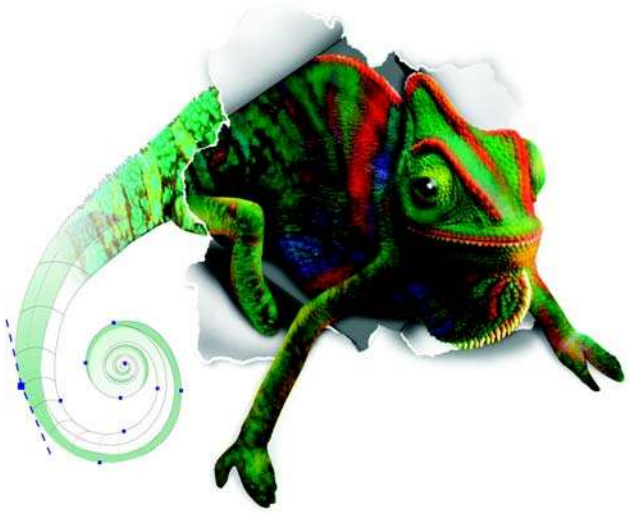


CorelDRAW® X3
GRAPHICS SUITE

Руководство по программированию для VBA

Русский перевод Владимир Ширкин

Содержание

Введение

Добро пожаловать в Руководство по программированию CorelDRAW® Graphics Suite X3 для VBA, ваш ресурс по разработке и распространению решений Microsoft® Visual Basic® для приложений (VBA) в CorelDRAW Graphics Suite X3.

О корпорации Corel

Корпорация Corel предоставляет инновационные программные решения, которые помогают миллионам предприятий и потребителей в более чем 75 странах повысить свою производительность. Компания славится своим мощным портфелем программного обеспечения, которое сочетает в себе инновационные приложения для редактирования фотографий, создания графики, векторной иллюстрации и техно-графики с офисными и персональными решениями.

Флагманские продукты Corel включают CorelDRAW Graphics Suite, WordPerfect® Office suite, Corel® Paint Shop Pro®, Corel® Painter™ и Corel DESIGNER® Technical Suite. Для получения дополнительной информации, пожалуйста, посетите www.corel.com

Графический комплекс (lDRAW) Graphics Suite X3

CorelDRAW Graphics Suite X3 - это комплексная программа векторного рисования, которая позволяет легко создавать профессиональные иллюстрации, от простых логотипов до технических иллюстраций. Инструменты CorelDRAW разработаны в соответствии с требованиями графического дизайнера.

Сведения о VBA в графическом пакете CorelDRAW X3

В 1995 году Corel включила автоматизацию в CorelDRAW 6, включив в нее язык Corel SCRIPT™. Это позволило разработчикам решений создавать интеллектуальные мини-приложения в CorelDRAW, например, для рисования фигур, изменения положения и размера фигур, открытия и закрытия документов или задания стилей.

Corel SCRIPT был включен в CorelDRAW версии с 6 по 9. Хотя редактор Corel SCRIPT не входит в состав CorelDRAW, в версиях 9 и более поздних включен движок "run-time", поэтому сценарии, написанные для более ранних версий CorelDRAW, можно легко перенести на последние версии.

В 1998 году Corel приняла стратегическое решение расширить функциональность Corel SCRIPT CorelDRAW 9, лицензировав движок Microsoft Visual Basic для приложений для обработки его закулисной автоматизации. Добавление VBA сделало CorelDRAW немедленно доступным для миллионов разработчиков VBA и Visual Basic по всему миру.

VBA в CorelDRAW может использоваться в качестве платформы для разработки мощных корпоративных графических решений, таких как авто-генераторы билетов, настраиваемые календари или пакетные процессоры файлов. VBA также можно использовать для улучшения и оптимизации рабочего процесса в CorelDRAW. Например, можно улучшить и настроить некоторые встроенные функции CorelDRAW (например, создание объектов, выравнивание или преобразование) или добавить макеты страниц на лету (например, для создания фирменных бланков компании).

VBA поставляется с полностью интегрированной средой разработки (IDE), которая предоставляет контекстные всплывающие списки, подсветку синтаксиса, построчную отладку и окна визуального конструктора. Эти полезные подсказки и вспомогательные средства создают особенно дружественную среду обучения для неопытных разработчиков. VBA доступен как для CorelDRAW, так и для Corel PHOTO-PAINT®.

Об этом руководстве

Это руководство, которое следует читать вместе со схемами объектных моделей CorelDRAW и Corel PHOTO-PAINT, устанавливаемыми вместе с CorelDRAW Graphics Suite X3, было разработано в качестве ресурса для следующего:

- изучение интегрированной среды разработки VBA и многих ее расширенных функций
- понимание наиболее важных функций CorelDRAW Graphics Suite X3 и способов их использования
- изучение возможностей VBA, разработанных для CorelDRAW Graphics Suite X3

Это руководство следует использовать всем, кто интересуется автоматизацией простых и сложных задач в CorelDRAW Graphics Suite X3 или разрабатывает коммерческие решения, интегрируемые с CorelDRAW Graphics Suite X3. Предполагается, что читатель уже имеет опыт работы хотя бы с одним процедурным языком программирования, таким как BASIC, Visual Basic, C, C++, Java™, Pascal, Cobol или Fortran. В этом руководстве не описываются основы процедурного программирования, такие как функции, условное ветвление и циклы. Непрограммисты должны изучить основы программирования на таких языках, как Visual Basic или VBA, прежде чем использовать этот документ для разработки решений CorelDRAW Graphics Suite X3.

Это руководство состоит из следующих глав, в которых рассматриваются конкретные аспекты автоматизации задач и создания решений в CorelDRAW Graphics Suite X3:

- **Глава 1**: Содержит краткое введение в VBA.
- **Глава 2**. Начало работы с VBA позволяет изучить рабочее пространство VBA в CorelDRAW и Corel PHOTO-PAINT.
- **Глава 3**: Работа с макросами показывает, как создавать, запускать и отлаживать макросы.
- **Глава 4**: Создание пользовательских интерфейсов для макросов демонстрирует, как создавать диалоговые окна, панели инструментов и кнопки, взаимодействие с пользователем и справку для ваших макросов.
- **Глава 5**: Организация и развертывание макросов поможет вам упорядочить создаваемые вами макросы.

В этом руководстве также есть приложение (стр. 55), в котором содержится подробная информация о объектной модели CorelDRAW.

Наконец, глоссарий (стр. 88) определяет термины, используемые в данном руководстве.

Соглашения в документации

В данном руководстве используются следующие условные обозначения:

Вы видите	Это значит
	Примечание — представляет информацию для выполнения процедуры.
	Совет — представляет такую информацию, как ярлыки процедур, варианты или преимущества.

Кроме того, в этом руководстве используется несколько соглашений о форматировании:

- Элементы пользовательского интерфейса отображаются **полужирным шрифтом**.
- Элементы глоссария выделены *курсивом*.
- Информация, которую можно заменить, например, путь или имя файла, отображается *<курсивом и в угловых скобках>*.
- Кодирование оформляется `моноширинным` шрифтом.

Дополнительная информация

VBA для CorelDRAW Graphics Suite X3 также устанавливается со следующей документацией:

- Схемы объектной модели VBA для CorelDRAW и Corel PHOTO-PAINT — доступны в формате PDF и устанавливаются в **..\Program Files\Corel\CorelDRAW Graphics Suite 13\Programs**.
- Файлы справки VBA для CorelDRAW и Corel PHOTO-PAINT — к ним можно получить доступ через редактор VB: выберите любой элемент в обозревателе объектов и нажмите F1, чтобы открыть раздел справки по этому элементу.

Подробную информацию о функциях CorelDRAW Graphics Suite X3 можно найти в руководстве пользователя, которое по умолчанию устанавливается в папку **..\Program Files\Corel\CorelDRAW Graphics Suite 13\ Languages\<*language*>\Help**. Кроме того, вы можете получить доступ к справочной системе из приложения CorelDRAW Graphics Suite X3, щелкнув **Help ▸ Help topics**.

Последние новости, советы и рекомендации, а также информацию об обновлении продуктов можно найти на веб-сайте CorelDRAW Graphics Suite X3. Перейдите на сайт www.corel.com и перейдите по ссылкам CorelDRAW Graphics Suite X3.

Для получения оперативной и точной информации о функциях продукта, спецификациях, ценах, доступности, услугах и технической поддержке обращайтесь в службу поддержки Corel. Для получения самой последней информации о доступной поддержке и профессиональных услугах для вашего продукта Corel посетите веб-сайт www.corel.com/support.

Если у вас есть комментарии или предложения по CorelDRAW Graphics Suite X3 или руководству по программированию CorelDRAW Graphics Suite X3 для VBA, вы можете отправить их, используя контактную информацию, указанную на странице www.corel.com/contact.

Глава 1.
Понятия VBA

Прежде чем приступить к работе с VBA в CorelDRAW Graphics Suite X3, важно немного разобраться в VBA в целом.

Эта глава отвечает на следующие вопросы:

- Что такое VBA?
- Для кого предназначен VBA?
- Чем VBA отличается от других языков программирования?
- Какие основные элементы используются в VBA?
- Как устроен код VBA?

Что такое VBA?

Visual Basic для приложений (более известный как VBA) — это встроенный язык программирования, который может автоматизировать повторяющиеся функции и создавать интеллектуальные решения в CorelDRAW и Corel PHOTO-PAINT.
CorelDRAW Graphics Suite X3 включает VBA версии 6.3.

VBA — это и язык и редактор. Невозможно иметь язык без редактора, также невозможно редактировать VBA в чем-либо, кроме редактора VB, или запускать программы VBA без редактора VB.

VBA разработан Microsoft и встроен почти во все ее настольные приложения, включая Microsoft® Office. VBA лицензируется Microsoft для других компаний, включая Corel Corporation (в CorelDRAW Graphics Suite, Corel DESIGNER® Technical Suite и WordPerfect Office), Autodesk, Inc. (в AutoCAD®) и IntelliCAD Technology Consortium (в IntelliCAD®). Это делает приложения Corel совместимыми с широким спектром приложений, поддерживающих VBA.

Полный список приложений, поддерживающих VBA, можно найти на веб-сайте Microsoft по адресу http://www.msdn.microsoft.com/vba/companies/company.asp.

Приложение не обязательно должно поддерживать VBA, чтобы ядро CorelDRAW Graphics Suite X3 VBA могло управлять этим приложением. Это означает, что вы можете создавать решения в CorelDRAW Graphics Suite X3, которые обращаются к базам данных, текстовым процессорам, специализированным редакторам содержимого, XML-документам и многому другому.

Что такое автоматизация?

Большинство действий, которые можно выполнять в CorelDRAW Graphics Suite X3, можно выполнять программно с помощью VBA. Эта программируемость CorelDRAW Graphics Suite X3 называется *автоматизацией*. Автоматизация повторяющихся задач может сэкономить время и уменьшить усилия, а автоматизация сложных задач может сделать возможным то, что иначе было бы невозможно.

В своей простейшей форме автоматизация просто записывает последовательность действий, чтобы вы могли воспроизводить их снова и снова. Термин «макрос» стал включать любой код, доступный для VBA во время выполнения внутри процесса, даже если часть этого кода может быть гораздо более сложной, чем простой набор записанных действий. Для целей данного руководства под макросом понимаются функции и подпрограммы VBA (которые объясняются в разделе «Создание функций и подпрограмм» на стр. 10).

Хотя в CorelDRAW Graphics Suite X3 можно записать последовательность действий, реальная сила автоматизации и VBA заключается в том, что эти записи можно редактировать для обеспечения условного и циклического выполнения. Например, простой макрос может установить красный цвет заливки выбранной фигуры и применить одноточечный контур; однако, добавив условие и механизм цикла в код VBA, макрос можно было бы, например, заставить искать каждую выбранную фигуру и применять только заливку к текстовым фигурам и только контур ко всем другим типам фигур.

Для кого предназначен VBA?

VBA может использоваться как программистами, так и непрограммистами.

VBA для непрограммистов

VBA основан на успешном языке программирования Microsoft Visual Basic (VB). Основное различие между VBA и VB заключается в том, что вы не можете создавать автономные исполняемые файлы (EXE) с помощью VBA, тогда как с VB вы можете. То есть, используя VBA, вы можете создавать только те программы, которые выполняются внутри главного приложения (в данном случае CorelDRAW или Corel PHOTO-PAINT).

VB — это «визуальная» версия языка программирования BASIC. Это означает, что это очень простой язык для изучения, особенно потому, что он предоставляет визуальные подсказки в редакторе. Microsoft многое добавила к исходному языку BASIC, и теперь это мощный и быстрый язык (хотя и не такой мощный, как Java или C++, и не такой быстрый, как C).

Цель этого руководства не в том, чтобы научить вас тому, как стать программистом, а в том, чтобы научить опытных программистов применять свои навыки для разработки полезных решений в CorelDRAW Graphics Suite X3. Если вы не программист, вам может быть полезно обратиться к множеству книг, написанных по VBA и VB, прежде чем продолжить чтение этого руководства.

VBA для программистов

VBA — это внутрипроцессный контроллер автоматизации. Другими словами, его можно использовать для управления функциями CorelDRAW Graphics Suite X3, которые можно автоматизировать. Кроме того, поскольку VBA работает «внутри процесса», он обходит механизмы межпроцессной синхронизации, чтобы работать намного эффективнее.

Вся автоматизация, доступная для внутрипроцессного VBA, также доступна для внешних внепроцессных контроллеров автоматизации или клиентов OLE. Сюда входят приложения, разработанные на языках программирования, которые можно использовать для разработки клиентов OLE, например:

- Microsoft Visual Basic, Visual C++ и Windows® Script Host
- Borland® Delphi™ и C++
- Движки VBA других приложений

Чем VBA отличается от других языков программирования?

VBA имеет много общего с большинством современных процедурных языков программирования, включая Java и JavaScript®, C и C++, а также Windows Script Host. Однако VBA работает как внутрипроцессный контроллер автоматизации, тогда как другие языки (кроме JavaScript) используются для компиляции автономных приложений.

VBA по сравнению с Java и JavaScript

VBA похож на Java и JavaScript в том, что это высокоуровневый процедурный язык программирования с полной сборкой мусора и очень небольшой поддержкой указателей памяти. (Дополнительную информацию см. в разделе «Использование указателей памяти и выделение памяти» на стр. 11.) Кроме того, код, разработанный на VBA, во многом так же, как и код, разработанный на Java и JavaScript, поддерживает компиляцию по запросу и может выполняться без компиляции.

VBA имеет еще одно сходство с JavaScript в том, что его нельзя выполнять как отдельное приложение. JavaScript встроен в веб-страницы как механизм для управления объектной моделью документа веб-браузера (или «DOM»). Точно так же программы VBA выполняются в хост-среде (в данном случае CorelDRAW или Corel PHOTO-PAINT) для управления объектной моделью хоста (что обсуждается в разделе «Что такое объектная модель?» на стр. 7).

Большинство приложений VBA можно скомпилировать в P-код, чтобы они работали быстрее, хотя разница едва заметна, учитывая сложность современного компьютерного оборудования. Аналогичным образом можно скомпилировать Java; JavaScript, однако, не может.

Наконец, в то время как VBA использует один знак равенства (=) как для сравнения, так и для присваивания, Java и JavaScript используют один знак равенства (=) для присваивания и два знака равенства (==) для логического сравнения. (Дополнительную информацию о логическом сравнении и присваивании в VBA см. в разделе «Использование логического сравнения и присваивания» на странице 12.)

VBA по сравнению с C и C++

Visual Basic — подобно C и C++ — использует функции. В VB функции могут использоваться для возврата значения, а подпрограммы — нет. Однако в C и C++ функции используются независимо от того, хотите ли вы вернуть значение. (Дополнительную информацию о функциях и подпрограммах см. в разделе «Создание функций и подпрограмм» на стр. 10.)

VBA выделяет и освобождает память «прозрачно». Однако в C и C++ за большую часть управления памятью отвечает разработчик. Это делает использование строк в VBA даже проще, чем использование класса CString в C++. Наконец, в то время как VBA использует один знак равенства (=) как для сравнения, так и для присваивания, C и C++ используют один знак равенства (=) для присваивания и два знака равенства (==) для логического сравнения. (Дополнительную информацию о логическом сравнении и присваивании в VBA см. в разделе «Использование логического сравнения и присваивания» на странице 12.)

VBA по сравнению с Windows Script Host

Windows Script Host (WSH) — это полезное дополнение к Windows для выполнения случайных сценариев и автоматизации задач Windows. WSH — это внепроцессный контроллер автоматизации, который можно использовать для управления CorelDRAW Graphics Suite X3. Однако, поскольку сценарии WSH не могут быть скомпилированы (и должны интерпретироваться по мере их выполнения) и должны запускаться вне процесса, они, как правило, работают медленно. WSH является хостом для ряда языков сценариев, каждый из которых имеет собственный синтаксис. Однако стандартный язык, используемый WSH, представляет собой макроязык, напоминающий Visual Basic, поэтому синтаксис стандартных скриптов такой же, как и в VBA.

Какие основные элементы используются в VBA?

Если вы когда-либо разрабатывали объектно-ориентированный код на C++, Borland Delphi или Java, вы уже знакомы с понятиями «классы», «объекты», «свойства» и «методы», но давайте еще раз рассмотрим их. их более подробно, поскольку они применимы к VBA.

Класс — это описание чего-либо. Например, класс «автомобиль» — это небольшое транспортное средство с двигателем и четырьмя колесами.

Объект — это экземпляр класса. Если мы расширим метафору автомобиля, то фактический физический автомобиль, который вы покупаете для вождения, является объектом (то есть экземпляром класса «автомобиль»). В контексте CorelDRAW каждый открытый документ — это экземпляр класса Document, каждая страница в документе — экземпляр класса Page, а каждый слой (и каждая фигура на каждом слое) — это дополнительные экземпляры большего количества классов.

Большинство классов имеют *свойства*. Например, свойства класса «автомобиль» заключаются в том, что он маленький, у него есть двигатель и четыре колеса. Каждый экземпляр класса «автомобиль» (то есть каждый объект в этом классе) также имеет такие свойства, как цвет, скорость и количество мест. Некоторые свойства, называемые свойствами «только для чтения», фиксируются при разработке класса; например, количество колес или сидений (обычно) не различается от машины к машине. Однако другие свойства можно изменить после создания объекта; например, скорость автомобиля может увеличиваться и уменьшаться, и с небольшой помощью можно изменить его цвет. В контексте CorelDRAW объекты документа имеют имя, разрешение и единицы горизонтальной и вертикальной линейки; отдельные фигуры имеют свойства контура и заливки, а также положение и коэффициент поворота; а текстовые объекты имеют текстовые свойства, которые могут включать в себя сам текст.

Метод — это операция, которую объект может выполнить сам с собой. В примере с классом «автомобиль» машину можно заставить двигаться быстрее и медленнее, поэтому для этого класса есть два метода: «ускорить» и «замедлить». В контексте CorelDRAW у документов есть методы для создания новых страниц, у слоев есть методы для создания новых фигур, а у фигур есть методы для применения преобразований и эффектов.

Объекты часто состоят из других более мелких объектов. Например, автомобиль содержит четыре объекта класса «колесо», два объекта класса «фара» и так далее. Каждый из этих дочерних объектов имеет те же свойства и методы, что и его тип класса. В контексте CorelDRAW объект документа содержит объекты страницы, которые содержат объекты слоя, которые содержат объекты формы, некоторые из которых содержат другие объекты. Важно распознавать эту родительско-дочернюю связь объектов, особенно при ссылке на отдельный объект.

Некоторые классы «наследуют» черты своих родителей. Например, в контексте CorelDRAW тип Shape имеет множество подтипов (или «унаследованных типов»), включая Rectangle, Ellipse, Curve и Text. Все эти подтипы могут использовать основные элементы типа Shape, включая методы перемещения и преобразования фигуры и установки ее цвета. Однако у подтипов также есть свои специалисты; например, Rectangle может иметь радиусы углов, тогда как Text имеет связанное свойство Font.

Что такое объектная модель?

VBA полагается на *объектную модель* приложения для связи с этим приложением и изменения его документов. Без объектной модели VBA не может запрашивать или изменять документы приложения.

Объектные модели в программном обеспечении обеспечивают высокий уровень структуры отношений между родительскими и дочерними объектами. Они также позволяют многократно использовать типы объектов или классы различными способами; например, объект Shape может иметь тип «группа» и может содержать другие объекты Shape, некоторые из которых также могут иметь тип «группа» или «прямоугольник», «кривая» или «текст». Этот высокий уровень организации и повторного использования делает объектную модель простой в использовании и в то же время мощной, а также удобной для навигации с помощью обозревателя объектов VBA (который обсуждается в разделе «Использование обозревателя объектов» на стр. 23).

Помните, что объектная модель — это карта, которую язык VBA использует для доступа к различным элементам (объектам, методам и свойствам) документа и для внесения изменений в эти элементы. Без объектной модели просто невозможно получить доступ к объектам в документе.

Понятие иерархии объектов

В любой объектной модели каждый объект является потомком другого объекта, который является потомком другого объекта. Кроме того, у каждого объекта есть собственные дочерние элементы — свойства, объекты и методы. Все это составляет иерархию объектов, которая является объектной моделью. В CorelDRAW корневым объектом всех объектов является объект «Приложение», поскольку все объекты являются дочерними или внучатыми по отношению к приложению.

Для «детализации» уровней иерархии, чтобы добраться до нужного объекта или члена, вы должны использовать стандартную нотацию. В VBA, как и во многих объектно-ориентированных языках, в нотации используется точка (.), чтобы указать, что объект справа является членом (или дочерним элементом) объекта слева.

```
Application.Documents(1).Pages(1).Layers(1).Shapes(1).Name = "Bob"
```

Обычно нет необходимости использовать полную иерархическую ссылку на объект или его свойства. Часть объектного синтаксиса в полной ссылке является обязательной или обязательной; однако другой синтаксис является необязательным (поскольку для него доступен сокращенный объект или потому, что он явный или неявный), и поэтому его можно либо включить для ясности, либо опустить для краткости.

Объект ярлыка (*shortcut object*) — это просто синтаксическая замена длинной версии объекта. Например, ярлык объекта ActiveLayer заменяет длинную версию `Application.ActiveDocument.ActivePage.ActiveLayer`, а ярлык объекта `ActiveSelection` заменяет длинную версию `Application.ActiveDocument.Selection`. Сведения об объектах-ярлыках, доступных в CorelDRAW, см. в разделе «Использование объектов-ярлыков» на странице 32.

Подробную информацию об объектной модели CorelDRAW см. в Приложении.

Как устроен код VBA?

Поскольку VBA — это процедурный язык, имеющий много общего со всеми процедурными языками, ваши текущие знания должны помочь вам быстро начать работу с VBA.

В этом разделе рассматриваются следующие темы по структуре и синтаксису VBA:

- Объявление переменных
- Построение функций и подпрограмм
- Концовка строк
- Комментарии
- Использование указателей памяти и выделение памяти
- Определение области действия
- Использование логического сравнения и присваивания
- Использование логических и побитовых операторов
- Предоставление окон сообщений и полей ввода

Редактор VB форматирует весь код за вас (как описано в разделе «Автоматическое форматирование кода» на стр. 20). Единственное пользовательское форматирование, которое вы можете сделать, это изменить размер отступов.

VBA может создавать объектно-ориентированные классы, хотя это особенность языка и подробно не рассматривается в данном руководстве.

Объявление переменных

В VBA объявление *переменных* выглядит следующим образом:

```
Dim foobar As Integer
```

Встроенные типы данных: Byte, Boolean, Integer, Long, Single, Double, String, Variant и несколько других менее используемых типов, включая Date, Decimal и Object.
Переменные могут быть объявлены в любом месте тела функции или в начале текущего модуля. Однако обычно рекомендуется объявлять переменную перед ее использованием; в противном случае компилятор интерпретирует его как Variant , и во время выполнения может возникнуть неэффективность.

Логические значения принимают False за ноль, а True за любое другое значение, хотя преобразование логического значения в длинное приводит к преобразованию True в значение –1.

Чтобы получить дополнительные сведения об одном из встроенных типов данных, введите его в окно кода, выберите его и нажмите клавишу F1.

Структуры данных могут быть построены с использованием следующего синтаксиса:

```
Public Type fooType
  item1 As Integer
  item2 As String
End Type
Dim myTypedItem As fooType
```

Доступ к элементам внутри переменной, объявленной как тип *fooType*, осуществляется с использованием записи через точку:

```
myTypedItem.item1 = 5
```

Объявление строк

Строки в VBA намного проще, чем в C. В VBA строки можно складывать вместе, усекать, выполнять поиск в прямом и обратном направлениях и передавать функции в качестве простых аргументов.

Чтобы сложить две строки вместе, просто используйте оператор конкатенации (и) или оператор сложения (+):

```
Dim string1 As String, string2 As String
string2 = string1 & " more text" + " even more text"
```

В VBA существует множество функций для работы со строками, включая InStr(), Left(), Mid(), Right(), Len() и Trim().

Объявление перечисляемых типов

Чтобы объявить *перечисляемый тип*, используйте следующую конструкцию:

```
Public Enum fooEnum
  ItemOne
  ItemTwo
  ItemThree
End Enum
```

Первому элементу перечисляемого типа по умолчанию присваивается нулевое значение.

Объявление массивов

Чтобы объявить массив, используйте круглые скобки, то есть символы (и):

```
Dim barArray (4) As Integer
```

Значение определяет индекс последнего элемента в массиве. Поскольку индексы массива по умолчанию отсчитываются от нуля, в предыдущем образце массива есть пять элементов (то есть элементы с 0 по 4 включительно).

Размер массивов можно изменить с помощью ReDim. Например, следующий код добавляет дополнительный элемент в *barArray*, но сохраняет существующее содержимое исходных пяти элементов:

```
ReDim Preserve barArray (6)
```

Верхние и нижние границы массива можно определить во время выполнения с помощью функций *UBound()* и *LBound()*.

Многомерные массивы можно объявить, разделив индексы измерений запятыми:

```
Dim barArray (4, 3)
```

Построение функций и подпрограмм

VBA использует как *функции*, так и *подпрограммы* (или «subs»). Функции могут использоваться для возврата значения, а подпрограммы — нет. В VBA функции и подпрограммы не нужно объявлять ни до их использования, ни до их определения. На самом деле функции и подпрограммы необходимо объявлять только в том случае, если они действительно существуют во внешних системных динамически подключаемых библиотеках (DLL).
Типичные функции в таких языках, как Java или C++, могут быть структурированы следующим образом:

```
void foo( string stringItem ) {
  // The body of the function goes here
}
double bar( int numItem ) { return 23.2; }
```

Однако в VBA функции структурированы, как в следующем примере:

```
Public Sub foo (stringItem As String)
  ' The body of the subroutine goes here
End Sub
Public Function bar (numItem As Integer) As Double bar = 23.2
End Function
```

Чтобы заставить функцию или подпрограмму немедленно выйти, вы можете использовать Exit Function или Exit Sub (соответственно).

Концовка строк

В VBA каждый оператор должен располагаться на отдельной строке, но для обозначения конца каждой строки не требуется специального символа. (Это отличается от многих языков программирования, которые используют точку с запятой для разделения отдельных операторов.)

Чтобы разбить длинную инструкцию VBA на две или более строк, каждая из строк (кроме последней) должна заканчиваться символом подчеркивания (_), которому предшествует хотя бы один пробел:

```
newString = fooFunction ("This is a string", _
                         5, 10, 2)
```

Также можно объединить несколько операторов в одну строку, разделив их двоеточиями:

```
a = 1 : b = 2 : c = a + b
```

Строка не может заканчиваться двоеточием. Строки, которые заканчиваются двоеточием, являются метками, используемыми оператором *Goto*.

Комментарии

Комментарии в VBA, как и в ANSI, C++ и Java, можно создавать только в конце строки. Комментарии начинаются с апострофа (') и заканчиваются в конце строки.

Каждая строка многострочного комментария должна начинаться со своего апострофа:

```
a = b ' This is a really interesting piece of code that
      ' needs so much explanation that I have had to break
      ' the comment over multiple lines.
```

Чтобы закомментировать большие участки кода, используйте следующий код (аналогично C или C++):

```
#If 0 Then ' That's a zero, not the letter 'oh'.
  ' All this code will be ignored by
  ' the compiler at run time!
#End If
```

Использование указателей памяти и выделение памяти

VBA не поддерживает указатели памяти в стиле C. Выделение памяти и сборка мусора выполняются автоматически и прозрачно, как в Java и JavaScript (и в некотором коде C++).

Передача значений «по ссылке» и «по значению»

Большинство языков, включая C++ и Java, передают аргумент в процедуру как копию оригинала. Если оригинал должен быть передан, то может произойти одно из двух:

- передается указатель памяти, который указывает на оригинал в памяти
- передается ссылка на оригинал

То же самое и в VB, за исключением того, что передача копии оригинала называется *передачей по значению*, а передача ссылки на оригинал называется *передачей по ссылке*.

По умолчанию параметры функции и подпрограммы передаются по ссылке. Это означает, что ссылка на исходную переменную передается в аргументе процедуры, поэтому изменение значения этого аргумента внутри процедуры, по сути, также изменяет значение исходной переменной. Это отличный способ вернуть более одного значения из функции или подпрограммы. Чтобы явно аннотировать код, указывающий, что аргумент передается по ссылке, вы можете добавить к аргументу префикс *ByRef*.

Если вы хотите, чтобы процедура не изменяла значение исходной переменной, вы можете принудительно скопировать аргумент. Для этого добавьте к аргументу префикс *ByVal*, как показано в следующем примере. Эта функциональность ByRef/ByVal аналогична способности C и C++ передавать копию переменной или передавать указатель на исходную переменную.

```
Private Sub fooFunc (ByVal int1 As Integer, _
                     ByRef long1 As Long, _
                     long2 As Long) ' Passed ByRef by default
```

В предыдущем примере аргументы *long1* и *long2* по умолчанию передаются по ссылке. Изменение любого аргумента в теле функции изменяет исходную переменную; однако изменение *int1* не изменяет оригинал, поскольку он является копией оригинала.

Определение области действия

Вы можете определить область действия типа данных или процедуры (или даже объекта). Типы данных, функции и подпрограммы (и члены классов), которые объявлены как частные, видны только в этом модуле (или файле), в то время как функции, объявленные как общедоступные, видны во всех модулях; однако вам, возможно, придется использовать полные ссылки, если модули почти выходят за рамки — например, если вы ссылаетесь на функцию в другом проекте.

В отличие от C, VBA не использует фигурные скобки, то есть символы { и }, для определения локальной области видимости. Локальная область в VBA определяется оператором открывающей функции или подопределения (то есть Function или Sub) и соответствующим оператором End (то есть End Function или End Sub). Любые переменные, объявленные внутри функции, доступны только в рамках самой функции.

Использование логического сравнения и присваивания

В VB логическое сравнение и присваивание выполняются с использованием одного знака равенства (=):

```
If a = b Then c = d
```

Это отличается от многих других языков, которые используют двойной знак равенства для логического сравнения и один знак равенства для присваивания:

```
if( a == b ) c = d;
```

Следующий код, допустимый в C, C++, Java и JavaScript, недопустим в VBA:

```
if( ( result = fooBar( ) ) == true )
```

Это должно быть написано на VBA следующим образом:

```
result = fooBar( )
If result = True Then
```

Для других логических сравнений VBA использует те же операторы, что и другие языки (за исключением операторов «равно» и «не равно»). Все булевы операторы сравнения представлены в следующей таблице:

Сравнение	Оператор VBA	Оператор C-стиля
равно	`=`	`==`
не равно	`<>`	`!=`
больше, чем	`>`	`>`
меньше чем	`<`	`<`
больше или равно	`>=`	`>=`
меньше или равно	`<=`	`<=`

Результатом использования одного из логических операторов всегда будет либо *True*, либо *False.*

Использование логических и побитовых операторов

В VBA логические операции выполняются с использованием ключевых слов And, Not, Or, Xor, Imp и Eqv, которые выполняют логические операции AND, NOT, OR, исключающее ИЛИ, логическую импликацию и логическую эквивалентность (соответственно). Эти операторы также выполняют логические сравнения.

Следующий код показывает сравнение, написанное на C или подобном языке:

```
if( ( a && b ) || ( c && d ) )
```

На VBA это будет записано следующим образом:

```
If ( a And b ) Or ( c And d ) Then
```

В качестве альтернативы вышеизложенное можно было бы записать в следующей полной рукописной форме:

```
If ( a And b = True ) Or ( c And d = True ) = True Then
```

В следующей таблице представлено сравнение четырех распространенных логических и побитовых операторов VBA, а также логических и побитовых операторов в стиле C, используемых в C, C++, Java и JavaScript.

Оператор VBA	Побитовый оператор в стиле C	Булев оператор в стиле C
`And`	`&`	`&&`
`Not`	`~`	`!`
`Or`	`|`	`||`
`Xor`	`^`	

Предоставление окон сообщений и полей ввода

Вы можете представить пользователю простые сообщения с помощью функции *MsgBox:*

```
Dim retval As Long
retval = MsgBox("Click OK if you agree.", _
```

```
                    vbOKCancel, "Easy Message")
    If retval = vbOK Then
      MsgBox "You clicked OK.", vbOK, "Affirmative"
    End If
```

Вы также можете получить строки от пользователя с помощью функции *InputBox:*

```
    Dim inText As String
    inText = InputBox("Input some text:", "type here")
    If Len(inText) > 0 Then
      MsgBox "You typed the following: " & inText & "."
    End If
```

Если пользователь нажимает кнопку «Отмена», длина строки, возвращаемой в *inText*, равна нулю.

Глава 2
Начало работы с VBA

еперь, когда вы немного разобрались в VBA, вы готовы приступить к работе с макросами. В этой главе рассматриваются следующие темы:

- Установка VBA для CorelDRAW Graphics Suite X3
- Использование панелей инструментов VBA
- Использование VB-редактора

Установка VBA для CorelDRAW Graphics Suite X3

Прежде чем вы сможете разрабатывать и запускать макросы в CorelDRAW Graphics Suite X3, вам может потребоваться установить компонент VBA.

При выполнении «обычной установки» CorelDRAW Graphics Suite X3, VBA устанавливается по умолчанию.

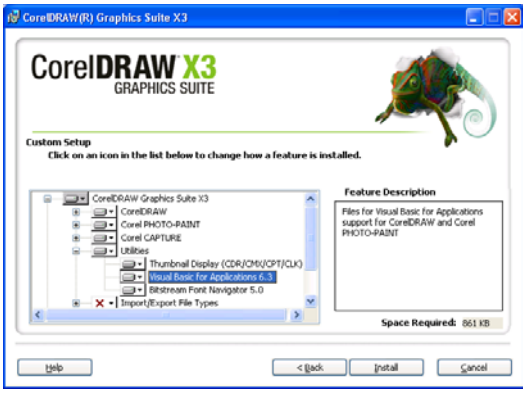
Чтобы установить VBA

1 Вставьте диск CorelDRAW Graphics Suite X3 Disc 1 в дисковод для компакт-дисков.

Если мастер установки не запустится автоматически, щелкните **Пуск > Выполнить** на панели задач Windows, а затем введите *X:\CGS13\Setup.exe* (где X — буква, соответствующая дисководу компакт-дисков).

2 Щелкните **Установить CorelDRAW Graphics Suite X3** и следуйте инструкциям на экране.

3 На странице **Выберите**, какие приложения вы хотите установить, щелкните **Дополнительные параметры**, а затем в появившемся окне **Выборочная установка** задайте для установки **Utilities\Visual Basic for Applications 6.3**.



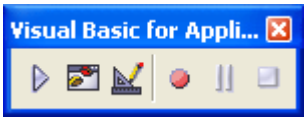
Выберите Utilities\Visual Basic for Applications 6.3 для установки

Использование панелей инструментов VBA

CorelDRAW и Corel PHOTO-PAINT имеют панель инструментов, которая позволяет получить доступ к общим функциям VBA.

Использование панели инструментов VBA в CorelDRAW

CorelDRAW имеет панель инструментов, которая обеспечивает легкий доступ к нескольким функциям VBA и редактору VB.



Панель инструментов Visual Basic для приложений в CorelDRAW

Кнопки панели инструментов выполняют следующие функции:

- запуск макросов
- открытие редактора VB
- переключение редактора VB между его режимами для разработки и запуска макросов
- запуск макросов
- приостановка записи макросов
- остановка записи макросов

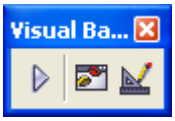
Отображение панели инструментов Visual Basic для приложений в CorelDRAW

- Кликните **Window ▸ Toolbars ▸ Visual Basic for Applications**.

  Флажок рядом с командой означает, что панель инструментов отображается.

Использование панели инструментов VB Editor в Corel PHOTO-PAINT

Corel PHOTO-PAINT имеет панель инструментов, обеспечивающую легкий доступ к VB Editor.



Панель инструментов редактора Visual Basic в Corel PHOTO-PAINT

Кнопки панели инструментов выполняют следующие функции:

- запуск макросов
- открытие редактора VB
- переключение редактора VB между его режимами для разработки и запуска макросов

Отображение панели инструментов VB Editor в Corel PHOTO-PAINT

- Кликните **Window ▸ Toolbars ▸ Visual Basic Editor**.

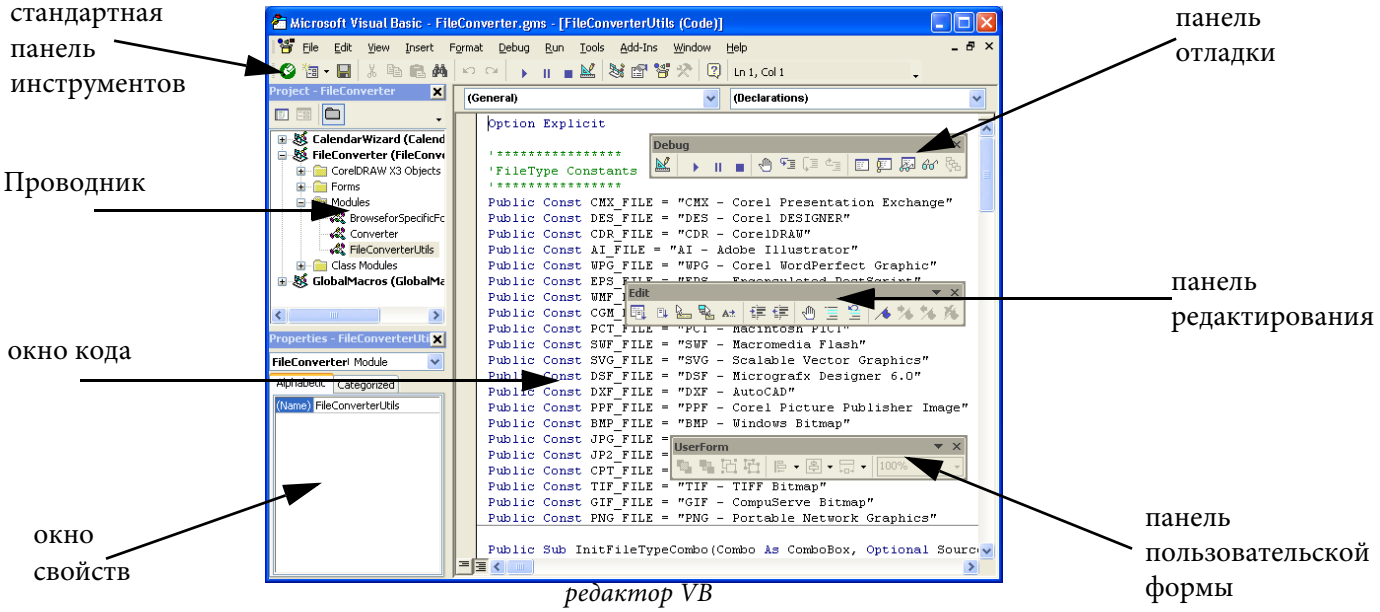
  Флажок рядом с командой означает, что панель инструментов отображается.

Редактор VB

Редактор VB, входящий в состав VBA, аналогичен редактору, входящему в полноценный Visual Basic.

Редактор VB позволяет разрабатывать код и диалоговые окна, просматривать дерево объектов и модули в каждом проекте, задавать индивидуальные свойства для объектов и отлаживать код. Однако важно отметить, что редактор VB для VBA не может компилировать исполняемые (EXE) программные файлы.

Редактор VB имеет несколько окон и панелей инструментов, все из которых обсуждаются в этом разделе. Доступны три окна: **Проводник** (см. стр. 17), **окно свойств** (см. стр. 18) и **окно кода** (см. стр. 19). Доступны четыре панели инструментов (см. стр. 22): **стандартная панель инструментов**, **панель отладки**, **панель редактирования** и **панель пользовательской формы**, из которых вы будете чаще всего использовать панели инструментов **стандартная** и **отладка**.



редактор VB

Редактор VB также позволяет вам получить доступ к обозревателю объектов (см. стр. 23).

Вы можете вызвать редактор VB из CorelDRAW или Corel PHOTO-PAINT. Хотя это запускает VBA как новое приложение в Windows, оно выполняется в процессе CorelDRAW или Corel PHOTO-PAINT.

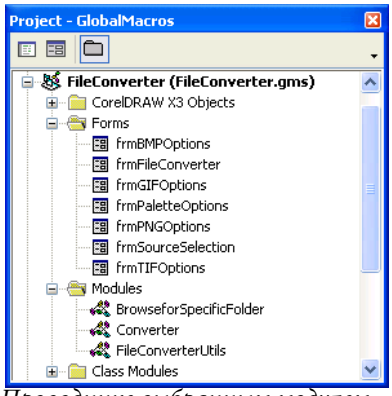
Запуск VB-редактора

- кликните **Tools** ▸ **Visual Basic** ▸ **Visual Basic Editor**, или нажмите **Alt** + **F11**.

Для переключения между VB-редактором и CorelDRAW или Corel PHOTO-PAINT используйте панель задач Windows или нажмите клавиши Alt + F11 или Alt + Tab.

Использование Проводника

Проводник необходим для навигации по проектам VBA и их составным документам/объектам, формам, *модулям* и *модулям классов*.



Проводникс выбранным модулем

Каждому типу элемента в Проводнике присвоен значок:

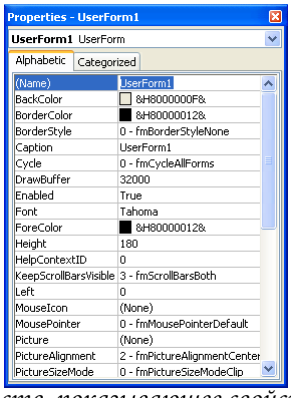
Значок	Значение
	проект
	папка
	документ/объект
	форма
	модуль
	модуль класса

Чтобы отобразить или скрыть Проводник

- Кликните **View** ▸ **Project Explorer**, или нажмите **Ctrl** + **R**.

Использование окна свойств

В окне «Свойства» перечислены все редактируемые свойства текущего выбранного объекта. Многие объекты в VBA, включая проекты, модули, формы и их элементы управления, имеют листы свойств, которые можно изменять.



Окно свойств, показывающее свойства формы

Окно «Свойства» автоматически обновляется при выборе объекта или при изменении свойств выбранного объекта с помощью других методов (например, с помощью мыши для перемещения и изменения размера элементов управления формы).

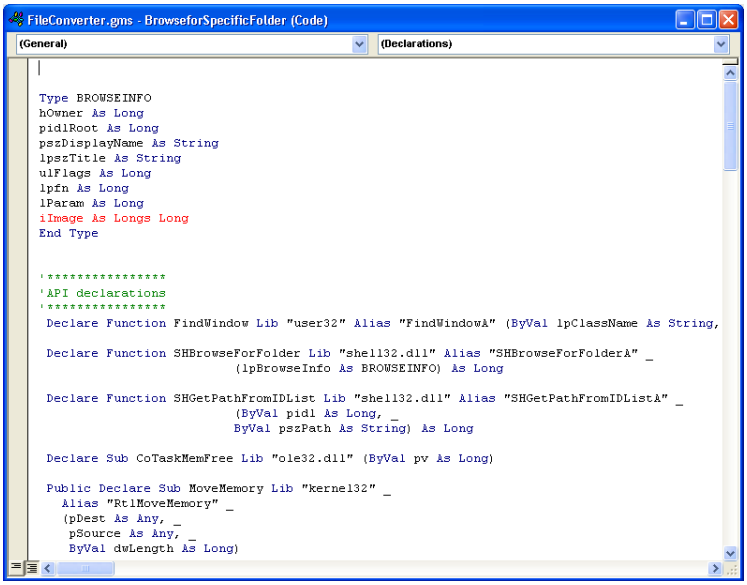
Чтобы отобразить или скрыть окно свойств

- Кликнуть **View** ▸ **Properties window**, или нажать **F4**.

Использование окна кода

Окно кода — это место, где вы проводите большую часть времени при работе с макросами. Стандартный редактор кода в стиле Microsoft® Visual Studio®, окно кода позволяет автоматически форматировать код, раскрашивать и автоматически проверять синтаксис, переходить к определениям и использовать контекстные всплывающие списки и автоматическое завершение.

Если вы уже знакомы с каким-либо из редакторов Microsoft Visual Studio, окно кода редактора VB будет вам хорошо знакомо.



Окно кода VBA

Автоматическое форматирование кода

Редактор VB автоматически форматирует код для вас — редактор VB позаботится даже о том, чтобы ключевые слова, функции, подпрограммы и переменные начинались заглавными буквами, независимо от того, что вы вводите. Вы не можете форматировать код в пользовательском формате, хотя вы можете установить отступ для каждой строки, а также размещение пользовательских разрывов строк.
При вызове функций и подпрограмм, вы должны придерживаться следующих правил:

- Если вы вызываете функцию и используете возвращаемое значение, круглые скобки вокруг параметров обязательны (как и в большинстве современных языков программирования):

```
a = fooFunc (b, c)
```

- Однако, если возвращаемое значение из вызова функции отбрасывается или если вы вызываете подпрограмму, круглые скобки должны быть опущены (в отличие от большинства других языков):

```
barFunc d, e

fooBarSub f
```

- Если вы предпочитаете всегда видеть круглые скобки, используйте ключевое слово *Call* перед функцией или подвызовом:

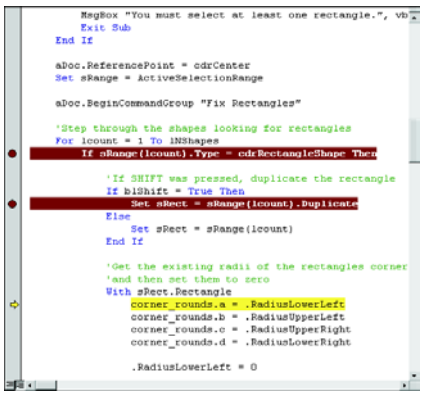
```
Call barFunc (d, e)

Call fooBarSub (f)
```

Автоматическая подсветка синтаксиса

При разработке в окне кода редактор раскрашивает каждое слово в соответствии с его классификацией:

- Ключевые слова и операторы программирования VBA обычно отображаются синим цветом.
- Комментарии отображаются зеленым цветом.
- Весь остальной текст отображается черным цветом.

Такое раскрашивание делает код намного легче для чтения.



Цветовая подсветка синтаксиса

Окно кода также использует следующие методы подсветки:

- Строки кода, содержащие ошибки, отображаются красным цветом.
- Выделенный текст отображается белым на синем фоне.
- Строка, выполнение которой было приостановлено для отладки, выделена желтым цветом.
- Точки останова, которые вы устанавливаете для целей отладки, отображаются в виде красной точки на левом поле с кодом белого цвета на красном фоне.

- Закладки (которые вы устанавливаете в коде) обозначаются синей точкой на левом поле.

Точки останова (вместе с закладками) теряются при выходе из приложения. Дополнительные сведения о них см. в разделе «Установка точек останова» на странице 35.

Вы можете изменить цвета по умолчанию для подсветки синтаксиса, щелкнув **Tools** ▸ **Options**, щелкнув вкладку **Editor format** и внеся необходимые изменения.

Автоматическая проверка синтаксиса

Каждый раз, когда вы перемещаете курсор за пределы строки кода, редактор проверяет синтаксис кода в этой строке; и если он находит ошибку, он меняет цвет текста этой строки на красный и отображает всплывающее предупреждение. Эта проверка в режиме реального времени полезна (особенно при изучении VBA), поскольку она указывает на возможные ошибки в коде без запуска кода.

Вы можете отключить всплывающие предупреждения, щелкнув **Tools** ▸ **Options**, щелкнув вкладку **Editor**, и отключив флажок **Auto syntax check**. Редактор VB по-прежнему проверяет синтаксис и выделяет ошибочные строки красным цветом, но перестает отображать предупреждение при вставке текста из другой строки кода.

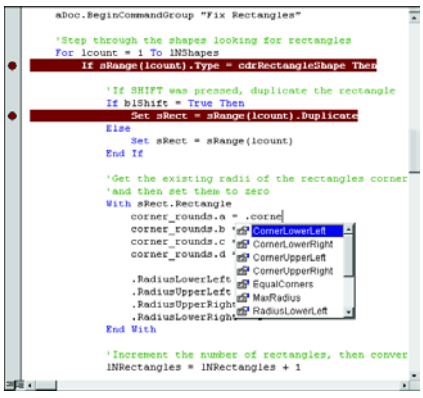
Переход к определениям

Вы можете перейти непосредственно к определению переменной, процедуры или объекта, щелкнув элемент правой кнопкой мыши в окне кода и выбрав Definition. Это приведет вас либо к определению переменной или функции в коде, либо к определению объекта в обозревателе объектов (Object Browser).

Чтобы вернуться туда, где вы запросили определение, щелкните правой кнопкой мыши и выберите **Last position** в окне кода.

Использование контекстных всплывающих списков и автоматического завершения

Когда вы пишете процедуры и определяете переменные, редактор VB добавляет эти элементы во внутренний список, который уже содержит все встроенные ключевые слова и перечисляемые значения. Когда вы печатаете, редактор VB предлагает вам список слов-кандидатов, которые вы можете захотеть вставить в текущую позицию; этот список контекстуален, поэтому редактор VB обычно представляет только слова, допустимые для текущей позиции.



Всплывающее меню автодополнения

Этот список делает разработку кода более быстрой и удобной, особенно потому, что вам не нужно запоминать каждую функцию и имя переменной, а вместо этого можно выбрать их из предоставленного списка. Если вы вводите первые несколько символов слова, которое хотите использовать, список переходит к ближайшему кандидату, который соответствует введенным вами символам. Выберите слово, которое вы хотите использовать, и либо введите символ, который должен следовать за словом (обычно это пробел, перевод строки, круглая скобка, точка или запятая), либо нажмите Tab или Ctrl + Enter, чтобы ввести только слово.

Чтобы принудительно отобразить всплывающее меню, вы можете нажать Ctrl + пробел. Меню прокручивается до слова, которое наиболее точно соответствует введенным вами символам. Этот метод также полезен для заполнения списков параметров при вызове функции или подпрограммы. Если имеется только одно точное совпадение, редактор VB вставляет слово, не открывая список. Чтобы отобразить всплывающий список для выбранного ключевого слова в любое время без его автоматического заполнения, нажмите Ctrl + J.

Панель инструментов

Редактор VB имеет четыре панели инструментов, которые можно использовать для выполнения задач VBA. Стандартная панель инструментов — это панель инструментов по умолчанию.

Стандартная панель инструментов

Панель инструментов «Отладка» содержит кнопки для общих задач отладки (как описано в «Отладка макросов» на странице 35).

Панель инструментов «Отладка»

Панель инструментов редактора содержит кнопки для обычных задач редактирования.

Панель инструментов редактора

Панель пользовательских форм содержит кнопки, предназначенные для создания форм (как обсуждалось в разделе «Создание диалоговых окон» на странице 42).

Панель пользовательских форм

Вы можете отображать или скрывать каждую панель инструментов.

Чтобы отобразить или скрыть панель инструментов

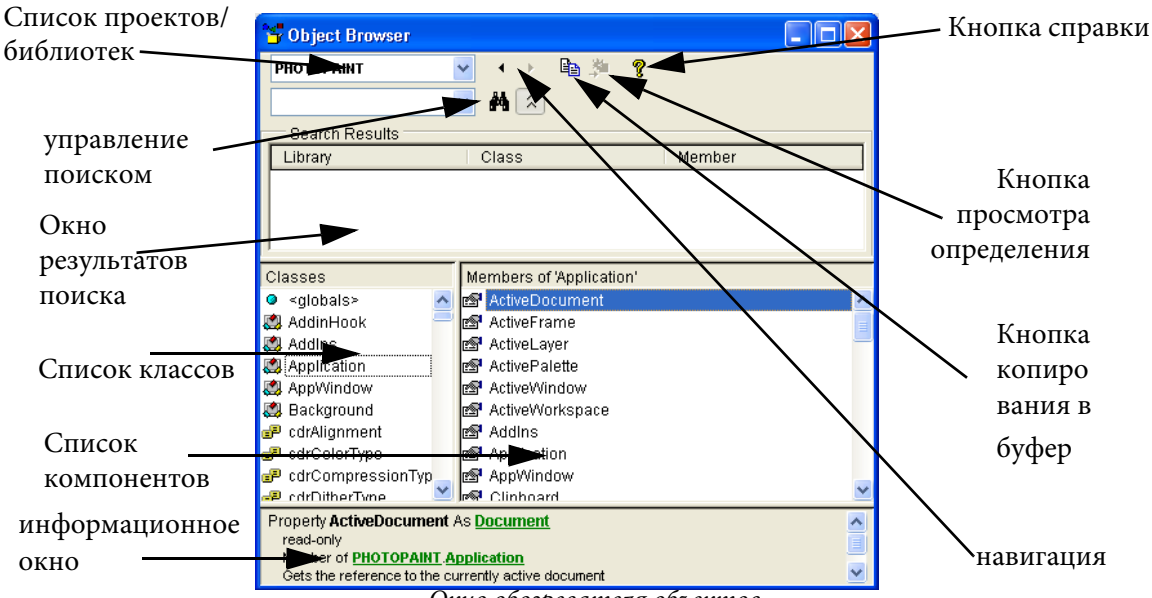
- Кликните **View** ▸ **Toolbars**, а затем выберите команду, соответствующую панели инструментов, которую вы хотите отобразить или скрыть.

  Флажок рядом с командой указывает на то, что в данный момент отображается ее панель инструментов.

Вы можете сделать панель инструментов «плавающей» , перетаскивая ее из строки меню. Вы можете закрепить панель инструментов, перетащив ее в строку меню.

Использование обозревателя объектов

Браузер объектов — один из самых полезных инструментов, предоставляемых редактором VB. Обозреватель объектов отображает полную объектную модель всех компонентов, на которые ссылаются (то есть все объекты ActiveX® или OLE, которые используются в проекте) и, что наиболее важно, объектную модель CorelDRAW или Corel PHOTO-PAINT — все в удобном виде. в использовании, структурированный формат.



Окно обозревателя объектов

Чтобы открыть обозреватель объектов, кликните **View** ▸ **Object Browser**, или нажмите **F2**.

Чтобы сослаться на объектные модели для других приложений, выберите **Tools** ▸ **References**. Доступ к указанным компонентам можно получить с помощью кода VBA.

Все объекты, на которые есть ссылки, а также текущий модуль перечислены в списке «Проект/ библиотека» в верхнем левом углу обозревателя объектов. По умолчанию все классы-члены для объектов, на которые имеются ссылки, представлены в списке классов.

Обозреватель объектов проще использовать, когда отображается только один проект или библиотека. Чтобы отобразить только один проект или библиотеку, выберите его в списке «Проект/библиотека».

Более подробная информация о некоторых элементах обозревателя объектов приведена ниже.

Использование списка классов

Список классов показывает все классы в текущем проекте или библиотеке.

Когда вы выбираете класс в списке овласс, члены этого класса отображаются в списке компонентов.

Каждый проект или библиотека имеет объектную модель, содержащую ряд классов-членов. Рядом с каждым элементом в списке классов есть значок, обозначающий тип класса:

Значок класса	Тип
	глобальное значение
	модуль
	перечисляемый тип
	шрифт
	модуль класса

Глобальные значения (которые применяются ко всему выбранному проекту) включают отдельные элементы из перечисленных типов (таких как выравнивание текстовых абзацев, типы фигур и фильтры импорта/экспорта). Классы-члены объекта имеют свои собственные члены.

Чтобы получить подробную информацию о выбранном элементе, нажмите кнопку справки в верхней части Обозревателя объектов.

Использование списка компонентов

Список компонентов показывает все свойства, методы и события, которые являются членами текущего класса. Каждому участнику дается значок в соответствии с его типом:

Значок класса	Тип
	свойство
	подразумеваемое свойство или свойство по умолчанию
	метод
	событие
	константа

Элементы свойств могут быть простыми типами (такими как логические значения, целые числа или строки) или они могут быть классом или перечисляемым типом из списка классов. Свойство, основанное на классе из списка Класс, наследует все члены этого класса.

У многих классов есть свойство по умолчанию, обозначенное синей точкой на значке. Свойство по умолчанию подразумевается, если имя свойства не указано при получении или установке значения родительского объекта. Например, типы коллекций имеют свойство Item по умолчанию, которое можно индексировать. Однако, поскольку Item обычно является свойством по умолчанию, в таких случаях нет необходимости указывать свойство item. Здесь,

```
ActiveSelection.Shapes.Item(1).Selected = False
```

то же самое, но короче

```
ActiveSelection.Shapes(1).Selected = False
```

потому что Item является свойством по умолчанию или подразумеваемым свойством набора фигур.

Методы широко известны как «функции-члены» — функции, которые класс может выполнять сам с собой. Хорошим примером является метод Move класса Shape, который используется для перемещения фигуры CorelDRAW с помощью вектора [x, y]. Следующий код перемещает выбранные фигуры на 2 единицы измерения вправо и на 3 единицы измерения вверх:

```
ActiveSelection.Move 2, 3
```

Если возвращаемое значение функции не используется, вызов функции не заключает в скобки список аргументов, если только не используется ключевое слово Call.

Дополнительные сведения об `ActiveSelection`, см. в Приложении.

С некоторыми классами связаны различные события. При настройке обработчика событий для события класса, когда это событие происходит в приложении, вызывается обработчик события. Эта функциональность позволяет разрабатывать сложные приложения, которые автоматически реагируют на то, что происходит внутри приложения.

Обычно обрабатываемые события включают события `BeforeClose`, `BeforePrint`, `BeforeSave`, `ShapeMove`, `PageActivate` и `SelectionChange` класса `Document`.

Только классы Document, GlobalMacroStorage и AddinHook имеют события в CorelDRAW. Класс AddinHook в данном руководстве не рассматривается.

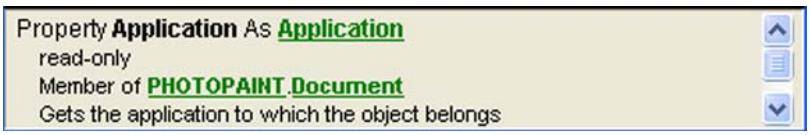
Константы, перечисленные в списке Компоненты, являются либо членами перечислимых типов, либо определены как Public в модуле. Перечислимые типы используются для группировки связанных элементов из закрытого списка, таких как типы фигур CorelDRAW, фильтры импорта/экспорта и выравнивания. Эти константы можно использовать везде, где требуется целочисленное значение.

Большинство констант CorelDRAW начинаются с cdr (например, cdrLeftAlignment, cdrEPS, cdrOutlineBevelLineJoin, cdrCurveShape и cdrSmetricNode). Некоторые константы начинаются с prn или pdf. Visual Basic также имеет свои собственные константы, в том числе для нажатий клавиш (например, vbKeyEnter) и кнопок диалоговых окон (например, vbOK).

Чтобы получить подробную информацию о выбранном элементе, нажмите кнопку «Справка» в верхней части Обозревателя объектов.

Использование информационного окна

Информационное окно предоставляет информацию о выбранном классе или члене класса. Эта информация включает «прототип» члена, его родителя и краткое описание члена, а также указывает, является ли член свойством только для чтения.



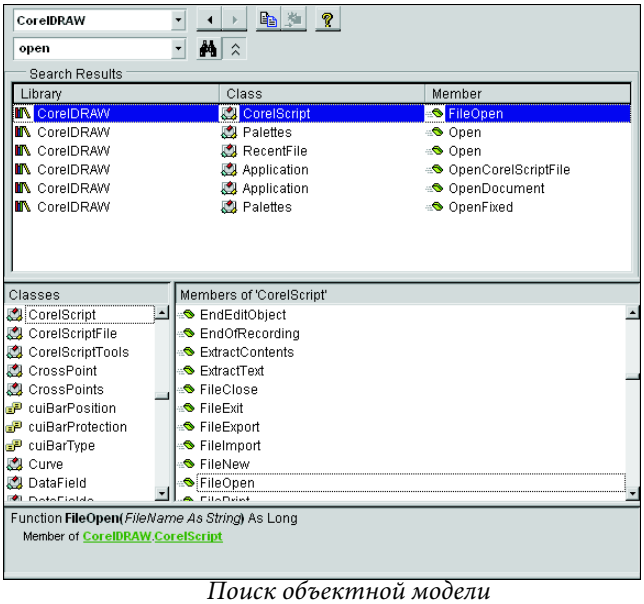
Информационное окно

Типы любых параметров и свойств функции задаются в виде гиперссылок на определение типа или самого класса, если тип определен в рамках текущей объектной модели. Например, свойство Application на предыдущем рисунке является членом класса PHOTOPAINT Document; однако Application также является типом свойства и является классом PHOTOPAINT, поэтому для просмотра класса и его членов можно щелкнуть гиперссылку Application.

Чтобы увеличить высоту информационного окна, перетащите верхнюю границу окна вверх, чтобы открыть его содержимое, или прокрутите вниз с помощью полосы прокрутки в правой части окна.

Управление поиском

Вы можете искать в объектной модели совпадающую строку. Это полезно для поиска класса или члена, имя которого вы можете вспомнить лишь частично, или для поиска классов и членов с похожими именами (например, именами, основанными на слове «open» или содержащими его).



Поиск объектной модели

Для поиска классов и элементов объектной модели введите строку в поле поиска и нажмите кнопку «Поиск». Появится список результатов поиска, отображающий все найденные совпадения в алфавитном порядке. Щелкнув найденное совпадение, вы перейдете к этому элементу в списках классов и компонентов (Member) и отобразите информацию о нем в информационном окне.

Соответствующие имена классов имеют пустой столбец Member в окне результатов поиска.

Чтобы скрыть окно результатов поиска, нажмите кнопку Скрыть результаты поиска  .

Глава 3
Работа с макросами

Теперь, когда вы знакомы с рабочим пространством VBA, можно приступить к работе с макросами. В этой главе рассматриваются следующие темы:

- Создание макросов
- Ссылки на объекты в макросах
- Предоставление обработчиков событий в макросах
- Запуск макросов
- Отладка макросов

 Подробную информацию об объектной модели CorelDRAW см. в Приложении.

Создание макросов

Вы можете создать макрос, либо написав его, либо записав его.

Написание макросов

Прежде чем вы сможете начать писать макрос, вы должны создать для него файл глобального хранилища макросов (GMS), также известный как файл проекта. Файл GMS хранится в папке GMS своего приложения, которая обычно находится по адресу X:\Program Files\Corel\CorelDRAW Graphics Suite X3\<приложение>\ (где X — буква, соответствующая диску, на который установлен CorelDRAW Graphics Suite X3). Редактор VB сохраняет все модули для этого проекта в файле GMS проекта.

Каждый создаваемый проект может содержать несколько модулей. **Проводник** (см. стр. 17) представляет каждый тип модуля в отдельной папке. Вы не можете переместить модуль из одной папки в другую в рамках одного проекта, но вы можете перетащить модуль в другой проект, чтобы сделать его копию там. Существует четыре типа модулей:

- Объекты CorelDRAW X3 или объекты Corel PHOTO-PAINT X3 — используются в основном для обработки событий, содержат один элемент (ThisMacroStorage или ThisDocument соответственно) и не должны использоваться для обычного кода.
- формы — используются для настраиваемых диалоговых окон и пользовательских интерфейсов, включая код для управления ими.
- модули — используются для общего кода и макросов
- модули классов — используются для объектно-ориентированных классов Visual Basic (которые не обсуждаются в этом руководстве)

После создания файла GMS вы можете изменить имя его проекта и добавить в него модули.

Чтобы написать макрос, вы должны использовать VB Editor. Макросы, разработанные в редакторе VB, могут использовать преимущества полного контроля над программированием, включая условное выполнение, циклы и ветвление. По сути, вы можете писать макросы, которые сами по себе являются программами. (Однако в данном руководстве весь код VBA называется макросом, хотя в некоторых контекстах макрос — это только те части кода, которые могут быть запущены CorelDRAW или Corel PHOTO-PAINT.)

Чтобы создать файл проекта

1 Закройте приложение (то есть CorelDRAW или Corel PHOTO-PAINT), для которого вы хотите создать файл проекта.

2 В проводнике Windows перейдите в папку GMS для приложения, для которого вы хотите создать файл проекта.

3 Кликните **File ▸ New ▸ Textdocument**.
В этой папке создается новый пустой текстовый документ.

4 Переименуйте файл как <имя файла>.gms, где <имя файла> — любое допустимое имя файла Windows.

5 Перезапустите приложение, для которого вы создали файл проекта.
Когда вы запускаете редактор VB, созданный вами файл проекта отображается в проводнике проектов как глобальные макросы (<имя файла>.gms).

Чтобы переименовать проект

1 В Проводнике выберите проект, который вы хотите переименовать.

2 В окне «Свойства» отредактируйте значение **(Name)**.
Имена должны соответствовать обычным соглашениям об именах переменных, поэтому они должны начинаться с буквенного символа и не должны содержать ни пробелов, ни специальных символов, кроме подчеркивания (_).

3 Нажмите Enter, чтобы зафиксировать изменения.

Чтобы добавить модуль в проект

1 В Проводнике щелкните правой кнопкой мыши проект, в который вы хотите добавить модуль.

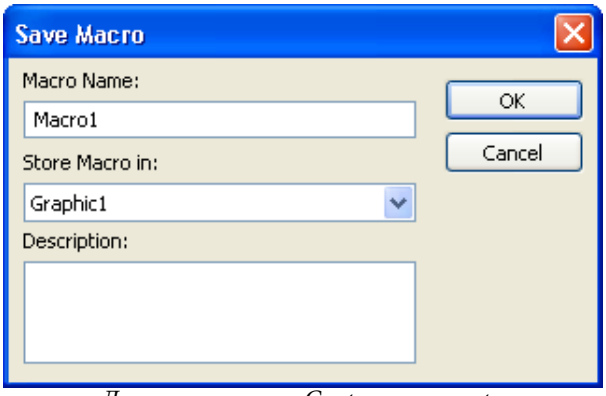
2 Щелкните одно из следующих

- **Insert ▸ Module** — вставляет обычный кодовый модуль
- **Insert ▸ UserForm** — вставляет новую форму (то есть диалоговое окно)
- **Insert ▸ Class Module** — вставляет новый модуль кода класса

Новый модуль сохраняется в папке проекта для этого типа модуля.

Запись макросов

Вы можете записывать макросы непосредственно в CorelDRAW (то есть без использования редактора VB). Запись макроса создает (в выбранном проекте) новый макрос VBA, который затем можно редактировать и настраивать с помощью редактора VB.



Диалоговое окно «Сохранить макрос»

Вы можете предпочесть создавать макросы путем записи, если вы не знакомы с объектной моделью приложения или не уверены, какие объекты и методы использовать. Для многих простых и повторяющихся задач записанные макросы являются быстрым и эффективным решением; в них хранится последовательность нажатий клавиш и действий мыши, выполняемых в CorelDRAW, чтобы вы могли повторно использовать эту последовательность в будущем.

Вы не можете записывать макросы в Corel PHOTO-PAINT.

Чтобы записать макрос

1 Кликните **Tools** ▸ **Visual Basic** ▸ **Record** или кнопку **Record** в **Visual Basic for Applications**

2 В списке «Сохранить макрос в» выберите файл проекта VBA (GMS) или файл CorelDRAW (CDR), в котором вы хотите сохранить записанный макрос.

3 В поле **Macro name** введите имя, уникальное для файла, в котором вы выбрали сохранение макроса.

Имена макросов могут содержать цифры, но должны начинаться с буквы. Имена макросов не могут содержать пробелы или не буквенно-цифровые символы, кроме подчеркивания (_).

4 При желании введите описание макроса в поле **Description**.

5 Кликните **OK**.

CorelDRAW начнёт записывать ваши действия. Чтобы приостановить или возобновить запись, выберите **Tools** ▸ **Visual Basic** ▸ **Pause**, или кнопку **Pause** на панели инструментов.

6 Чтобы остановить запись, нажмите **Tools** ▸ **Visual Basic** ▸ **Stop**, или нажмите кнопку Stop

Макрос сохраняется с указанными вами настройками.

Не все действия можно записать из-за их сложности (хотя многие такие действия можно закодировать вручную в VB Editor). Когда действие не может быть записано, в код помещается следующий комментарий: The recording of this command is not supported (Запись этой команды не поддерживается).

Ссылки на объекты в макросах

Если вы хотите создать ссылку на объект, чтобы вы могли обращаться с этой ссылкой как с переменной (sh в следующем примере), вы можете использовать ключевое слово Set:

```
Dim sh As Shape
Set sh = ActiveSelection.Shapes.Item(1)
```

После того, как вы создадите эту ссылку, вы можете обращаться с ней так, как будто это сам объект:

```
sh.Outline.Color.GrayAssign 35
```

Если выделение изменено, пока sh все еще находится в области действия, sh ссылается на исходную форму из старого выделения и не затрагивается новым выделением. Вы не можете просто присвоить объект переменной, как в следующем примере:

```
Dim sh As Shape
sh = ActiveSelection.Shapes.Item(1)
```

Чтобы освободить объект, вы должны установить для него значение Nothing:

```
Set sh = Nothing
```

Вы также можете проверить, ссылается ли переменная на допустимый объект, используя ключевое слово Nothing:

```
If sh Is Nothing Then MsgBox "sh is de-referenced."
```

Объекты не нужно явно освобождать. В большинстве случаев Visual Basic освобождает объект при удалении переменной при выходе из функции или подпрограммы.

Подробную информацию об объектной модели CorelDRAW см. в Приложении.

Ссылки на коллекции в макросах

Многие объекты являются членами коллекций объектов. Коллекция похожа на массив, за исключением того, что она содержит объекты, а не значения. Несмотря на это, к членам коллекций можно обращаться так же, как к массивам. Например, коллекция, которая часто используется в CorelDRAW, — это коллекция фигур на слое: объект ActiveLayer ссылается либо на текущий слой, либо на слой, выбранный в окне настройки «Диспетчер объектов CorelDRAW».

CorelDRAW содержит множество коллекций: документ содержит страницы, страница содержит слои, слой содержит фигуры, кривая содержит вложенные пути, вложенный путь содержит сегменты и узлы, текстовый диапазон содержит линии и слова, группа содержит фигуры, а приложение содержит окна. Все эти коллекции обрабатываются VBA одинаково.

Подробную информацию об объектной модели CorelDRAW см. в Приложении.

Ссылки на элементы в коллекции

Для ссылки на фигуры слоя используется коллекция фигур этого слоя: ActiveLayer.Shapes. Для ссылки на отдельные фигуры в коллекции используется свойство Item():

```
Dim sh As Shape
Set sh = ActiveLayer.Shapes.Item(1)
```

Большинство элементов коллекции начинаются с 1 и увеличиваются. Для коллекции ActiveLayer.Shapes Item(1) — это элемент «сверху» или «спереди» слоя — другими словами, тот, который находится перед всеми остальными фигурами. Поскольку каждый элемент в коллекции ActiveLayer является объектом типа Shape, вы можете ссылаться на компоненты, просто добавляя соответствующий элемент с записью через точку:

```
ActiveLayer.Shapes.Item(1).Outline.ConvertToObject
```

Иногда отдельные предметы имеют имена. Если у элемента, который вы ищете, есть связанное имя (и вы знаете, какое это имя и в какой коллекции находится элемент), вы можете напрямую сослаться на элемент, используя его имя:

```
Dim sh1 As Shape, sh2 As Shape
Set sh1 = ActiveLayer.CreateRectangle(0, 5, 7, 0)
sh1.Name = "myShape"
Set sh2 = ActiveLayer.Shapes.Item("myShape")
```

Кроме того, поскольку элемент обычно является подразумеваемым членом коллекции или членом по умолчанию, он не является строго обязательным, поэтому последнюю строку предыдущего кода можно переписать следующим образом:

```
Set sh2 = ActiveLayer.Shapes("myShape")
```

Подсчет элементов в коллекции

У всех коллекций есть свойство Count. Это свойство, доступное только для чтения, дает количество элементов в коллекции:

```
Dim count As Long
count = ActiveLayer.Shapes.Count
```

Возвращаемое значение — это не только количество элементов в коллекции, но поскольку коллекция начинается с 1, это также индекс последнего элемента.

Разбор элементов в коллекции

Часто необходимо проанализировать элементы коллекции, чтобы проверить или изменить свойства элементов.

Используя элементы Item() и Count, можно легко пройти через коллекцию элементов. С каждой итерацией можно проверить свойства текущего элемента или вызвать его методы. Следующий код ограничивает все фигуры на слое шириной не более десяти единиц:

```
Dim I As Long, count As Long
count = ActiveLayer.Shapes.Count
For I = 1 to count
  If ActiveLayer.Shapes.Item(i).SizeWidth > 10 Then
    ActiveLayer.Shapes.Item(i).SizeWidth = 10
  End If
Next I
```

Однако есть более удобный способ разбора коллекции в VBA. Вместо использования свойства Count и цикла For-Next в этом методе используется цикл For-Each-In:

```
Dim sh As Shape
For Each sh In ActiveLayer.Shapes
  If sh.SizeWidth > 10 Then
```

```
      sh.SizeWidth = 10
    End If
  Next sh
```

Если вы хотите скопировать выделение, а затем проанализировать его позже, когда оно больше не будет выделено, скопируйте выделение в объект ShapeRange:

```
Dim sr As ShapeRange
Dim sh As Shape
Set sr = ActiveSelectionRange
For Each sh In sr
  ' Do something with each shape
Next sh
```

Использование ярлыков объектов

CorelDRAW предоставляет несколько ярлыков для часто используемых объектов. Эти ярлыки проще в использовании, чем их полноформатные версии, потому что они требуют меньшего набора текста. (Кроме того, использование ярлыков может повысить производительность во время выполнения, поскольку компилятору не нужно определять каждый объект в длинной ссылке, разделенной точками.)
В следующей таблице представлены сочетания клавиш и их полные формы, а также описание каждого из них:

Ярлык	Длинная форма	Описание
`ActiveLayer`	`ActivePage.ActiveLayer`	Получает текущий редактируемый слой на текущей странице документа
`ActivePage`	`ActiveDocument.ActivePage`	Получает текущую страницу активного документа
`ActiveSelection`	`ActiveDocument.Selection`	Получает выделение в активном документе
`ActiveSelectionRange`	`ActiveDocument.SelectionRange`	Получает диапазон выбора в активном документе
`ActiveShape`	`ActiveDocument.Selection.Shapes(1)`	Получает последнюю выбранную фигуру в активном документе
`ActiveView`	`ActiveWindow.ActiveView`	Получает активное представление документа (то есть представление документа в ActiveWindow)
`ActiveWindow`	`ActiveDocument.ActiveWindow`	Получает активное окно (то есть окно, в котором отображается активный документ)

Ярлык можно использовать сам по себе как свойство объекта приложения CorelDRAW.

В качестве элементов данного объекта Document также можно использовать следующие ярлыки: ActiveLayer, ActivePage, ActiveShape и ActiveWindow. Объект Document также имеет свойства Selection и SelectionRange, которые позволяют получить выделение или диапазон выделения (соответственно) из указанного документа независимо от того, активен ли этот документ.

Предоставление обработчиков событий в макросах

Во время работы CorelDRAW генерирует различные события, на которые VBA может реагировать с помощью обработчиков событий — подпрограмм с определенными именами в модуле ThisMacroStorage в CorelDRAW. Каждый файл проекта CorelDRAW VBA (то есть GMS) имеет один модуль ThisMacroStorage в подпапке CorelDRAW X3 Objects для проекта. (Аналогичным образом каждый проект Corel PHOTO-PAINT VBA имеет один модуль ThisDocument в подпапке Corel PHOTO-PAINT X3 Objects для проекта.)

Объект GlobalMacroStorage — это виртуальный объект, представляющий все без исключения открытые документы CorelDRAW. Объект GlobalMacroStorage имеет несколько событий, которые вызываются во время любого события, такого как открытие, печать, сохранение или закрытие документа CorelDRAW (хотя диапазон событий на самом деле больше, поскольку у каждого из них есть «до» и « после» события).

Чтобы отреагировать на событие, необходимо предоставить обработчик события — подпрограмму в любом модуле ThisMacroStorage с определенным именем, для которого CorelDRAW предварительно запрограммирован на поиск. Однако CorelDRAW проверяет все модули ThisMacroStorage во всех установленных проектах, поэтому вы можете создать решение, управляемое событиями, и распространять его как единый файл проекта, как если бы вы предоставили решение любой другой формы. В каждом проекте может быть только один модуль ThisMacroStorage, и он автоматически создается при первом создании проекта.

Например, решению может потребоваться реагировать на закрытие документа, регистрируя закрытие в файле как часть системы управления рабочим процессом и записи. Чтобы реагировать на открытие документа, решение должно реагировать на событие OpenDocument класса GlobalMacroStorage. Для этого откройте модуль ThisMacroStorage для редактирования в VB Editor; затем выберите GlobalMacroStorage из списка объектов в верхней части окна кода, а затем выберите DocumentOpen из списка процедур. Редактор VB создает новую пустую подпрограмму с именем GlobalMacroStorage_DocumentOpen() — или, если она уже существует, помещает в нее курсор. Затем вам нужно только написать код, который добавляет имя открытого файла в журнал. Рекомендуемый способ сделать это — создать общедоступную подпрограмму в другом модуле, который фактически выполняет работу, так что обработчик событий просто вызывает эту подпрограмму при необходимости; это сохраняет размер модуля ThisMacroStorage как можно меньшим, чтобы интерпретатору времени выполнения было легче анализировать все модули ThisMacroStorage каждый раз, когда возникает событие. Следующий код иллюстрирует этот пример:

```
Private Sub GlobalMacroStorage_OpenDocument( ByVal Doc As Document, _
                                             ByVal FileName As String)
            Call LogFileOpen(FileName)
End Sub
```

Вот небольшой пример событий, доступных в CorelDRAW:

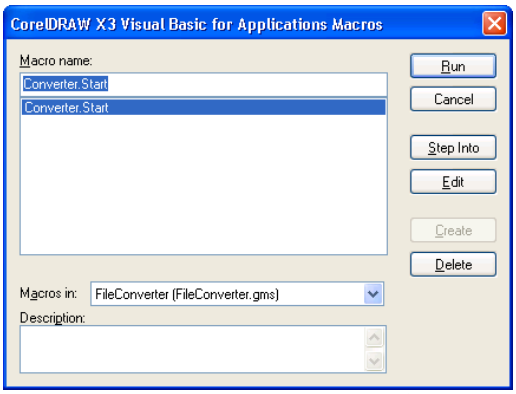
Событие	Описание
`Start`	Возникает, когда пользователь запускает CorelDRAW
`DocumentNew`	Возникает при создании документа; передает ссылку на документ

Событие	Описание
`DocumentOpen`	Возникает при открытии документа; передает ссылку на документ
`DocumentBeforeSave`	Возникает перед сохранением документа; передает имя файла документа в качестве параметра
`DocumentAfterSave`	Возникает после сохранения документа; передает имя файла документа в качестве параметра
`DocumentBeforePrint`	Возникает перед отображением диалогового окна «Печать»
`DocumentAfterPrint`	Возникает после печати документа
`SelectionChange`	Возникает при изменении выбора
`DocumentClose`	Возникает перед закрытием документа
`Quit`	Возникает, когда пользователь выходит из CorelDRAW

Обработчики событий для частых событий, таких как события, связанные с классом Shape, должны быть максимально эффективными, чтобы приложение работало как можно быстрее.

Запуск макросов

Макросы можно запускать либо непосредственно из CorelDRAW или Corel PHOTO-PAINT, либо из редактора VB.



Диалоговое окно CorelDRAW X3 Visual Basic для макросов приложений

Запуск макроса в CorelDRAW

1 Кликните **Tools** ▸ **Visual Basic** ▸ **Play**, или кнопку **Play** в **Visual Basic for Applications**.

2 В списке Макросы в выберите файл проекта VBA (GMS) или файл CorelDRAW (CDR), в котором хранится записанный макрос.

3 Выберите макрос в списке имен макросов.

4 Кликните **Run**.

Запуск макроса в Corel PHOTO-PAINT

1 Кликните Tools Visual Basic Play, или кнопку Play в Visual Basic for Applications.

2 В списке Макросы в выберите файл проекта VBA (GMS) в котором хранится записаный макрос.

3 Выберите макрос в списке имен макросов.

4 Кликните Run.

Чтобы запустить макрос из редактора VB

- Щелкните в любом месте подпрограммы, которая формирует макрос, а затем нажмите **Run ▸ Run macro**.

Отладка макросов

Редактор VB предоставляет мощные средства отладки, общие для языковых редакторов. Можно устанавливать точки останова и выполнять код пошагово.

Вы также можете вносить изменения в код во время его выполнения, а также наблюдать и изменять переменные, но это расширенные методы, которые не обсуждаются в данном руководстве.

Установка точек останова

Точка останова — это маркер в строке кода, который приводит к приостановке выполнения. Чтобы продолжить, вы должны либо перезапустить выполнение, либо выполнить последующие строки кода.

Чтобы установить или снять точку останова, щелкните строку, а затем выберите **Debug ▸ Toggle breakpoint**. По умолчанию строка выделяется темно-красным цветом, а на полях ставится красная точка. Чтобы очистить все точки останова, нажмите **Debug ▸ Clear all breakpoints**.

Чтобы перезапустить код после того, как он остановился в точке останова, нажмите **Run ▸ Continue**. Чтобы приостановить выполнение кода (с немедленным выходом из всех функций и отбрасыванием всех возвращаемых значений), нажмите **Run ▸ Reset**.

Вы также можете «выполнить до курсора», то есть выполнять код до тех пор, пока он не достигнет строки, на которой находится курсор, а затем сделать паузу на этой строке. Для этого щелкните строку, в которой вы хотите приостановить выполнение, а затем щелкните **Debug ▸ Run to cursor**.

Если строка с точкой останова (или курсор при «выполнении до курсора») не выполняется, потому что находится в условном блоке (if-then-else), код не останавливается на этой строке.

Точки останова не сохраняются. Они теряются, когда вы закрываете VB Editor.

Пошаговое выполнение кода

Когда выполнение приостанавливается в точке останова, вы можете продолжить выполнение кода по одной строке за раз. Это позволяет вам проверять значения отдельных переменных после каждой строки и определять, как код влияет на эти значения (и как значения влияют на код). Это называется «пошаговое выполнение кода».

Для пошагового выполнения кода (по одной строке за раз) нажмите **Debug ▸ Step into**. Выполнение переходит к каждой строке во всех вызываемых функциях и подпрограммах.

Чтобы пройти через каждую строку текущей функции или подпрограммы, но не через строки каждой вызванной функции или подсистемы, нажмите **Debug ▸ Step over**. Вызываемые функции и подпрограммы выполняются, но не построчно.

Чтобы выполнить оставшуюся часть текущей функции или подпрограммы, но сделать паузу, когда функция или подпрограмма вернутся в точку, в которой она была вызвана, нажмите **Debug ▸ Step out**. Это быстрый способ вернуться к точке входа функции, чтобы продолжить пошаговое выполнение кода вызывающей функции.

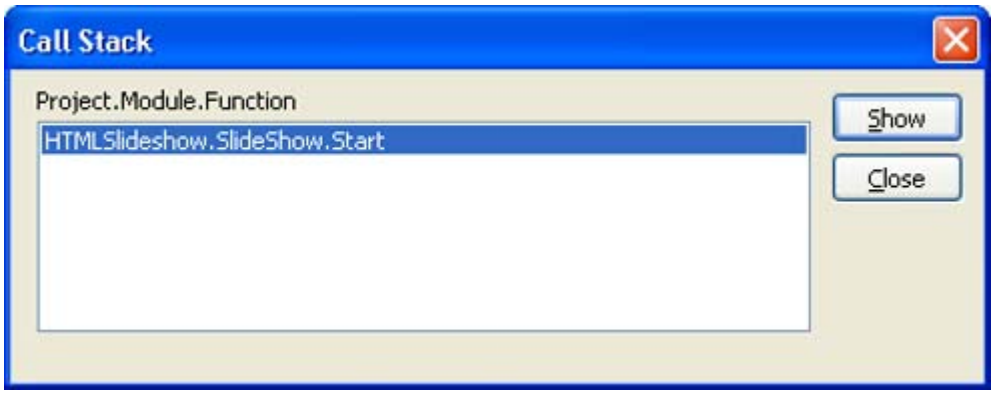
Использование окон отладки

Существует четыре окна, которые используются при отладке кода: окно стека вызовов **Call Stack**, окно немедленной обработки **Immediate**, окно локальных переменных **Locals** и окно контрольных значений **Watches**. Все эти окна предоставляют важную информацию о состоянии функций и переменных во время работы приложения.

Использование окна стека вызовов Call Stack

Окно стека вызовов представляет собой модальное диалоговое окно, в котором указано, какая функция вызывает какую функцию. В длинных сложных приложениях это полезно для отслеживания шагов до конкретной вызываемой функции. Чтобы посетить функцию, указанную в окне, выберите имя функции, а затем нажмите **Show**, или закройте окно.

Чтобы отобразить окно стека вызовов, нажмите **View ▸ Call Stack**….



Окно стека вызовов

Использование окна немедленной обработки Immediate

Окно **Immediate** позволяет вам вводить и запускать произвольные строки кода, пока макрос приостановлен. Это полезно для получения или установки свойства объекта в документе или для установки значения переменной в коде. Чтобы запустить фрагмент кода, введите его в окне Immediate и нажмите клавишу Enter. Код выполняется немедленно.

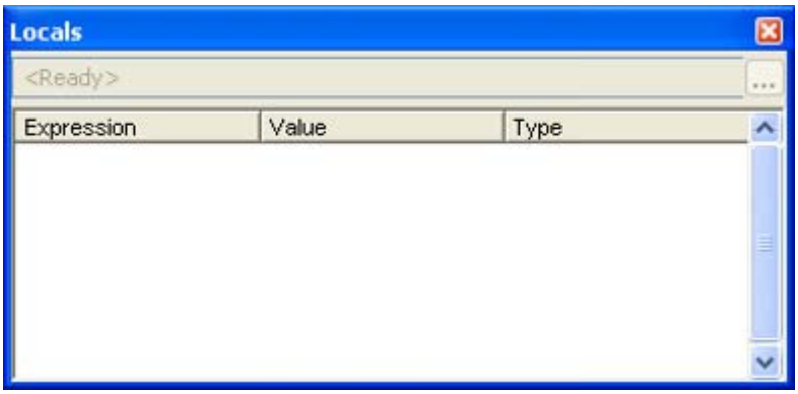
Чтобы отобразить окно Immediate, нажмите **View ▸ Immediate window**.

Окно немедленной обработки

Использование окна локальных переменных Locals

В окне Locals отображаются все переменные и объекты, существующие в текущей области. Тип и значение каждой переменной перечислены в столбцах рядом с именем переменной. Некоторые переменные и объекты могут иметь несколько дочерних элементов, которые можно отобразить, нажав кнопку развертывания дерева рядом с родителем. Многие переменные позволяют редактировать их значение, щелкнув по нему.

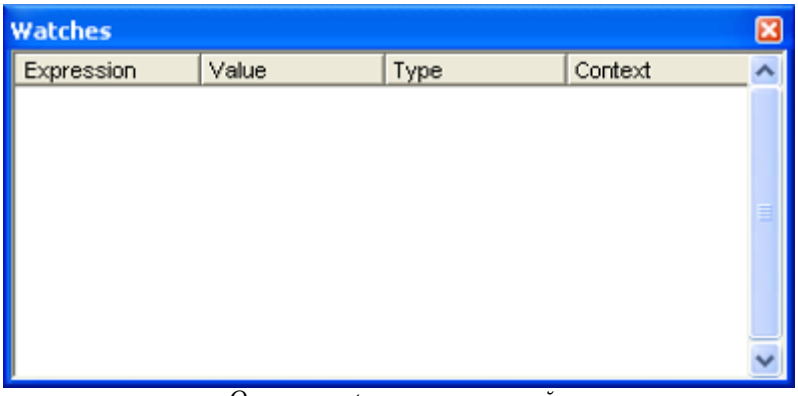
Чтобы отобразить окно Locals, нажмите **View ▸ Locals window**.



Окно локальных переменных

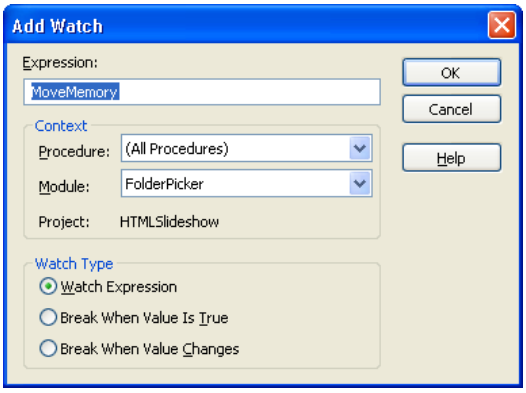
Использование окна контрольных значений Watches

Окно Watches используется для наблюдения за определенными переменными или свойствами объекта. Это очень полезно для выбора только одного или двух значений для просмотра, а не для поиска нужного значения во всех значениях в окне **Locals**.



Окно контрольных значений

Чтобы добавить значение в окно Watches, выберите переменную или объект и его свойство, а затем перетащите выделение в окно Watches; в качестве альтернативы, щелкните элемент, а затем щелкните **Debug ▸ Quick Watch…** чтобы добавить элемент непосредственно в окно **Watches**.



Диалоговое окно «Добавить наблюдение»

Выберите элемент, который вы хотите наблюдать, выберите любые условия для этого наблюдения, а затем нажмите «ОК». Если условие становится истинным, приложение приостанавливает работу, чтобы вы могли изучить код.

Глава 4
Создание пользовательских интерфейсов для макросов

Важной частью многих решений VBA является пользовательский интерфейс. Хорошо спроектированный пользовательский интерфейс делает решение VBA настолько простым в использовании и настолько мощным, что пользователь без колебаний использует его.

Большинство пользовательских интерфейсов для сложных решений VBA основаны на диалоговом окне или форме, но более простые пользовательские интерфейсы можно создать с помощью панелей инструментов и кнопок (которые, в свою очередь, могут быть дополнены заголовками и всплывающими подсказками, а также изображениями или значками). Другие пользовательские интерфейсы требуют, чтобы пользователь щелкал или перетаскивал мышью.

Однако, независимо от характера пользовательского интерфейса, решение VBA можно упростить для развертывания и поддержки, предоставив пользователю некоторую помощь.

В этой главе рассматриваются следующие темы:

- Создание диалоговых окон для макросов
- Создание панелей инструментов и кнопок для макросов
- Обеспечение взаимодействия с пользователем для макросов
- Предоставление справки по макросам

Создание диалоговых окон для макросов

Все диалоговые окна должны соответствовать следующим правилам:

- У них должен быть осмысленный заголовок.
- Они должны предоставлять очевидную функциональность для их отмены или закрытия.
- Их компоновка должна облегчать их использование, но они также должны иметь кнопку «Справка», с помощью которой пользователи могут получить доступ к документации с практическими рекомендациями.
- Каждый их элемент управления должен содержать строку *ControlTipText*, чтобы пользователи могли получать информацию о каждом элементе управления, наводя на него указатель.

Однако существует два вида диалоговых окон: **модальные** (*modal)* и **немодальные** (*modeless)*.

Прежде чем пользователь сможет возобновить выполнение макроса, необходимо выполнить действия в модальных диалоговых окнах. Приложение блокируется до тех пор, пока диалоговое окно не будет закрыто (путем отправки или отмены). Встроенные диалоговые окна, которыми можно управлять с помощью VBA, почти всегда являются модальными.

Немодальные диалоговые окна не блокируют приложение, поэтому их можно оставить открытыми, пока пользователь продолжает работать в приложении. Таким образом, они ведут себя как докеры.

Прежде чем вы сможете закодировать и спроектировать диалоговое окно, вы должны решить, сделать ли его модальным или немодальным.

Выбор между модальными и немодальными диалоговыми окнами

Тип диалогового окна, которое вы должны предоставить, зависит от того, чего вы хотите достичь.

Например, если вы создаете решение, позволяющее применить эффект или действие к одной или нескольким фигурам, и если вы считаете, что пользователь захочет впоследствии применить тот же эффект или действие к другому набору фигур, то вы должен предоставлять немодальное диалоговое окно, чтобы пользователь мог настроить эффект в диалоговом окне один раз, а затем применить его много раз.

С другой стороны, если вы создаете решение, которое является «однократным» сквозным решением (например, замена диалогового окна «Печать» или «Сохранить»), то модальное диалоговое окно будет более подходящим; в таком случае маловероятно, что пользователь захочет повторно применять одни и те же настройки, поэтому повторное открытие диалогового окна будет менее удобным для него, чем закрытие его вручную.

Модальные диалоговые окна обычно имеют следующие особенности:

- кнопка **OK** — выполняет окончательное действие диалогового окна, а затем скрывает диалоговое окно. Эта кнопка присутствует по умолчанию.
- Кнопка **Cancel** — закрывает диалоговое окно, не выполняя действие диалогового окна. Кнопка «Закрыть» в правом верхнем углу диалогового окна обеспечивает ту же функциональность.

Для некоторых модальных диалоговых окон требуется кнопка «Применить», которая выполняет действие диалогового окна, не делая его постоянным, так что отмена диалогового окна отменяет действие.

Если диалоговое окно выполнено в стиле мастера, оно должно иметь кнопки «Назад» и «Далее», а также кнопку «Отмена». На первой странице последовательности кнопка «Назад» должна быть отключена (то есть для ее свойства «Включено» должно быть установлено значение *False*), а на последней странице кнопка «Далее» должна стать кнопкой «Готово», чтобы указать, что последняя страница достигнута. .

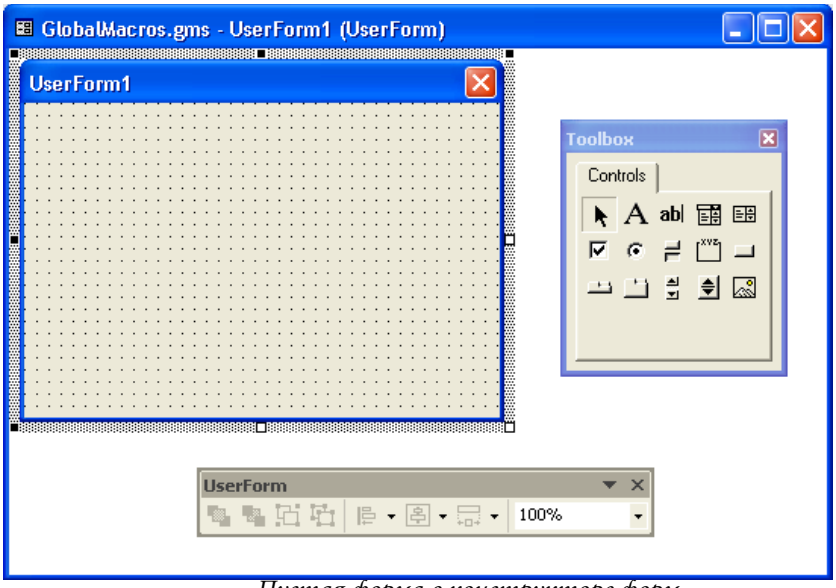
Немодальные диалоговые окна обычно имеют следующие особенности:

- кнопка **Apply** (Применить) или **Create** (Создать) — выполняет действие диалогового окна и может быть специально помечено для описания действия диалогового окна. Эта кнопка должна быть по умолчанию.
- Кнопка **Close** (Закрыть) – закрывает диалоговое окно. Эта кнопка используется после того, как пользователь применил желаемое действие.

После того, как вы выбрали, должно ли ваше диалоговое окно быть модальным или немодальным, вы готовы приступить к его настройке.

Настройка диалоговых окон

Чтобы настроить диалоговое окно для использования в VBA, используется конструктор форм в редакторе VB. Конструктор форм обеспечивает легкий доступ к инструментам для кодирования и разработки форм.



Пустая форма в конструкторе форм

Доступ к конструктору форм можно получить, создав новую пустую форму.

Информацию о доступе к панели инструментов пользовательской формы, которую можно использовать при разработке формы, см. в разделе «Использование панелей инструментов» на стр. 22.

Вы можете протестировать форму в любое время, запустив ее.

Чтобы создать новую пустую форму

- В Проводнике щелкните правой кнопкой мыши проект, к которому вы хотите добавить диалоговое окно, и выберите **Insert** ▸ **UserForm**.

Чтобы изменить заголовок формы, щелкните форму, чтобы выбрать ее, а затем в окне **Properties** (Свойства) измените свойство **Caption** (Заголовок).

Настоятельно рекомендуется давать каждой форме уникальное описательное имя. Вы можете сделать это из окна свойств, но не забывайте следовать правилам именования переменных в VBA.

Чтобы протестировать форму, запустите ее

- Нажмите **F5**.

Диалоговые окна кодирования

Пользовательские диалоговые окна могут быть установлены как модальные или немодальные с помощью параметра *Modal* метода *Show* формы.

Например, форму *frmFooForm* можно отобразить с помощью следующего кода:

```
frmFooForm.Show
```

Поскольку необязательный параметр Modal по умолчанию имеет значение vbModal (или 1), этот код создает модальное диалоговое окно.

Если для параметра Modal задано значение vbModeless (или 0), как в следующем примере, создается немодальное диалоговое окно:

```
frmFooForm.Show vbModeless
```

Чтобы открыть диалоговое окно из макроса, доступного из самого приложения, вы должны создать общедоступную подпрограмму в модуле VBA; если подпрограмма существует в коде формы или в модуле класса, она не может быть доступна из приложения. Подпрограмма, которую вы создаете, не может принимать никаких параметров.

В следующем примере подпрограмма запускает frmFooForm как немодальную форму:

```
Public Sub showFooForm()
  frmFooForm.Show vbModeless
End Sub
```

Когда форма загружается, она запускает собственное событие UserForm_Initialize. Из этого обработчика событий вы должны инициализировать все элементы управления в форме, которые должны быть инициализированы. Дополнительные сведения об обработчиках событий см. в разделе «Предоставление обработчиков событий в макросах» на стр. 33..

Можно запустить одну или несколько других форм из текущей формы с помощью ее функции Show:

```
UserForm2.Show vbModal
```

Однако VBA не возвращает управление, пока все открытые формы не будут выгружены.

Разработка диалоговых окон

Панель инструментов конструктора форм — это основная утилита, которую вы будете использовать при разработке диалоговых окон. Она позволяет добавлять элементы управления в форму путем перетаскивания соответствующего элемента управления из панели инструментов в форму.

Набор инструментов конструктора форм

Панель инструментов конструктора форм позволяет добавлять в форму следующие элементы управления.

Значок	Элемент управления	Функция
	Label	Позволяет предоставить пользователю статический текст (например, инструкции или подписи)
	TextBox	Позволяет указать область, в которую пользователь может вводить текст
	ComboBox	Позволяет предоставить список, из которого пользователь может выбрать элемент и (необязательно) в который пользователь также может ввести текст
	ListBox	Позволяет предоставить список, из которого пользователь может выбрать несколько элементов.
	CheckBox	Позволяет указать флажок, который пользователь может включить (щелкнув, чтобы вставить в него галочку), или отключить (щелкнув, чтобы снять с него галочку), или который может быть затенен (чтобы сделать его недоступным для пользователя)
	OptionButton	Позволяет предоставить «переключатель», связанный с другими переключателями с тем же свойством GroupName, чтобы пользователь мог включить только один из них одновременно.
	ToggleButton	Позволяет предоставить кнопку, которую пользователь может щелкнуть для переключения (например, она нажата или не отображается)
	Frame	Позволяет группировать элементы вместе: элементы, нарисованные в рамке, перемещаются вместе с рамкой
	CommandButton	Позволяет предоставить кнопку, которую пользователь может щелкнуть, чтобы совершить назначенное действие.
	TabStrip	Позволяет предоставить пользователю отдельные представления связанных элементов управления.
	MultiPage	Позволяет предоставить пользователю несколько страниц элементов управления
	ScrollBar	Позволяет предоставить пользователю немедленный доступ к диапазону значений путем прокрутки
	SpinButton	Позволяет улучшить другой элемент управления (например, TextBox), чтобы пользователь мог быстрее установить значение этого элемента управления.
	Image	Позволяет предоставить изображение

Более подробная информация о некоторых из этих элементов управления приведена ниже.

На панели инструментов конструктора форм также имеется инструмент «Указатель»  который позволяет выбирать и перемещать элементы управления в форме.

Для получения дополнительных сведений о любом из этих элементов управления (или о других элементах управления, поддерживаемых VBA) перетащите элемент управления в форму, а затем нажмите клавишу F1, чтобы отобразить соответствующий раздел справки.

Предоставление текстовых полей

Текстовые поля (то есть элементы управления TextBox) являются основой пользовательского ввода. Они просты в использовании, быстро программируются и очень гибки.

Чтобы установить текст в текстовом поле при его инициализации, установите свойство Text элемента управления TextBox (по умолчанию или неявное):

```
txtWidth.Text = "3"
txtHeight = "1"
```

Чтобы получить значение элемента управления TextBox, получите его свойство Text:

```
Call SetSize(txtWidth.Text, txtHeight.Text)
```

Предоставление комбинаций и списков

В поле со списком (то есть в элементе управления ComboBox) пользователь может либо выбрать элемент из списка, либо ввести его в текстовое поле. Вы можете запретить пользователям вводить данные в элемент управления ComboBox, задав для его свойства Style значение fmStyleDropDownList.

В поле со списком (то есть в элементе управления ListBox) пользователь может выбрать один или несколько элементов (обычно от трех до десяти элементов).

Чтобы заполнить список любого типа, вы должны вызвать функцию-член списка AddItem. Эта функция принимает два параметра: строку или числовое значение и позицию в списке. Параметр position является необязательным, поэтому если его опустить, элемент вставляется в последнюю позицию в списке. Например, следующий код заполняет список ComboBox1 четырьмя элементами:

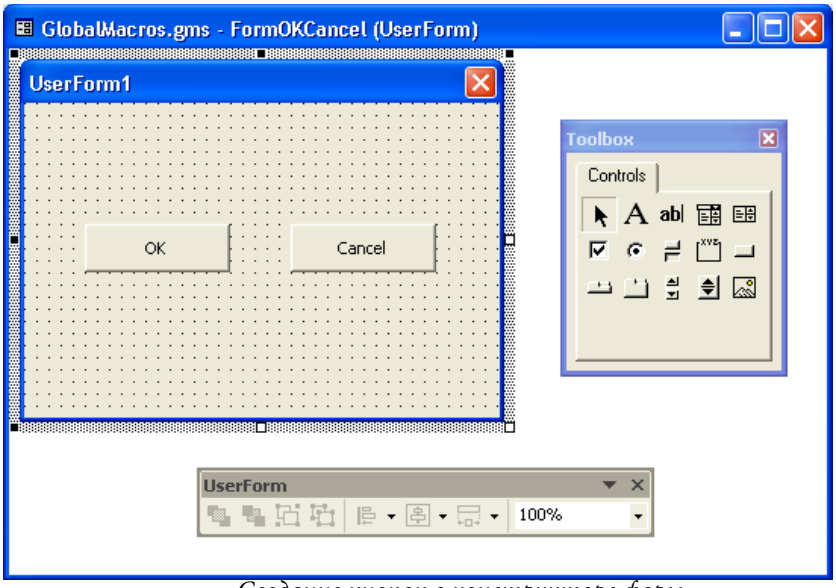
```
ComboBox1.AddItem 1
ComboBox1.AddItem 2
ComboBox1.AddItem 3
ComboBox1.AddItem 0, 0
```

Чтобы проверить, какой элемент выбирается при нажатии кнопки OK, проверьте свойство списка ListIndex. Чтобы получить значение заголовка выбранного элемента, проверьте свойство Text для ComboBox или ListBox:

```
Dim retList As String
retList = ComboBox1.Text
```

Предоставление кнопок

Как обсуждалось ранее, вы можете добавить кнопку в форму с помощью элемента управления CommandButton. Щелкните форму, чтобы добавить кнопку с размерами по умолчанию, или перетащите ее, чтобы создать ее в соответствии с вашими требованиями. Щелкните заголовок, чтобы отредактировать его, или выберите кнопку и измените ее свойство Caption (Заголовок) в окне Properties (Свойства). Вы также можете изменить имя кнопки на что-то более описательное, например, buttonOK или buttonCancel.



Создание кнопок в конструкторе форм

В большинстве форм есть кнопки «OK» и «Отмена». Однако ни одна кнопка не будет работать, пока в ней не будет кода для обработки события нажатия кнопки. (Это связано с тем, что формы VBA управляются событиями.)

Кнопка «Отмена» — самый простой элемент управления: она должна закрыть форму, ничего больше не делая. Чтобы добавить действие отмены к кнопке «Отмена», дважды щелкните кнопку в конструкторе форм, чтобы отобразить ее код в окне **Code**. Это создает новую подпрограмму с именем *cmdCancel_Click:*

Окно кода с кодом для кнопки «Отмена»

Следующий код, примененный к кнопке «Отмена», указывает, что форма должна быть закрыта при нажатии кнопки:

```
Private Sub cmdCancel_Click()
    Unload Me
End Sub
```

Если вы продолжите, установив для свойства Cancel формы значение True, вы обнаружите, что когда пользователь нажимает Escape, запускается событие cmdCancel_Click, и предоставленный вами код выгружает форму.

Точно так же вы можете выбрать кнопку OK и установить для ее свойства по умолчанию значение True, чтобы, когда пользователь нажимал Enter для активации формы, вызывался обработчик события кнопки OK; обработчик события нажатия кнопки OK выполняет функции формы, а затем выгружает форму.

Если форма используется для установки размера выбранных фигур путем установки их ширины и высоты, то обработчик события нажатия кнопки OK может напоминать следующий пример кода (который предполагает, что вы уже создали два текстовых поля с именами *txtWidth* и *txtHeight*):

```
Private Sub buttonOK_Click()
  Me.Hide
  Call SetSize(txtWidth.Text, txtHeight.Text)
  Unload Me
End Sub
```

Точно так же подпрограмма установки размера может выглядеть следующим образом:

```
Private Sub SetSize(width As String, height As String)
  ActiveDocument.Unit = cdrInch
  ActiveSelection.SetSize CDbl(width), CDbl(height)
End Sub
```

Изнутри собственного модуля кода формы объект формы является неявным, поэтому ко всем элементам управления можно просто получить доступ по имени. Доступ к элементам управления из других модулей должен осуществляться по их полному имени, которое может быть *UserForm1.buttonOK*.

Предоставление изображений

Элемент управления Image используется для размещения графики на форме. Изображение (растровое изображение) содержится в свойстве Picture, поэтому вы можете либо загрузить изображение RGB из файла (например, GIF, JPEG или файл BMP Bitmap Windows), либо вставить его в свойство.

Во время выполнения новые изображения можно загрузить в элемент управления Image, изменив свойство **Picture** — используя функцию VBA *LoadPicture* и указав путь к новому файлу изображения — как в следующем примере:

```
Image1.Picture = LoadPicture("C:\Images\NewImage.gif")
```

Создание панелей инструментов и кнопок для макросов

Панели инструментов для любого решения VBA почти всегда видны. Панели инструментов полезны, потому что их графические значки запоминаются, хотя они занимают лишь небольшую площадь, а также потому, что их кнопки могут иметь осмысленные заголовки, которые отображаются как лицевая часть, и всплывающие подсказки, которые отображаются при наведении на них указателя.

Создание панелей инструментов для макросов

Создание панелей инструментов следует тщательно планировать. Лучше иметь много маленьких панелей инструментов, содержащих несколько связанных кнопок, чем одну большую панель инструментов, содержащую все кнопки для всех ваших макросов. Разбив ваши кнопки на небольшие группы, их будет намного проще развернуть с проектами, к которым они относятся.

Чтобы создать панель инструментов

1 Кликните **Tools** ▸ **Options**.

2 Кликните **Workspace** ▸ **Customization** ▸ **Command bars**.

3 Кликните **New**.

4 Введите новое имя для панели инструментов.

5 Установите флажок рядом с названием панели инструментов.

Чтобы добавить макрос-кнопки на панель инструментов

1 Кликните **Workspace** ▸ **Customization** ▸ **Commands**.

2 Выберите **Macros** (в CorelDRAW) или **VBA Macros** (в Corel PHOTO-PAINT) в списке **Commands**.

В списке отображаются полные имена всех общедоступных подпрограмм без параметров из всех файлов установленного проекта (GMS).

3 Перетащите макрос из списка на панель инструментов.

Макрос появляется на панели инструментов со значком макроса по умолчанию.

Добавление заголовков и всплывающих подсказок к макросам

Каждый макрос может иметь как заголовок, так и всплывающую подсказку. Заголовок отображается всякий раз, когда используется команда меню, и может отображаться как часть кнопки, а всплывающая подсказка появляется, когда указатель наводится на элемент меню или кнопку.

Чтобы установить заголовок для макроса

1 Кликните **Workspace** ▸ **Customization** ▸ **Commands**.

2 Выберите **Macros** в списке **Commands**.

В списке отображаются полные имена всех общедоступных подпрограмм без параметров из всех файлов установленного проекта (GMS).

3 Выберите макрос в списке команд.

4 Щелкните вкладку **Appearance**.

5 Введите заголовок в поле **Caption**.

Чтобы связать с заголовком клавишу быстрого доступа, которую можно активировать в сочетании с клавишей **Alt**, введите амперсанд (&) перед любым символом в заголовке, который вы хотите установить в качестве клавиши быстрого доступа. Эта клавиша быстрого доступа применяется только к командам меню, в которых символы сочетания клавиш отображаются с символом подчеркивания (_).

Чтобы установить всплывающую подсказку для макроса

1 Кликните **Workspace** ▸ **Customization** ▸ **Commands**.

2 Выберите **Macros** в списке **Commands**.

В списке отображаются полные имена всех общедоступных подпрограмм без параметров из всех файлов установленного проекта (GMS).

3 Выберите макрос в списке команд.

4 Кликните вкладку **General**.

5 Введите всплывающую подсказку в

поле **Tooltip help**

Связывание изображений или значков с макросами

С командами может быть связано маленькое изображение или значок. Этот значок можно отображать или скрывать на панелях инструментов и в меню, а его размер может быть маленьким (16 × 16 пикселей), средним (32 × 32 пикселя) или большим (48 × 48 пикселей).

Чтобы связать изображение или значок с макросом

1 Кликните **Workspace** ▸ **Customization** ▸ **Commands**.

2 Выберите **Macros** в списке **Commands**.

В списке отображаются полные имена всех общедоступных подпрограмм без параметров из всех файлов установленного проекта (GMS).

3 Выберите макрос в списке команд.

4 Щелкните вкладку **Appearance**.

5 Выполните одно из следующих действий:

- Чтобы применить растровое изображение Windows (BMP) к макросу, нажмите **Import**, перейдите к месту хранения изображения и выберите его. Цвета изображения сопоставляются с ближайшими доступными совпадениями.
- Чтобы создать индивидуальный значок для макроса, отредактируйте пиксели, отображаемые в области **Image**.

Обеспечение взаимодействия с пользователем макросов

Объектная модель CorelDRAW предоставляет два основных способа получения данных от пользователей при работе с документами: захват действий мыши и захват координат.

Захват действий мыши

Вы можете зафиксировать положение щелчка мыши с помощью функции *Document.GetUserClick*. Вы можете зафиксировать положение перетаскивания мышью с помощью функции *Document.GetUserArea*.

Захват щелчков мыши

Чтобы получить от пользователя положение одного щелчка мыши, используйте функцию-член *GetUserClick* объекта *Document*. Эта функция приостанавливает макрос на указанный период времени или до тех пор, пока пользователь не щелкнет где-нибудь в документе или не нажмет Escape. Вот пример использования функции *GetUserClick*:

```
Dim doc As Document, retval As Long
Dim x As Double, y As Double, shift As Long
Set doc = ActiveDocument
doc.Unit = cdrCentimeter
retval = doc.GetUserClick(x, y, shift, 10, True, cdrCursorPick)
```

В этом примере закодированы следующие параметры для функции *GetUserClick*:

- В переменных **x** и **y** возвращается позиция щелчка мыши.
- В параметре *shift* возвращается комбинация клавиш Shift, Ctrl и Alt, которую пользователь удерживает нажатой при щелчке мышью. Клавишам Shift, Ctrl и Alt назначаются значения 1, 2 и 4 (соответственно); эти значения складываются перед возвратом.
- Пользователю дается 10 секунд, чтобы щелкнуть где-нибудь в документе.
- Параметр *SnapToObjects* включен (то есть установлен в значение *True)*.
- Значок инструмента *Pick* используется для значка курсора. (Вы не можете использовать пользовательский значок.)

Возвращаемое значение — 0, 1 или 2, в зависимости от того, успешно ли пользователь завершает щелчок, отменяет ли пользователь нажатием Escape или время ожидания операции истекло (соответственно).

Возвращаемые координаты щелчка относятся к источнику страницы и находятся в единицах VBA документа, поэтому часто необходимо явно указать единицы, в которых вы хотите, чтобы возвращалось значение.

Чтобы получить фигуры, находящиеся под возвращенной точкой клика, используйте функцию *SelectShapesAtPoint*, которая является членом *Page*:

```
doc.ActivePage.SelectShapesAtPoint x, y, True
```

Значение указывает, следует ли выбирать объекты без заливки (True) или нет (False).

Захват перетаскивания мышью

Чтобы получить от пользователя положение перетаскивания мышью, область или прямоугольник, используйте функцию-член *GetUserArea* объекта *Document*. Эта функция приостанавливает макрос на указанный период времени или до тех пор, пока пользователь не щелкнет мышью, не перетащит и не отпустит ее где-нибудь в документе, или пока пользователь не нажмет Escape. Вот пример использования функции *GetUserArea*:

```
Dim doc As Document, retval As Long, shift As Long
Dim x1 As Double, y1 As Double, x2 As Double, y2 As Double
Set doc = ActiveDocument
doc.Unit = cdrCentimeter
retval = doc.GetUserArea(x1, y1, x2, y2, shift, 10, True, cdrCursorExtPick)
ActivePage.SelectShapesFromRectangle x1, y1, x2, y2, False
```

В этом примере закодированы следующие параметры для функции *GetUserArea*:

- В переменных x1, y1, x2 и y2 положение области возвращается как два противоположных угла прямоугольника.

- В параметре *shift* возвращается комбинация клавиш Shift, Ctrl и Alt, которую пользователь удерживает нажатой при щелчке мышью. Клавишам Shift, Ctrl и Alt назначаются значения 1, 2 и 4 (соответственно); эти значения складываются перед возвратом.
- Пользователю дается 10 секунд, чтобы щелкнуть где-нибудь в документе.
- Параметр *SnapToObjects* включен (то есть установлен в значение True).
- Используемый значок курсора определяется последним параметром.

В этом примере код заканчивается выбором фигур, полностью лежащих в пределах области, с помощью метода *SelectShapesFromRectangle* объекта *Page*.

Возвращаемое значение — 0, 1 или 2, в зависимости от того, успешно ли пользователь завершает выбор области, отменяет ли пользователь нажатием Escape или истекает время операции (соответственно).

Возвращаемые координаты области относятся к исходной точке страницы и находятся в единицах VBA документа, поэтому часто необходимо явно указать единицы, в которых вы хотите, чтобы возвращалось значение.

Этот метод возвращает две точки, интерпретируемые как углы прямоугольника. Однако эти две точки также можно использовать в качестве начальной и конечной точек перетаскивания мышью.

Захват координат

Вы можете преобразовать экранные координаты в координаты документа или из документа в экранные координаты с помощью метода *Window.ScreenToDocument* или *Window.DocumentToScreen* (соответственно).

Вы можете определить, находится ли набор координат на фигуре, используя метод *Shape.IsOnShape*.

Преобразование координат

При захвате действий мыши или при разработке сложного решения VBA может быть полезно преобразовать координаты экрана в координаты документа. Это делается с помощью функций-членов *ScreenToDocument* и *DocumentToScreen* объекта *Window*.

Следующий код берет координату с экрана и преобразует ее в точку в документе, которая видна в активном окне.:

```
Dim docX As Double, docY As Double
ActiveDocument.Unit = cdrMillimeter
ActiveWindow.ScreenToDocument 440, 500, docX, docY
```

Следующий код возвращает экранную координату точки в документе, отображаемую на экране:

```
Dim screenX As Long, screenY As Long
ActiveDocument.Unit = cdrMillimeter
ActiveWindow.DocumentToScreen 40, 60, screenX, screenY
```

В обоих случаях преобразованные координаты возвращаются в последних двух параметрах.

Координаты экрана начинаются с верхнего левого угла экрана, поэтому положительные значения Y находятся внизу экрана, а отрицательные значения Y в документе — вверху экрана.

Проверка размещения координат

Вы можете проверить, находится ли набор координат (то есть точка) внутри, снаружи или на контуре кривой, используя функцию-член *IsOnShape* фигуры. Эта функция принимает координату документа и возвращает *cdrInsideShape*, если координата находится внутри фигуры, *cdrOutsideShape*, если координата находится вне фигуры, или *cdrOnMarginOfShape*, если координата находится на контуре фигуры или рядом с ним.

Например, следующий код проверяет, где точка (4, 6) находится по отношению к *ActiveShape*:

```
Dim onShape As Long
ActiveDocument.Unit = cdrInch
onShape = ActiveShape.IsOnShape(4, 6)
```

Предоставление справки по макросам

Настоятельно рекомендуется предоставлять пользователям документацию по вашим макросам.

Одним из решений является предоставление им файла Readme или печатного руководства. Другой вариант — встроить инструкции прямо в диалоговое окно, но при этом используется ценная «недвижимость» на экране. Еще одной альтернативой является создание интерактивной справочной системы, но для этого требуются инструменты и значительный объем дополнительной работы.

По этим причинам вместо этого вы можете предоставить справку в виде обычного текстового файла. Если вы создаете значение реестра при установке проекта, указывающее на расположение файла, вы можете использовать следующую функцию, чтобы открыть его:

```
Public Sub launchNotepad(file As String)
  Shell "Notepad.exe" & " " & file, vbNormalFocus
End Sub
```

Передайте полный путь к текстовому файлу, например *C:\ReadMe.txt*, в файле параметров.

Гораздо более мощное, но все же легко создаваемое решение — использовать HTML. Используя HTML, вы можете включать графику в свою документацию, а также можете направить пользователя в определенное место на странице с помощью «хэш»-ссылки (например, index.html#middle). Если вы знаете, где установлен файл HTML, вы можете использовать следующую функцию, чтобы открыть его:

```
' Put this Declare statement before all Subs and Functions!
Declare Function ShellExecute Lib "shell32.dll" _
    Alias "ShellExecuteA" (ByVal hwnd As Long, _
    ByVal lpOperation As String, ByVal lpFile As String, _
    ByVal lpParameters As String, ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long
Public Sub launchBrowser(url As String)
    ShellExecute 0, vbNullString, url, vbNullString, vbNullString, 5
End Sub
```

Просто передайте имя файла (например, *C:\Program Files\ReadMe.htm*) или URL-адрес в параметре *url*.

Глава 5
Организация и развертывание макросов

Когда вы закончите разработку своего макроса, вы можете сделать его доступным для других пользователей. В этой главе рассматриваются следующие темы:

- Организация макросов
- Развертывание макросов

Организация макросов

Чтобы упростить развертывание макросов, рекомендуется организовать их.

Лучший способ организовать и поддерживать макросы — использовать отдельный модуль для каждого макроса, а затем сгруппировать связанные макросы в один файл проекта (GMS).

Чтобы помочь пользователю найти точку входа в ваши макросы, рекомендуется поместить все общедоступные подпрограммы в один модуль, а затем проинструктировать пользователя, как их найти; таким образом, макросы можно вызывать из CorelDRAW или Corel PHOTO-PAINT.

Развертывание макросов

Вы можете развернуть макросы для установки пользователями. Вы можете развернуть файлы проекта или рабочие области, или и то, и другое.

Развертывание файлов проекта

Поскольку каждый документ CorelDRAW или Corel PHOTO-PAINT имеет встроенный файл проекта (GMS), вы можете явно распространять макросы как часть документа, чтобы при открытии этого документа пользователь имел немедленный доступ к макросам, даже если они не установлены. Таким образом, вы можете, например, настроить макрос, чтобы отслеживать, сколько времени было потрачено на редактирование документа.

Вы должны разработать механизм распространения файлов вашего проекта (GMS) среди пользователей для установки. Хотя можно распространять модули самостоятельно, для ваших пользователей будет намного проще, если вместо этого вы будете распространять файлы проекта (чтобы пользователям не приходилось вручную интегрировать модуль в свой существующий файл проекта).

Развертывание файла GMS

1 Сделайте файл GMS доступным для пользователей.

2 Попросите пользователей установить файл GMS, сохранив его в своей папке GMS.

Развертывание рабочих областей

Когда вы разработали настроенное рабочее пространство, содержащее настроенные панели инструментов, меню и сочетания клавиш, эти настройки могут стать неотъемлемой частью вашего решения для работы с макросами.

Вы можете развернуть всю рабочую область для установки пользователями.

Кроме того, вы можете экспортировать некоторые функции вашего рабочего пространства, чтобы пользователи могли импортировать в свое рабочее пространство только эти функции. Вы можете распространять отдельные панели инструментов и меню, а также полные наборы сочетаний клавиш. Пользователи могут импортировать рабочую область или функции рабочей области с помощью метода *Application.ImportWorkspace* или вручную импортировать файлы рабочей области Corel (XSLT).

Экспорт элементов рабочей области

1 Щелкните правой кнопкой мыши на панели инструментов и выберите **Customize ▸ Workspace ▸ Export workspaces…**.

2 В списке установите флажки рядом с функциями рабочей области, которые вы хотите экспортировать.

3 Щелкните **Save**.

4 В поле **File name** введите имя файла.

Указанные вами функции рабочей области сохраняются в один файл Corel Workspace (XSLT) с указанным вами именем.

Также можно экспортировать строку состояния, а также размеры и положение окон настройки.

При желании вы можете экспортировать каждую функцию рабочей области в отдельный файл. Просто экспортируйте один элемент за раз, чтобы создать серию файлов XSLT.

При экспорте сочетаний клавиш вы экспортируете все сочетания клавиш. Если вы хотите распределить только несколько клавиш, создайте новую рабочую область и удалите из нее все сочетания клавиш, а затем создайте только те клавиши, которые хотите экспортировать.

Чтобы импортировать функции рабочей области вручную

1 Щелкните правой кнопкой мыши на панели инструментов и выберите **Customize-Workspace-Import workspaces…**.
2 Щелкните **Browse**.
3 Найдите и выберите файл рабочей области Corel (XSLT), из которого вы хотите импортировать объекты.
4 Щелкните **Next**.
5 В списке установите флажки рядом с функциями рабочей области, которые вы хотите экспортировать.
6 Щелкните **Next**.
7 Выберите место назначения для функций рабочей области, выполнив одно из следующих действий:

- Включите параметр **Current workspace**, чтобы импортировать указанные вами функции рабочей области в текущую рабочую область, а затем нажмите **Next.**
- Включите параметр **New workspace**, чтобы импортировать функции рабочей области в новую рабочую область. Нажмите **Next** и укажите сведения о рабочей области. Нажмите **Next**.

8 Подтвердите данные импорта и нажмите **Finish**.

Указанные вами функции рабочей области импортируются в указанную вами рабочую область.

Если имя входящей панели инструментов конфликтует с именем существующей панели инструментов, входящая панель инструментов переименовывается.
Если импортированная команда вызывает макрос VBA, который не установлен, он не работает.

Приложение
Понятия объектной модели CorelDRAW

В CorelDRAW объект *Application* является корневым объектом для всех остальных объектов. Чтобы сослаться на объектную модель CorelDRAW из внешнего контроллера, вы используете его объект *Application*. Например, в VB вы должны использовать следующий код:

```
Dim cdr As CorelDRAW.Application
Set cdr = CreateObject("CorelDRAW.Application.13")
```

Хотя приведенный выше код можно использовать в VBA, он не требуется для CorelDRAW, поскольку объект *Application* используется по умолчанию, если не указан другой корневой объект.

Объект *Application* содержит все объекты *Document*, которые создаются при открытии документов в его свойстве *Documents*. Он также содержит все объекты *Window*. Дополнительные сведения см. в разделе «Работа с документами» на стр. 55.

Объекты *Document* содержат коллекции *Pages* всех своих объектов *Page*. Дополнительную информацию см. в разделе «Работа со страницами» на стр. 65.

Отдельные объекты *Page* содержат коллекции *Layers* всех своих объектов *Layer*. Дополнительную информацию см. в разделе «Работа со слоями» на стр. 68.

Наконец, объекты *Layer* содержат коллекции *Shapes* всех своих объектов *Shape*. Дополнительные сведения см. в разделе «Работа с фигурами» на странице 71.

Чтобы просмотреть схему объектной модели CorelDRAW, обратитесь к файлу *CorelDRAW VBA Object Model.pdf* в *Corel\CorelDRAW Graphics Suite X3\Programs* (который обычно устанавливается в папку Windows Program Files).

Если вам нужна информация об объектной модели Corel PHOTO-PAINT, вы можете просмотреть ее схему в *Corel\CorelDRAW Graphics Suite X3\Programs\PP VBA Object Model.pdf* (который обычно устанавливается в папку Windows Program Files).

Работа с документами

При каждом открытии файла CorelDRAW в объекте приложения для этого документа создается новый объект *Document*. Объект *Application* содержит коллекцию *Documents*, обеспечивающую доступ ко всем открытым документам. Порядок документов в коллекции устанавливается в порядке, в котором документы были созданы и открыты.

Полный список объектов *Document* можно просмотреть в объектной модели, но вот список наиболее полезных (некоторые из них более подробно описаны в этой главе):

Компоненты **Document**	**Описание**
`Activate`	Активирует данный документ, помещая его в начало стопки в CorelDRAW. *ActiveDocument* установлен для ссылки на него.
`ActiveLayer`	Представляет слой, который установлен как активный в диспетчере объектов.
`ActivePage`	Представляет активную страницу в документе, то есть ту, которая редактируется в CorelDRAW.
`AddPages` `AddPagesEx`	Добавляет страницы в конец документа
`BeginCommandGroup` `EndCommandGroup`	Создает «группу команд», то есть серию действий, которые отображаются как один элемент в списке **Undo**.
`ClearSelection`	Очищает выделение документа, чтобы отменить выбор всех фигур в документе.
`Close`	Закрывает документ
`Export`	Выполняет простой экспорт из документа
`ExportEx`	Выполняет настраиваемый экспорт из документа
`ExportBitmap`	Выполняет экспорт в растровое изображение с полным контролем
`FileName`	Получает имя файла
`FilePath`	Получает путь к файлу
`FullFileName`	Получает полный путь и имя файла документа
`GetUserArea`	Позволяет добавить интерактивности в макрос, позволяя пользователю перетаскивать область
`GetUserClick`	Позволяет добавить интерактивность в макрос, позволяя пользователю щелкнуть
`InsertPages` `InsertPagesEx`	Вставляет страницы в документ в указанном месте
`Pages`	Предоставляет доступ к коллекции *Pages*
`PrintOut` `PrintSettings`	Печать документа с использованием параметров печати документа
`PublishToPDF` `PDFSettings`	Публикует документ в формате Adobe Acrobat Reader (PDF).
`ReferencePoint`	Получает/устанавливает опорную точку, используемую многими функциями *Shape* (например, для преобразования фигуры или получения положения фигуры).
`Save`	Сохраняет документ, используя текущее имя файла
`SaveAs`	Сохраняет документ в файл с новым именем или с новыми настройками.
`Selection`	Получает выделение как *Shape*
`SelectionRange`	Получает выделение как *ShapeRange*

Компоненты Document	Описание
`Unit` `WorldScale`	Задает единицы измерения документа, используемые функциями, которые принимают значение измерения, например функциями, связанными с размером и положением. Это не зависит от единиц измерения, на которые настроены линейки. Также получает/устанавливает масштаб чертежа. Это изменяет значение в документе; однако оно должно быть явно вычислено в функциях, которые принимают значение измерения, которое по умолчанию использует 1:1.

Объект *Application* также содержит все объекты *Window*.

Объекты *Document* и *Window* имеют множество свойств-членов и методов для выполнения таких действий, как следующие:

- Создание документов
- Открытие документов
- Импорт файлов в документы
- Переключение между документами
- Просмотр документов
- Изменение содержимого в документах
- Установка строки *Undo* (отмена) для документов
- Экспорт файлов из документов
- Печать документов
- Публикация документов в формате PDF
- Закрытие документов

Создание документов

Объект *Application* имеет два метода для создания новых документов.

Первая функция создает новый пустой документ на основе размера страницы, ориентации и стилей по умолчанию:

```
Application.CreateDocument() As Document
```

Вторая функция создает новый документ без названия из указанного шаблона CorelDRAW (.CDT):

```
Application.CreateDocumentFromTemplate(Template As String, _
                [IncludeGraphics As Boolean = True])As Document
```

Новый документ становится активным немедленно, поэтому *ActiveDocument* ссылается на новый документ (а не на старый).

Обе эти функции возвращают ссылку на новый документ, поэтому они обычно используются следующим образом:

```
Dim newDoc as Document
Set newDoc = CreateDocument
```

Класс *Document* не имеет метода для создания экземпляра (объекта) самого себя; только объект *Application* может создавать документы.

Открытие документов

Чтобы открыть документ, вы можете использовать функцию-член *OpenDocument* глобального объекта *Application*:

```
Dim doc As Document
Set doc = OpenDocument("C:\graphic1.cdr")
```

Импорт файлов в документы

Файлы всех поддерживаемых форматов можно импортировать в CorelDRAW. Однако файлы импортируются в слои, поэтому информацию об импорте файлов см. в разделе «Импорт файлов в слои» на стр. 70.

Переключение между документами

Свойство *ActiveDocument* обеспечивает прямой доступ к активному документу, то есть документу, который находится перед всеми другими документами в окне CorelDRAW. *ActiveDocument* является объектом типа *Document* и поэтому имеет все те же элементы — свойства, объекты и методы — что и класс *Document*.

Если открытых документов нет, *ActiveDocument* ничего не возвращает. Вы должны проверить это с помощью следующего кода:

```
If Documents.Count = 0 Then
  MsgBox "There aren't any open documents.", vbOK, "No Docs"
  Exit Sub
End If
```

В классе *Document* также есть метод *Activate*, который активирует документ и выводит его на передний план среди всех открытых документов в CorelDRAW, так что *ActiveDocument* ссылается на активированный документ. Следующий пример кода активирует третий открытый документ (при условии, что в CorelDRAW открыто три или более документов):

```
Documents(3).Activate
```

Использование метода *Activate* для свойства *ActiveDocument* не имеет никакого эффекта.

Если вы хотите, вы можете проверить свойства *FilePath, FileName* и *FullFileName* документа, чтобы активировать правильный из них:

- `FilePath` — для проверки только пути к файлу (например, *C:\My Documents\)*
- `FileName` — для проверки только имя файла (например, *Graphic1.des*)
- `FullFileName` — для проверки как полного пути, так и имени (например, *C:\My Documents\Graphic1.des)*

Вы можете проверить имя, используя следующий код:

```
Public Function findDocument(filename As String) As Document
  Dim doc As Document
  For Each doc In Documents
    If doc.FileName = filename Then Exit For
    Set doc = Nothing
```

```
    Next doc
    Set findDocument = doc
End Function
```

Затем вы можете вызвать метод *Activate* возвращенного документа.

Документы в коллекции *Documents* упорядочены в том порядке, в котором они были созданы и открыты. Чтобы отразить текущий порядок размещения документов в CorelDRAW, необходимо использовать коллекцию *Windows*.

Просмотр документов

В CorelDRAW пользователь может одновременно отображать несколько окон для просмотра одного документа. Для большого документа одно окно может быть увеличено до правого верхнего угла, а другое — до правого нижнего угла. Хотя отдельные окна можно масштабировать и панорамировать независимо друг от друга, перелистывая страницу в одном окне, вы переворачиваете страницу во всех них.

Используя View Manager, пользователь также может сохранять индивидуальные настройки масштабирования и отображения. При выборе представления отображается это местоположение на странице.

В VBA основное различие между объектом *Window* и объектом *View* заключается в том, что объект *Window* обеспечивает доступ к окнам, содержащим каждое представление документа. *Window* — это фрейм, а *View* — это отображение документа, который находится внутри него.

Работа с окнами

Каждый объект *Document* имеет коллекцию *Windows* для просмотра документа. Для переключения между окнами используйте метод *Activate* окна:

```
ActiveDocument.Windows(2).Activate
```

На следующее и предыдущее окна для текущего документа ссылаются в свойствах окна *Next* и *Previous*:

```
ActiveWindow.Next.Activate
```

Чтобы создать новое окно, вызовите функцию-член *NewWindow* объекта *Window:*

```
ActiveWindow.NewWindow
```

Чтобы закрыть окно, вызовите его функцию члена *Close*. Если это последнее окно документа, документ также закрывается:

```
ActiveWindow.Close
```

Работа с представлениями

Элементы *Views* и *ActiveView* буквально являются представлениями документа. Разница между ними заключается в том, что каждый объект *Window* имеет только один член *ActiveView* (который является текущим представлением документа), тогда как объект *View* — это просто записанная память одного конкретного члена *ActiveView* (например, повторная активация этого объекта *View* масштабирует и перемещает элемент *ActiveView* объекта *Window* в положение и настройки масштабирования, хранящиеся в свойствах объекта *View*).
Каждый объект *Window* имеет элемент *ActiveView*. Каждый объект *Document* имеет коллекцию объектов *View* в своем элементе *Views*.

Единственный способ получить доступ к члену *ActiveView* — это свойство *ActiveView* объекта *Window*.

Вы можете создать новый объект *View* и добавить его в коллекцию *Views* объекта *Document* из самой коллекции. Следующий код добавляет текущие настройки *ActiveView* в коллекцию *Views:*

```
ActiveDocument.Views.AddActiveView "New View"
```

Вы также можете создать новое представление с определенными параметрами, используя функцию члена *CreateView* объекта *Document*. Следующий код создает новый объект *View* в позиции (3, 4) в дюймах, используя коэффициент масштабирования 95% и отображая страницу 6:

```
ActiveDocument.Unit = cdrInch
ActiveDocument.CreateView "New View 2", 3, 4, 95, 6
```

Чтобы восстановить представление, вызовите функцию члена *Activate* этого представления. Активное окно документа соответственно настроено на этот вид:

```
ActiveDocument.Views("New View").Activate
```

Масштабирование

Чтобы увеличить масштаб до заданного значения, задайте свойство *Zoom* члена *ActiveView*. Масштаб устанавливается как двойное значение в процентах. Например, следующий код устанавливает коэффициент масштабирования равным 200 %:

```
ActiveWindow.ActiveView.Zoom = 200.0
```

Вы также можете масштабировать элемент *ActiveView* с помощью различных функций-членов: *ToFitAllObjects, ToFitArea, ToFitPage, ToFitPageHeight, ToFitPageWidth, ToFitSelection, ToFitShape, ToFitShapeRange* и *SetActualSize.*

Панорамирование

Чтобы панорамировать элемент *ActiveView*, переместите его исходную точку. Это легко сделать, изменив свойства *OriginX* и *OriginY* члена *ActiveView*. Следующий код перемещает документ на 5 дюймов влево и на 3 дюйма вверх:

```
Dim av As ActiveView
ActiveDocument.Unit = cdrInch
Set av = ActiveWindow.ActiveView
av.OriginX = av.OriginX - 5
av.OriginY = av.OriginY + 3
```

Вы также можете использовать функцию-член *SetViewPoint*:

```
Dim av As ActiveView
ActiveDocument.Unit = cdrInch
Set av = ActiveWindow.ActiveView
av.SetViewPoint av.OriginX - 5, av.OriginY + 3
```

Изменение содержимого в документах

Вы можете изменять содержимое документов независимо от того, активны ли они. Например, если у вас есть ссылка на документ, вы можете добавить новый слой с именем *fooLayer,* используя следующий код:

```
Dim doc As Document
Set doc = Documents(3)
doc.ActivePage.CreateLayer "fooLayer"
```

Если вы хотите создать новый слой в неактивном документе, имя которого вам известно (в следующем примере это *barDoc.cdr*), вы можете использовать следующий код для вызова функции *findDocument()*:

```
Dim doc As Document
Set doc = findDocument("barDoc.cdr")
If Not doc Is Nothing Then doc.ActivePage.CreateLayer "fooLayer"
```

Изменение содержимого в неактивном документе не активирует этот документ. Чтобы активировать документ, вызовите его методом *Activate*.

Установка строки отмены для документов

Две очень полезные функции-члены объекта *Document* позволяют отображать любое количество запрограммированных действий CorelDRAW как одно действие в списке отмены. Это методы *BeginCommandGroup()* и *EndCommandGroup()*, как в следующем примере кода:

```
Dim sh As Shape
ActiveDocument.BeginCommandGroup "CreateCurveEllipse"
    Set sh = ActiveLayer.CreateEllipse(0, 1, 1, 0)
    sh.ConvertToCurves
ActiveDocument.EndCommandGroup
```

После запуска этого кода строка **Undo** в меню **Edit** отображает **Undo CreateCurveEllipse**, а нажатие кнопки **Undo** отменяет не только операцию *ConvertToCurves*, но и операцию *CreateEllipse*.

При необходимости группа команд может содержать несколько сотен команд. Это помогает сделать ваши макросы полностью интегрированными в CorelDRAW.

Экспорт файлов из документов

Файлы всех поддерживаемых форматов можно экспортировать из CorelDRAW.
Файлы экспортируются из объекта «Документ», а не из объектов «Слой», поскольку диапазон экспортируемых фигур часто распространяется на несколько слоев (или даже на несколько страниц). Объект *Document* имеет три функции экспорта — *Export, ExportEx* и *ExportBitmap* — все они могут использоваться для экспорта в растровый или векторный формат.
Для экспорта страницы вам потребуется только имя файла и тип фильтра. Следующий код экспортирует текущую страницу в растровый файл TIFF:

```
ActiveDocument.Export "C:\ThisPage.tif", cdrTIFF
```

Однако такое кодирование дает мало контроля над выводом изображения. Больше контроля можно получить, включив объект *StructExportOptions*, как в следующем коде:

```
Dim expOpts As New StructExportOptions
expOpts.ImageType = cdrCMYKColorImage
```

```
expOpts.AntiAliasingType = cdrNormalAntiAliasing
expOpts.ResolutionX = 72
expOpts.ResolutionY = 72
expOpts.SizeX = 210
expOpts.SizeY = 297
ActiveDocument.Export "C:\ThisPage.tif", cdrTIFF, cdrCurrentPage, expOpts
```

Объект *StructPaletteOptions* также можно включить в вызов функции для форматов изображений на основе палитры, чтобы предоставить настройки для автоматического создания палитры.

Функция ExportEx аналогична функции Export, за исключением того, что она может получить доступ к диалоговому окну фильтра, чтобы получить его настройки, а затем экспортировать файл:

```
Dim eFilt As ExportFilter
Set eFilt = ActiveDocument.ExportEx("C:\ThisPage.eps", cdrEPS)
If eFilt.HasDialog = True Then
  If eFilt.ShowDialog = True Then
      eFilt.Finish
    End If
Else
  eFilt.Finish
End If
```

Третья функция, ExportBitmap, аналогична *ExportEx* тем, что возвращает объект *ExportFilter*, который можно использовать для отображения диалогового окна экспорта. Однако эта функция принимает в качестве параметров отдельные члены объекта StructExportOptions, тем самым упрощая использование функции:

```
Dim eFilt As ExportFilter
Set eFilt = ActiveDocument.ExportBitmap("C:\Selection.tif", _
              cdrTIFF, cdrSelection, cdrCMYKColorImage, _
              210, 297, 72, 72, cdrNormalAntiAliasing, _
              False, True, False, cdrCompressionLZW)
eFilt.Finish
```

Печать документов

Печать документов с помощью VBA проста: почти все параметры, доступные в диалоговом окне «Печать» CorelDRAW, доступны как свойства члена *PrintSettings* объекта *Document*. Когда свойства установлены, печать документа — это просто вызов функции-члена *PrintOut* документа.

Например, следующий код печатает 3 копии страниц 1, 3 и 4 на принтер PostScript® уровня 3:

```
With ActiveDocument.PrintSettings
  .Copies = 3
  .PrintRange = prnPageRange
  .PageRange = "1, 3-4"
```

```
    .Options.PrintJobInfo = True
    With .PostScript
      .DownloadType1 = True
      .Level = prnPSLevel3
    End With
  End With
  ActiveDocument.PrintOut
```

Для каждой страницы в диалоговом окне «Печать» CorelDRAW существует соответствующий объект в объектной модели. В следующей таблице перечислены объекты, соответствующие каждой странице в диалоговом окне «Печать».

Страница параметров в диалоговом окне	Член объекта PrintSettings
General	Properties of PrintSettings
Layout	Properties of PrintSettings
Prepress	Prepress
PostScript	PostScript
Misc	Options

Каждый объект содержит все свойства с соответствующей страницы диалогового окна «Печать». Единственными параметрами печати, которые нельзя задать в VBA, являются параметры макета, однако при необходимости вы можете запустить диалоговое окно «Печать» с помощью функции-члена объекта *PrintSettings ShowDialog*.

Чтобы сбросить настройки печати, вызовите функцию члена *Reset* объекта *PrintSettings:*

```
ActiveDocument.PrintSettings.Reset
```

Ваш код также может получить доступ к любым профилям печати, сохраненным пользователем, из диалогового окна «Печать» с помощью функции-члена *Load* объекта *PrintSettings*:

```
ActiveDocument.PrintSettings.Load "C:\Corel DESIGNER Defaults.prs"
```

Обратите внимание, что для этой функции требуется полный путь к профилю печати.

Вы можете сохранить профили печати, используя функцию сохранения элемента.

Чтобы напечатать только выбранные фигуры, задайте для *Document.PrintSettings.PrintRange* значение *prnSelection*. Чтобы выбрать конкретный принтер, установите *Document.PrintSettings.Printer* для ссылки на соответствующий принтер в коллекции *Application.Printers*.

Публикация документов в PDF

Публикация в формате Adobe Acrobat Reader (PDF) — это двухэтапный процесс. Первым шагом является указание параметров PDF, но этот шаг можно пропустить, если пользователь задает параметры документа в CorelDRAW или предпочитает использовать параметры по умолчанию. Второй шаг — экспорт файла.

Чтобы установить параметры PDF, измените свойства члена *PDFSettings* объекта *Document*. Этот элемент является объектом типа *PDFVBASettings* и имеет свойства для всех параметров PDF, которые можно задать в диалоговом окне CorelDRAW **PublishToPDF**.

The following code exports pages 2, 3, and 5 to a PDF file called MyPDF.pdf:

```
Dim doc As Document
Set doc = ActiveDocument
With doc.PDFSettings
  .Author = "Corel Corporation"
  .Bookmarks = True
  .ColorMode = pdfRGB
  .ComplexFillsAsBitmaps = False
  .CompressText = True
  .DownsampleGray = True
  .EmbedBaseFonts = True
  .EmbedFonts = True
  .Hyperlinks = True
  .Keywords = "Test, Example, Corel, Corel DESIGNER, PublishToPDF"
  .Linearize = True
  .PageRange = "2-3, 5"
  .pdfVersion = pdfVersion13
  .PublishRange = pdfPageRange
  .TrueTypeToType1 = True
End With
doc.PublishToPDF "C:\MyPDF.pdf"
```

Вы можете предоставить больше возможностей пользователю, используя следующий код для отображения диалогового окна «Параметры PDF»:

```
Dim doc As Document
Set doc = ActiveDocument
If doc.PDFSettings.ShowDialog = True Then
  doc.PublishToPDF "C:\MyPDF.pdf"
End If
```

Профили для параметров PDF можно сохранять и загружать с помощью функций-членов *Save* и *Load* объекта *PDFSettings*.

Закрытие документов

Чтобы закрыть документ, вызовите его метод *Close:*

```
ActiveDocument.Close
```

В приведенном выше коде активный документ закрывается, а документ, который был за ним в CorelDRAW, становится новым активным документом. Если код закрывает документ, который не является активным документом, документ, на который ссылается *ActiveDocument*, не изменяется.

Вы должны явно протестировать свойство *Dirty* документа и предпринять соответствующие действия, если документ изменился.

Вы также можете закрыть документ, используя функцию-член *Close* самого объекта *Document:*

```
doc.Close
```

Работа со страницами

Каждая страница в документе является членом коллекции *Pages* объекта *Document*.

Страницы упорядочены в коллекции *Pages* в том же порядке, что и в документе; например, пятая страница документа — это та же страница, что и *ActiveDocument.Pages.Item(5)*. При изменении порядка страниц в представлении **Page Sorter** или при добавлении или удалении страниц коллекция *Pages* немедленно обновляется, отражая новый порядок страниц в документе..

В этом разделе показано, как выполнить следующие действия:

- Создание страниц
- Переключение между страницами
- Изменение порядка страниц
- Изменение размера страниц
- Удаление страниц

Создание страниц

Функция-член для создания страниц на самом деле является членом класса *Document*, а не класса *Page*.

Функция-член документа	Описание
`Document.AddPages()`	Добавляет указанное количество страниц стандартного размера в конец документа.
`Document.AddPagesEx()`	Добавляет указанное количество страниц указанного размера в конец документа
`Document.InsertPages()`	Вставляет страницы размера по умолчанию в указанную позицию в документе.
`Document.InsertPagesEx()`	Вставляет страницы заданного размера в указанную позицию в документе

Например, следующая функция добавляет в конец документа 3 страницы стандартного размера:

```
Public Function AddSomeSimplePages() as Page
    Set AddSomeSimplePages = ActiveDocument.AddPages(3)
End Function
```

Следующий пример функции добавляет 3 страницы размером 8,5 на 11 дюймов в конец документа.

```
Public Function AddSomeSpecifiedPages() as Page
    Dim doc as Document
    Set doc = ActiveDocument
    doc.Unit = cdrInch
```

```
    Set AddSomeSpecifiedPages = doc.AddPagesEx(3, 8.5, 11)
End Function
```

Обе эти функции-члены возвращают первую добавленную страницу. Поэтому, если вы знаете количество созданных страниц, вы можете получить доступ к любой из них, потому что они следуют за страницей, возвращаемой в коллекции *Pages* документа.

Вы можете использовать свойство *Index* возвращаемой страницы, чтобы найти нужное место в коллекции *Pages*, а затем увеличить его для доступа к последующим добавленным страницам:

```
Dim firstNewPage As Page, secondNewPage As Page
Set firstNewPage = AddSomeSimplePages
Set secondNewPage = ActiveDocument.Pages(firstNewPage.Index + 1)
```

Переключение между страницами

Чтобы получить доступ к активной странице активного документа, используйте *Application.ActivePage, ActiveDocument.ActivePage или просто ActivePage*. Они возвращают ссылку на активную страницу в активном документе типа *Page*:

```
Dim pg As Page
Set pg = ActivePage
```

Чтобы получить доступ к активной странице любого документа (независимо от того, активен ли он), используйте свойство *Document.ActivePage* данного документа:

```
Public Function getDocsActivePage(doc As Document) As Page
    Set getDocsActivePage = doc.ActivePage
End Function
```

Чтобы переключаться между страницами, найдите страницу, к которой вы хотите получить доступ, а затем вызовите ее функцию-член *Activate*. Следующий код активирует страницу 3 в документе CorelDRAW:

```
ActiveDocument.Pages(3).Activate
```

Нет необходимости активировать страницу, чтобы внести в нее изменения. Явно ссылаясь на страницу, которую вы хотите отредактировать, вы можете внести эти изменения, не активируя страницу. Следующий код удаляет все фигуры на странице 3 активного документа, не активируя эту страницу.

```
Public Sub DeleteShapesFromPage3()
  Dim doc As Document
  Set doc = ActiveDocument
  doc.Pages(3).Shapes.All.Delete
End Sub
```

Активация страницы в неактивном документе не активирует этот документ. Используйте метод *Document.Activate* для активации документа.

Изменение порядка страниц

Отдельные страницы можно перемещать внутри документа с помощью функции-члена *MoveTo* каждой из страниц. Следующий код перемещает страницу 2 на позицию страницы 4:

```
ActiveDocument.Pages(2).MoveTo 4
```

Активация страницы в неактивном документе не активирует этот документ. Используйте метод *Document.Activate* для активации документа.

Изменение размера страниц

Вы можете изменять размер страниц и устанавливать их ориентацию, устанавливать размер страницы по умолчанию и использовать определенные размеры страниц.

Изменение размера страниц и установка их ориентации

Размер страниц можно изменять по отдельности с помощью функции-члена *SetSize* класса *Page*. Эта функция принимает два значения размера (ширину и высоту) и применяет их к странице. Следующий код изменяет размер активной страницы в активном документе на A4:

```
ActiveDocument.Unit = cdrMillimeter
ActivePage.SetSize 210, 297
ActivePage.Orientation = cdrLandscape
```

Для метода *SetSize* первое число всегда является шириной страницы, а второе число — всегда высотой страницы. Если два числа поменять местами, CorelDRAW изменит ориентацию страницы.

Установка размера страницы по умолчанию

Чтобы установить размер страницы по умолчанию для документа, установите значение элемента в коллекции *Pages* документа с индексом 0:

```
Dim doc As Document
Set doc = ActiveDocument
doc.Unit = cdrMillimeter
doc.Pages(0).SetSize 297, 210
```

В качестве альтернативы вы можете использовать свойство ярлыка *MasterPage* объекта *Document*:

```
Dim doc As Document
Set doc = ActiveDocument
doc.Unit = cdrMillimeter
doc.MasterPage.SetSize 297, 210
```

Использование определенных размеров страницы

Все размеры страниц, определенные (либо CorelDRAW, либо пользователем), хранятся в коллекции *PageSizes* класса *Application*. Вы можете получить все имена размеров страниц, проанализировав эту коллекцию и получив свойство *Name* каждого объекта *PageSize*:

```
Dim pageSizeName As String
```

```
pageSizeName = Application.PageSizes(3).Name
```

Размеры страниц можно указать, используя их имя. Например, следующий код получает значение *PageSize*, называемое визитной карточкой (Business Card):

```
Dim thisSize As PageSize
Set thisSize = Application.PageSizes("Business Card")
```

Вы можете получить фактические размеры каждого объекта *PageSize*, используя свойства *Width* и *Height*. Следующий код извлекает ширину и высоту (в миллиметрах) объекта PageSize 3:

```
Dim pageWidth As Double, pageHeight As Double
Application.Unit = cdrMillimeter
pageWidth = Application.PageSizes(3).Width
pageHeight = Application.PageSizes(3).Height
```

Объекты *PageSize* имеют функцию-член *Delete*, которую можно использовать только для определенных пользователем размеров страниц. Чтобы определить, определяется ли *PageSize* пользователем, проверьте его логическое свойство *BuiltIn*:

```
Public Sub deletePageSize(thisSize As PageSize)
    If Not thisSize.BuiltIn Then thisSize.Delete
End Sub
```

Если вам нужен размер страницы в определенной единице измерения, всегда устанавливайте единицы измерения документа перед получением ширины и высоты.

Удаление страниц

Страницы можно удалить, назвав функцию-член *Delete* каждой страницы следующим образом:

```
ActivePage.Delete
```

Это удалит все фигуры, которые существуют на этой странице, и удалит страницу из коллекции *Pages* документа. Коллекция немедленно обновляется, чтобы отразить изменение. *Delete* нужно вызывать отдельно для каждой страницы, которую вы хотите удалить.

Вы не можете удалить все страницы в документе. Используя следующий код, вы можете избежать попыток удалить последнюю оставшуюся страницу в документе:

```
If ActiveDocument.Pages.Count > 1 Then ActivePage.Delete
```

Работа со слоями

Все слои в документе содержатся в объектах *Page* документа.

В CorelDRAW все видимые фигуры содержатся в слоях на каждой странице. Хотя один слой может иметь разные свойства на каждой странице, все страницы имеют одинаковые слои, и все эти слои имеют одинаковые имена на каждой странице.

Если вы хотите создавать новые фигуры в CorelDRAW с помощью VBA, вы должны использовать функции-члены создания фигур класса *Layer*. Дополнительные сведения см. в разделе «Работа с фигурами» на странице 71.

В этом разделе описывается, как выполнять следующие задачи:

- Создание слоев
- Активация слоев
- Блокировка и скрытие слоев
- Изменение порядка слоев
- Переименование слоев
- Импорт файлов в слои
- Удаление слоев

Создание слоев

Чтобы создать слой, используйте функцию-член *CreateLayer класса Page*.

Следующий код создает новый слой под названием `My New Layer`:

```
ActivePage.CreateLayer "My New Layer"
```

Новый слой всегда располагается вверху списка неосновных слоев.

Активация слоев

Чтобы активировать слой, вызовите функцию-член *Activate* этого слоя:

```
ActivePage.Layers("Layer 1").Activate
```

Это делает слой активным, но не включает слой и не делает его видимым, если он уже заблокирован или скрыт.

Блокировка и скрытие слоев

Объекты слоя имеют свойства *Enabled* и *Visible*, которые определяют (соответственно), можно ли редактировать слой и отображается ли его содержимое в CorelDRAW. Оба свойства являются булевыми. Установив для обоих свойств значение *True*, вы разблокируете и отобразите слой для редактирования. Однако, установив для любого свойства значение *False*, вы заблокируете слой таким образом, что его нельзя будет редактировать.
Следующий пример кода блокирует, но отображает слой на активной странице:

```
ActivePage.Layers("Layer 1").Visible = True
ActivePage.Layers("Layer 1").Editable = False
```

Результат любых изменений этих свойств немедленно отображается в диспетчере объектов CorelDRAW.

Предыдущий пример кода влияет только на активную страницу. Вы можете установить параметры слоя для данной страницы, указав страницу из коллекции *Pages* или сославшись на *ActivePage*. Чтобы внести изменения во все страницы документа, используйте свойство *MasterPage* документа:

```
ActiveDocument.MasterPage.Layers("Layer 1").Visible = True
```

Изменение порядка слоев

Вы можете использовать VBA для изменения порядка слоев. Класс *Layer* имеет две функции-члена: *MoveAbove* и *MoveBelow*. Оба метода принимают объект *Layer* в качестве единственного параметра, поэтому слой перемещается выше или ниже этого слоя.

Следующий код перемещает слой под названием **Layer 1** сразу под **Guides** слоя:

```
Dim pageLayers As Layers
Set pageLayers = ActivePage.Layers
pageLayers("Layer 1").MoveBelow pageLayers("Guides")
```

Изменение немедленно отражается в диспетчере объектов CorelDRAW (хотя иногда эффекты проявляются только в диспетчере слоев).

Переименование слоев

Слои можно переименовать, отредактировав свойство *Name* слоя.

Следующий код переименовывает Layer 1:

```
ActivePage.Layers("Layer 1").Name = "Layer with a New Name"
```

Импорт файлов в слои

Файлы всех поддерживаемых форматов можно импортировать в CorelDRAW.

Файлы импортируются в слои; поэтому функции Import и ImportEx являются членами объекта *Layer*. Следующий код импортирует файл *C:\logotype.gif* на активный слой в центре страницы:

```
ActiveLayer.Import "C:\logotype.gif"
```

При импорте любые формы, которые были выбраны ранее, отменяются, а содержимое импортируемого файла выделяется.

Чтобы изменить положение или размер импортированных фигур, получите выделение документа:

```
ActiveDocument.Unit = cdrInch
ActiveSelection.SetSize 3, 2
```

Некоторые форматы файлов, в частности Encapsulated PostScript (EPS) и PDF, можно импортировать с помощью одного из двух фильтров. Используя фильтр EPS, вы можете импортировать файл EPS как размещаемый объект, который можно распечатать, но не изменить; вы также можете интерпретировать часть PostScript файла EPS, импортируя фактическую иллюстрацию из EPS, а не только помещаемый заголовок TIFF или WMF с низким разрешением (который не так легко редактировать). Чтобы указать, какой фильтр использовать, включите необязательный параметр *Filter*:

```
ActiveLayer.Import "C:\map.eps", cdrPSInterpreted
```

Функция *ImportEx* обеспечивает гораздо лучший контроль над фильтром импорта за счет дополнительного использования объекта *StructImportOptions*. Следующий код импортирует файл как связанный файл:

```
Dim iFilt As ImportFilter
Dim importProps As New StructImportOptions
importProps.LinkBitmapExternally = True
Set iFilt = ActiveLayer.ImportEx("C:\world-map.tiff", cdrAutoSense, importProps)
iFilt.Finish
```

Удаление слоев

Слои можно удалить, вызвав функцию *Delete* класса *Layer*. Доступ к коллекции *Layers* возможен только из объекта *Page*. Вызов функции *Delete* полностью удаляет слой из документа, удаляя все фигуры в этом слое на всех страницах документа.

Следующий код удаляет слой с именем Layer 1:

```
ActivePage.Layers("Layer 1").Delete
```

Работа с фигурами

Все фигуры в документе содержатся в объектах слоя документа.

Объекты типа *Shape* — это фактические фигуры, существующие в документе CorelDRAW. Если вы измените свойства фигуры в документе (например, переместите фигуру, измените ее размер или заполните ее новой заливкой), эти изменения сразу же отобразятся в объектной модели VBA..

Объекты *Shape* существуют как члены объектов *Layer*. Каждый объект *Layer* содержит коллекцию *Shapes*, содержащую все фигуры слоя. Первый элемент этой коллекции — фигура вверху слоя (другими словами, та, что поверх всех остальных), а последний элемент — фигура внизу; если вы измените порядок фигур на странице, коллекция обновится, чтобы отразить это изменение.

Каждый объект *Page* в документе также имеет коллекцию *Shapes*, содержащую все коллекции *Shapes* на всех слоях (включая все основные слои) на странице. Первая фигура в коллекции — это фигура в самом верху страницы, а последняя — фигура внизу.

В этом разделе описывается, как выполнять следующие задачи:

- Создание фигур
- Выбор фигур
- Определение типа фигуры
- Изменение свойств формы
- Раскрашивание фигур
- Дублирование фигур
- Применение эффектов к фигурам

Создание фигур

Объекты *Shape* представляют собой фигуры, которые вы создаете в документе CorelDRAW с помощью инструментов рисования. Среди форм, которые вы можете создавать, есть прямоугольники, эллипсы, кривые и текстовые объекты.

Поскольку каждый объект *Shape* является членом коллекции *Shapes*, которая является членом одного из объектов *Layer* на странице, методы для создания новых фигур принадлежат классу *Layer*, и все они начинаются со слова *Create*.

Создание прямоугольников

Есть две функции для создания новых прямоугольных фигур — *CreateRectangle* и *CreateRectangle2*, обе из которых возвращают ссылку на новый объект *Shape*. Эти две функции отличаются только параметрами, которые они принимают.

Например, следующий код, использующий *CreateRectangle*, создает простой прямоугольник размером два на один дюйм, расположенный на шесть дюймов снизу вверх и на три дюйма слева от страницы:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
Set sh = ActiveLayer.CreateRectangle(3, 7, 6, 5)
```

Параметры задаются как левое, верхнее, правое, нижнее и измеряются в единицах измерения документа (которые можно явно задать перед созданием фигуры).

Альтернативный метод *CreateRectangle2* создает прямоугольник, указывая координаты его нижнего левого угла, а также его ширину и высоту. Следующий код создает тот же прямоугольник, что и выше:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
Set sh = ActiveLayer.CreateRectangle2(3, 6, 2, 1)
```

Предлагаются альтернативные методы, упрощающие разработку решений; они обеспечивают идентичную функциональность.

Прямоугольники со скругленными углами также можно создавать с помощью методов *CreateRectangle* и *CreateRectangle2*. Обе функции имеют четыре необязательных параметра, которые задают скругленность углов при создании прямоугольника, но эти значения немного отличаются для двух функций.

Четыре необязательных параметра метода *CreateRectangle* принимают целые значения в диапазоне от 0 до 100 (по умолчанию ноль). Эти значения определяют радиус четырех углов как целое число в процентах от половины длины самой короткой стороны. Следующий код воссоздает прямоугольник размером два на один дюйм из предыдущего, но для четырех угловых радиусов установлены значения 100 %, 75 %, 50 % и 0 % от половины самой короткой стороны; другими словами, радиусы будут 0,5 дюйма, 0,375 дюйма, 0,25 дюйма и вершина:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
Set sh = ActiveLayer.CreateRectangle(3, 7, 6, 5, 100, 75, 50, 0)
```

Четыре параметра определяют радиусы углов в следующем порядке: верхний левый, верхний правый, нижний левый, нижний правый.

Метод *CreateRectangle2* определяет радиусы углов в том же порядке, за исключением того, что он принимает двойные значения (с плавающей запятой), которые являются измерениями радиуса в единицах измерения документа. Следующий код создает такой же прямоугольник со скругленными углами:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
ActiveDocument.ReferencePoint = cdrBottomLeft
Set sh = ActiveLayer.CreateRectangle2(3, 6, 2, 1, 0.5, 0.375, 0.25, 0)
```

Вы должны ограничить радиусы, передаваемые методу *CreateRectangle2*, менее чем половиной меньшего размера прямоугольника.

Создание эллипсов

Существует два метода создания эллипсов: *CreateEllipse* и *CreateEllipse2*. Они различаются параметрами, которые они принимают, поэтому вы можете создать эллипс на основе его ограничивающей рамки или вместо этого на основе его центральной точки и радиуса. Обе функции также создают дуги или частичные эллипсы, сегменты или сектора круговой диаграммы.

Метод *CreateEllipse* принимает четыре параметра, которые определяют его ограничивающую рамку так же, как и для *CreateRectangle*, — другими словами, слева, сверху, справа, снизу в единицах измерения документа. Следующий код создает круг диаметром 50 миллиметров.:

```
Dim sh As Shape

ActiveDocument.Unit = cdrMillimeter

Set sh = ActiveLayer.CreateEllipse(75, 150, 125, 100)
```

Для создания дуги или сегмента требуются три дополнительных параметра: начальный угол, конечный угол и логическое значение, определяющее, является ли дуга (False) или сегментом (True). Углы измеряются так, что ноль располагается горизонтально прямо на странице, а положительные значения представляют собой градусы от нуля, движущиеся против часовой стрелки. Дуга или круговая диаграмма рисуются от начального угла до конечного угла. Следующий код создает форму буквы «С»:

```
Dim sh As Shape

ActiveDocument.Unit = cdrMillimeter

Set sh = ActiveLayer.CreateEllipse(75, 150, 125, 100, 60, 290, False)
```

Метод *CreateEllipse2* создает эллипсы на основе их центральных точек, а также радиусов по горизонтали и вертикали. (Если задан только один радиус, создается круг.) Следующий код создает такой же круг диаметром 50 миллиметров, как и предыдущий код:

```
Dim sh As Shape

ActiveDocument.Unit = cdrMillimeter

Set sh = ActiveLayer.CreateEllipse2(100, 125, 25)
```

Чтобы создать эллипс, укажите второй радиус. (Первый радиус — это горизонтальный радиус, второй — вертикальный радиус.)

```
Dim sh As Shape

ActiveDocument.Unit = cdrMillimeter

Set sh = ActiveLayer.CreateEllipse2(100, 125, 50, 25)
```

Для создания дуги или сегмента используйте те же дополнительные параметры, что и для метода *CreateEllipse*.

Создание кривых

Кривые состоят из нескольких объектов: у каждой кривой есть один или несколько объектов-членов *SubPath*, у каждого *SubPath* есть один или несколько объектов *Segment*, а у каждого *Segment* есть два объекта *Node*, а также два свойства позиции/угла контрольной точки.

Вы можете создать объект кривой в CorelDRAW, используя метод *CreateCurve* объекта *Application*.

Создайте новый *SubPath* внутри объекта *Curve* с помощью функции-члена *CreateSubPath*. Это создаст первый узел (*Node*) кривой.

Затем добавьте новый сегмент линии или кривой к *SubPath* с помощью функций-членов *AppendLineSegment* и *AppendCurveSegment*. Это добавляет еще один узел и устанавливает позиции управляющих ручек для этого сегмента. Повторяйте это столько раз, сколько потребуется, чтобы создать кривую.

Создайте форму кривой на слое с помощью функции-члена *CreateCurve*.

Вы можете добавить дополнительные подпути к кривой и построить их с собственными узлами. Вы также можете закрыть кривую, установив для ее свойства *Closed* значение *True*.

Следующий код создает замкнутую кривую D-образной формы:

```
Dim sh As Shape, spath As SubPath, crv As Curve
ActiveDocument.Unit = cdrCentimeter
Set crv = Application.CreateCurve(ActiveDocument) ‡Create Curve object
Set spath = crv.CreateSubPath(6, 6) ‡ Create a SubPath
spath.AppendLineSegment 6, 3 ‡ Add the short vertical segment
spath.AppendCurveSegment 3, 0, 2, 270, 2, 0 ‡ Lower curve
spath.AppendLineSegment 0, 0 ‡ Bottom straight edge
spath.AppendLineSegment 0, 9 ‡ Left straight edge
spath.AppendLineSegment 3, 9 ‡ Top straight edge
spath.AppendCurveSegment 6, 6, 2, 0, 2, 90 ‡ Upper curve
spath.Closed = True ‡ Close the curve
Set sh = ActiveLayer.CreateCurve(crv) ‡ Create curve shape
```

Для метода *AppendLineSegment* требуется только одна позиция узла для конца сегмента. Однако для метода *AppendCurveSegment* требуется одна декартова координата и две полярные координаты; декартова координата предназначена для конечного узла, а полярные координаты — для двух ручек управления, где первый параметр — это длина ручки, а второй параметр — угол поворота ручки управления. Кроме того, вы можете использовать метод *AppendCurveSegment2*, чтобы указать координаты обоих ручек управления.

Класс *Layer* имеет три дополнительные функции-члена для создания объектов кривых: *CreateLineSegment, CreateCurveSegment* и *CreateCurveSegment2*. Эти функции являются ярлыками для создания формы *Curve* вместе с ее первым сегментом на первом *SubPath*, и все это в одной функции.

Новые сегменты добавляются к последнему узлу вложенного пути *SubPath*. Существует необязательный параметр, который вызывает добавление сегмента к первому узлу вложенного пути.

Создание текстовых объектов

Текстовые объекты — это еще один тип объекта *Shape*. Однако работать с текстом сложнее, чем с другими фигурами.

В CorelDRAW текст обрабатывается как объект *TextRange*. Доступ к текстовым диапазонам можно получить с помощью любого из следующих свойств *Shape.Text*:

- Коллекция *Frames* — в этом сценарии каждый член объекта *TextFrame* по умолчанию является объектом *TextRange*, то есть текстом внутри фрейма.
- *Story* — в этом сценарии объект *TextRange* содержит весь текст во всех фреймах, связанных с текущим объектом *Shape* на всех страницах документа.

С текстом в объекте *TextRange* можно работать как с единым текстовым блоком, а его свойства (например, шрифт, размер и стиль) можно задавать одновременно. В качестве альтернативы объект *TextRange* имеет несколько свойств, представляющих собой коллекции небольших текстовых диапазонов: столбцы, абзацы, строки, слова и символы (*Columns, Paragraphs, Lines, Words* и *Characters*).

Вы можете создать два типа текста: фигурный текст (*artistic*) и текст абзаца (*paragraph*).

Чтобы создать новый фигурный текст, используйте функцию-член *CreateArtisticText* объекта *Layer*. Следующий код создает фигурный текст из слов **Hello World**, с левым концом строки на расстоянии 1 дюйм и базовой линией на расстоянии 4 дюйма от начала координат:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
Set sh = ActiveLayer.CreateArtisticText(1, 4, "Hello World")
```

Для этой функции существует множество необязательных параметров, например, вы можете установить такие атрибуты, как курсив, полужирный шрифт, размер и выравнивание (*italic, bold, size* и *alignment*)

Чтобы создать новый текст абзаца, используйте функцию-член *CreateParagraphText* объекта *Layer*. Абзацный текст отличается от художественного текста тем, что он располагается внутри прямоугольного контейнера, а не имеет необходимую ширину перед переводом строки. Первые четыре параметра функции — левый, верхний, правый, нижний:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
Set sh = ActiveLayer.CreateParagraphText(1, 4, 5, 2, "Hi There", _
                                         Alignment := cdrCenterAlignment)
```

Вы можете форматировать текст. Для этого сначала получите ссылку на объект *TextRange*, а затем примените к нему форматирование. *Frames, columns, paragraphs, lines* и *story* могут использоваться для получения ссылки на объект *TextRange*.

Следующий код преобразует первый абзац статьи в стиль заголовка, а второй и третий абзацы — в стиль основного текста:

```
Dim txt As TextRange
' Format the first paragraph
Set txt = ActiveShape.Text.Story.Paragraphs(1)
txt.ChangeCase cdrTextUpperCase
txt.Font = "Verdana"
txt.Size = 18
txt.Bold = True
' Format the second and third paragraphs
Set txt = ActiveShape.Text.Story.Paragraphs(2, 2)
txt.Font = "Times New Roman"
txt.Size = 12
txt.Style = cdrNormalFontStyle
```

Все параметры форматирования в диалоговом окне **Format Text** в CorelDRAW можно применять программно с помощью VBA. Для получения дополнительной информации см. объект *Text* в обозревателе объектов VBA.

Вы можете подогнать текст к контуру, используя функцию-член объекта *Text FitToPath*, которая просто прикрепляет текстовый объект к контуру фигуры таким образом, чтобы текст обтекал контур.

Следующий код создает новый текстовый объект и прикрепляет его к выбранной фигуре:

```
Dim sh As Shape, sPath As Shape
ActiveDocument.Unit = cdrInch
Set sPath = ActiveShape
Set sh = ActiveLayer.CreateArtisticText(1, 4, "Hello World")
sh.Text.FitToPath sPath
```

Фигуры текста *Paragraph* могут располагаться внутри замкнутых фигур, образуя непрямоугольные рамки. Это делается путем помещения объекта *Text* внутрь объекта *Shape* с помощью функции-члена *PlaceTextInside* объекта *Shape*. Учитывая, что в CorelDRAW выбран текст — *artistic* или *paragraph*, следующий код создает эллипс размером 5 на 2 дюйма и помещает в него выделенный текст:

```
Dim txt As Shape, sh As Shape
ActiveDocument.Unit = cdrInch
Set txt = ActiveShape
Set sh = ActiveLayer.CreateEllipse(0, 2, 5, 0)
sh.PlaceTextInside txt
```

Выбор фигур

Чтобы определить, выбрана ли фигура, вы можете проверить Boolean свойства *Selected* :

```
Dim sh As Shape
Set sh = ActivePage.Shapes(1)
If sh.Selected = False Then sh.CreateSelection
```

Вы можете добавить фигуру к выделению, просто установив для свойства *Selected* значение *True*; это выбирает форму, не отменяя выбор всех других фигур. Чтобы выбрать только одну фигуру без каких-либо других фигур, используйте метод *CreateSelection*, как в предыдущем коде.

Чтобы снять выделение со всех фигур, вызовите метод *ClearSelection* документа:

```
ActiveDocument.ClearSelection
```

Чтобы выбрать все фигуры на странице или слое, используйте следующий код:

```
ActivePage.Shapes.All.CreateSelection
```

Это вызывает *CreateSelection* объекта *ShapeRange*, который возвращается функцией-членом *All* коллекции *Shapes* на активной странице. Не будут выбраны только фигуры на заблокированных или скрытых слоях.

Чтобы получить выделение — другими словами, чтобы получить доступ к фигурам, выделенным в документе, — у вас есть выбор: получить ссылку на объект *Selection* документа или вместо этого получить копию свойства *SelectionRange* документа. Разница между этими двумя объектами типа выбора заключается в том, что *Selection* обновляется всякий раз, когда выбор изменяется в документе, в то время как *SelectionRange* является копией состояния выбора в данный момент и не обновляется.

Ссылка на объект ActiveSelection

Ярлык для *ActiveDocument.Selection* это *ActiveSelection*. Это возвращает объект *Shape*, который является ссылкой на объект *Selection* документа. Поскольку это ссылка, всякий раз, когда выделение в документе изменяется (либо пользователем, либо программно), фигуры, содержащиеся в этом объекте *Selection*, отражают изменение.

```
Dim sel As Shape
Set sel = ActiveDocument.Selection
```

ActiveSelection возвращает объект *Shape* подтипа *cdrSelectionShape*. Этот подтип *Shape* имеет коллекцию элементов под названием *Shapes*, которая представляет собой набор всех выбранных фигур в документе. Доступ к элементам коллекции *ActiveSelection.Shapes* можно получить обычным образом:

```
Dim sh As Shape, shs As Shapes
Set shs = ActiveSelection.Shapes
For Each sh In shs
  sh.Rotate 15 'Rotate each shape thru 15° counterclockwise
Next sh
```

Вы не можете напрямую запоминать и вызывать объекты, выбранные с помощью *ActiveSelection*. Чтобы скопировать ссылки на выбранные объекты, необходимо заполнить массив *Shape* или набор типов *Shapes* либо использовать *ShapeRange*.

Ссылка на объект ActiveSelectionRange

ActiveSelectionRange — это ярлык для *ActiveDocument.SelectionRange*. Это свойство документа типа *ShapeRange*:

```
Dim selRange As ShapeRange
Set selRange = ActiveDocument.SelectionRange
```

Объект *ShapeRange*, возвращаемый *ActiveSelectionRange*, содержит коллекцию ссылок на фигуры, которые были выбраны в момент чтения свойства. Поскольку эти ссылки относятся к самим фигурам (а не к выделению), при изменении выделения *ShapeRange* не обновляется.

Фигуры, на которые ссылается *ActiveSelectionRange*, хранятся в объекте *ShapeRange*. Эту коллекцию можно разобрать обычным способом:

```
Dim sh As Shape, shRange As ShapeRange
Set shRange = ActiveSelectionRange
For Each sh In shRange
  sh.Skew 15 ' Skew each shape thru 15° counterclockwise
Next sh
```

Поскольку *SelectionRange* представляет собой статическую копию ссылок на объекты, выбранные в данный момент, это идеальный способ запоминания и повторения выбора. Кроме того, фигуры можно добавлять и удалять из объекта *ShapeRange*, поэтому это идеальный способ управления выделением.

Сравнение объектов ActiveSelection и ActiveSelectionRange

Чтобы получить выделение, у вас есть выбор: вы можете получить ссылку на объект *Selection* документа (типа Shape), который обновляется при изменении выделения, или вы можете получить ссылку на новый объект *ShapeRange*, который подобен коллекции ссылок на объекты в выборке в данный момент времени и не обновляется.

Хотя *ActiveSelection*, как правило, проще в использовании, большое преимущество *ActiveSelectionRange* по сравнению с ним состоит в том, что даже при изменении выделения содержимое *ShapeRange* не меняется (в то время как коллекция *ActiveSelection Shapes* изменяется). Кроме того, *ShapeRange* можно дублировать, и он даже позволяет добавлять и удалять фигуры и применять преобразования, и все это без активного в данный момент времени *ShapeRange*.

Отдельные фигуры в *ShapeRange* можно выбрать, вызвав функцию-член *CreateSelection* диапазона:

```
Dim shRange As ShapeRange
Set shRange = ActiveSelectionRange
shRange.Remove 1
shRange.Remove 2
shRange.CreateSelection
```

Приведенный выше код создает *ShapeRange* из выделения, а затем удаляет первую и вторую — фактически третью фигуру исходного диапазона, потому что она становится второй фигурой, когда вы удаляете первую — фигуры из диапазона; затем он создает новый выбор из тех фигур, которые остались в диапазоне (то есть, все фигуры исходного выбора за вычетом двух фигур, которые были удалены из коллекции).

Если вы хотите добавить *ShapeRange* к текущему выбору, а не заменить его, используйте функцию-член *AddToSelection* для *ShapeRange*.

Разбор выбранных фигур

Анализ выделенных фигур выполняется одинаково как для выделений, так и для диапазонов выделений.

Вот пример кода для выбора:

```
Dim shs As Shapes, sh As Shape
Set shs = ActiveSelection.Shapes
For Each sh In shs
  ' Do something with the shape, sh
Next sh
```

А вот пример кода для диапазонов выбора:

```
Dim sRange As ShapeRange, sh As Shape
Set sRange = ActiveSelectionRange
For Each sh In sRange
  ' Do something with the shape, sh
Next sh
```

Однако диапазоны выбора имеют то преимущество, что даже если выбор впоследствии изменяется, диапазон не обновляется, и поэтому «память» выбора не теряется. Однако получение ссылки на *ActiveSelection* не создает копию ссылок на фигуры в выделении, а создает ссылку на встроенный объект *ActiveSelection.* Это означает, что при изменении выбора все ссылки на *ActiveSelection* также обновляются, и поэтому «память» выбора теряется.

Упорядочивание выбранных фигур в коллекции

Порядок объектов *Shape* в коллекциях *ActiveSelection.Shapes* и *ActiveSelectionRange* является обратным порядку, в котором фигуры были выбраны пользователем. Следовательно, первая фигура в обеих коллекциях — это последняя фигура, выбранная пользователем, а последняя фигура в обеих коллекциях — это первая фигура, выбранная пользователем. Это свойство этих коллекций полезно для макросов, которым необходимо различать, какая фигура была выбрана первой или последней.

Определение типа фигуры

Каждый объект *Shape* имеет свойство *Type*, которое возвращает подтип фигуры, то есть тип фигуры (прямоугольник, эллипс, кривая, текст, группа и т. д.). Это свойство доступно только для чтения. Поскольку некоторые типы фигур имеют функции-члены и свойства, которых нет у других, часто необходимо проверить тип фигуры перед вызовом метода, который не обязательно к ней применим.

Следующий пример кода проверяет фигуру, чтобы определить, является ли она текстом. Затем он проверяет, является ли это художественным или абзацным текстом; если он определяет, что это художественный текст, он поворачивает его на 10 градусов:

```
Dim sh As Shape
Set sh = ActiveShape
If sh.Type = cdrTextShape Then
  If sh.Text.IsArtisticText = True Then
    sh.Rotate 10
  End If
End If
```

Изменение свойств фигуры

Вы можете изменить свойства фигур, которые вы создаете, с помощью VBA.

Размеры фигуры

Чтобы получить ширину или высоту фигуры, проверьте ее свойства *SizeWidth* или *SizeHeight*:

```
Dim width As Double, height As Double
ActiveDocument.Unit = cdrMillimeter
width = ActiveShape.SizeWidth
height = ActiveShape.SizeHeight
```

Возвращаемое значение **Double** зависит от настройки *ActiveDocument.Unit*, поэтому, если вам нужен размер в определенных единицах, начните с его установки. То же самое относится ко всем свойствам и методам размера в объектной модели.

Чтобы получить ширину и высоту одной командой, используйте функцию-член *GetSize:*

```
Dim width As Double, height As Double
ActiveDocument.Unit = cdrMillimeter
ActiveShape.GetSize width, height
```

Чтобы задать размер фигуры, задайте для ее свойств *SizeWidth* или *SizeHeight* новые значения. Следующий код задает размер активной формы 50 на 70 миллиметров:

```
ActiveDocument.Unit = cdrMillimeter
ActiveShape.SizeWidth = 50
ActiveShape.SizeHeight = 70
```

В качестве альтернативы вы можете использовать метод *SetSize*, который устанавливает оба параметра за один вызов:

```
ActiveDocument.Unit = cdrMillimeter
ActiveShape.SetSize 50, 70
```

Существует дополнительный метод установки размера фигуры: *SetSizeEx*. Этот метод принимает центральную точку в дополнение к ширине и высоте; изменение размера выполняется относительно этой точки, а не относительно центральной точки фигуры. Например, следующий код изменяет размер выделения с 10 на 8 дюймов относительно точки (6, 5) в документе:

```
ActiveDocument.Unit = cdrInch
ActiveSelection.SetSizeEx 6, 5, 10, 8
```

Объект *ActiveSelection* относится к объекту *Shape*, который содержит все выбранные фигуры в документе.

Весь предыдущий код относится к размеру объекта с точки зрения его векторных кривых и игнорирует тот факт, что любая фигура с контуром может фактически выходить за пределы прямоугольника, содержащего кривые. Существует один метод, который можно использовать для получения размера ограничивающей рамки фигуры с возможностью включения ширины любых контуров. Это метод *GetBoundingBox*:

```
Dim width As Double, height As Double
Dim posX As Double, posY As Double
ActiveDocument.Unit = cdrInch
ActiveDocument.ReferencePoint = cdrBottomLeft
ActiveShape.GetBoundingBox posX, posY, width, height, True
```

Основное использование метода — получение ограничивающей рамки фигуры, включая положение ее нижнего левого угла. Последний параметр — это логическое значение, указывающее, следует ли возвращать ограничивающую рамку фигуры, включая ее контур (True) или исключая ее (False). Метод установки ограничивающей рамки фигуры не включает в себя необходимый для этого параметр по отношению к ее ограничивающей рамке, хотя при использовании *GetBoundingBox* дважды (один раз с включением контура и второй раз без контура) можно вычислить размер и положение ограничительной рамки вектора, не включая контур:

```
Public Sub SetBoundingBoxEx(X As Double, Y As Double, _
                            Width As Double, Height As Double)
  Dim sh As Shape
  Dim nowX As Double, nowY As Double
```

```
    Dim nowWidth As Double, nowHeight As Double
    Dim nowXol As Double, nowYol As Double
    Dim nowWidthol As Double, nowHeightol As Double
    Dim newX As Double, newY As Double
    Dim newWidth As Double, newHeight As Double
    Dim ratioWidth As Double, ratioHeight As Double
    Set sh = ActiveSelection
    sh.GetBoundingBox nowX, nowY, nowWidth, nowHeight, False
    sh.GetBoundingBox nowXol, nowYol, nowWidthol, nowHeightol, True
    ratioWidth = Width / nowWidthol
    ratioHeight = Height / nowHeightol
    newWidth = nowWidth * ratioWidth
    newHeight = nowHeight * ratioHeight
    newX = X + (nowX - nowXol)
    newY = Y + (nowY - nowYol)
    sh.SetBoundingBox newX, newY, newWidth, newHeight, False, cdrBottomLeft
End Sub
```

Растягивание и масштабирование фигур

Формы могут быть растянуты или масштабированы пропорционально. Объект *Shape* имеет две функции-члена — *Stretch* и *StretchEx*, которые выполняют эту операцию. Обе функции принимают десятичное значение для горизонтального и вертикального растяжения, где 1 равно 100% (или без изменений); вы не можете использовать ноль, поэтому вместо этого вы должны использовать очень маленькое значение. Следующий код растягивает выделение до половины текущей высоты и удвоенной ширины примерно до середины нижнего края ограничивающего прямоугольника:

```
ActiveDocument.ReferencePoint = cdrBottomMiddle
ActiveSelection.Stretch 2, 0.5
```

Растяжение также можно выполнить в любой точке страницы с помощью функции-члена *StretchEx*. Следующий код выполняет то же растяжение, что и выше, но относительно точки (4, 5) на странице в дюймах:

```
ActiveDocument.Unit = cdrInch
ActiveSelection.StretchEx 4, 5, 2, 0.5
```

Обе вышеупомянутые функции имеют необязательный логический параметр, который, если он равен True, растягивает текстовые символы абзаца на заданную величину; когда оно равно False, растягивается только ограничительная рамка текста, а текст перестраивается внутри рамки.

Позиционирование фигур

Положение объекта *Shape* можно определить с помощью свойств *PositionX* и *PositionY*, а также с помощью методов *GetPosition* и *GetBoundingBox*. Положение фигуры можно указать, установив свойства *PositionX* и *PositionY* или используя методы *SetPosition, SetSizeEx* и *SetBoundingBox*.

Следующий код получает положение фигуры *ActiveSelection* относительно текущего свойства *ReferencePoint ActiveDocument*, которое код явно устанавливает в нижний левый угол:

```
Dim posX As Double, posY As Double
ActiveDocument.ReferencePoint = cdrBottomLeft
ActiveSelection.GetPosition posX, posY
```

Приведенный выше код возвращает положение контрольной точки выделения без учета ширины контуров любой из выбранных фигур. Чтобы учесть ширину контуров, используйте функцию *GetBoundingBox*.

Следующий код устанавливает положение нижнего правого угла каждой выбранной фигуры в активном документе на (3, 2) в дюймах:

```
Dim sh As Shape
ActiveDocument.Unit = cdrInch
ActiveDocument.ReferencePoint = cdrBottomRight
For Each sh In ActiveSelection.Shapes
  sh.SetPosition 3, 2
Next sh
```

Вращение фигур

Фигуры можно поворачивать с помощью функций-членов *Rotate* и *RotateEx* объекта *Shape*.

Функция-член *Rotate* просто поворачивает фигуру на заданный угол (в градусах) вокруг центра вращения фигуры. Следующий код поворачивает выделение на 30 градусов относительно его центра вращения:

```
ActiveSelection.Rotate 30
```

Положительные углы поворота всегда равны градусам против часовой стрелки.

Чтобы найти центр вращения фигуры, получите значения свойств *RotationCenterX* и *RotationCenterY*. Изменение значений этих свойств перемещает центр вращения для следующего вызова этой функции.

Функция *RotateEx* принимает дополнительные параметры, определяющие центр вращения. Это более быстрый и простой метод, чем установка каждого из *RotationCenterX* и *RotationCenterY* с последующим вызовом *Rotate*. Следующий код поворачивает каждую из выбранных фигур на 15 градусов по часовой стрелке вокруг нижнего правого угла каждой фигуры:

```
Dim sh As Shape
ActiveDocument.ReferencePoint = cdrBottomRight
For Each sh In ActiveSelection.Shapes
  sh.RotateEx -15, sh.PositionX, sh.PositionY
Next sh
```

Перекос фигур

Фигуры можно деформировать с помощью функций-членов *Skew* и *SkewEx* объекта *Shape*. Эти две функции аналогичны функциям-членам *Rotate* и *RotateEx*, за исключением того, что они принимают два параметра угла: первый — горизонтальная деформация (положительные значения смещают верхний край влево, а нижний — вправо), а второй — вертикальная деформация (положительные значения смещают правый край вверх, а левый вниз). Горизонтальное смещение применяется перед вертикальным смещением..

Следующий код смещает выделение на 30 градусов по горизонтали и на 15 градусов по вертикали:

```
ActiveSelection.Skew 30, 15
```

Перекосы углов, близких или превышающих 90°, не допускаются.

Раскрашивание фигур

Цвет можно применять как к заливке, так и к контуру объекта, а также к фону страницы, линзам и градиентной заливке.

Вы можете скопировать один цвет в другой с помощью метода *CopyAssign*:

```
Dim sh As Shape
Set sh = ActiveShape
sh.Outline.Color.CopyAssign sh.Fill.UniformColor
```

Вы можете получить цветовую модель объекта, получив его свойство *Type*:

```
Dim colType As cdrColorType
colType = ActiveShape.Outline.Color.Type
```

Когда у вас есть тип цвета (обычно один из *cdrColorRGB, cdrColorCMYK* или *cdrColorGray*, но не ограничиваясь ими), вы можете получить следующие компоненты цвета:

- для цвета `CMYK` — `CMYKCyan`, `CMYKYellow`, `CMYKMagenta` и `CMYKBlack`
- для цвета `RGB` — `RGBRed`, `RGBGreen` и `RGBBlue`
- для оттенков серого — `Gray`

Чтобы преобразовать цвет объекта *Color* в другую цветовую модель, используйте функции-члены *ConvertToCMYK, ConvertToRGB, ConvertToGray* и т. д. Цвет преобразуется с помощью параметров управления цветом CorelDRAW. Например, следующий код преобразует заливку в RGB:

```
ActiveShape.Fill.UniformColor.ConvertToRGB
```

Чтобы задать новый цвет, используйте такие методы, как *CMYKAssign, RGBAssign* и *GrayAssign*. Количество параметров, которые принимают эти методы, зависит от того, сколько цветовых компонентов требуется для этой цветовой модели. Например, следующий код назначает темно-синий цвет RGB контуру активной фигуры:

```
ActiveShape.Outline.Color.RGBAssign 0, 0, 102
```

Диапазон значений каждого компонента зависит от цветовой модели.

Некоторые функции объектной модели CorelDRAW принимают в качестве параметра объект *Color*. Чтобы создать новый объект *Color*, используйте ключевое слово VBA *New*, как в следующем примере:

```
Dim col As New Color
```

```
col.RGBAssign 0, 255, 102
ActiveShape.Outline.Color.CopyAssign col
```

Цвет “none” не существует. Чтобы установить цвет “none” для контура или заливки, вы должны фактически установить тип контура или заливки на “none”.

Обведение фигур

Каждый объект Shape имеет свойство *Outline*, которое ссылается на объект *Outline*. Контурные объекты имеют такие свойства, как *Type, Width, Color* и *Style.*

Свойство *Type* определяет, имеет ли фигура контур: у фигуры есть контур, если для него установлено значение *cdrOutline*, но нет контура, если для него установлено значение *cdrNoOutline*. Установка для этого свойства значения *cdrOutline* для фигуры, не имеющей контура, дает фигуре контур документа по умолчанию, а задание для этого свойства значения *cdrNoOutline* удаляет контур из фигуры. Это то же самое, что установить *Width* равным нулю.

Свойство *Width* задает ширину контура в единицах измерения документа; например, чтобы установить ширину в пунктах, сначала установите для свойства *Unit* документа значение *cdrPoint*. В следующем примере кода для контура выбранных фигур задано значение 1 миллиметр:

```
ActiveDocument.Unit = cdrMillimeter
ActiveSelection.Outline.Width = 1
```

Любые фигуры, свойство *Type* которых имеет значение *cdrNoOutline*, изменяют это свойство на *cdrOutline* при установке цвета или ширины контура.

Свойство *Color* — это объект цвета, определяющий цвет контура. Установка цвета контура автоматически устанавливает для свойства *Type* контура значение *cdrOutline* и придает контуру ширину по умолчанию:

```
ActiveSelection.Outline.Color.GrayAssign 0 ' Set to black
```

Свойство *Style* элемента *Outline* задает пунктирные свойства контура. Он имеет четыре свойства, из которых можно установить только три:

- *DashCount* — устанавливает количество штрихов в стиле
- *DashLength* — массив значений, указывающий длину каждого штриха, при этом размер штриха определяется значением *DashCount*.
- *GapLength* — массив значений, указывающий длину каждого промежутка после каждого штриха, при этом размер штриха определяется значением *DashCount*.
- *Index* — доступное только для чтения свойство, которое дает индекс стиля структуры в коллекции *OutlineStyles* документа, которая представлена пользователю в диалоговом окне **Outline**.

Значения в массивах *DashLength* и *DashGap* отображаются как кратные ширине линии; следовательно, если *DashLength(1)* равно 5, а ширина линии равна 0,2 дюйма, длина штриха равна 1 дюйму; если ширина линии изменена на 0,1 дюйма, длина штриха становится равной 0,5 дюйма.

Чтобы использовать один из контурных стилей приложения, вы должны сослаться на стиль из коллекции *OutlineStyles*, указав индекс стиля, который вы хотите использовать. Проблема заключается в том, что установка CorelDRAW для каждого пользователя может иметь различный набор стилей контуров (в первую очередь, если пользователь изменил их), поэтому не гарантируется, что заданный проиндексированный стиль будет одинаковым для всех пользователей. Чтобы использовать один из стилей, назначьте его контуру:

```
ActiveShape.Outline.Style = Application.OutlineStyles(3)
```

OutlineStyles.Item(0) всегда представляет собой сплошную линию; чтобы сделать контур сплошным, установите его равным этому стилю.

Контурные объекты имеют много других свойств, в том числе следующие:

- *StartArrow, EndArrow* — устанавливает наконечник стрелки для использования на каждом конце незамкнутой кривой.
- *LineCaps, LineJoin* — задает тип окончания линии (встык, круглый или квадратный) и соединения (скос, под углом или круглый)
- *NibAngle, NibStretch* — задает форму пера, используемого для рисования линии.
- *BehindFill, ScaleWithShape*: рисует контур за заливкой и масштабирует контур по фигуре.

Контурные объекты также имеют два метода:

- *ConvertToObject* — преобразует контур в объект.
- *SetProperties* — единый метод, который можно использовать для установки большинства свойств контура за один вызов, метод, который намного эффективнее и быстрее, чем указание каждого свойства по отдельности при одновременной установке свойств сотен или тысяч контуров.

Заливка фигур

В CorelDRAW существует много типов заливок, в том числе однородные, фонтанные, PostScript-заливки, заливки узором и текстурные заливки, и каждый тип должен обрабатываться по-разному.

Здесь обсуждаются только однородные и фонтанные заливки.

Свойство *Type* объекта *Fill*, доступное только для чтения, указывает тип заливки или ее отсутствие. Следующий код получает тип заливки:

```
Dim fillType As cdrFillType
fillType = ActiveShape.Fill.Type
```

Тип заливки не может быть установлен с помощью этого свойства; он устанавливается при создании и применении заливки.

Чтобы удалить заливку любого типа, используйте функцию-член *ApplyNoFill* объекта *Fill*.

Однородные заливки состоят из одного сплошного цвета. Этот цвет представлен свойством заливки *UniformColor*, которое является объектом *Color*.

Чтобы применить однородную заливку к фигуре, используйте функцию-член *ApplyUniformFill* свойства *Fill* фигуры:

```
ActiveShape.Fill.ApplyUniformFill CreateRGBColor(255, 0, 0)
```

Если вы не копируете цвет из существующего объекта *Color*, вам необходимо сначала создать новый объект *Color*, а затем применить цвет к свойству *Fill* фигуры.

Чтобы изменить цвет однородной заливки, измените свойство *UniformColor* объекта *Fill*:

```
ActiveShape.Fill.UniformColor.RGBAssign 0, 0, 102
```

Однородные заливки имеют свойство *Type cdrUniformFill*.

Фонтанные заливки определяются свойством *Fountain*, которое является объектом *FountainFill*. Объекты *FountainFill* имеют множество свойств, включая тип, угол и тип перехода. Однако наиболее важным свойством является набор цветов, составляющих фонтанную заливку.

Чтобы создать новую фонтанную заливку, используйте функцию-член *ApplyFountainFill* объекта *Fill*. Это создает простой двухцветный фонтан с основными свойствами. Следующий код создает простую линейную фонтанную заливку от красного до желтого цвета под углом 30 градусов к горизонтали:

```
Dim startCol As New Color, endCol As New Color
startCol.RGBAssign 255, 0, 0
endCol.RGBAssign 255, 255, 0
ActiveShape.Fill.ApplyFountainFill startCol, endCol, cdrLinearFountainFill, 30
```

Все параметры функции-члена *ApplyFountainFill* являются необязательными, и не все они использовались в предыдущем примере кода. Можно установить среднюю точку, количество шагов и тип смешения цветов в одном и том же вызове функции.

После создания фонтанной заливки можно добавить в смесь дополнительные цвета, добавив новые элементы *FountainColor* в коллекцию *Colors*, которая является свойством *Fill.Fountain*. Например, следующий код добавляет зеленый цвет в коллекцию *Colors* на расстоянии примерно одной трети (33%) от красного:

```
Dim fFill As FountainFill
Set fFill = ActiveShape.Fill.Fountain
fFill.Colors.Add CreateRGBColor(0, 102, 0), 33
```

Отдельные цвета фонтана можно перемещать с помощью функции-члена *FountainColor.Move.* Следующий код перемещает зеленый цвет из предыдущего кода в положение, которое составляет 60% от красного к желтому:

```
ActiveShape.Fill.Fountain.Colors(1).Move 60
```

Позиции цвета представляют собой целые значения в процентах, где 0 % — это позиция начального цвета, а 100 % — позиция конечного цвета.

Количество цветов, сообщаемое свойством Count коллекции Colors, — это количество цветов между начальным и конечным цветами. Таким образом, для фонтанной заливки, созданной выше, значение свойства *Count* равно 1. Первый цвет в коллекции *Colors* имеет номер 0 и является начальным цветом; его нельзя переместить, но можно изменить его цвет. Последний цвет в коллекции *Colors* — элемент (*Count* + 1) и является конечным цветом; его также нельзя переместить, но можно изменить его цвет. Следующий код изменяет цвет конечного цвета с желтого на синий:

```
Dim cols As FountainColors
Set cols = ActiveShape.Fill.Fountain.Colors
cols(cols.Count + 1).Color.RGBAssign 0, 0, 102
```

Существуют дополнительные свойства (не описанные здесь), которые определяют заполнение краев, положение центра (для конических, радиальных и прямоугольных фонтанов) и количество шагов, используемых для рисования заливки.

Фонтанные заливки имеют значение свойства *Type cdrFountainFill.*

Дублирование фигур

Чтобы дублировать фигуры, вы можете использовать функцию-член *Duplicate* объекта *Shape:*

```
ActiveSelection.Duplicate
```

Эта функция принимает два значения параметра, которые задают смещение дубликата от оригинала. Следующий код размещает дубликат на два дюйма правее и на один дюйм выше оригинала:

```
ActiveDocument.Unit = cdrInch
ActiveSelection.Duplicate 2, 1
```

Применение эффектов к фигурам

Эффекты можно применять непосредственно к фигурам с помощью соответствующей функции-члена объекта *Shape*.

Создание перехода

Функция-член *CreateBlend* объекта *Shape* создает переход между текущей фигурой и фигурой в списке параметров. Следующий код создает 10-ступенчатый переход:

```
Dim sh As Shapes, eff As Effect
Set sh = ActiveSelection.Shapes
Set eff = sh(1).CreateBlend(sh(2), 10)
```

Количество фигур в переходе равно 12 — начальная и конечная формы плюс десять созданных шагов.

Есть несколько необязательных параметров (не показаны) для метода *CreateBlend,* которые управляют ускорением перехода, а также задают путь, по которому создается переход.

Объект *Effect*, возвращаемый предыдущим кодом, является ссылкой на переход. Установив свойства *Blend* объекта *Effect*, можно точно настроить переход. Сам объект *Effect* имеет функцию-член для разделения эффекта смешивания, а также *Clear* для его удаления.

Создание контуров

Контуры можно создавать с помощью функции-члена *CreateContour* объекта *Shape*.
Следующий код создает трехступенчатый контур с шагом 5 миллиметров:

```
Dim eff As Effect
ActiveDocument.Unit = cdrMillimeter
Set eff = ActiveShape.CreateContour(cdrContourOutside, 5, 3)
```

Существует несколько необязательных параметров (не показаны) для метода *CreateContour*, которые управляют цветами и ускорением форм контура.

Создание других эффектов

Объект *Shape* имеет несколько других функций для создания эффектов: *CreateDropShadow, CreateEnvelope, CreateExtrude, CreateLens, CreatePerspective, CreatePushPullDistortion, CreateTwisterDistortion* и *CreateZipperDistortion*. Для получения дополнительной информации о них обратитесь к справочной системе VBA из обозревателя объектов.

Глоссарий

array *(массив)*
Набор последовательно индексируемых элементов одного и того же типа данных. По умолчанию индексы массива отсчитываются от нуля.

automation *(автоматизация)*

Процесс записи или сценарий макроса

class *(класс)*

Определение (то есть описание) объекта

class module *(модуль класса)*

Модуль, содержащий определение *класса*, включая определения его *свойств* и *методов*.

collection *(коллекция)*
Набор *объектов*

constant (*константа)*

Именованный элемент, сохраняющий постоянное значение во время выполнения макроса.

enumerated type (enumeration) *перечисляемый тип (перечисление)*

Тип данных, в котором перечислены все возможные значения переменных, которые его используют.

event *(событие)*

Действие, распознаваемое формой или элементом управления

event handler *(обработчик события)*

Подпрограмма, запрограммированная так, чтобы приложение реагировало на определенное событие.

function *(функция)*
Процедура, которая выполняет заданную задачу в макросе и может использоваться для возврата значения. Функциональная процедура начинается с оператора Function и заканчивается оператором End Function. В VBA функции не нужно объявлять ни до их использования, ни до их определения.

global value *(глобальное значение)*

Значение, которое применяется к данному проекту в целом

macro *(макрос)*

Сценарий или записанный набор действий, которые можно неоднократно вызывать в приложении.

method *(метод)*

Операция, которую объект может выполнить над собой

modal dialog box *(модальное диалоговое окно)*

Диалоговое окно, в котором необходимо действовать, прежде чем пользователь сможет возобновить выполнение макроса. Приложение блокируется до тех пор, пока диалоговое окно не будет закрыто путем его отправки или отмены. Встроенные диалоговые окна, которыми можно управлять с помощью VBA, почти всегда являются модальными.

modeless dialog box *(немодальное диалоговое окно)*

Диалоговое окно, которое не блокирует приложение, поэтому его можно оставить открытым, пока пользователь продолжает работать в приложении. Таким образом, немодальные диалоговые окна ведут себя как докеры.

module *(модуль)*

Набор объявлений, за которыми следуют процедуры

object *(объект)*

Экземпляр класса. Объект может быть родителем для дочерних объектов.

object model *(объектная модель)*

Высокоуровневая структура отношений между родительскими и дочерними объектами. Без объектной модели VBA не может получить доступ к объектам в документе, а также запросить или изменить документы приложения.

passing by reference *(передача по ссылке)*

Действие по передаче аргумента подпрограмме или функции с использованием ссылки на оригинал. По умолчанию параметры функции и подпрограммы передаются по ссылке, но если вы хотите явно аннотировать код, чтобы указать, что аргумент передается по ссылке, вы можете добавить к аргументу префикс ByRef.

passing by value *(передача по значению)*

Акт передачи аргумента функции или подпрограмме с использованием копии оригинала. Чтобы указать, что вы хотите передать аргумент по значению, добавьте к аргументу префикс ByVal.

property *(свойство)*

Характеристика класса. Свойства, фиксированные конструкцией класса, называются «только для чтения».

scope *(область видимости)*

Видимость типа данных, процедуры или объекта

shortcut object *(ярлык объекта)*

Синтаксическая замена длинной версии объекта.

subroutine (sub) *- подпрограмма*

Процедура, которая выполняет заданную задачу в макросе, но не может использоваться для возврата значения. Процедура подпрограммы начинается с оператора Sub и заканчивается оператором End Sub. В VBA подпрограммы не нужно объявлять ни до их использования, ни до их определения.

variable *(переменная)*

Элемент, который может быть создан (или «объявлен») для целей хранения данных. Встроенные типы данных: Byte, Boolean, Integer, Long, Single, Double, String, Variant и несколько других менее используемых типов, включая Date, Decimal и Object. Если переменная не объявлена до ее использования, компилятор интерпретирует ее как Variant.

Visual Basic for Applications (VBA) *- Visual Basic для приложений (VBA)*

Встроенный язык программирования, позволяющий автоматизировать повторяющиеся функции и создавать интеллектуальные решения в CorelDRAW и Corel PHOTO-PAINT.

Index

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

Z

CorelDRAW® Graphics Suite X3 Programming Guide for VBA

Product specifications, pricing, packaging, technical support and information ("specifications") refer to the retail English version only. The specifications for all other versions (including other language versions) may vary.

011115