Глава 55

Надежный код: отладка и перехват ошибок

В этой главе...

- Нужно быть реалистом (или немного о Законах Мерфи)
- Отладка кода VBA
- Использования контрольных точек для нахождения ошибок
- Использование окон Immediate, Locals и Add Watch
- Перехват ошибок с использованием специализированных объектов баз данных

Написать код на VBA достаточно просто. Запустить его и получить нужные результаты гораздо сложнее. Нахождение и исправление ошибок занимает основную часть времени кодирования, и в данной главе рассказывается, как это сделать в VBA. Также описывается процесс перехвата ошибок, позволяющий программе предпринимать конкретные действия, когда что-то пошло не так.

Что может работать неправильно, то будет работать неправильно

Программу, написанную на VBA или любом другом языке, преследуют три типа проблем.

- **Ошибки синтаксиса.** Синтаксис и другие ошибки, которые не позволяют запустить программу.
- **Ошибки логики.** Проколы при разработке программы, в результате чего она не выполняет того, что желает пользователь, и делает то, что не требуется — программа работает, но неправильно.
- **Ошибки выполнения.** Останавливают программу во время работы. Ошибки выполнения могут быть результатом серьезных логических ошибок или неучтенных условий.

Из этих трех типов ошибки синтаксиса, определенно, наиболее простые для обнаружения и исправления. Они описаны вкратце, а большая часть главы посвящена двум другим типам, — настоящим ошибкам.

Исправление синтаксических ошибок

Когда пользователь допускает ошибки первого типа, синтаксического, редактор Visual Basic поможет найти их до запуска. Как только пользователь вводит что-то, что редактор не может понять, строка выделяется красным цветом. Затем, если установлен флажок Auto Syntax Check (Автоматическая проверка синтаксиса) во вкладке Editor (Редактор)

диалогового окна Tools⇨Options (Сервис⇨Параметры), сразу по переходу курсора на другую строку появится сообщение описывающее проблему. Например, если ввести **If x=3**, но забыть **Then**, в сообщении будет написано `"Compile error: Expected: Then or Goto"`. (VBA не отлавливает некоторые ошибки до запуска программы.)

Когда пользователь знает об ошибке исправить ее достаточно легко. Если есть какие-либо сомнения насчет того, как правильно воплотить задуманный код, проконсультируйтесь с нужным разделом этой книги или разделом справки по VBA.

Отладка для программистов на VBA

Когда исправлены все ошибки, что появлялись при компиляции, появляется чувство гордости и облегчения. Однако, когда программа сообщает что `2 + 2 = 22` или весь текст документа приобретает бледно-зеленую окраску, наступает прозрение: в код вкралась ошибка.

Это примеры логических ошибок. Программа запускается; на самом деле она четко следует инструкциям, заложенным программистом. Проблема заключается в том, написанный код заставляет программу делать не то, что хотелось бы. В этом случае нужно найти, какой оператор приводит к неверному результату. Такой процесс называется *отладкой* программы. Инструменты и способы отладки рассматриваются в этом разделе.

Конечно, способы отладки, используемые для устранения логических ошибок, применяются и для исправления ошибок выполнения, которые связаны с ошибками логики. Но поскольку программа не может предусмотреть все возможные обстоятельства, следует встраивать в программы код для обработки ошибок выполнения.

Проверка, проверка, проверка

Очень удобно, когда ошибки объявляют о своем существовании, выводя заведомо ложные результаты или изменяя цвет текста. К сожалению, многие логические ошибки намного сложнее. Чтобы убедиться, что программа всегда работает нормально, приходится ее тестировать при различных условиях.

Если предполагается, что программа будет работать с различными документами, то и тестировать ее нужно с различными документами. Запустите программу, когда два или три документа уже открыты в используемом приложении VBA и когда не открыт ни один документ. Посмотрите, что произойдет, если запускать программу в различных состояниях окна (свернутом, полном или восстановленном). Запустите программу, когда в окне документа выбраны различные элементы или группы.

Если программа требует от пользователя ввести что-либо в окне ввода, проверьте, как программа воспринимает нормальные и неподходящие значения. Если предполагается, что должно быть введено целое число, нужно попробовать, как отреагирует программа на ввод числа с плавающей точкой, даты или строки. А если программа работает с датой или временем, проверьте, как она реагирует на различные даты, включая 29 февраля и различное время, включая полночь.

Если обнаружены ошибки, пора начинать отладку. Даже если ошибки не обнаружены, это еще не значит, что их нет, лучше предположить, что они есть.

Периодически во время написания кода проверяйте его. Создайте таблицы и сценарии для проверки модулей. Их можно применять для проверки работоспособности приложения.

Использование режима останова для отладки

Ключом к отладке программы является *режим останова*. В этом режиме программа выполняется, но может приостанавливать свое выполнение на определенном операторе кода. Поскольку программа все еще выполняется, неплохо бы проверить текущее значение переменных. Начиная с этого момента, можно использовать команду Step (Шаг) для выполнения одного оператора программы и отслеживания изменения значений переменных после каждого шага. В следующих разделах детально описывается работа с переменными при помощи команд Step.

На рис. 55.1 приведена процедура VBA в режиме останова. За исключением желтой подсветки и стрелки, обозначающей следующий оператор, редактор Visual Basic выглядит так же, как и при написании кода.

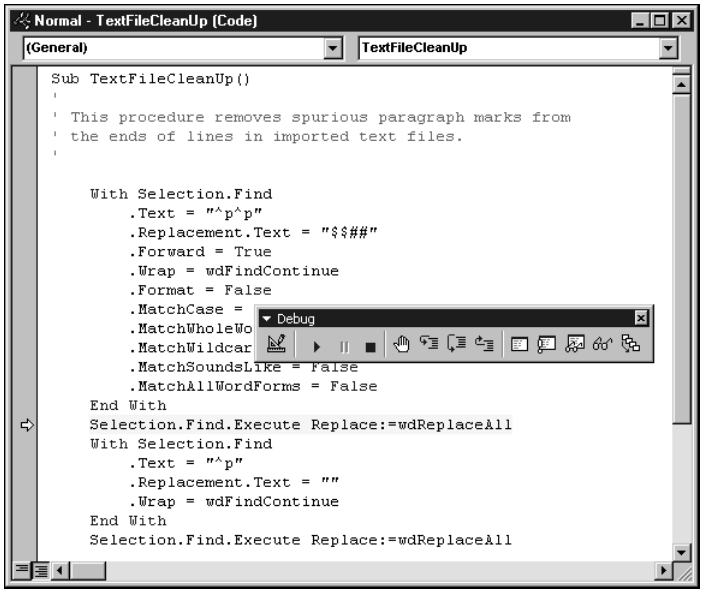

Рис. 55.1. Процедура VBA в режиме останова

На самом деле, в режиме останова можно редактировать программу, изменяя текущие и добавляя новые строки. Это не роскошь, а основное преимущество отладки, которое следует научиться использовать с пользой.

Включение режима останова

Можно использовать многие способы для включения режима останова, — компьютерный аналог слайдов. Далее приводится список этих способов.

- Начать выполнение программы в режиме останова, используя команду Step Into (Следующий шаг) (см. "Пошаговое выполнение кода").
- Установить контрольную точку (breakpoint) перед строкой кода. Когда программа достигнет оператора за контрольной точкой, она приостановит выполнение и войдет в режим останова.
- Разместить оператор `Stop` в коде. Когда программа будет запущена, она войдет в режим останова, готовая выполнить оператор, следующий за оператором `Stop`.

- Щелкнуть на кнопке Break (Останов), выбрать команду меню Run⇨Break (Выполнить⇨Останов) или нажать комбинацию клавиш <Ctrl+Break> во время выполнения. Эта же техника используется, чтобы вернуть контроль исполнения обратно программе. Не известно, в каком именно месте программа будет прервана, но, по крайней мере, можно увидеть, что в ней происходит.
- Создать флажок Break When True (Останов при значении Истина) или Break When Change (Останов при изменении). Программа войдет в режим останова, когда значение выражения будет `True` или изменится.

Еще одно обстоятельство, при котором программа переходит в режим останова — это возникновение ошибок во время исполнения. VBA отображает диалоговое окно, описывающее ошибку (см. рис. 55.2). Щелчок на кнопке End (Завершить), останавливает выполнение программы, а щелчок на кнопке Debug (Отладка) переводит ее в режим останова. Можно отследить, какие ошибки во время исполнения переводят программу в режим останова, воспользовавшись опциями вкладки General (Общие) диалогового окна Tools⇨Options (Сервис⇨Параметры).

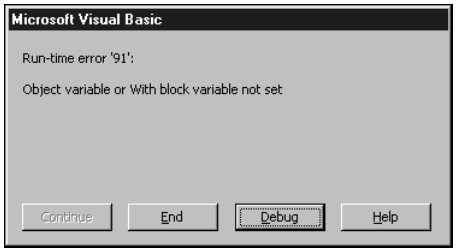

Рис. 55.2. Диалоговое окно ошибки во время исполнения

Установка контрольных точек

Когда имеется подозрение, что фрагмент кода содержит логическую ошибку, установите контрольную точку перед сомнительным оператором. На рис. 55.3 подсветка и большая точка на границе окна кода показывают, как редактор Visual Basic выделяет на экране контрольную точку.

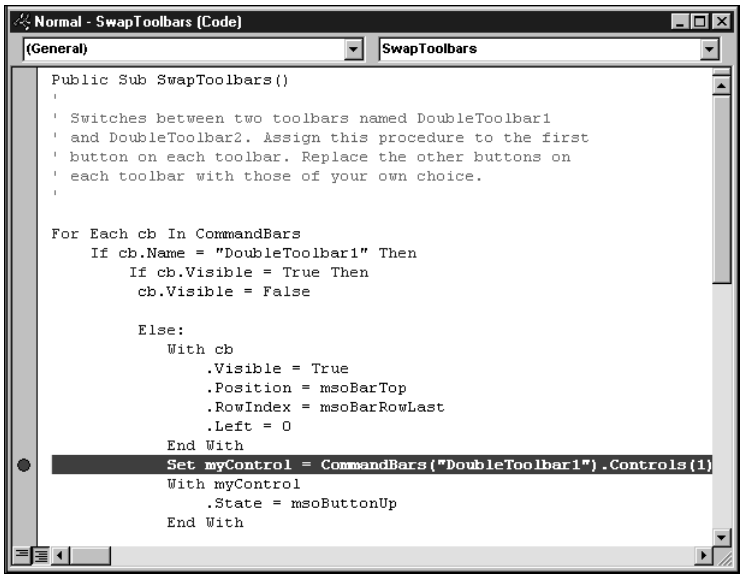

Рис. 55.3. Когда установлена контрольная точка, редактор Visual Basic подсветкой обозначает строку кода, на которой выполнение программы должно быть остановлено

С установленной контрольной точкой можно запустить программу до контрольной точки, пропуская код, который (будем надеяться) работает нормально. Когда VBA достигает оператора, перед которым расположена контрольная точка, программа приостанавливается. После активизации режима останова, можно проверить значения переменных и использовать команды Step (Шаг), чтобы отследить, как они изменяются по мере последовательного выполнения операторов.

Чтобы установить контрольную точку, щелкните в левой части окна кода напротив строки, следующей за нужной.

Контрольных точек можно установить столько, сколько нужно. Однако их нельзя устанавливать напротив комментариев и операторов, которые VBA не исполняет (вроде объявлений переменных).

Помните: VBA останавливает программу и переходит в режим останова при достижении контрольных точек. Другими словами, оператор, напротив которого расположена контрольная точка, не выполняется сразу, он будет выполнен только тогда, когда работа программы будет продолжена.

Удаление контрольных точек

Когда ошибка исправлена или пользователь отчаялся исправить ее в этот раз, контрольную точку следует удалить, что позволит VBA в следующий раз выполнить программу, как обычно. Для удаления точки применяются те же методы, что использовались для ее установки: щелкните в левой границе окна кода или нажмите клавишу <F9>.

Чтобы удалить все контрольные точки, выберите команду Debug⇨Clear All Breakpoints (Отладка⇨Снять все контрольные точки) или нажмите комбинацию клавиш <Ctrl+Shift+F9>.

Осторожно! После удаления всех контрольных точек команда Undo (Отменить) их не восстановит.

Как узнать в режиме останова, в каком месте остановлена программа

Когда программа запущена в режиме останова, редактор Visual Basic всегда подсвечивает оператор, который будет выполнен следующим. Чтобы пользователь точно не ошибся, за границей окна кода появляется стрелка, указывающая на следующий оператор. На рис. 55.4 показано, как выглядит это напоминание, хотя черно-белая картинка оставляет богатый простор для воображения.

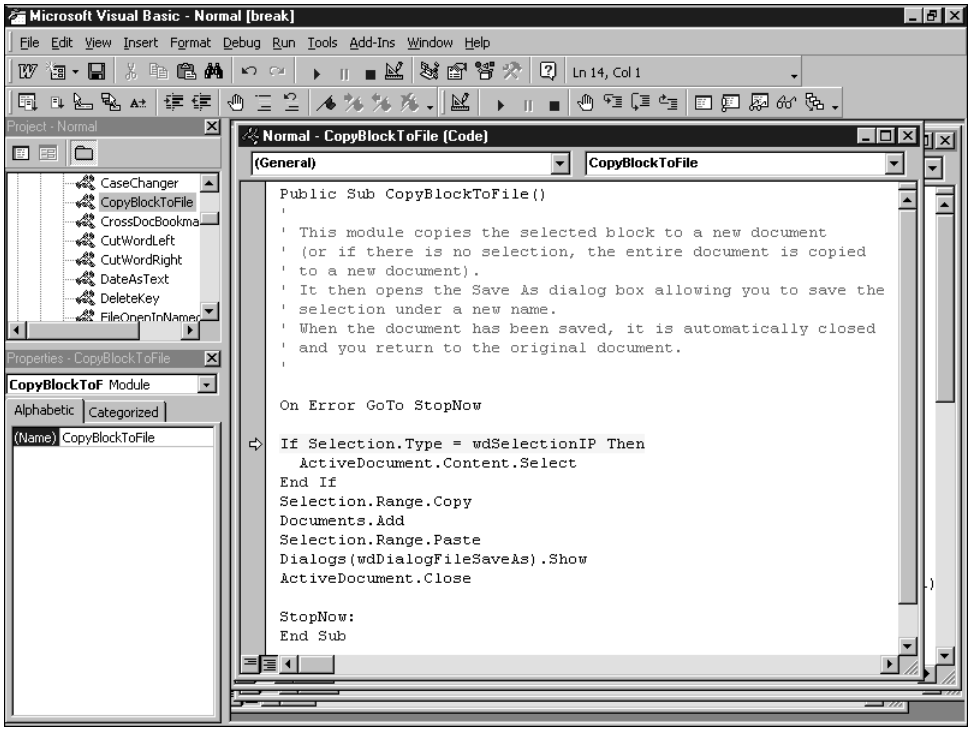

Рис. 55.4. Пользователь не пропустит следующий оператор, который должен быть выполнен, пока его видно на экране

Когда пользователь видит на экране следующий оператор, у него не возникает вопросов насчет подсветки. Что произойдет, если перейти в другую часть программы, используя полосу прокрутки, или перейти в окно кода другого модуля? Могут появиться проблемы с нахождением следующего оператора.

Вот в этот момент и может понадобиться команда Show Next Statement (Показать следующий оператор). При выборе команды Debug⇨Show Next Statement (Отладка⇨Показать следующий оператор) редактор Visual Basic переходит к нужному оператору. Есть еще одна команда, которая позволяет выбрать в качестве следующего другой оператор (см. раздел "Выбор иного следующего оператора" далее в этой главе).

Альтернатива контрольной точке — оператор Stop

Контрольные точки очень легко использовать, но у них есть один недостаток — они временны. При отладке сложной программы не всегда удается сразу верно расставить точки. А поскольку контрольные точки не хранятся вместе с кодом, при следующем запуске редактора Visual Basic придется их расставлять заново.

Оператор `Stop` является решением данной проблемы. Добавьте его в код там, где нужно, чтобы программа перешла в режим останова. Как и любой другой код, операторы `Stop` сохраняются в проекте. Например:

```
...
intDataFromMars = GetDataFromMars(1.5454)
Stop
MsgBox "Результат " & intDataromMars | Z
...
```

Как можно догадаться, оператор `Stop` используется для проверки результатов, отображаемых в окне сообщения. Находясь в программе, оператор `Stop` переводит ее в режим останова сразу после процедуры, присваивающей значение переменной. Таким образом, можно проверить значение обеих переменных, используемых в дальнейших вычислениях, и определить, какая из них выходит из диапазона допустимых значений.

Выход из режима останова

Когда программа находится в режиме останова, можно заставить VBA перейти к ее нормальному выполнению. Для этого подойдет любая команда запуска программы. В режиме останова элемент меню Run (Выполнить) и кнопка на панели управления выглядят, как обычно, но их названия изменятся на Continue (Продолжить). А можно использовать клавишу <F5>. Продолженная программа может быть заново переведена в режим останова, например, следующей контрольной точкой или другими событиями, активизирующими режим останова.

Чтобы совсем остановить выполнение программы, используйте команду Reset (Сброс). Существует кнопка Reset и команда Reset в меню Run, однако вызвать ее с помощью горячих клавиш не удастся по причине отсутствия таковых.

Пошаговое выполнение кода

Как и все хорошие отладчики, редактор Visual Basic позволяет выполнять за один шаг один оператор программы. Такой подход предоставляет фантастические возможности по перехвату любого места, где логические ошибки приводят к появлению критического запредельного значения.

Чтобы использовать технику *пошагового выполнения*, нужен способ отслеживания значений переменных, изменяемых программой. Редактор предоставляет для этой цели Auto Data Tips (Автоматические подсказки), однако окна Locals (Локальные значения) и Watch (Наблюдения) обладают большим потенциалом и позволяют при необходимости даже изменять значения. Эти возможности будут освещены немного дальше.

Теперь речь пойдет о трех командах Step, используемых в редакторе Visual Basic: Step Into, Step Over и Step Out. Все они находятся в меню Debug (Отладка) и в виде кнопок на панели инструментов Debug (Отладка), их можно вызвать и при помощи горячих клавиш.

Использование команды Step Into

Команда Step Into применяется при выполнении операторов программы в определенной последовательности. Каждый раз, когда используется эта команда, VBA выполняет следующий оператор программы и вновь возвращается в режим останова, позволяя просмотреть внесенные изменения. Наиболее простым способом вызова команды Step Into является нажатие клавиши <F8>.

Эта команда получила свое название в связи с тем, что она может входить в процедуры, вызываемые программой. Когда следующий оператор вызывает процедуру `Function` или `Sub`, команда Step Into открывает вызываемую процедуру в окне кода, где можно увидеть, что именно она делает. Этим данная команда отличается от Step Over.

Можно воспользоваться командой Step Into, даже когда режим останова выключен. Если необходимо пошагово выполнить программу с самого начала, просто дайте команде выполнить первый оператор и войти в режим останова. Применяемая таким образом команда Step Into запускает процедуру, содержащую точку вставки, всегда начиная с первой строки процедуры.

Использование команды Step Over и Step Out

Нажатие комбинации клавиш <Shift+F8> — наиболее быстрый способ выполнить команду Step Over (Пошаговое исполнение). Команда Step Over работает так же, как и Step Into, но со следующими отличиями:

- что наиболее важно, Step Over не переходит по отдельным операторам вызываемой процедуры;
- нельзя начать выполнение программы в режиме останова при помощи команды Step Over.

Когда следующий оператор вызывает другую процедуру, Step Over выполняет процедуру как единое целое, переходя к следующему оператору текущей процедуры. Это полезно для проверки работы процедуры в целом, не переходя к отдельным операторам, что сохраняет время при включении в подпрограмму уже отлаженной процедуры.

Команда Step Out является удобным дополнением к команде Step Into. Эффективнее всего вызывать команду Step Out комбинацией клавиш <Ctrl+Shift+F8>. Когда работа вызываемой процедуры проверена, ошибка исправлена, а также использована команда Step Into вместо Step Over, можно сэкономить время, не выполняя оставшихся операторов процедуры. Достаточно воспользоваться командой Step Out, чтобы быстро проскочить остаток процедуры. VBA перейдет к вызывавшей процедуре и выделит оператор, следующий за вызовом процедуры.

Внесение изменений во время отладки

Когда программа находится в режиме останова, это еще не значит, что она стала застывшим памятником. VBA достаточно сложен, чтобы допустить внесение определенных изменений. То есть, можно редактировать существующий код и изменять последовательность, в которой выполняются операторы.

Добавление и редактирование кода в режиме останова

Возможности редактирования программы в окне кода ничуть не ограничены в режиме останова. Можно вводить новые операторы и изменять существующие или вообще удалять их. Но что особенно интересно, большинство внесенных изменений сразу работают, становясь частью исполняемой программы. Например, можно объявить новые переменные и сразу же использовать их в вычислениях, возможно даже в сочетаниях с уже существующими. Некоторые изменения приводят к остановке программы, например, такие как изменение типа переменной в объявлении.

Другие способы пропустить фрагмент кода

Существуют другие способы пропустить фрагмент кода, содержащий ошибку. Эти методы работают независимо от того, выполняется ли программа

пошагово в режиме останова или код редактируется перед запуском. Один из подходов предполагает размещение апострофа в начале каждой строки, которую нужно пропустить (таким образом, оператор превращается в комментарий). Можно воспользоваться панелью управления Comment Block (Комментарий), чтобы закомментировать выбранную строку или все выделенные строки кода (см. главу 53).

Иногда лучшим способом миновать участок кода является использование метки в сочетании с временными операторами `GoTo`. В следующем коде VBA полностью пропускает все операторы между `GoTo AfterTheSkip` и оператором, следующим за меткой `AfterTheSkip:`. Чтобы вновь запустить пропущенный код, достаточно удалить оператор `GoTo` или просто превратить его в комментарий.

```
...
    GoTo AfterTheSkip
    A = B + C
    D = A + E/F
    G = B + D
    AfterTheSkip:
    MsgBox "Сегодня " & Format(Now, "dddd")
...
```

Выбор иного следующего оператора

Предположим, при пошаговом выполнении программы вы пришли к выводу, что следующий оператор содержит страшную ошибку. Вместо выполнения этого оператора и отправки программы в штопор, лучше просто пропустить его и заняться им позже.

Редактор Visual Basic позволяет выбрать в качестве следующего оператора разные операторы практически по всему коду, используя мышь или клавиатуру. Таким образом можно перейти в любое место кода и не только пропускать неверный код, но и по несколько раз выполнять одни и те же операторы, пока не станет ясной суть их работы.

Представим, что в переменных остались данные, которые находились там до внесения изменений. Эти значения могут существенно отличаться от тех, которые бы были, при условии нормального выполнения программы. Если необходимо, можно присвоить переменным любое значение, используя окна Immediate, Watch или Locals.

Выбрать следующий оператор можно так.

- Используя клавиатуру, передвиньте курсор на строку, содержащую оператор. Затем нажмите <Ctrl+F9> или выберите команду Debug⇨Set Next Statement (Отладка⇨Назначить следующий оператор).
- При помощи мыши перетащите желтую стрелку указателя на границе окна кода к следующему оператору (см. рис. 55.5).

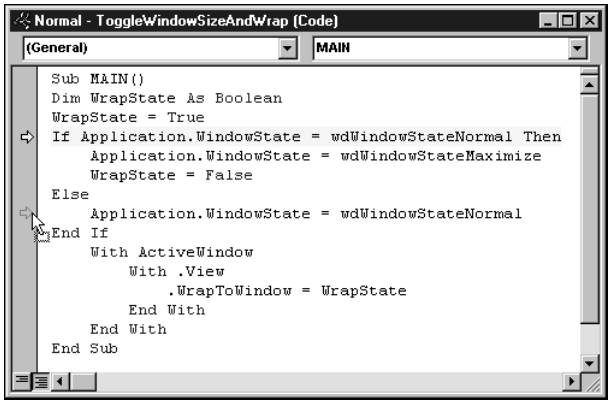

Рис. 55.5. Выбор иного следующего оператора позволяет повторять участки кода, действие которых не до конца изучено

Автоматические подсказки

Опция автоматических подсказок VBA позволяет увидеть текущее значение любой переменной, где бы в коде она ни использовалась. Находясь в режиме останова, наведите указатель мыши на любую переменную — появится окно подсказки, содержащее имя и текущее значение переменной. На рис. 55.6 показан пример использования данной подсказки.

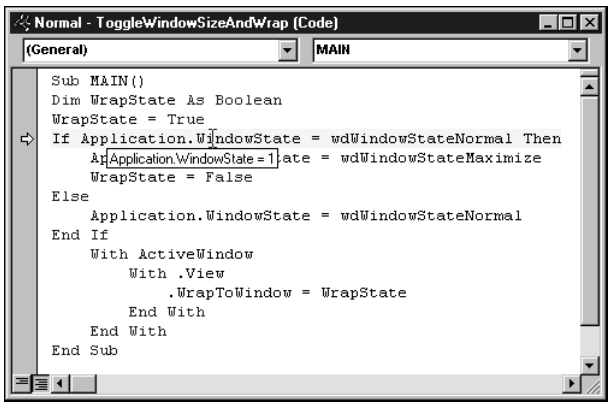

Рис. 55.6. Окошко автоматической подсказки содержит значение переменной, над которой находится указатель мыши

В случае, если окно автоматической подсказки не появляется, установите соответствующий флажок во вкладке Editor диалогового окна Tools⇨Options.

Можно использовать и другую опцию Editor, чтобы показывать тип и область видимости переменной, хотя данный процесс не столь автоматизирован. Поместите курсор в или перед именем переменной и нажмите комбинацию клавиш <Ctrl+I>, чтобы появилось окно Quick Info, содержащее все необходимые данные. При этом не обязательно находиться в режиме останова, что особенно полезно при работе в середине большой программы, переменные которой объявлены где-нибудь вначале. На рис. 55.7 показано, какого типа информация отображается в окне.

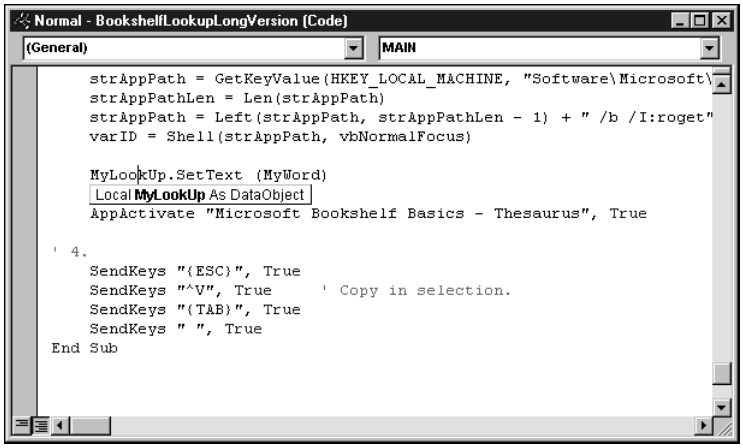

Рис. 55.7. Окно Quick Info, содержащее информацию о переменной

Окно Immediate

Окно Immediate (рис. 55.8) можно вызвать при помощи комбинации клавиш <Ctrl+G>.

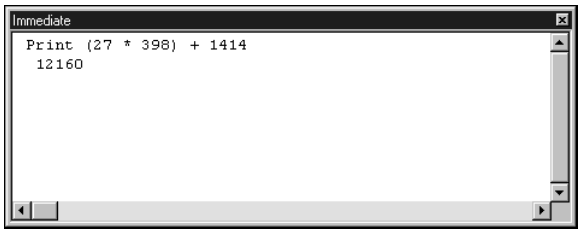

Рис. 55.8. Окно Immediate в действии

Оно позволяет делать следующее:

- видеть значения вычислений и переменных используя метод `Debug.Print`;
- выполнять отдельные операторы в коде напрямую, без запуска их в теле процедуры. Чтобы выполнить оператор в окне Immediate, просто введите его и нажмите <Enter>.

"Ну и зачем все это нужно?",— возникнет вопрос у читателя. Позвольте объяснить.

- Можно использовать окно Immediate как калькулятор. Если ввести выражение типа `Print (27 * 398) + 1414` и нажать <Enter>, получим результат, причем, *немедленно*. При работе в окне Immediate не нужно определять объект `Debug`.
- Можно отслеживать промежуточные значения переменных и выражений в различных точках выполняемой программы в окне Immediate, добавляя метод `Debug.Print` в код программы (а не в само окно Immediate). После завершения программы можно быстро проверить, правильные ли получились результаты, не выводя диалоговое окно для каждого значения (рис. 55.9).

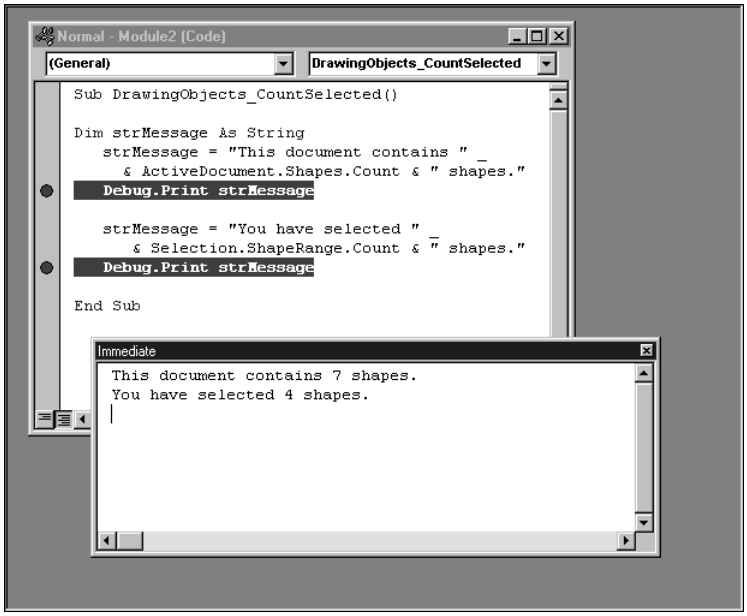

Рис. 55.9. Вывод результатов программы с использованием метода `Debug.Print`

- В режиме останова можно вывести значение любой переменной или свойства объекта в окне Immediate при помощи оператора `Print` или изменить значения, используя стандартный оператор присваивания, например `intTheirNumber = 25`
- Также можно вызвать процедуру `Sub` или `Function`, как обычно. Следует заметить, что в режиме останова операторы, которые выполняются в окне Immediate, могут работать только с теми переменными, объектами и процедурами, которые находятся в области видимости текущей исполняемой процедуры программы.

Немного интересных фактов. Можно перетащить выделенный текст из окна кода в окно Immediate удерживая нажатой клавишу <Ctrl>, а это значит, что можно не набирать заново длинные выражения и имена переменных (если не удерживать клавишу <Ctrl>, текст будет перенесен в окно Immediate, а не скопирован). Клавиша <F1> действует в окне Immediate так же, как и в окне кода, вызывая раздел справки, посвященный ключевому слову, на котором находится курсор. Автоматические подсказки в окне Immediate не функционируют.

Окно Locals

Если на экране хватает места, окно Locals должно находиться на экране на протяжении всего времени отладки программы в режиме останова. Для вызова этого окна щелкните на одноименной кнопке в панели инструментов Debug или выберите нужный пункт меню View (Вид).

Как показано на рис. 55.10, окно Locals автоматически отображает все переменные, доступные в текущей процедуре, показывая их имена, значения и типы данных. Редактор Visual Basic обновляет информацию при выполнении каждого оператора, т.е. значения в окне Locals всегда отвечают текущим значениям переменных.

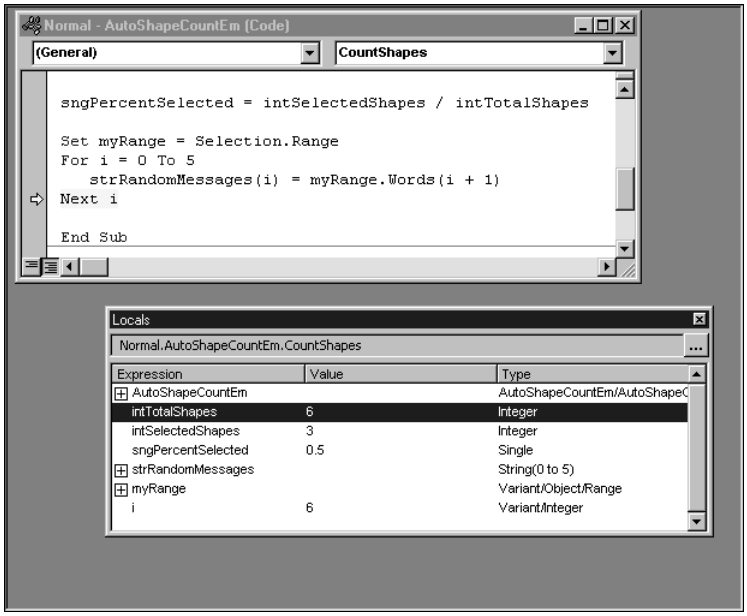

Рис. 55.10. Окно Locals в режиме останова

Как работает окно Locals

Подобно большинству окон редактора Visual Basic, окно Locals имеет свое место, заданное по умолчанию, но оно также может быть и плавающим отдельным окном. Основы работы с окнами редактора Visual Basic описаны в главе 52.

Если информация в столбцах не видна, помните, что их размер можно изменить. Наведите указатель мыши на разделитель между столбцами в заголовке в верхней части окна. Когда указатель приобретет вид двунаправленной стрелки, перетащите границу влево или вправо, изменяя, таким образом, ширину выбранных столбцов.

На рис. 55.10 видно, что в некоторых строках перечислены индивидные переменные типа массивов, переменных пользовательских типов и объектов. Такие элементы не имеют своих значений, они содержат переменные и другие "контейнерные" элементы. Когда процедура выполняется, окно Locals показывает их в свернутом виде — содержащиеся в них элементы не видны. Чтобы развернуть такой элемент, щелкните на квадратике с плюсом слева от элемента.

Только объявленные (явно или неявно) в текущей процедуре переменные появляются вверху иерархической структуры окна Locals сразу в развернутом виде. Чтобы увидеть переменные уровня модуля, доступные всем процедурам текущего модуля, разверните самый первый элемент в окне Locals. Этот элемент всегда принадлежит модулю, в котором выполняется текущая процедура. В окне Locals нельзя работать с глобальными переменными или переменными других проектов.

Зачем редактировать значения переменных

Перед описанием техники изменения значения переменной, остановимся на том, зачем это нужно. Причины таковы.

- В предыдущей строке кода содержалась ошибка, в результате которой переменной было присвоено неправильное значение. Местоположение ошибки найдено, но нужно продолжить проверку остальной части программы. Для этого необходимо изменить значение переменной так, чтобы ошибка не повлияла на дальнейшие результаты.
- Необходимо проверить некоторые альтернативные значения переменной, не внося изменений в код. Можно использовать команду `Set Next Statement`, чтобы последовательно выполнять один и тот же участок кода, вводя каждый раз различные значения в окне Locals.
- Значения, которые программа использует в реальной жизни, не всегда доступны. Например, программа должна читать информацию из базы данных, которая размещена в Internet. В этом случае можно подставить другие значения при помощи окна Locals.

Как изменять значения переменных

Чтобы изменить значение переменной при помощи окна Locals, выполните следующее.

1. Щелкните дважды (а не выполните двойной щелчок) на текущем значении переменной в окне так, чтобы значение было подсвечено. Щелчок в любом месте одной строки выделит ее всю.
2. Введите новое значение. Если вводится строковая величина, добавьте открывающую и закрывающую кавычки. Литералы даты заключаются в символы #.
3. Чтобы отменить ввод и восстановить предыдущее значение, нажмите <Esc>. Чтобы подтвердить ввод, нажмите <Enter>. Редактор не позволит ввести неправильное значение и выдаст сообщение об ошибке.

Можно изменять значения любых переменных, включая отдельные элементы массивов и пользовательских типов. Представим ситуацию, когда необходимо развернуть массив переменных пользовательского типа перед тем, как получить доступ к элементам данных. Что касается объектной переменной, нельзя изменить не присвоенный ей объект, но можно редактировать свойства объекта (кроме свойств только-для-чтения). Опять-таки, чтобы увидеть редактируемые элементы, разверните объект. На рис. 55.11 показан развернутый массив пользовательского типа и объектные переменные.

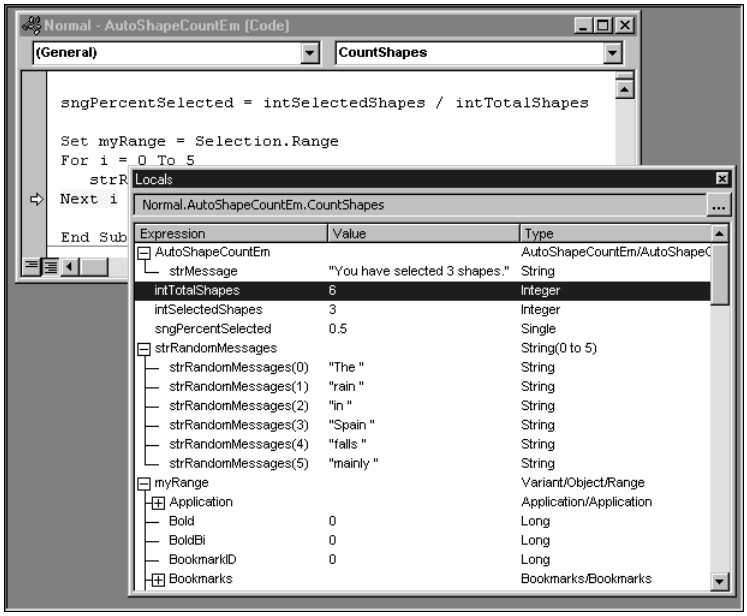

Рис. 55.11. Окно Locals отображает ту же процедуру, что показана на рис. 55.10, но свернутые ранее элементы здесь развернуты

Окно Watch — основной инструмент отладки

После освоения окна Locals работа с окном Watch не покажется сложной. Оно функционирует практически так же, разница лишь в отображаемых значениях. На рис. 55.12 представлено окно Watch в действии. Чтобы отобразить окно Watch, выберите команду View⇨Watch Window (Вид⇨Окно наблюдений).

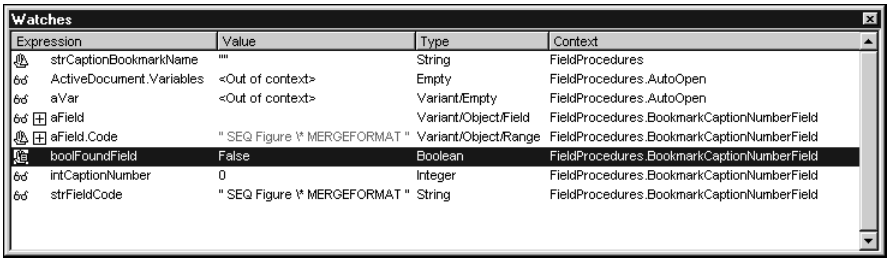

Рис. 55.12. Окно Watch редактора Visual Basic

Окно Watch используется для отладки больших программ со множеством различных процедур и глобальными переменными. Добавив эти глобальные переменные в окно Watch, можно отслеживать их текущее значение, независимо от того, какая часть программы выполняется.

Окно Watch появляется автоматически, как только определено, какое значение следует отследить. Если же места на экране немного, окно Watch можно убрать, щелкнув на кнопке закрытия в правом верхнем углу.

Чем еще окно Watch отличается от окна Locals

Помимо того, что содержимое окна зависит от пользователя, существуют еще два серьезных отличия между окнами Watch и Locals.

- Каждая строка окна Watch позволяет пользователю отследить значение любого выражения VBA, а не только индивидной переменной. Выражения типа `(X - Y) > 15`, `"Цвет " & strColor` или даже `2 +2` вполне подходят. Такие выражения называются просматриваемыми.
- Поскольку окно Watch может отслеживать переменные из любого модуля или процедуры проекта, в нем добавляется четвертый столбец — Context (Контекст). В этом столбце описывается область видимости, где отображается значение переменной или выражения. При выполнении процедуры вне указанного контекста в нем находится сообщение `<Out of context>`.

Добавление просматриваемых выражений

Если вам нравится всегда иметь несколько вариантов, то окно Watch вполне подойдет (просто удивительно, сколькими способами можно добавить просматриваемые выражения).

Неважно, какую технику вы используете, начните с выделения нужной переменной или выражения в окне кода. Под *выделением* подразумевается то, что должно быть выделено все выражение, как в любом текстовом процессоре, при помощи мыши или используя комбинацию клавиши <Shift> и клавиш управления курсором.

Однако, если нужно отследить значение только одной переменной (другими словами, элемента, который не является свойством объекта или элементом пользовательского типа), выделять всю переменную необязательно, достаточно установить курсор на имени переменной.

Добавить просматриваемое выражение можно:

- щелкнув правой кнопкой мыши на выделении и выбрав Add Watch (Добавить в окно наблюдений), чтобы вызвать одноименное диалоговое окно;
- выбрав команду Debug⇨Add Watch (Отладка⇨Добавить в окно наблюдений), что также приведет к появлению диалогового окна Add Watch;
- щелкнув на кнопке Quick Watch на панели Debug или выбрав команду меню Debug⇨Quick Watch. Редактор выдаст подтверждающее сообщение, описывающее просматриваемое выражение, которое собирается добавить пользователь, но изменить он ничего не сможет.
- перетащив выделение в окно Watch, что позволит создать просматриваемое выражение без промежуточных шагов.

Если же вы приверженец ручного труда, в окне кода ничего не нужно выделять. Просто щелкните на кнопке Add Watch и затем с нуля введите нужное выражение в одноименном диалоговом окне.

Диалоговое окно Add Watch

На рис. 55.13 приведено диалоговое окно Add Watch, которое появляется на экране при выборе команды Debug⇨Add Watch или команды Add Wtch из контекстного меню. В диалоговом окне Add Watch можно определить параметры просматриваемого выражения.

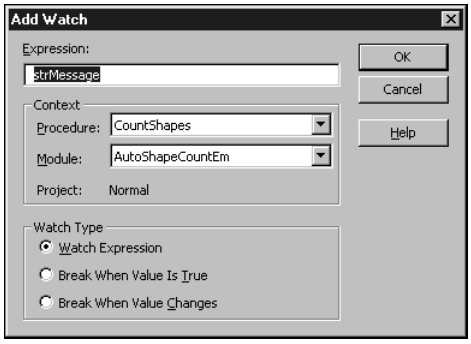

Рис. 55.13. Диалоговое окно Add Watch

В поле Expression (Выражение) задайте просматриваемое выражение. Если изначально выбрана правильная переменная или выражение, ничего менять не нужно. Хотя, при желании, конечно, можно.

Раздел Context позволяет определить процедуры, в которых редактор Visual Basic вычисляет и отображает значение просматриваемого выражения. Значения полей Procedure (Процедура) и Module (Модуль) задают процедуру, из которой переменная добавлена в окно Watch. Если нужно увидеть значение при запуске разных процедур, выберите их имена. Если процедура находится в другом модуле, вначале выберите модуль, а потом процедуру.

Чтобы переменная оставалась видимой не зависимо от того, какая процедура в данном модуле исполняется в настоящий момент, выберите All Procedures (Все процедуры) — первый элемент списка Procedure. А чтобы видеть переменную всегда, выберите All modules (Все модули) в списке Module. Чем шире выбранная область видимости, тем дольше VBA будет подсчитывать значение. Однако, иногда возможность отследить значение переменной во всей программе более важна чем небольшая задержка.

Обычно в разделе Watch Type (Тип наблюдения) лучше оставить переключатель Watch Expression (Выражение). Другие возможные значения рассмотрены в разделе "Использование просматриваемых выражений для определение контрольных точек" далее в этой главе.

Редактирование просматриваемых выражений

Как и в окне Locals, значение переменной можно изменить во время работы программы. Однако настройки раздела Context в окне Watch изменить нельзя.

Для этой цели вызовите диалоговое окно Edit Watch, копию диалогового окна Add Watch, показанного на рис. 55.13, используя комбинацию клавиш <Ctrl+W>, либо из меню Debug. Когда это диалоговое окно открыто, вы сможете внести изменения в выражение или его контекст подобно тому, как если бы оно определялось заново.

Проще всего отредактировать существующее просматриваемое выражение путем внесения изменений в столбец Expression окна Watch.

Щелкните в строке, содержащей выражение, чтобы выбрать его, а затем еще раз щелкните на самом выражении, чтобы его выделить. Далее можно редактировать выражение, как угодно. Контекстное меню в окне Watches также содержит команду Edit Watch.

Использование просматриваемых выражений для определения контрольных точек

Используя переключатели группы Watch Type в диалоговом окне Add Watch или Edit Watch, можно установить два типа включения режима останова.

- Можно сделать так, что программа будет автоматически входить в режим останова по выполнению любого оператора в VBA, который изменяет выражение на ненулевое значение (напомним, 0 это `False`, а все другие выражения — это `True`). Для постановки контрольной точки такого типа выберите Break When Value Is True (Останов при достижении значения Истина).
- Можно заставить программу автоматически входить в режим останова сразу после исполнения программой оператора, который изменяет значение выражения. Для установки контрольной точки такого типа выберите Select Break When Value Changes (Останов при изменении значения).

Несмотря на неординарные способности, эти просматриваемые выражения работают в окне Watch, как и обыкновенные. Вы можете отличать просматриваемые выражения разных типов по маленьким пиктограммам, расположенным слева от выражений (рис. 55.14). Когда контрольная точка запускается изменением выражения, VBA приостанавливает выполнение программы и входит в режим останова. В окне кода оператор, вызвавший изменение, находится перед подсвеченным оператором `Exit For`. Просматриваемое выражение, которое вызвало остановку, подсвечено в окне Watch, так что его легко обнаружить (см. рис.55.14).

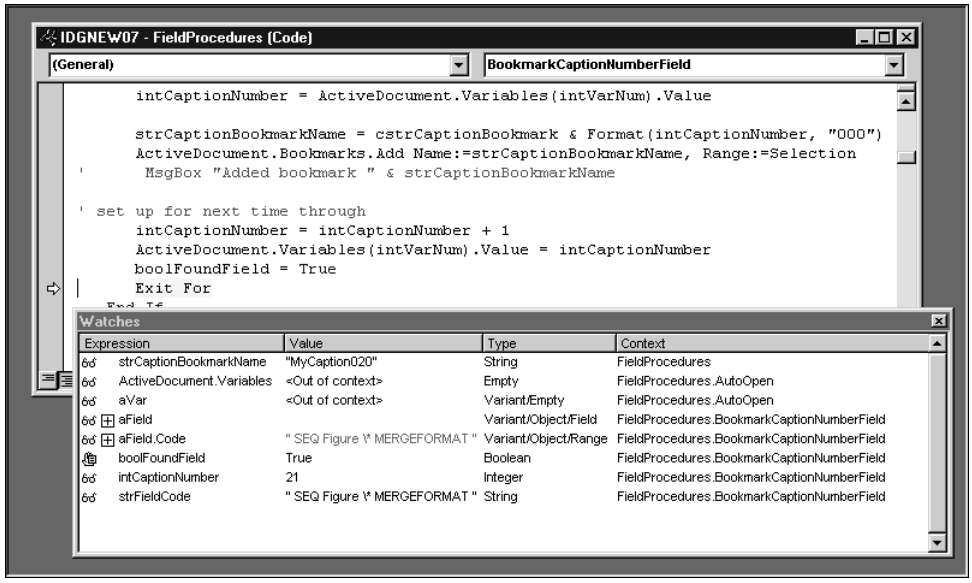

Рис. 55.14. Просматриваемое выражение Break When Value Changes, примененное к переменной `boolFoundFiles`, привело к переводу программы в режим останова

Чем больше программа, тем тяжелее отслеживать, какие ее процедуры и операторы изменяют определенные переменные. Часто вы подозреваете, что окончательное значение переменной неверно, но не знаете, в каком месте кода искать проблему. В таком случае контрольная точка, основанная на просматриваемом выражении, может спасти жизнь, ну, или, по крайней мере, код.

Перехват ошибок с использованием оператора On Error и объекта Error

Когда во время выполнения программы что-то идет не так, как задумывалось, результат обычно оказывается катастрофическим. VBA останавливает выполнение и выводит диалоговое окно с сообщением об ошибке, чтобы сжато сообщить плохие новости. В диалоговом окне можно выбрать одну из следующих опций: End (Завершение программы), Debug (Отладка) и Help (Справка), которая выводит раздел справки о появившейся ошибке.

Выбор опции не достаточно очевиден, особенно если программа выполняется в рабочем режиме, а не для проверки. Гораздо лучше, если программа сможет определить ошибку и исправить ее или, по крайней мере, обойти, до того как VBA вывалит свое зловещее окно с сообщением об ошибке во время исполнения. Даже если не принимать во внимание такие неприятности, можно хотя бы порадовать пользователя более интересным сообщением. Однако потребуется добавить в программу код обработки ошибок, как описано в данном разделе.

Откуда берутся ошибки во время исполнения

Как упоминалось ранее, ошибки периода исполнения происходят по двум причинам.

- Серьезная ошибка в логике программы приводит к возникновению ситуации, с которой VBA не может справиться. Предположим, в операторе некое значение делится на значение переменной. Если где-то в коде этой переменной присвоен нуль, произойдет неприятность —деление на нуль строго запрещено, попытка выполнить это действие — значит повесить систему, поэтому VBA прекратит выполнение программы.
- Произошли некие непредвиденные обстоятельства. Например, соединение с файлом базы данных или Internet разорвано, и программа не получает значения, необходимые для работы.

Разные причины, но результат один. Нужно защитить программу и пользователей с помощью кода обработки ошибок.

Как работает код обработки ошибок

Чтобы предотвратить сообщения VBA об ошибках периода исполнения и обработать их самостоятельно, снабдите каждую процедуру *обработчиком ошибок* — фрагментом кода, единственная цель которого вмешиваться в выполнение программы при возникновении ошибки. В случае неожиданностей VBA передает управление обработчику ошибок, код которого определяет, какая ошибка имеет место, и предпринимает шаги, которые требует сложившаяся ситуация.

Как написать код обработки ошибок

Чтобы добавить в процедуру код обработки ошибок, выполните следующее.

- Добавьте в начало процедуры оператор `On Error`, чтобы указать VBA, где находится код обработки ошибок.
- Введите оператор `Exit Sub` или `Exit Function` после тела процедуры, перед кодом обработчика ошибок.
- Напишите код обработки ошибок.

Все эти этапы важны: без начального оператора `On Error` VBA не будет знать о существовании обработчика ошибок, а без оператора `On Error` ошибки могут привести к остановке программы.

Написание операторов On Error

Оператор `On Error` включает обработчик ошибок и сообщает VBA его положение в коде процедуры. Полностью оператор имеет синтаксис `On Error GoTo` *метка*, где *метка* — метка в любом месте процедуры, определяющая первую строку кода обработки ошибки.

Вот как это выглядит в процедуре:

```
Sub ErrorHandlerDemo()
On Error GoTo ErrorHandler
    MamaVariable = DoThisFunction (X,Y,Z)
    PapaVariable = DoThatProcedure
    BabyVariable = MamaVariable + PapaVariable
Exit Sub ' Остановить процедуру здесь, если ошибки не
произошло
ErrorHandler:
(здесь должен быть код обработки ошибки)
End Sub
```

Есть и две другие формы оператора `On Error`. Они работают следующим образом:

- `On Error GoTo 0` отключает обработку ошибок, начиная с этого места и далее в процедуре, и очищает объект `Err`;
- `On Error Resume Next` предписывает VBA игнорировать строку, содержащую ошибку, и продолжать выполнение со следующей строки. Эта схема полезна при проверке объекта `Err` сразу после строки, в которой находится ошибка для обработки ошибок в теле процедуры.

Добавление в процедуру оператора Exit

Обработчик ошибок является частью процедуры, в которой он вызывается. Это значит, что VBA выполняет код обработчика ошибок каждый раз при выполнении проце-

дуры, пока пользователь явно не укажет обратное. Очевидно, что это неправильно. Нужно, чтобы обработчик ошибок выполнялся только при возникновении ошибки, а не когда попало.

Чтобы заставить VBA работать таким образом, нужно лишь добавить оператор `Exit` перед кодом обработки ошибок, как показано в примере, приведенном в предыдущем разделе. При создании подпрограмм `Sub` оператором выхода должен быть `Exit Sub`; для функции `Function`, соответственно — `Exit Function`.

Написание обработчика ошибок

Обработчик ошибок должен содержать несколько компонентов, описанных в следующих подразделах.

Добавление метки

Чтобы определить начало кода обработки ошибки, в первой строке обработчика ошибок вводится стандартная метка VBA. Как пояснялось ранее, оператор `On Error` при ошибке в процедуре направляет выполнение программы к этой метке. В данном примере, `ThisIsTheErrorHandler` и есть такой меткой.

```
...
ThisIsTheErrorHandler
...
    (Операторы обработки ошибок)
...
End Sub
```

Получение информации об ошибках и их обработка

В тело обработчика ошибок закладываются две основные задачи: проверка типа ошибки и связанные с ней действия. Эти операции существенно отличаются, но на практике помещаются рядом.

Сначала необходимо определить, что не так с VBA-объектом `Err`. Если известно, что с некоторой переменной возможны проблемы, перед проверкой объекта `Err` убедитесь, не выходит ли ее значение за границы диапазона. В противном случае, ключом к разгадке будет именно он.

Объект `Err` доступен в любой программе VBA. Его можно использовать, не создавая копию. VBA автоматически сохраняет информацию о последней ошибке в объекте `Err`. Ваш код должен лишь получить свойства объекта `Err`, `Number` и `Description`.

Свойство `Number` сообщает номер текущей ошибки. Его можно присвоить переменной или использовать прямо в условном операторе, как в следующем примере:

```
Sub YetAnotherFineMess()
On Error GoTo ImTryinToThingButNothingHappens
...
(операторы, которые могут содержать ошибку)
...
Exit Sub
ImTryinToThingButNothingHappens:

Select Case Err.Number
    Case 7 ' Ошибка выхода за границы памяти
        (Код обработки ошибки номер 1)
    Case 11 ' Ошибка деления на ноль
```

```
        (Код обработки ошибки номер 2)
        (и т.д.)
    Case Else
        (Код обработки всех остальных ошибок)
End Select
End Sub
```

Здесь оператор `Select Case` сверяет свойство `Number` с рядом значений, для которых написаны операторы обработки ошибок. Такие операторы должны выполнять следующее:

- информировать пользователя об ошибке и запросить инструкции при помощи окна ввода или стандартной формы;
- вновь попробовать получить верную информацию из источника, не доступного ранее, либо из альтернативного источника;
- изменить неправильное значение на подходящее, записав в файле или в окне сообщения, что значение было изменено принудительно.

Для эффективного использования свойства `Number` требуется знать, что обозначает каждый номер ошибки. Информацию о наиболее общих ошибках можно найти в справочной системе VBA.

Если код не может справиться с ошибкой или вам не хочется возиться с проблемами, стандартного сообщения VBA об ошибке можно по-прежнему избежать, используя свойство `Description` объекта `Err`. В следующем примере выдается вежливое сообщение:

```
strMyErrMessage = "Вынужден с прискорбием сообщить, что " _
    & "не все ладно в этой прекрасной " _
    & "программе. Согласно VBA, причиной " _
    & "неполадок является "
MsgBox strMyErrMessage & Err.Description
```

Некоторые приложения VBA обеспечивают дополнительные объекты и функции, предоставляющие информацию о проблемах, связанных с работой приложения. В Microsoft Access, например, существует отдельный объект `Error`.

Продолжение выполнения программы

Когда обработчик ошибок завершает работу, можно либо продолжить процедуру с точки, в которой она была остановлена, либо передать управление в вызывающую ее процедуру.

Поместите оператор `Resume` в обработчике ошибок, если нужно перейти в текущую процедуру. Оператор `Resume` может использоваться по-разному.

- Оператор `Resume` в отдельной строке передает управление в процедуру, вызвавшую ошибку. Используйте этот вариант только, когда причина ошибки устранена.
- Чтобы пропустить оператор, вызвавший ошибку, поместите в обработчик ошибок оператор `Resume Next`. Выполнение будет продолжено с оператора, следующего за вызвавшим ошибку.

- Чтобы перейти к определенной точке в процедуре, используйте оператор `Resume` *метка* после кода обработки ошибок. *Метка* ссылается на ту метку, которая расположена в процедуре, а не на метку, определяющую обработчик ошибок. Конечно, при использовании такого подхода нужно добавить в процедуру еще одну метку.