непустую, так как проверку на наличие пересечения и на полную принадлежность мы уже провели), которая и будет сохранена в качестве геометрии итогового объекта.

```
    If pGeometry.GeometryType <> esriGeometryPoint Then

     Set pGeometry = pTopologicalOperator.Difference( _
      pRelationalOperator)

    End If
```

Далее мы опустим вопрос о разбиении получившейся составной геометрии на простые части, а также действия, связанные с сохранением объектов. По этим вопросам обращайтесь к соответствующим частям данной главы.

```
    ...
    pInsertCursor.InsertFeature pFeatBuf

...
Set pFeature = pFeatureCursor.NextFeature
...

Loop
```

```
Set pFeatBuf = Nothing
Set pInsertCursor = Nothing

...
```

Таким образом, мы рассмотрели все ключевые моменты кода команды Erase Features.

Другой интересный и более сложный способ обработки геометрии объектов заключен в команде XTools Identity. Мы снова опускаем весь вспомогательный код и переходим непосредственно к геометрическим преобразованиям. Взятые нами названия переменных для ссылок на слои подразумевают, что в операции Identity участвуют два слоя — исходный и идентифицирующий, из которого будут добавляться данные во всевозможные пересечения с объектами исходного слоя.

```
...
Set pInputLayer = (the layer to execute identity operation on)
Set pOverlayLayer = (the layer for performing identity)
...
Set pOutputFC = CreateShapefile(...)
```

Мы повторим здесь описание процесса создания полей и установки соответствия между старым и новым наборами полей, поскольку эта процедура будет отличаться от ранее описанных и в ее основе лежит более универсальный подход. В данной ситуации мы сталкиваемся с двумя проблемами: мы не знаем заранее формулу соответствия между полями слоев (мы точно не знаем даже списка полей, которые нужно добавить к полям исходного слоя перед началом процедуры); и мы должны решить проблему совпадающих имен полей в независимых исходных коллекциях, так как нам нужно объединить эти коллекции в одну.

Прежде всего, необходимо получить коллекции имен новых полей и объявить для этого несколько переменных. Обратите внимание на массив pNewFieldNames, который в случае необходимости будет хранить новые имена полей.

```
Dim pNewFieldNames() As String
Dim pInputFieldsList As Collection
Dim pOverlayFieldsList As Collection
Set pInputFieldsList = {collection of the field names for the
  source layer for processing specified by user for adding
  to the new file}
Set pOverlayFieldsList = {collection of the field names for the
  identity accomplishing layer specified by user for adding to
  the new file}

. . .

Dim pFieldsCorrespondence() As Long
Dim pFields As IFields

Dim i As Long
Dim pField As IField

...
```

Для установки соответствия между полями исходных слоев и слоя результата мы используем один массив, причем его размер должен соответствовать сумме общего числа полей в каждом из исходных слоев.

```
ReDim pFieldsCorrespondence( _
  pInputLayer.FeatureClass. Fields.FieldCount + _
  pOverlayLayer.FeatureClass.Fields.FieldCount) As Long
```

```
For i = 0 To pInputLayer.FeatureClass.Fields.FieldCount +
  pOverlayLayer.FeatureClass.Fields.FieldCount – 1
 pFieldsCorrespondence(i) = –1
Next i
```

Далее происходит копирование первой порции выбранных пользователем полей исходного слоя. Обратите внимание на то, что для каждого выбранного поля в массиве соответствия устанавливается значение –2. Это повысит скорость обработки и даст возможность избежать дополнительных проверок, поскольку выбранные пользователем поля будут явно помечены. Заметьте также, что мы не копируем целиком весь набор входных полей, а только выбранное подмножество.

```
For i = 0 To pInputFieldsList.Count – 1

 nIndex = pInputLayer.FeatureClass.Fields.FindField( _
  pInputFieldsList.Item(i + 1))
 pFieldsCorrespondence(nIndex) = –2
 Set pClone = pInputLayer.FeatureClass.Fields.Field(Index)
 pOutputFC.AddField pClone.Clone

Next i
```

Далее мы инициализируем массив для тех полей, у которых изменились имена. Очевидно, мы должны это сделать только для второго слоя (если мы работаем с шейп-файлами), поскольку оба слоя содержат свои уникальные корректные наборы полей. Это значит, что возможна только одна проблема — размещение полей идентифицирующего слоя в результирующем слое, уже заполненном полями исходного слоя. Следовательно, нам нужно будет изменить имена полей идентифицирующего слоя, совпадающих с именами полей исходного, при переносе их в результирующий класс объектов.

```
Dim pUpdateField As IField
ReDim pNewFieldNames( _
  pOverlayLayer.FeatureClass.Fields.FieldCount) As String

For i = 0 To pOverlayLayer.FeatureClass.Fields.FieldCount – 1
  pNewFieldNames(i) = ""
Next i
```

После этих приготовлений мы помещаем новые поля в результирующий класс пространственных объектов. Вы можете воспользоваться приведенными ниже приемами, если необходимо добавить поля в класс объектов и изменить их имена.

```
Dim pFieldEdit As IFieldEdit
Dim sTrueFieldName As String

For i = 0 To pOverlayFieldsList.Count – 1
```

Вначале выполняется последовательность операций по клонированию поля: получение индекса для текущего поля из списка выбранных пользователем в идентифицирующем классе объектов, пометка его как существующего (–2) в массиве, поддерживающем соответствие между полями, и, наконец само клонирование.

```
nIndex = pOverlayLayer.FeatureClass.Fields.FindField(
 pOverlayFieldsList.Item(i + 1))
pFieldsCorrespondence(
  index + pInputLayer.FeatureClass.Fields.FieldCount) = –2
Set pClone = pOverlayLayer.FeatureClass.Fields.Field(nIndex)
Set pUpdateField = pClone.Clone
```

Далее мы проверяем, следует ли изменить имя поля, т. е. имеет ли уже результрующий класс объектов поле с таким же именем. Если да, то выполняем код, решающий эту проблему.

```
If pOutputFC.Fields.FindField(pUpdateField.Name) >= 0 Then
```

Мы воспользуемся функцией UniqueFieldName, возвращающей новое уникальное имя поля на основе текущего имени и заданного набора полей. Эта функция добавляет некоторое число в конец имени поля и увеличивает это суффикс до тех пор, пока имя не станет уникальным для заданной коллекции полей. После такой процедуры вы можете безопасно поместить поле в результирующий класс объектов. Поскольку в подобных случаях мы должны запомнить поле, подлежащее добавлению, под другим именем, соответствующий элемент массива pNewFieldNames будет содержать это новое уникальное имя.

Кроме того, необходимо задать псевдоним для поля, так как обычно пользователь не предполагает увидеть измененные имена полей вместо хорошо знакомых ему старых. Поэтому мы создаем псевдоним, содержащий старое имя поля. Вы можете также использовать этот подход вместо использования массивов и коллекций для поддержки изменений имен, однако не стоит полностью полагаться на псевдонимы, так как правила именования псевдонимов менее строгие, чем для имен полей.

```
    Set pFieldEdit = pUpdateField
    sTrueFieldName = pUpdateField.Name
    pFieldEdit.Name = UniqueFieldName(
      pOutputFC.Fields, pUpdateField.Name)
    pNewFieldNames(nIndex) = pUpdateField.Name
    pOutputFC.AddField pUpdateField
    Set pFieldEdit = pOutputFC.Fields.Field(
      pOutputFC.Fields.FindField(pUpdateField.Name))
    pFieldEdit.AliasName = sTrueFieldName

  Else

    pOutputFC.AddField pUpdateField

  End If

Next i
```

На заключительном шаге устанавливается соответствие между полями. Первая часть массива pFieldsCorrespondence будет хранить соответствия индексов для полей входного класса, тогда как вторая часть будет определять соответствие с полями идентифицирующего класса. Поскольку теперь отсутствуют совпадения имен между полями результирующего класса объектов, эта процедура будет несложной.

Для каждого поля исходного слоя мы будем проверять, нужно ли переносить его в результирующий класс, с помощью значения переменной pFieldsCorrespondence для его индекса в исходном классе. В случае утвердительного ответа мы ищем его индекс в результирующем классе объектов.

```
Set pFields = pInputLayer.FeatureClass.Fields

For i = 0 To pFields.FieldCount – 1

  If pFieldsCorrespondence(i) = –2 Then
    pFieldsCorrespondence(i) = pOutputFC.Fields.FindField( _
      pFields.Field(i).Name)
  End If

Next i
```

Работа с полями идентифицирующего слоя отличается только способом отслеживания изменений имен полей. Если текущее поле было выбрано пользователем и добавлено в результирующий набор, тогда инициируется специальная процедура для модификации имени поля. В случае возвращения массивом pNewFieldNames непустой строки это будет означать, что ее надо использовать в качестве реального имени поля в результирующем классе вместо имени в идентифицирующем классе.

```
Set pFields = pOverlayLayer.FeatureClass.Fields
```

```
For i = 0 To pFields.FieldCount - 1

 If pFieldsCorrespondence(
  pInputLayer.FeatureClass.Fields.FieldCount + i) = -2 Then

  If pNewFieldNames(i) = "" Then

   pFieldsCorrespondence(
    pInputLayer.FeatureClass.Fields.FieldCount + i) = _
   pOutputFC.Fields.FindField(pFields.Field(i).Name)

  Else

   pFieldsCorrespondence(
    pInputLayer.FeatureClass.Fields.FieldCount + i) = _
   pOutputFC.Fields.FindField(pNewFieldNames(i))

  End If

 End If

Next i
```

Следующий фрагмент кода мы приводим исключительно для сохранения целостности изложения. Поскольку он включает в себя уже рассмотренные ранее шаги, мы оставили его без комментариев.

```
Dim pFeatureCursor As IFeatureCursor
Dim pOverlayCursor As IFeatureCursor
Dim pInsertCursor As IFeatureCursor
Dim pOverlayFeature As IFeature
Dim pFeatBuf As IFeatureBuffer
Dim pGeometry As IGeometry
Dim pSF As ISpatialFilter
Set pSF = New SpatialFilter
Dim pFeature As IFeature
```

```
Dim pDataset As IDataset
Dim pRelationalOperator As IRelationalOperator

Dim isNoIdentity As Boolean
isNoIdentity = True

If isInputSelection Then

  pInputSelSet.Search pQF, True, pFeatureCursor

Else

  Set pFeatureCursor = pInputLayer.Search(pQF, True)

End If

Set pInsertCursor = pOutputFC.Insert(True)

Set pFeatBuf = pOutputFC CreateFeatureBuffer

Dim nTotalCount As Long
Dim nActualCount As Long
nTotalCount = pInputLayer.FeatureClass FeatureCount(pQF)

If isInputSelection Then
  nActualCount = pInputSelSet.Count
Else
  nActualCount = nTotalCount
End If

nCounter = 0
Dim nCorrespondingField As Long
Dim pTopologicalOperator As ITopologicalOperator2

Dim pGeometryCollection As IGeometryCollection
Set pGeometryCollection = New Polygon
```

...

Далее мы снова должны создать уже встречавшееся нам ранее объединение геометрий, так как операция Identity предполагает две различные ситуации: работу с частью исходного объекта, которая имеет пересечение с идентифицирующим слоем, или наоборот. Поэтому мы выполним шаги, уже обсуждавшиеся выше для предыдущей команды приложения, включающие создание коллекции геометрий, построение объединения, проецирование и коррекцию.

```
Dim pEnumGeometry As IEnumGeometry
Set pEnumGeometry = New EnumFeatureGeometry
Dim pEnumGeometryBind As IEnumGeometryBind
Set pEnumGeometryBind = pEnumGeometry

If isOverlaySelection Then
  pEnumGeometryBind.BindGeometrySource Nothing,
    pOverlaySelSet
Else
  pEnumGeometryBind.BindGeometrySource pQF,
    pOverlayLayer
End If

Set pTopologicalOperator = New Polygon
pTopologicalOperator.ConstructUnion pEnumGeometry

Set pGeometry = pTopologicalOperator
pGeometry.Project pSR

pTopologicalOperator.IsKnownSimple = False

If Not pTopologicalOperator.IsSimple Then
  pTopologicalOperator.Simplify
End If
```

```
Set pRelationalOperator = pTopologicalOperator
Dim pIntersectedGeometry As IGeometry
Dim pOverlayGeometry As IGeometry
```

Для построения пересечений нам необходимо получить размерность геометрии объектов входного слоя. Позже мы обсудим этот шаг. Затем мы начинаем перебор объектов исходного слоя и выполнение операции Identity.

```
Dim nDimension As esriGeometryDimension

Set pFeature = pFeatureCursor.NextFeature

If Not pFeature Is Nothing Then
  nDimension = pFeature.Shape.Dimension
End If

Do While (Not pFeature Is Nothing)
```

Есть два способа создания набора пересечений между геометриями исходного и идентифицирующего слоев в задаче Identity. Вы можете организовать перебор всех объектов идентифицирующего класса для каждого объекта исходного класса либо вы можете перебрать только те объекты идентифицирующего класса, которые пересекают текущий объект исходного класса. Очевидно, второй подход обеспечивает лучшую производительность, и поэтому мы воспользуемся именно им. Мы инициализируем новый объект-фильтр для определения пространственного запроса к слою и поместим в него геометрию текущего объекта исходного слоя как геометрию, с которой надо найти пересечение.

Если вы раньше не работали с пространственными фильтрами, запомните этот способ, поскольку он позволяет вам комбинировать атрибутивный запрос с пространственным.

Поскольку нам не нужно иметь свою копию геометрии, мы извлекаем ее из свойства IFeature.Shape (а не IFeature.ShapeCopy).

```
Set pGeometry = pFeature.Shape

Set pSF.Geometry = pGeometry
pSF.GeometryField = pInputLayer.FeatureClass.Shape
  FieldName
pSF.SpatialRel = esriSpatialRelIntersects
```

Далее обычным способом получаем новый курсор для перебора объектов, но теперь для этого используем пространственный фильтр в качестве критерия запроса. Это небольшое изменение дает в результате существенный прирост производительности.

```
If isOverlaySelection Then
  pOverlaySelSet.Search pSF, True, pOverlayCursor
Else
  Set pOverlayCursor = pOverlayLayer.Search(pSF, True)
End If

Set pOverlayFeature = pOverlayCursor.NextFeature
```

В следующей строке мы модифицируем процедуру обработки ошибок. Иногда бывает полезно ограничить диапазон влияния ошибки некоторой локальной областью кода во избежание сбоя всей процедуры, например, если некоторый объект имеет некорректную геометрию (иногда это происходит даже с неповрежденными данными). Смысл этого обработчика заключается в установке флага bIgnoreGeometry и отмене всех прочих операций с данным объектом.

```
On Error GoTo FeatureErrorHandler
```

Итак, начнем процедуру идентификации объектов. Здесь мы сбросим флаг bIgnoreGeometry, поскольку еще неизвестно, является ли текущая геометрия некорректной.

```
Do While (Not pOverlayFeature Is Nothing)

  bIgnoreGeometry = False
  Set pOverlayGeometry = pOverlayFeature.ShapeCopy
```

Далее мы проверяем, что идентифицирующая геометрия является непустой и двухмерной (размерности 2), поскольку этот шаг является необходимым для правильного выполнения операции пересечения.

```
If (Not pOverlayGeometry.IsEmpty) And ( _
  pOverlayGeometry.Dimension = esriGeometry2Dimension)
  Then

  ...
  Set pTopologicalOperator = pOverlayGeometry
  ...
```

Наконец, мы можем выполнить операцию пересечения геометрий. С этой целью мы определяем геометрию, с которой должна производиться операция пересечения, и размерность. Размерность задается равной размерности исходного слоя с помощью полученного ранее значения. Очевидно, она соответствует ожидаемой размерности, т. е. идентификация полилинейного слоя с помощью полигонального слоя дает в результате полилинейный слой и т. д.

```
Set pIntersectedGeometry = pTopological
  Operator.Intersect(pGeometry, nDimension)
```

Если пересечение не пустое и при этом не возникли ошибки (на это указывает флажок bIgnoreGeometry), мы можем начать копирование атрибутивной информации.

```
...

If (Not bIgnoreGeometry) And ( _
  Not pIntersectedGeometry.IsEmpty) Then
```

Не будет никаких проблем также скопировать атрибуты с помощью уже построенного массива соответствия.

```
...

Set pFields = pInputLayer.FeatureClass.Fields

For i = 0 To pFields.FieldCount – 1

 nCorrespondingField = pFieldsCorrespondence(i)

 If nCorrespondingField >= 0 Then

  pFeatBuf.Value( _
   nCorrespondingField) = pFeature.Value(i)

 End If

Next i

Set pFields = pOverlayLayer.FeatureClass.Fields

For i = 0 To pFields.FieldCount – 1

 nCorrespondingField =
  pFieldsCorrespondence( _
   pInputLayer.FeatureClass.Fields.FieldCount + i)

 If nCorrespondingField >= 0 Then

  pFeatBuf.Value( _
   nCorrespondingField) = pOverlayFeature.Value(i)

 End If

Next i
```

Сохраняем новый пространственный объект.

```
        pInsertCursor.InsertFeature pFeatBuf
      End If
    End If

    Set pOverlayFeature = pOverlayCursor.NextFeature
  Loop
  bIgnoreGeometry = False
```

Последнее, что мы делаем, — это обрабатываем уже рассмотренную ранее ситуацию, связанную с обработкой не перекрытых частей геометрии входного объекта. При этом будем различать случай точечной геометрии и остальные случаи, поскольку первый не требует расчета геометрической разности и эта операция может не выполняться. И вновь, для этого типа геометрии мы заменяем топологическую операцию проверкой пространственных соотношений (метод IRelationalOperator.Overlaps проверяет, перекрывает ли оператор данную геометрию).

```
  If (pGeometry.GeometryType <> esriGeometryPoint) Then

    Set pTopologicalOperator = pGeometry

    On Error GoTo NonSimpleErrorHandler

    Set pIntersectedGeometry = pTopological
      Operator.Difference(pRelationalOperator)

    On Error GoTo FeatureErrorHandler
```

```
ElseIf Not pRelationalOperator.Overlaps(pGeometry) Then

  Set pClone = pGeometry
  Set pIntersectedGeometry = pClone.Clone

Else

  bIgnoreGeometry = True

End If
```

Поскольку ранее мы уже рассмотрели операцию копирования атрибутов, нет необходимости в ее повторном обсуждении.

```
    ...

    pInsertCursor.InsertFeature pFeatBuf

  End If

  Set pFeature = pFeatureCursor.NextFeature
  ...

Loop

...
```

На этом мы закончим. Как мы видим из приведенных примеров, иногда действительно трудно избежать использования топологических операций и анализа пространственных соотношений, создавая что-то интересное и полезное.

Как сохранить настройки программы между запусками?

При работе с XTools и некоторыми другими приложениями мы столкнулись с необходимостью сохранения данных между сеансами работы ArcMap. Требуется при каждом запуске ArcMap, а с ним и нашего приложения, извлекать настройки программы перед началом ее работы и сохранять их, когда пользователь прекращает работу с программой. К информации такого рода относятся различные параметры программы, выбранные пользователем: система координат по умолчанию, имена полей, списки, варианты работы программы и т. д. Имеется два хорошо всем известных способа хранения подобной информации: файлы на диске и системный реестр MS Windows. Оба имеют свои преимущества и недостатки. К реестру легче получить доступ, и он не требует какой-либо разработки структуры хранения, однако он не способен хранить большие двоичные объекты, за исключением каких-то их свойств. Очевидно, файлы могут служить хранилищем для информации любого сорта, однако вряд ли целесообразно использовать их для хранения небольшого списка параметров запуска программы. Поэтому мы рекомендуем вам сохранять настройки, выражаемые строками или целыми числами и задающие ключевые свойства приложения, в реестре, а все другие свойства размещать в файлах.

Например, формат XML-документа особенно подходит для хранения структурированной информации достаточно большого объема. Для получения доступа к такой информации вы просто можете поместить XML-документ в папку, путь к которой прописан в реестре системы.

В нашем приложении мы использовали оба подхода: через реестр и с помощью файла на диске. Объяснение этому очень простое: с одной стороны, у нас есть объект, для которого требуется хранить пространственную привязку по умолчанию между сеансами работы приложения, таким образом, использование файлов неиз-

бежно. С другой стороны, нам нужно было хранить набор строковых и булевых параметров, а их, по-видимому, легче хранить в реестре. Поэтому далее мы продемонстрируем оба способа.

Давайте начнем с чтения параметров из реестра. Для реализации связанных между собой операций мы использовали отдельный модуль программы. Этот модуль импортирует Win32 API функции для работы с реестром и создает для них подходящую форму вызова в Visual Basic. Мы думаем, что не стоит здесь проводить детальный анализ этих функций, поскольку их код служит только одной цели — поддержке доступа к реестру, и может быть легко написан любым программистом или взят из примеров MSDN. Вы также легко можете найти аналогичный набор функций в Интернете. Поэтому здесь мы рассмотрим только непосредственный способ использования этих функций.

В качестве примера мы возьмем код из XTools Defaults, поскольку он содержит вызовы двух основных функций, которые мы использовали: сохранение и чтение данных из реестра.

Для извлечения данных мы использовали функцию ***getstring***, возвращающую строковое значение по ключу. Первым параметром является раздел реестра HKEY, и он задается как константа в разделе общих объявлений модуля. Вторым аргументом является строковый ключ, а третьим — имя параметра.

```
...
Dim sValue As String

sValue = getstring(HKEY_CURRENT_USER,
  "Software\DataEast\XTools", "MapUnits")
```

Функция сохранения (*savestring*) строковых данных выглядит почти так же, за исключением того, что ***savestring*** имеет еще один параметр, определяющий значение, сохраняемое в реестре.

```
If sValue = "" Then

  Call savestring(HKEY_CURRENT_USER, "Software\ _
    DataEast\XTools", "MapUnits", "Degrees")
  Call savestring(HKEY_CURRENT_USER, "Software\ _
    DataEast\XTools", "OutputMapUnits", "Feet")

  ...

End If
```

Теперь перейдем к проблеме сохранения проекции. Сохраненная система координат считывается, когда пользователь открывает диалог Defaults. В действительности здесь ArcObjects могут выполнить за вас большую часть работы, благодаря реализации интерфейса ***IPersistStream***. Этот интерфейс позволяет программисту сохранить некоторый объект в файл (как в поток) и извлечь его при необходимости. При этом вы можете не беспокоиться о формате хранения этого объекта в файле и не пытаться считывать объект вручную, в этом просто нет необходимости.

Координатная система будет сохраняться при закрытии диалога всякий раз, когда флаг, указывающий на ее изменение, устанавливается в True.

```
If m_isCSChanged Then
```

Кроме этого, нам нужно получить путь к сохраненному файлу. Мы могли бы задать его во время инсталляции приложения путем создания соответствующего параметра реестра, но на этот раз мы будем просто использовать фиксированную директорию {ArcGIS Home}\arcexe8x\ETC. Путь к этой папке хранится в реестре, поэтому мы получим его с помощью той же функции, которую мы использовали при работе с нашими собственными настройками.

```
Dim sETCPath As String
sETCPath = getstring(HKEY_LOCAL_MACHINE, "Software\ _
 ESRI\ArcMap\Settings", "APSupportDir")
```

Затем мы сбрасываем флаг, связанный с индикацией изменений системы координат, и создаем новый объект ***FileSystemObject***. Позже мы им воспользуемся. Обратите внимание на то, что мы используем функцию ***CreateObject*** вместо оператора ***New***. Проблема заключается в том, что любая скомпилированная библиотека работает недостаточно стабильно, если в ней имеется явная ссылка на этот объект.

```
m_isCSChanged = False
Dim fso As Object
Set fso = CreateObject("Scripting FileSystemObject")
```

Мы воспользуемся этим объектом для перезаписи файла, содержащего информацию о системе координат. Если этот файл существует, объект ***FileSystemObject*** удалит его и затем создаст новый файл с тем же именем. Полное описание этого класса вы можете найти в библиотеке MSDN, поскольку он может оказаться весьма полезным при решении ваших задач.

```
If fso.FileExists(sETCPath & "\XToolsCS.prj") Then
  fso.DeleteFile sETCPath & "\XToolsCS.prj"
End If

Dim pTextStream As Object
Set pTextStream = fso.CreateTextFile( _
  sETCPath & "\XToolsCS.prj", True)
pTextStream.Close
```

Теперь у нас есть пустой файл на диске, готовый к приему данных. На его основе мы создадим новый объект ***FileStream***. Затем с помощью метода ***Open*** интерфейса ***IFile*** мы свяжем этот объект с файлом.

```
Dim pFile As IFile
Set pFile = New FileStream
pFile Open sETCPath & "\XToolsCS.prj", esriReadWrite
```

Теперь мы воспользуемся интерфейсом ***IPersistStream***, поддерживаемым многими COM объектами библиотеки ArcObjects. Мы запрашиваем этот интерфейс у объекта системы координат и сохраняем координатную систему в вышеупомянутом файле методом Save.

```
Dim pPersistStream As IPersistStream
Set pPersistStream = m_pSR

pPersistStream.Save pFile, 1
Set pFile = Nothing

End If
```

Вот и все. Теперь наши параметры сохранятся между сеансами работы программы.

Измерительный инструмент. Как рассчитывать площади, периметры и длины объектов?

Пространственные объекты (features) — это объекты, моделирующие предметы и явления реального мира. Наиболее часто задаваемые вопросы при анализе пространственных объектов относятся к их размерам, т. е. к площади, периметру и длине. На первый взгляд, проблема измерения размеров объектов в ArcMap кажется очень простой. И иногда это действительно так. Давайте рассмотрим некоторые простейшие ситуации получения размеров объекта.

Например, у нас есть полилинейный слой, и мы хотим узнать длину каждой полилинии в нем. Наиболее просто это можно сделать, используя свойство Length интерфейса IPolyline. Для этого

организуем перебор коллекции пространственных объектов исходного класса объектов — полилиний и получим ссылку на геометрию каждого объекта. Поскольку мы работаем со слоем типа polyline, каждый тип геометрии будет иметь тип Polyline (полилиния) и поддерживать требуемый интерфейс IPolyline. Ситуация понятна, и мы можем написать следующий код:

```
...
Dim pFeature As IFeature
Dim pPolyline As IPolyline
Set pFeature = pCursor.NextFeature

Do Until pFeature Is Nothing
  Set pPolyline = pFeature.Shape
  pFeature.Value(pFeature.Fields.FindField("Length")) =
    pPolyline.Length
  Set pFeature = pCursor.NextFeature
Loop
...
```

Конечно, нужно не забыть создать курсор для перебора объектов и новое поле Length типа Double в классе объектов перед началом цикла. Но все эти вопросы уже были затронуты в предыдущих разделах главы.

Почти аналогичный код используется для расчета площадей и периметров объектов полигонального слоя. Здесь, однако, мы должны использовать другие интерфейсы. Существует специальный интерфейс IArea, который можно использовать для получения размеров площадных объектов. Каждый полигональный объект поддерживает этот интерфейс. Для того чтобы измерить периметр полигона, вначале мы должны получить его границу в виде кривой, а затем применить свойство Length интерфейса ICurve. Границу мы можем получить с помощью свойства Boundary уже известного нам интерфейса ITopologicalOperator, поддерживаемого объектами типа Polygon. Вот требуемый код:

```
...
Dim pFeature As IFeature
Dim pArea As IArea
Dim pCurve As ICurve
Dim pTO As ITopologicalOperator

Set pFeature = pCursor.NextFeature

Do Until pFeature Is Nothing
  Set pArea = pFeature.Shape
  pFeature.Value(pFeature.Fields.FindField("Area")) =
    pArea.Area
  Set pTO = pArea ' we can do it because pArea is a reference
    to one of the Polygon interface
  Set pCurve = pTO.Boundary
  pFeature.Value(pFeature.Fields.FindField("Perimeter")) =
    pCurve.Length
  Set pFeature = pCursor.NextFeature
Loop
...
```

Все кажется простым и понятным. Но при использовании приведенного выше кода вы столкнетесь со следующими проблемами:

- Полученные значения площадей, периметров и длин заметно отличаются от реальных размеров объектов. Например, вы можете открыть географический справочник и обнаружить, что площадь вашего округа не совпадает с рассчитанным вами значением поля Area. Проблема заключается в проекции, примененной к вашим данным. Многие географические проекции не обеспечивают сохранения площадей или длин объектов земной поверхности при проектировании на плоскость. Поэтому вы должны заранее подготовить данные, т. е. правильно их спроектировать.
- Иногда вы будете получать правильные значения, но единицы измерения, в которых они посчитаны, не будут вас

устраивать. В этом случае вы должны конвертировать значение из одних единиц измерения в другие.

- Наконец, вы можете правильно выполнить все операции (правильно спроектировать данные и выставить нужные единицы измерения), но, тем не менее, получить неверный результат. Почему так может произойти? Вследствие наличия некорректных представлений геометрии у ваших объектов. Мы уже говорили о том, что иногда геометрия объектов становится топологически некорректной. Это может происходить во время редактирования или в результате пространственных операций. Поэтому дополнительно вы должны привести геометрию к корректному виду, вызвав метод Simplify интерфейса ITopologicalOperator (детали работы с этим методом описаны в предыдущих разделах книги и в справочной системе разработчика ArcObjects).

Учитывая все перечисленные выше обстоятельства, рассмотрим способ решения этих проблем. В качестве примера возьмем инструмент Calculate приложения XTools.

Перед тем, как начать анализ кода, обсудим общее поведение инструмента Calculate приложения XTools. Одной из наиболее важных составляющих наших расчетов является, как отмечалось выше, пространственная привязка объектов. Типы пространственных привязок (координатных систем) могут быть разделены на три основные группы:

- Неопределенные координатные системы (Unknown Coordinate Systems)
- Географические координатные системы (Geographic Coordinate Systems)
- Проекционные координатные системы (Projected Coordinate Systems)

Каждая группа требует свои параметры для работы и свой тип реализации процесса измерения.

Если в ваших данных отсутствует информация о пространственной привязке (случай неопределенной координатной системы), вы не можете спроектировать их в какую-либо другую координатную систему. В данном случае у вас есть только одна возможность — получить значения площади, периметра или длины и преобразовать их в другие единицы измерения, если вы знаете, в каких единицах измерения находятся ваши исходные данные, и, наконец, заполнить результирующие поля значениями. В такой ситуации вам достаточно знать тип единиц измерения исходных данных и требуемых единиц измерения для результата.

В случае географической системы координат исходными единицами измерения являются градусы. Вы можете рассчитать значения площади, периметра и длины в градусах (для этого используйте соответствующие интерфейсы, как показано выше). Либо вы можете спроектировать данные в нужную вам проекцию с заданными единицами измерения.

Наконец, если у вас уже есть спроектированные данные, то вы можете использовать эту проекцию для расчетов либо спроектировать данные в другую нужную вам проекцию.

Все эти варианты поддерживаются специальным диалогом Calculate в XTools. Здесь мы рассмотрим только ключевую часть инструмента — процедуру ***CalcSize2***.

Перед тем, как использовать эту процедуру, вы должны объявить три переменных уровня модуля: ***m_pSR*** — требуемая на выходе пространственная привязка (проекция), ***m_sInputUnits*** — единицы измерения исходных данных и ***m_sOutputUnits*** — требуемые единицы измерения для результатов измерений. В XTools эти параметры получают свое значения через диалог Calculate, если вы используете команду Calculate из панели инструментов XTools. Либо параметры берутся из диалога Defaults, если расчеты выполняются автоматически как заключительная часть работы некоторых других инструментов.

Единственным параметром метода *CalcSize2* является ссылка на слой пространственных объектов, который вы хотите дополнить полями с измерениями площадей, периметров или длин. То, что передается ссылка на слой, а не на сами данные, — это просто соглашение, принятое в XTools, но вы можете использовать ссылки на объекты класса FeatureClass для доступа к вашим данным.

```
Public Sub CalcSize2(pLayer As IFeatureLayer)
  On Error GoTo ErrHandler
```

Прежде всего получим ссылку на класс пространственных объектов.

```
  Dim pFClass As IFeatureClass
  Set pFClass = pLayer.FeatureClass
```

Далее мы определяем набор служебных переменных, необходимых для дальнейшей работы функции. Среди них переменные, служащие для выборки объектов из FeatureClass (***pQueryFilter***, ***pCursor***, ***pFeature***), переменные для создания новых полей и переменные, соответствующие геометрическим типам объектов (***pPolyline***, ***pPolygon***, ***pCurve***, ***pTO*** и др.).

```
  ' new field
  Dim pFEdit As IFieldEdit

  ' variables for iterating through the featureclass
  Dim pFeature As IFeature
  Dim pCursor As IFeatureCursor
  Dim pQueryFilter As IQueryFilter
  Set pQueryFilter = New QueryFilter

  ' feature geometry and boundary
  Dim pPolygon As IPolygon
  Dim pPolyline As IPolyline
  Dim pCurve As ICurve
```

```
Dim pArea As IArea
Dim pTO As ITopologicalOperator2

' polygon area and boundary length
Dim dArea As Double

' polyline feature length
Dim dLength As Double
Dim bConvertUnits As Boolean
```

Следующим подготовительным шагом является создание информативного индикатора выполнения процесса, поскольку исходный слой может содержать большое число объектов и поэтому обработка может быть длительной. Здесь мы опять используем стандартный объект диалога хода выполнения процесса и механизм отмены дальнейшего выполнения операции.

```
Dim pTrackCancel As ITrackCancel
Set pTrackCancel = New CancelTracker

Dim pPDlgFact As IProgressDialogFactory
Set pPDlgFact = New ProgressDialogFactory

Dim pStepPro As IStepProgressor
Dim boolCont As Boolean

Dim nFeatureCount As Long
nFeatureCount = pFClass.FeatureCount(pQueryFilter)
Dim pPDlg As IProgressDialog2
Set pPDlg = pPDlgFact.Create(pTrackCancel, m_pApp.hWnd)
pPDlg.CancelEnabled = True
pPDlg.Description = "Calculating area, perimeter, length of _
  " & nFeatureCount & " features"
pPDlg.Title = "XTools"
pPDlg.Animation = esriDownloadFile
```

```
Set pStepPro = pPDlg
pStepPro.MinRange = 0
pStepPro.MaxRange = nFeatureCount
pStepPro.StepValue = 1
pStepPro.Message = "Calculating..."
```

Далее мы должны разбить задачу на два возможных сценария. Первый сценарий имеет место, когда мы получаем на вход слой полигонов и хотим рассчитать площадь и периметр его объектов. При этом необходимо проверить существование полей Area и Perimeter и при необходимости добавить их.

```
If (pFClass.ShapeType = esriGeometryPolygon) Then

  If (pFClass.Fields.FindField("Area") = -1) Then
    Set pFEdit = New Field
    With pFEdit
      .Name = "Area"
      .Editable = True
      .Type = esriFieldTypeDouble
    End With
    pFClass.AddField pFEdit
  End If

  If (pFClass.Fields.FindField("Perimeter") = -1) Then
    Set pFEdit = New Field
    With pFEdit
      .Name = "Perimeter"
      .Editable = True
      .Type = esriFieldTypeDouble
    End With
    pFClass.AddField pFEdit
  End If
```

Мы также можем добавить два необязательных поля: Acres и Hectares для площади в акрах и гектарах соответственно. Уста-

новки по умолчанию в приложении XTools, задаваемые в диалоге Defaults, определяют, должны ли мы добавлять их или нет. Для получения этих установок мы используем специальную функцию *GetAreaUnits()*.

```
If (GetAreaUnits() = "Acres" Or GetAreaUnits() = _
  "Both Acres and Hectares") Then

  If (pFClass.Fields.FindField("Acres") = -1) Then
     Set pFEdit = New Field
     With pFEdit
        .Name = "Acres"
        .Editable = True
        .Type = esriFieldTypeDouble
     End With
     pFClass.AddField pFEdit
  End If

End If

If (GetAreaUnits() = "Hectares" Or GetAreaUnits() = _
  "Both Acres and Hectares") Then

  If (pFClass.Fields.FindField("Hectares") = -1) Then
     Set pFEdit = New Field
     With pFEdit
        .Name = "Hectares"
        .Editable = True
        .Type = esriFieldTypeDouble
     End With
     pFClass.AddField pFEdit
  End If

End If
```

Еще мы должны не забыть подготовить курсор для перебора объектов в цикле Do...Loop.

```
Set pQueryFilter = New QueryFilter
Set pCursor = pFClass.Update(pQueryFilter, True)
```

Как было замечено выше, иногда необходимо преобразовывать полученные значения площадей и длин из одних единиц измерения в другие. Здесь мы будем придерживаться следующего правила: если исходные единицы измерения совпадают с требуемыми на выходе единицами измерения или исходные единицы измерения являются десятичными градусами, то нам не нужно их конвертировать. Дадим краткое обоснование такому подходу.

Первый случай (типы входных и выходных единиц измерения совпадают) вполне ясен — нет никаких причин для пересчета. Второй случай (единицы измерения — десятичные градусы) не так очевиден. Если исходные единицы измерения являются градусами, это означает, что мы можем получить измеренные значения только в градусах, поскольку невозможно обеспечить простой способ пересчета градусов, например, в метры или футы. Причиной является земная поверхность. Один градус соответствует различному числу метров, футов и т. д. в зависимости от места измерения на поверхности Земли. Следовательно, единственным способом получения метров, футов и т. д. является предварительное проектирование данных в соответствующую проекцию с нужными единицами измерения (единицы измерения определяются проекцией). Поэтому, если система координат определена, мы просто проектируем данные. Если мы хотим проводить измерения в градусах, пространственная привязка устанавливается в Nothing.

```
If ((m_sInputUnits = m_sOutputUnits) Or (m_sInputUnits = _
  "Degrees")) Then
  bConvertUnits = False
Else
```

```
    bConvertUnits = True
  End If
```

Итак, запишем начало цикла. Объекты извлекаются с помощью созданного ранее курсора.

```
  Set pFeature = pCursor.NextFeature
  Do While (Not pFeature Is Nothing)
```

Далее получаем копию геометрии объекта. Ее необходимо спроектировать, если ссылка на систему координат не пустая.

```
    Set pPolygon = pFeature.ShapeCopy

    'project geometry if necessary
    If (Not m_pSR Is Nothing) Then
      pPolygon.Project m_pSR
    End If
```

Ну а теперь нам следует вспомнить о последней проблеме — некорректной геометрии. Мы должны выполнить операцию Simplify на геометрии каждого объекта для того, чтобы убедиться в ее корректности.

```
    Set pTO = pPolygon
    pTO.Simplify
```

Итак, геометрия объекта подготовлена для измерений, и мы запрашиваем у нее интерфейсы IArea и ICurve и затем извлекаем значения площади и периметра.

```
    Set pArea = pPolygon
    Set pCurve = pPolygon

    'get polygon area and length of the curve
    dArea = pArea.Area
    dLength = pCurve.Length
```

Последней операцией является преобразование единиц измерения в случае необходимости. Для преобразования одного типа единиц измерения в другой мы используем несколько простых функций, смысл которых понятен из их названий. Детали этого преобразования видны из приведенного ниже кода.

```
If (bConvertUnits) Then
  dArea = ConvertAreaFromMeters(ConvertAreaToMeters( _
   dArea, m_sInputUnits), m_sOutputUnits)
 dLength = ConvertFromMeters(ConvertToMeters( _
  dLength, m_sInputUnits), m_sOutputUnits)
End If
```

Нужные значения получены, и мы сохраняем их в соответствующих полях класса пространственных объектов.

```
pFeature.Value(pFeature.Fields.FindField("Area")) = dArea
pFeature.Value(pFeature.Fields.FindField("Perimeter")) = _
 dLength
```

Дополнительные строки кода рассчитывают площадь в гектарах и акрах. Сначала мы должны преобразовать единицы в метры, а затем метры — в гектары и акры. Здесь все очевидно.

```
If (GetAreaUnits() = "Acres" Or GetAreaUnits() = "Both Acres
 and Hectares") Then
   pFeature.Value(pFeature.Fields.FindField("Acres")) = _
    ConvertAreaFromMeters(CDbl(pArea.Area) *
     meterPerUnit * meterPerUnit, "Feet") / _
    CDbl(43560)
End If

If (GetAreaUnits() = "Hectares" Or GetAreaUnits() = "Both
 Acres and Hectares") Then
```

```
      pFeature.Value(pFeature.Fields.FindField("Hectares")) = _
        CDbl(pArea.Area) * _
        meterPerUnit * meterPerUnit / CDbl(10000)
    End If
```

Ниже показана заключительная часть цикла: отслеживание нажатия пользователем на Cancel и подготовка новой итерации цикла. Для сохранения обновленных значений полей мы должны вызвать метод курсора UpdateFeature.

```
    boolCont = pTrackCancel.Continue
    If Not boolCont Then Exit Do

    pCursor.UpdateFeature pFeature
    Set pFeature = pCursor.NextFeature
  Loop
```

А теперь мы вспомним, что должны рассмотреть второй сценарий использования этого инструмента, когда исходным слоем является слой полилиний. Для него измерительный процесс реализуется еще проще. Первой операцией является добавление поля Length в класс пространственных объектов, если оно отсутствует.

```
  ElseIf (pFClass.ShapeType = esriGeometryPolyline) Then

    If (pFClass.Fields.FindField("Length") = -1) Then
      'create length field
      Set pFEdit = New Field
      With pFEdit
        .Name = "Length"
        .Editable = True
        .Type = esriFieldTypeDouble
      End With
      pFClass.AddField pFEdit
    End If
```

Подготовим курсор для пошаговой выборки объектов.

```
Set pQueryFilter = New QueryFilter
Set pCursor = pFClass.Update(pQueryFilter, True)
```

Используя те же приемы, анализируем исходные и требуемые для результата единицы измерения и решаем, необходимо ли их преобразование.

```
If ((m_sInputUnits = m_sOutputUnits) Or (m_sInputUnits = _
 "Degrees")) Then
 bConvertUnits = False
Else
 bConvertUnits = True
End If
```

Начинаем цикл.

```
Set pFeature = pCursor.NextFeature
Do While (Not pFeature Is Nothing)
```

Затем получаем копию каждой геометрии и проектируем ее при необходимости.

```
 Set pPolyline = pFeature.ShapeCopy

 If (Not m_pSR Is Nothing) Then
  pPolyline.Project m_pSR
 End If
```

Не забудьте выполнить коррекцию геометрии!

```
 Set pTO = pPolyline
 pTO.Simplify
```

Наконец, получаем значение длины, конвертируем его в другие единицы и сохраняем в поле Length текущего объекта.

```
    dLength = pPolyline.Length

    If (bConvertUnits) Then
      dLength = ConvertFromMeters(ConvertToMeters( _
        dLength, m_sInputUnits), _ m_sOutputUnits)
    End If

    pFeature.Value(pFeature.Fields.FindField("Length")) = _
      dLength
```

Проверяем, не нажал ли пользователь кнопку Cancel, и продолжаем процесс.

```
    boolCont = pTrackCancel.Continue ' step progressor
    If Not boolCont Then Exit Do

    pCursor.UpdateFeature pFeature
    Set pFeature = pCursor.NextFeature

  Loop
```

Закрываем оператор If — все сценарии использования инструмента мы рассмотрели.

```
  End If
```

Наконец, закрываем диалог хода выполнения процесса и освобождаем ресурсы.

```
    pPDlg.HideDialog
    Set pPDlg = Nothing
    Exit Sub

ErrHandler:
  If (Not pPDlg Is Nothing) Then
    pPDlg.HideDialog
```

```
    End If
    MsgBox "Application Error. " & Err.Description, vbCritical,
    "XTools"
End Sub
```

На протяжении этой главы мы подробно рассмотрели различные процедуры, используемые программистами, работающими в среде ArcGIS. Вот некоторые из них: создание и открытие нового класса пространственных объектов и таблиц; преобразование геометрий пространственных объектов; обработка сложных геометрий объектов; топологические операции; расчет площадей и длин; и многие другие. На протяжении этой главы мы использовали большое количество интерфейсов и классов ArcObjects, с которыми вы обязательно встретитесь на практике: IMxDocument, IMap, IFeatureLayer, IFeatureCursor, IFeature, IGeometry, ITopologicalOperator, ISpatialReference и др. Эта глава явилась как бы иллюстрацией тех многочисленных выгод, которые обеспечивает вам использование модели ArcObjects, а также показала множество эффективных способов использования этой модели при создании приложения XTools.

Заключение

Наконец, мы достигли финиша в нашем небольшом турне по технологии ArcObjects. Мы надеемся, что оно окажется полезным и поможет вам в создании множества замечательных и эффективных программ для системы ArcGIS. К сожалению, нет такой книги, которая содержала бы всю необходимую информацию по рассматриваемой проблеме, и данная книга не является исключением. Поэтому, если вы столкнетесь с какой-то проблемой, не затронутой в нашей книге, попробуйте найти ответ на сайте ESRI (www.esri.com), где можно получить доступ к большому количеству полезной информации и использовать опыт и знания других программистов участвуя в дискуссионных форумах. Мы рекомендуем также прочесть книги издательства ESRI, посвященные ArcObjects, самой полной из которых является, несомненно, *Exploring ArcObjects*.

Если у вас есть комментарии, предложения или замечания, касающиеся данной книги, пожалуйста, направляйте их авторам книги (ipopov@dataeast.ru и mchikinev@dataeast.ru). Мы приветствуем любые мнения, касающиеся данной книги.

Эта работа была выполнена в рамках контракта и при финансовой поддержке со стороны компании "Дата Ист". Авторы благодарны И. С. Забадаеву, Б. Ю. Берхину, Н. Н. Добрецову и И. Д. Зольникову за предложения и замечания, касающиеся содержания данной книги. Мы благодарны всей команде разработчиков компании "Дата Ист" и в особенности М.А. Саттарову, чьи ценные советы помогли улучшить приводимые в книги примеры, а также всех сотрудников компании, которые помогли нам в подготовке иллюстраций и в правке корректуры.

Дополнительную информацию по книге и новым приложениям вы можете найти на сайте компании "Дата Ист" (www.dataeast.ru).

Класс для поддержки работы с файлами

```
'*************************************************************

'FILE NAME     FileInfo.cls
'PURPOSE       This class supports file path analysis and splitting
'COPYRIGHT (c) DATAEAST LLC., 2002
'
'NOTES:
'*************************************************************

Option Explicit

Private sFullPath As String
'*************************************************************

'NAME          ParentDirPath (Property Get)
'PURPOSE       This is the read-only property representing the full path
  to the ParentDir (that is the workspace path for the shape file) of the file.
'ARGUMENTS:
'RETURNS:    String  -  full path to the ParentDir(
  that is the workspace path for the shape file) of the file.
'EXTERNAL:
'NOTES:
'*************************************************************

Public Property Get ParentDirPath() As String
  ParentDirPath = GetDirNameByPath(sFullPath)
End Property
'*************************************************************
```

```
'NAME        FileName (Property Get)
'PURPOSE   This is the read-only property representing the file name
  without extension (that is the dataset name for the shape file) of the file.
'ARGUMENTS:
'RETURNS:   String  -  file name without extension (that is the dataset
  name for the shape file) of the file.
'EXTERNAL:
'NOTES:
'*************************************************************

Public Property Get FileName() As String

  FileName = GetFileNameByPath(sFullPath)
  Dim sExt As String
  sExt = GetFileExtensionByFileName(FileName)

  If Len(sExt) > 0 Then
    FileName = Left(FileName, Len(FileName) - Len(sExt) – 1)
  End If

End Property
'*************************************************************

'NAME        FileNameExtension (Property Get)
'PURPOSE   This is the read-only property representing the file name
  extension of the file.
'ARGUMENTS:
'RETURNS:   String  -  file name extension of the file.
'EXTERNAL:
'NOTES:
'*************************************************************

Public Property Get FileNameExtension() As String
  FileNameExtension = GetFileExtensionByFileName
    (GetFileName ByPath(sFullPath))
End Property
'*************************************************************
```

```
'NAME        FullPath (Property Let)
'PURPOSE     This is the read-write property representing the full path\
  to the file.
'ARGUMENTS:   s  -  full path to the file.
'RETURNS:
'EXTERNAL:
'NOTES:
'*************************************************************

Public Property Let FullPath(s As String)
  sFullPath = s
End Property

'*************************************************************

'NAME        FullPath (Property Get)
'PURPOSE     This is the read-write property representing the full path
  to the file.
'ARGUMENTS:
'RETURNS:    String  - full path to the file
'EXTERNAL:
'NOTES:
'*************************************************************

Public Property Get FullPath() As String
  FullPath = sFullPath
End Property

'*************************************************************

'NAME        StrToCollection
'PURPOSE     This function splits the string according to the given
'            delimiter and forms the collection of resultant substrings
'
'ARGUMENTS:  Src       -   source string
'             delimiter -   delimiter string
'RETURNS:     String    -   the collection of resultant substrings.
```

```
'EXTERNAL:
'NOTES:
'*****************************************************************

Private Function StrToCollection(Src As String, delimiter As String)
  As Collection
  Dim src2 As String
  Dim si As Long

  Dim pCol As Collection
  Set pCol = New Collection

  src2 = Src

  While Len(src2) > 0

    si = InStr(src2, delimiter)

    If si > 0 Then

      si = si - 1
      pCol.Add Left(src2, si)
      src2 = Right(src2, Len(src2) - Len(delimiter) - si)

    Else

      pCol.Add src2
      src2 = ""

    End If

  Wend

  Set StrToCollection = pCol

  Exit Function
```

```
End Function
'*************************************************************

'NAME        GetFileNameByPath
'PURPOSE     This function gets the file name according to path
'
'ARGUMENTS:  path    -   file full path
'RETURNS:    String  -   the file name.
'EXTERNAL:
'NOTES:
'*************************************************************

Private Function GetFileNameByPath(path As String) As String
  Dim pCol As Collection
  Set pCol = StrToCollection(path, "\")

  GetFileNameByPath = pCol.Item(pCol.Count)

  Exit Function
End Function
'*************************************************************

'NAME        GetFileExtensionByFileName
'PURPOSE     This function gets the file extension according to file
  name
'
'ARGUMENTS:  sFileName -  source file name
'RETURNS:        String   -  the file extension.
'EXTERNAL:
'NOTES:
'*************************************************************

Private Function GetFileExtensionByFileName(sFileName As String)
As String
  Dim pCol As Collection
  Set pCol = StrToCollection(sFileName, ".")
```

```
  Dim strPart As Variant

  If pCol.Count > 1 Then

    For Each strPart In pCol

      GetFileExtensionByFileName = strPart

    Next

  Else

    GetFileExtensionByFileName = ""

  End If

  Exit Function
End Function
'*************************************************************

'NAME         GetDirNameByPath
'PURPOSE    This function gets the file folder name according to file
  path
'
'ARGUMENTS:  path     -   source file path
'RETURNS:      String   -   directory name.
'EXTERNAL:
'NOTES:
'*************************************************************

Private Function GetDirNameByPath(path As String) As String
  Dim FileName As String
  FileName = GetFileNameByPath(path)

  Dim LengthOfDirName As Long
```

```
  LengthOfDirName = Len(path) - Len(FileName)

  GetDirNameByPath = Left(path, LengthOfDirName)

End Function
```

Функции для проверки имен полей к шейп-файлам

```
'*************************************************************

'NAME          ToDBFFieldName
'PURPOSE       Creates correct DBF field name based on given field
  and collection
'             of fields
'ARGUMENTS:  pFld     -  original field
'                  pFldCol  -  collection of fields
'RETURNS:      String   -  correct DBF field name
'EXTERNAL:
'NOTES:
'*************************************************************

Function ToDBFFieldName(pFld As IField, pFldCol As IFields)
  As String

  Dim strTmp As String
  Dim shch As Integer
  Dim j As Integer

  shch = 1

  If Len(pFld.Name) > 10 Then
    strTmp = Left(pFld.Name, 10)
  Else
```

```
    strTmp = pFld.Name
  End If

  strTmp = RemoveAllBadChars(strTmp)

  For j = 0 To pFldCol.FieldCount - 1

    If (strTmp = pFldCol.Field(j).Name) Then
      strTmp = Left(strTmp, Len(strTmp) - Len(
        CStr(shch))) & CStr(shch)
      shch = shch + 1
      j = 0
    End If

  Next j

  ToDBFFieldName = strTmp

End Function
'*************************************************************

'NAME          RemoveAllBadChars
'PURPOSE    Replaces all non numeric and non alphabetical chars
  with underscore
'             in the given string
'ARGUMENTS:  Src      -  string to correct
'RETURNS:      String   -  corrected string
'EXTERNAL:
'NOTES:
'*************************************************************

Function RemoveAllBadChars(Src As String) As String
  Dim strTmp As String
  Dim i As Integer

  strTmp = ""
```

```
 For i = 1 To Len(Src)

  If (IsAlphabeticalOrNumerical(Mid(Src, i, 1))) Then
   strTmp = strTmp & Mid(Src, i, 1)
  Else
   strTmp = strTmp & "_"
  End If

 Next i

 RemoveAllBadChars = strTmp

End Function
'*************************************************************

'NAME         IsAlphabeticalOrNumerical
'PURPOSE   Checks whether given char is alphabetical or numerical
'ARGUMENTS:  sChar      -  char to check
'RETURNS:      Boolean   -  whether given char is alphabetical or
  numerical
'EXTERNAL:
'NOTES:
'*************************************************************

Function IsAlphabeticalOrNumerical(sChr As String) As Boolean

  Dim nCode As Byte
  nCode = Asc(Mid(sChr, 1, 1))
  IsAlphabeticalOrNumerical = ((nCode >= Asc("A")) And (
   nCode <= Asc("Z"))) _
      Or ((nCode >= Asc("a")) And (nCode <= Asc("z"))) Or _
      ((nCode >= Asc("0")) And (nCode <= Asc("9")))

End Function
```