

24.3. Создание программ инсталляции.....	573
24.4. Как писать и распространять Shareware программы.....	583
24.5. Создание программ маленького размера.....	586
24.6. Оптимизация скорости выполнения программы.....	593

24.3. Создание программ инсталляции

Когда твоя программа готова, надо задуматься о том, как бы её установить на компьютер пользователя. Если программа состоит только из одного файла, то никаких проблем. А что если программа состоит из множества файлов? В этом случае используют программы установки, которые запускаясь копируют всё необходимое на компьютер клиента. Именно этот способ я советую тебе использовать и в этой части мы познакомимся с программой *InstallShield Express*.

Программу *InstallShield Express* чаще всего можно найти на том же диске, что и Delphi. Она устанавливается отдельно и если ты ещё не установил её, то сделай это сейчас.

Запустив программу ты увидишь окно, как на рисунке 24.3.1. Главное окно содержит файл помощи, в котором показано, как работать с программой. Слева окна расположено дерево, в котором можно выбирать раздел, который тебя интересует. В центре окна будет отображаться информация, найденная по выбранному разделу. Если у тебя нет проблем с английским, то ты сможешь разобраться по файлу помощи, но если проблемы есть, то лучше почитать эту часть книги, потому что здесь я постараюсь описать всё необходимое для создания собственных программ инсталляции.

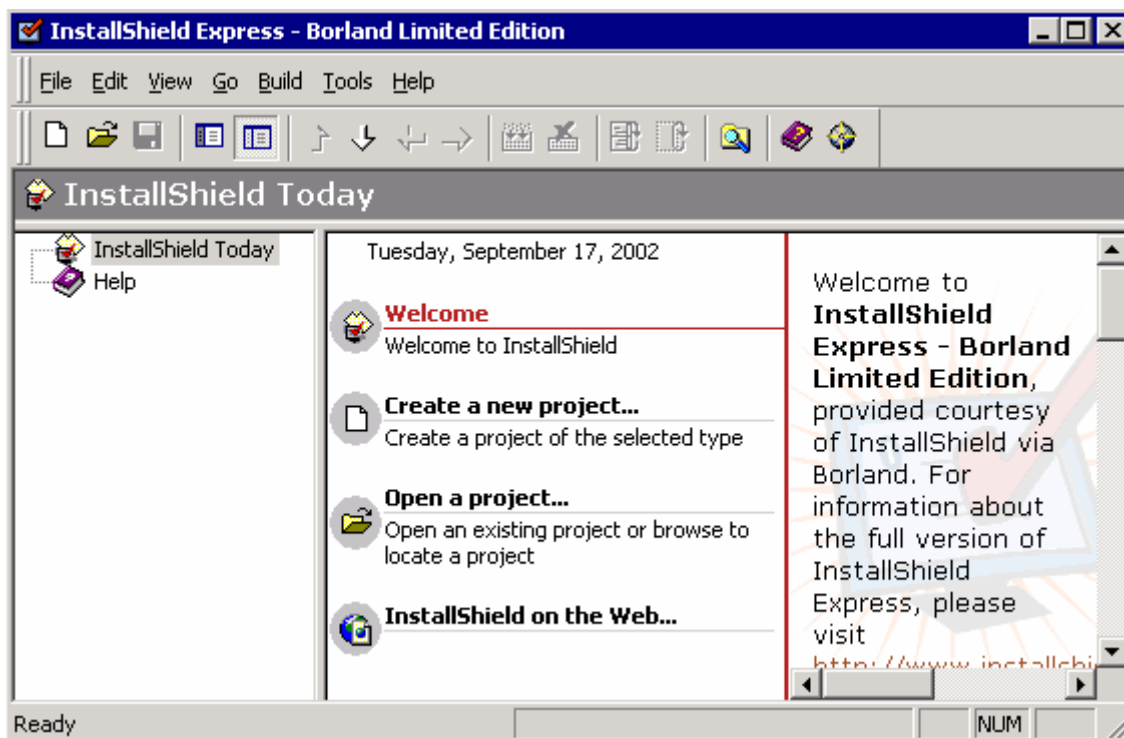


Рис 24.3.1. Главное окно программы *InstallShield Express*

Для создания нового проекта выбери из меню *File* пункт *New*. Перед тобой откроется окно создания нового проекта. В этом окне нужно указать лишь путь проекта и имя файла. У имени файла должно быть расширение *ism* и старайся ему задавать вполне понятное имя, потому что оно будет использоваться в качестве имени проекта. Указав путь и имя проекта нажимай *OK*.

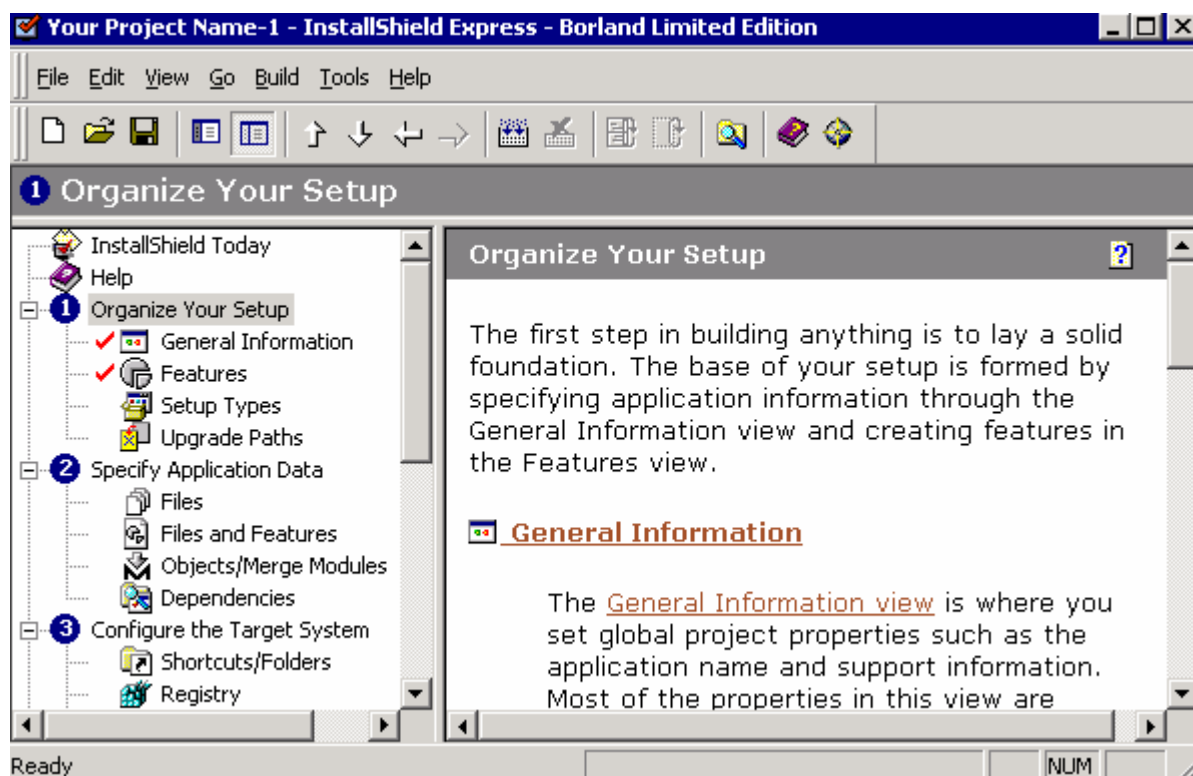


Рис 24.3.2. Главное окно программы после создания нового проекта

Теперь главное окно измениться, и примет вид подобный рисунку 24.3.2. Дерево слева, где были пункты помощи, теперь пополнилось новыми пунктами. Выбирая эти пункты, ты слева будешь указывать параметры будущего инсталлятора. Давай рассмотрим эти пункты более подробно:

Organize Your Setup – это имя раздела, в котором ты найдёшь подпункты основных параметров программы установки. Если щёлкнуть по этому пункту, то в левой половине окна ты увидишь исчерпывающую информацию о подпунктах, что в них храниться и для чего они предназначены. Далее пойдёт описание подпунктов этого раздела.

General Information – здесь ты должен указать основные сведения о себе, как о разработчике, указать свой сайт в сети интернет и контактную информацию (рисунок 24.3.3). Давай подробно рассмотрим основные свойства, которые ты можешь изменить в этом пункте:

Author	
Authoring Comments	
Subject	Your Product Name
Keywords	Installer; MSI; Database
Product Name	Default
Display Icon	
Product Version	1.00.0000
INSTALLDIR	[ProgramFilesFolder]\{Your Company Name}
Publisher/Product URL	http://www.yourcompany.com
Product Update URL	http://www.yourcompany.com
Publisher	Your Company Name
Support Contact	
Support URL	http://www.yourcompany.com

Рис 24.3.3. Свойства пункта General Information

1. *Author* – здесь нужно ввести имя автора программы. Введи своё имя или название своей компании.
2. *Authoring Comments* – Комментарии автора.
3. *Subject* – здесь указывается имя программы, которую нужно установить.
4. *Product Name* – имя продукта.
5. *Display Icon* – иконка программы.
6. *Product Version* – версия продукта.
7. *INSTALLDIR* – директория, в которую будет установлена программа. По умолчанию используется директория *[ProgramFilesFolder]\Your Company Name\Default*. Здесь *[ProgramFilesFolder]* указывает на то, что программа должна устанавливаться в папку Program Files на компьютере клиента, после чего указывается подкаталог этой папки. В качестве примера предлагается ввести имя компании (*Your Company Name*).
8. *Publisher/Product URL* и *Product Update URL* – адрес в сети интернет, по которому можно найти программу.
9. *Publisher* – здесь опять укажи своё имя или название компании.
10. *Support Contact* – контактная информация со службой поддержки. Здесь указывается e-mail службы поддержки (твой e-mail).

Остальное – специфичные настройки, которые используются редко. Не советую тебе указывать в качестве службы поддержки свой телефон или реальный почтовый адрес, используй только e-mail.

Features – следующий раздел, в котором ты устанавливаешь возможности инсталлятора. Если ты выберешь этот раздел, то в левой части окна увидишь нечто похожее на рисунок 24.3.4.

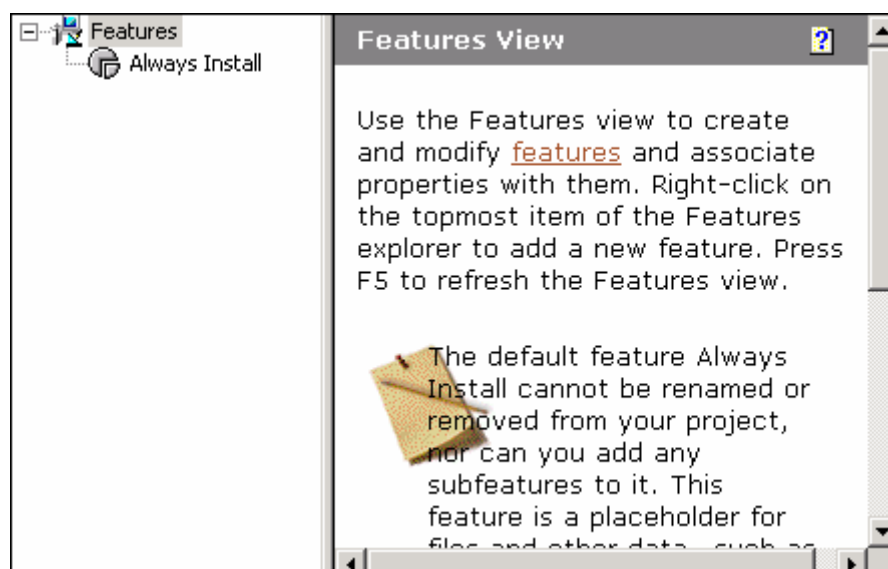


Рис 24.3.4. Свойства пункта Features

Здесь слева находится дерево, в котором перечислены разные возможности инсталлятора. По умолчанию создана только одна возможность – *Always Install* (устанавливать всегда). Щёлкни правой кнопкой по верхнему элементу или нажми клавишу Ins, чтобы создать новую возможность – назови её *Organizer*. Допустим, что мы создаём программу установки какой-нибудь программы. Все файлы этой программы мы поместим в пункт *Always Install*. У нашей программы будет и дополнительная возможность – органайзер, который пользователь должен уметь выбирать – устанавливать или нет.

Выбирая одну из возможностей ты можешь указать её свойства:

Description - описание типа установки.

Required – обязательность типа.

Visible – видимость. Для пункта *Always Install* – здесь указывается невидимость, потому что здесь будут указываться пункты, которые должны инсталлироваться всегда. Для пункта *Organizer* – который должен быть виден в окне выборочной установки здесь нужно ставить *Visible and Collapsed*, чтобы пользователь мог выбирать, устанавливать ему дополнительную возможность - *Организер* или нет.

Setup Types – типы установки. Если выбрать этот пункт, то в левой половине окна ты увидишь окно, похожее на рисунок 24.3.5. Слева окна находится список разных типов установки: *Typical* (типичная), *Minimal* (минимальная), *Custom* (выборочная). У каждого пункта есть элемент управления *ComboBox*, в котором ты можешь выбрать – нужен этот тип установки или нет. По умолчанию во всех типах стоят галочки. Для нашего примера оставим всё так, как есть, пусть будет три типа установки.

Выделяя в левом списке тип установки, ты можешь в правом списке указывать то, что должно устанавливаться в данном случае. При выборе *Typical* должно устанавливаться все, поэтому в левом списке оставляем выделенными все пункты. При выборе типа *Minimal* должен устанавливаться необходимый минимум, т.е. организер не обязателен и с него снимаем галочку. При выборе *Custom* опять же оставляем всё выделенное, чтобы пользователь сам мог убрать то, что ему надо.

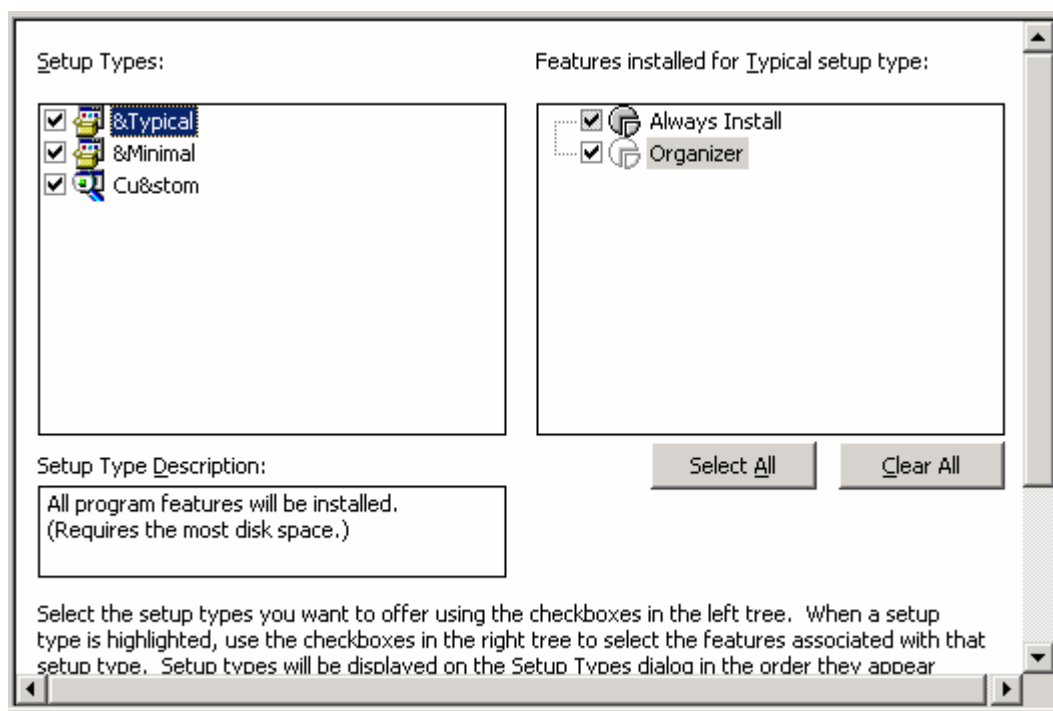


Рис 24.3.5. Свойства пункта *Setup Types*

Specify Application Data – это следующий раздел, в котором нужно указать какие файлы нужно устанавливать на компьютер клиента. Этот раздел состоит из следующих пунктов:

Files – здесь ты указываешь, какие файлы нужно устанавливать на компьютер клиента. Если выбрать этот пункт, то в левой половине окна ты увидишь рисунок 24.3.6.

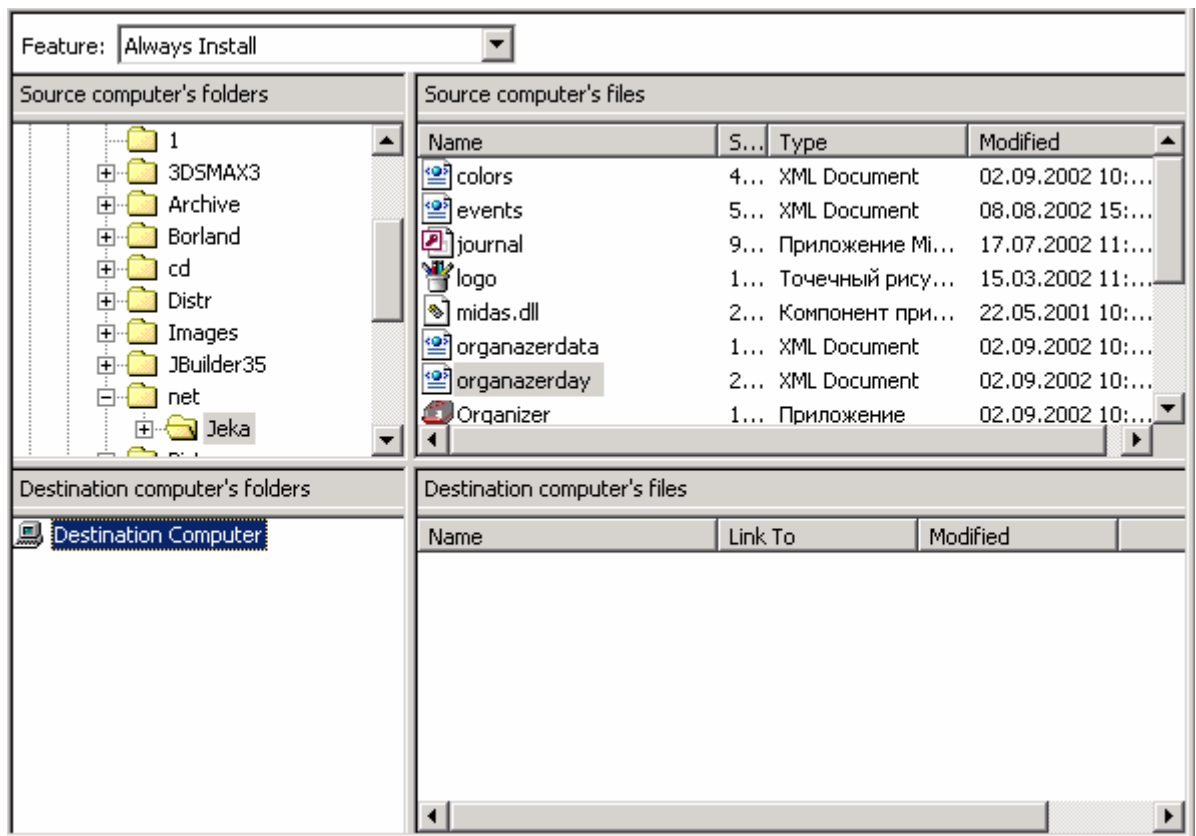


Рис 24.3.6. Свойства пункта Files

Сверху ты можешь выбирать в выпадающем списке тип установки. У нас должно быть там два пункта:

1. *Always Install*.
2. *Organizer*.

Чуть ниже слева находится список дисков и директорий твоего компьютера. Выбирая директорию ты справа будешь видеть файлы выбранной папки. Перейди в директорию, где находятся файлы твоей программы, для которой ты создаёшь программу установки.

Внизу слева ты можешь видеть окно, в котором мы должны выбирать папку на компьютере клиента, в которую нужно устанавливать файлы. Щёлкни в этом окне правой кнопкой мыши по пункту *Destination Computer* и в появившемся меню выбери пункт *Show Predefined Folder* и затем выбери пункт *[INSTALLDIR]*. В дереве этого окна появиться соответствующий пункт. В этот пункт надо переместить основные файлы, которые надо поместить в папку, в которую устанавливается программа. Если какие-то файлы должны быть помещены в папку Windows, то щёлкни в этом окне правой кнопкой мыши по пункту *Destination Computer* и в появившемся меню выбери пункт *Show Predefined Folder* и затем выбери пункт *[WindowsFolder]*. В созданный пункт дерева можно переносить файлы, которые должны быть в папке Windows.

Теперь в выпадающем списке *Features* (сверху окна) выбери пункт *Organizer*. Снова создай в левом нижнем окне пункт *[INSTALLDIR]* и перенеси в него файлы органайзера.

Files and Features – здесь ты можешь просмотреть информацию о копируемых файлах в виде списка (рисунок 24.3.7).

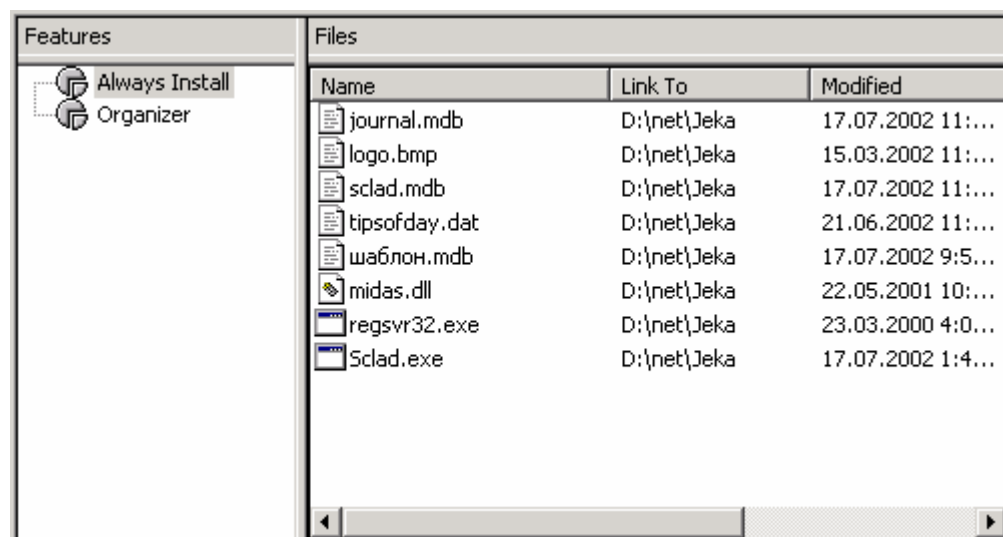


Рис 24.3.7. Свойства пункта Files and Features

Objects/Merge Modules – здесь ты можешь указать, какие модули нужно перенести на компьютер клиента. Выделив этот пункт, перед тобой откроется список установленных на твоём компьютере модулей, которые можно перенести на компьютер клиента (рисунок 24.3.8).

Допустим, что твоя программа использует базы данных MS Access. В этом случае на компьютере клиента должен быть установлена надстройка DAO. Твоя программа установки может автоматически перенести эту надстройку на компьютер клиента. Достаточно только найти её в списке левого верхнего окна и поставить напротив этой строки галочку. Программа сама определит файлы, которые надо скопировать на компьютер клиента и при установке внесёт необходимые изменения в реестр.

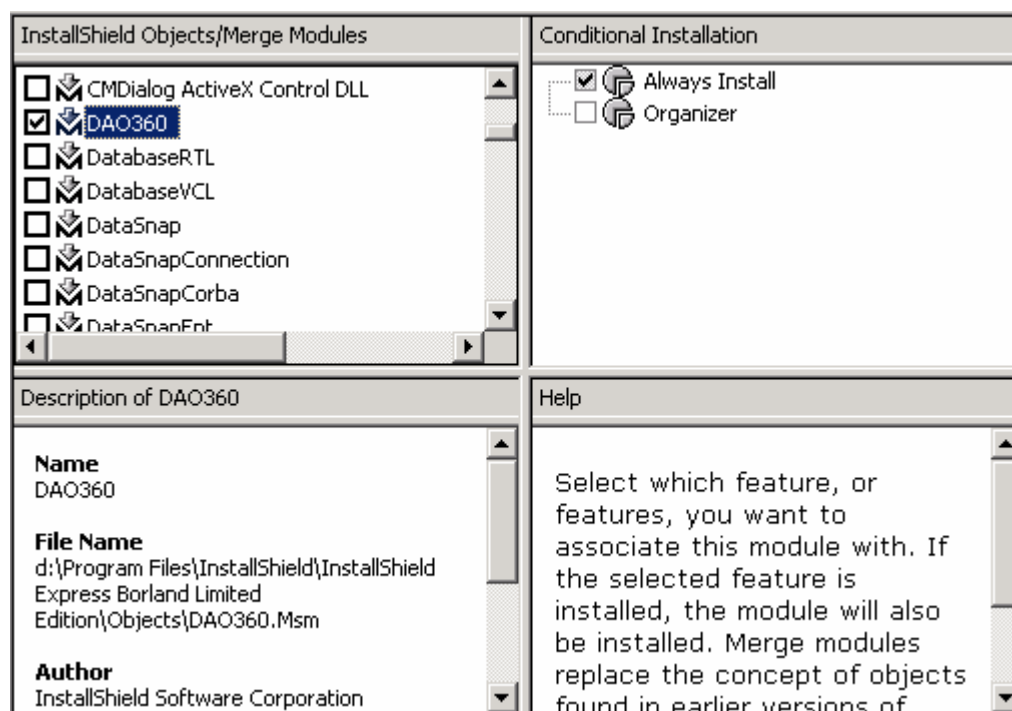


Рис 24.3.8. Свойства пункта Objects/Merge Modules

Если тебе необходимо установить на компьютер клиента какой-нибудь компонент ActiveX, то опять же его можно найти в этом списке. Созданная программа установки

сама регистрирует этот компонент во время инсталляции и ты избавишься от всех проблем по установке и регистрации ActiveX компонентов.

Configure the Target System – это следующий раздел, в котором ты можешь указать, какие изменения надо произвести на компьютере клиента.

Shortcuts/Folders – здесь мы указываем ярлыки, которые нужно создать в программном меню для вызова программ. Выдели этот пункт и ты увидишь в правой половине окна нечто похожее на рисунок 24.3.9.

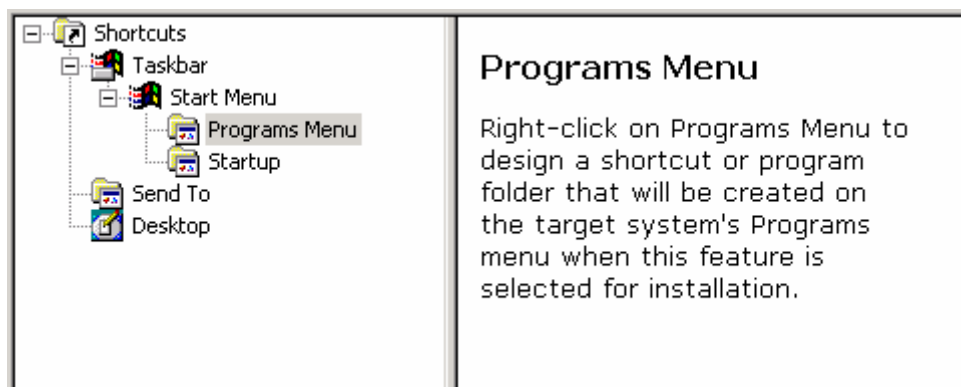


Рис 24.3.9. Свойства пункта Shortcuts/Folders

Если ты хочешь создать ярлык для вызова программы из меню *Пуск*, то выдели пункт *Program Menu* и щёлкни по нему правой кнопкой. В появившемся меню выбери пункт *New Folder*, чтобы создать отдельную папку для своей программы. Теперь выдели эту папку и щёлкни правой кнопкой по ней. Здесь выбери пункт *New Shortcut*, чтобы создать ярлык для программы.

Выделив имя созданной иконки ты увидишь следующие её свойства:

Description – описание программы.

Feature – когда создавать ярлык. По умолчанию стоит «Всегда», но ты можешь выбрать пункт *Organizer*, если иконку нужно создавать только если пользователь требует установку органайзера.

Arguments – здесь указываются параметры, которые нужно передавать программе.

Target – файл, который надо запускать при выборе этой иконки. Здесь нужно указывать имя запускного файла, относительно компьютера клиента, например `[INSTALLDIR]\organizer.exe`.

Icon File – файл иконки для ярлыка.

Icon Index – если у программы есть своя иконки, то ты можешь указать её индекс. Например, если здесь указать 0, то будет использоваться первая иконка программы.

Run – здесь указываются параметры запуска программы. По умолчанию стоит нормальный запуск – *Normal Window*.

Working Directory – рабочая директория программы. Здесь так же нужно указывать директорию относительно компьютера клиента, например, `[INSTALLDIR]`.

Registry – здесь ты можешь указывать изменения, которые надо произвести в реестре. Окно разделено на две части: в верхней ты видишь свой текущий реестр, снизу ты видишь изменения, которые надо произвести.

Для создания нового параметра, который нужно будет создать на машине клиента ты должен щёлкнуть правой кнопкой по нужному разделу и в появившемся меню выбрать пункт *Key* для создания нового ключа. В этом ключе ты можешь создать ещё один ключ или параметр (снова щёлкнув правой кнопкой).

ODBC Resources – здесь ты можешь увидеть в виде дерева все установленные к тебе ODBC драйвера. Эти драйвера используются для доступа к базам данных с через

компоненты ADO. Мы в этой книге использовали MS Jet драйвер, который относится к DAO, но тебе могут понадобиться и ODBC драйвера. Если нужен такой драйвер, то найди его в левом верхнем окошке и поставь галочку напротив имени.

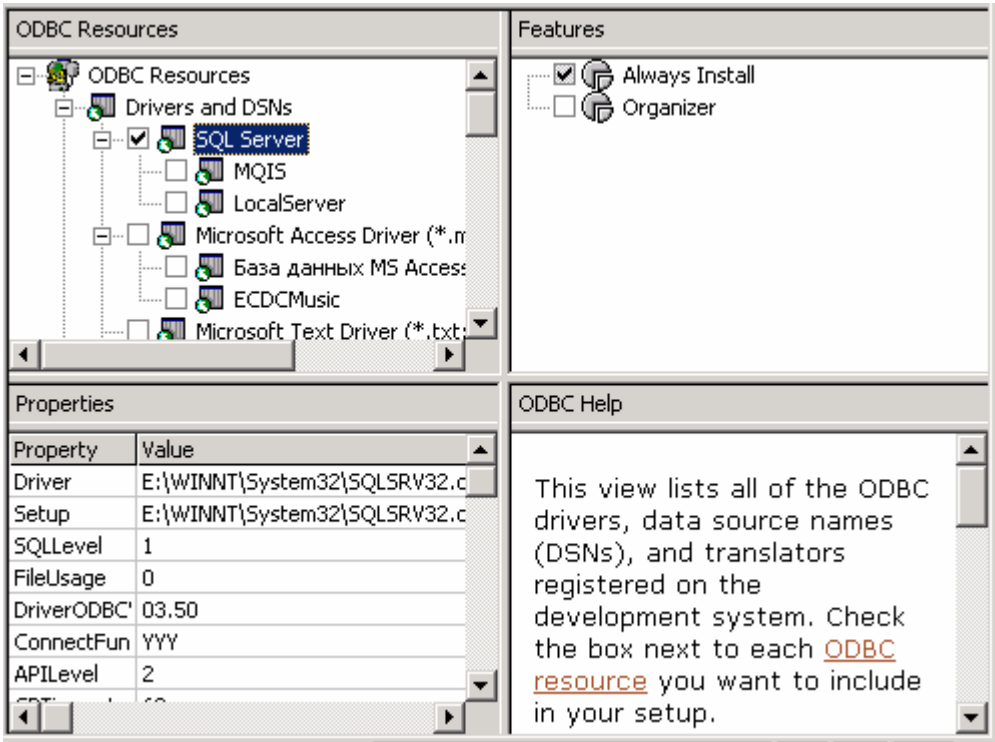


Рис 24.3.10. Выбор ODBC драйвера

Как только ты выбрал ODBC драйвер, так сразу активизируются правое верхнее и левое нижнее окна. В правом верхнем ты можешь указать в каких типах установки нужно устанавливать этот драйвер. В левом нижнем видны свойства драйвера.

INI File Changes – здесь ты можешь указать, какие изменения нужно произвести в ini файлах. Я эти фалы не использую по причине их устарелости и тебе не советую. Поэтому я не буду останавливаться на этом пункте и двинусь дальше.

File Extensions – здесь ты указываешь, какие расширения нужно зарегистрировать под твою программу. Выбери этот пункт и слева появиться окно, как на рисунке 24.3.11.

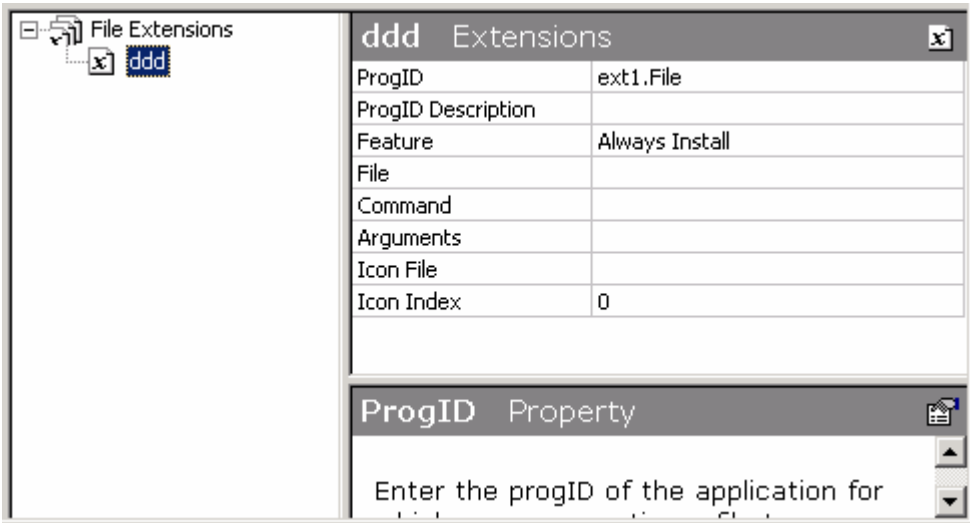


Рис 24.3.11. Окно регистрации нового расширения

Чтобы создать новое расширение, щёлкни правой кнопкой по пункту *File Extensions* в дереве слева. В дереве появится новый пункт в котором надо ввести имя расширения. В правой половине окна ты должен в свойстве *File* указать имя твоей программы, которая должна запускаться. Ты так же можешь указать дополнительные аргументы программы и иконку, которая будет отображаться для всех файлов этого типа.

Customize the Setup Appearance – в этом разделе находятся пункты настроек, в которых ты можешь указать, как должна выглядеть программа инсталляции.

Dialogs – здесь ты указываешь, какие диалоги должны быть видны во время инсталляции. Выбери этот пункт и ты увидишь окно, как на рисунке 24.3.12. В левом верхнем окне находится дерево, в котором перечислены все доступные диалоги. В левом нижнем окне ты сможешь видеть примерный внешний вид выделенного окна. В правом верхнем окне будут отображаться свойства, которые ты можешь изменить.

Slash Bitmap – если напротив этого пункта стоит галочка, то вначале загрузки будет отображаться рисунок. В свойстве *Splash Bitmap* этого окна ты можешь указать картинку, которая должна отображаться в окне.

Install Welcome – окно приглашения в программу установки. Это окно невозможно отключить и оно обязательно должно присутствовать в программе установке. Ты можешь только изменять свойства окна:

1. Bitmap Image – картинка, которая будет отображаться в окне.
2. Show Copyright – показывать строку Copyright.
3. Copyright Text – текст строки Copyright.

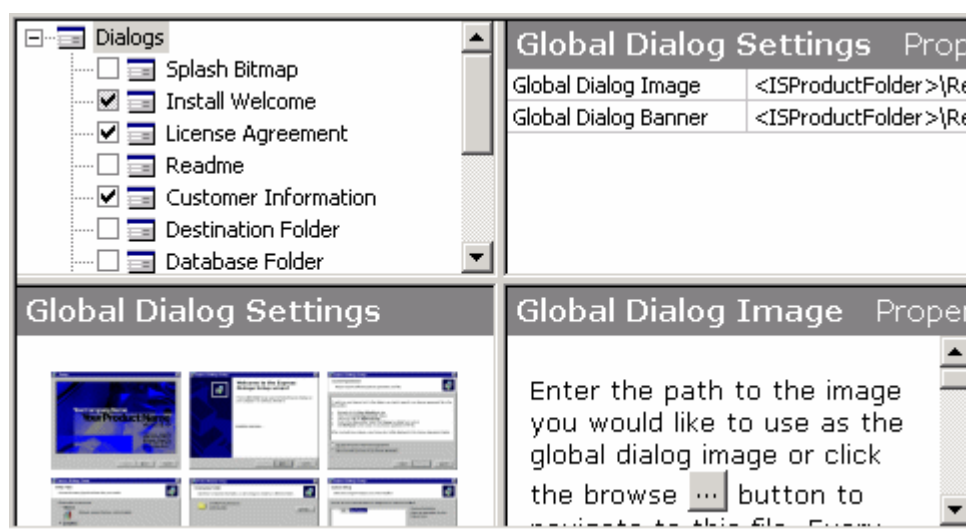


Рис 24.3.12. Окно регистрации нового расширения

License Agreement – окно лицензии. У этого окна два свойства:

1. Banner Bitmap – картинка иконки.
2. License File – файл с текстом лицензии.

Readme – окно с дополнительной текстовой информацией об устанавливаемой программе. У этого окна есть два свойства:

1. Banner Bitmap – картинка иконки.
2. Readme File – файл с текстом.

Customer Information – окно, в котором пользователь должен вводить данные о себе – имя, название компании. У этого окна много свойств и все они достаточно интересны:

1. Banner Bitmap – картинка иконки.
2. Show Serial Number – показывать строку для ввода серийного номера продукта.
3. Serial Number Template – шаблон серийного номера. Здесь ты можешь указать шаблон, по которому будет вводиться серийный номер. Например, у тебя серийный номер состоит из двух чисел (каждое по три цифры) между которыми стоит знак тире. В этом случае шаблон будет выглядеть так «###-###». Если вначале должны быть какие-то обязательные буквы, например «sernum», то шаблон можно записать так: «sernum###-###».
4. Serial Number Validation DLL – здесь можно указать DLL файл, который будет отвечать за проверку правильности введенного серийного номера.
5. Validate Function – здесь указывается функция из библиотеки, которая будет проверять серийный номер.
6. Success Return Value – значение, которое должно возвращаться при положительном результате проверки.
7. Retry Limit – количество попыток, которые пользователь может ввести при регистрации.
8. Show All Users Option – показать возможность выбора, для каких пользователей будет доступна программа. В Windows NT/2000/XP есть возможность давать доступ к программе только тому пользователю, который её устанавливает или всем пользователям компьютера.

Destination Folder – показывать окно выбора папки, в которую надо устанавливать программу.

Database Folder – окно выбора папки, куда будут устанавливаться базы данных.

Setup Type – окно выбора типа установки. В этом окне пользователь сможет выбирать, какая установка ему нужна – типичная, минимальная или выборочная.

Custom Setup – окно, в котором будут отображаться компоненты твоей программы, если пользователь выбрал выборочную установку. Если у этого окна в свойстве Show Change Destination установить true, то пользователь сможет менять директорию установки из этого окна.

Ready to Install – окно, предупреждающее о начале копирования файлов.

Setup Progress – показывать окно хода установки. Если у этого окна в свойстве Show Progress Bar установить false, то пользователь не будет видеть Progress Bar, в котором отображается ход установки.

Setup Complete Success – окно окончания установки. У этого окна целая куча интересных свойств:

1. Banner Bitmap – картинка иконки.
2. Show Launch Program – показывать возможность запуска программы по окончании процесса установки.
3. Program File – имя файла программы, которую надо запустить.
4. Command Line Parameters – параметры командной строки, которые надо передать программе.
5. Show Readme – показывать текстовый файл с дополнительной информацией.
6. Readme file – текстовый файл с дополнительной информацией.

Define Setup Requirements and Actions – в этом разделе мы будем делать последние настройки нашей программы установки.

Requirements – здесь можно указать ОС, тип процессора, количество памяти и др. параметры необходимые твоей программе. Программа установки при старте будет проверять эти параметры и если что-то не совпадает, то установка не пройдет.

Prepare For Release – в этом разделе ты создаёшь программу установки.

Build You Release – здесь ты должен выбрать носитель, на котором будет распространяться твоя программа. В зависимости от типа носителя программа установки может отличаться. Для запуска процесса создания программы установки нужно нажать F7 или выбрать из меню *Build* пункт *Build*.

Test You Release – тут можно протестировать созданную тобой программу установки.

Distribute You Release – в этом разделе ты можешь скопировать программу установки на носитель.

24.4. Как писать и распространять Shareware программы

Сегодня я решил рассказать тебе о том, как писать программы и что именно надо писать. Многие программисты говорят: "Дайте мне точку опоры, и я переверну весь мир". Где-то я это уже слышал, и возможно это сработает, но не в нашем случае. Самое главное при написании программы это не идея. А что же можно назвать главным фактором успеха программного продукта? Вот именно это я постараюсь тебе сегодня рассказать.

Какую программу всё же написать? Для этого не надо далеко идти. Просто подумай, что тебе нужно? Если ты работаешь с базами данных, то займись написанием этих баз. Если ты работаешь с графикой, то напиши простенькую программу, которая будет делать несколько крутых выкрутасов или эффектов. Твоя будущая программа должна удовлетворять нескольким критериям:

1. Программа должна быть нужна тебе. Если она не нужна даже разработчику, то ею не будет пользоваться никто. Если ты дизайнер, то кто, как не ты лучше знает, что нужно настоящему художнику? Вот я, например, в графике не особо смыслю. Я умею её программировать, но ничего не понимаю в художествах. Поэтому я никогда не лезу в эту сферу, хотя мне очень хочется и нравится работать с различными алгоритмами.

2. Программа должна быть уникальна. Если в твоей программе есть какая-то изюминка, то она точно найдёт своего пользователя.

3. Она должна быть простой в использовании. Не стоит загромождать программу лишними действиями. Если ты гений в графических фильтрах, то напиши простенький plug-in к PhotoShop, но не надо писать целый графический редактор ради одного эффекта. Это только усложнит твоё детище и отпугнёт потенциальных пользователей. Программа должна выполнять только самые необходимые действия и содержать как можно меньше лишних функций. Поэтому не стоит встраивать в графический редактор, текстовый процессор, такие большие продукты уже есть и ты всё равно не сможешь с ними конкурировать. Чем проще пользоваться программой, тем больше у неё шансов. Вот ещё один пример: CyD GIF Studio Pro - отличный GIF редактор, но в нём очень много функций, которыми не все пользуются, поэтому многие переходят на более дорогую

программу, но выполняющую только самое необходимое. Это принцип жизни всех американцев и европейцев. Я этого не понимаю, но приходится к нему прислушиваться. Для меня лучше взять программу немного более сильную и дешевле, чем дороже, но менее функциональную. Приведу ещё один пример: Linux - дешёвая и навороченная, поэтому она не получила распространения, а Windows простая и дорогая и стоит на большинстве компьютеров. Парадокс, но это так.

4. Интерфейс - должен быть удобным и симпатичным. Цвета желательно выбирать не сильно яркие, чтобы они не резали глаз. На счёт цвета, я бы вообще-то посоветовал бы использовать только системные, чтобы они изменялись в зависимости от выбранной в компьютере цветовой схемы. Все часто используемые функции, нужно выносить из меню на панель, чтобы можно было получить к ним быстрый доступ. Короче, посмотри на все программы Windows и старайся не сильно выделяться, а то это иногда шокирует и люди не очень охотно используют такие вещи.

5. Документация - она должна быть полной и желательно с конкретными примерами. Пользователи любят, когда описаны конкретные действия при работе с программой. Не надо ограничиваться простым описанием возможностей. Постарайтесь дать как можно больше информации и конкретных примеров.

Этот список можно продолжать бесконечно, но я останавлиюсь. В течении всей статьи я ещё вернусь к уже описанным требованиям и укажу ещё некоторые вещи.

Прежде чем писать программу, поставьте перед собой конкретные цели. Главное, чтобы твоя цель была достижима и как можно в кратчайшие сроки. Не надо ставить перед собой задачу написать текстовый редактор, потому что их уже достаточно, ты не напишешь лучше чем Microsoft, но даже если это и так, то пока ты будешь писать, Microsoft выпустит уже десять версий. Ты не сможешь угнаться за гигантами. Поэтому делай свою цель менее фантастической.

Я уже говорил, что желательно сделать программу маленькой. Для этого есть несколько причин:

1. Пользователи не любят слишком навороченные программы. Об этом я уже сказал и привёл несколько примеров.

2. Первая версия твоей программы должна быть готова максимум через месяц. Если ты затянешь написание программы на год, то через этот период может пропасть необходимость в ней или кто-то уже реализует твою идею, и ты потратишь время зря.

Именно поэтому ставь реальные и быстро достижимые цели.

После того, как ты поставил перед собой цель, начинай писать программу. Во время написания нужно чётко придерживаться поставленной цели. Не в коем случае нельзя обращать внимание на мысли: "надо бы добавить вот такую возможность или вот такую". Если ты хоть раз обратишь на неё внимание и начнёшь выполнять, то тебя затянет. Ты будешь вечно добавлять новые возможности и так и не выпустишь свою программу в свет. А если программа и появиться, то в недоработанном варианте, потому что ты просто выбросишь промежуточный вариант, в надежде, что в ближайшее время будет полный, а сам снова уйдёшь в постоянные доработки. Лучше записывай появляющиеся мысли на бумагу и засовывай подальше в карман.

Когда ты закончишь выполнять первоначальный план, то протестируй готовую программу и начинай заниматься раскруткой (об этом я расскажу чуть ниже). На своём сайте можешь указать всё, что у тебя накопилось на бумажках (то что ты собираешься добавить) и указать, что это можно увидеть в следующей версии программы. Потенциального пользователя это заинтересует, и он запомнит ссылку на твою страничку среди "Избранных", чтобы вернуться за новой версией с обещанными возможностями.

Пока все работают с твоей первой версией программы, ты достаёшь из кармана все свои мысли и начинаешь их выполнять, оформляя новую версию.

Постарайся хорошенечко тестировать свою работу. Для этого можно набрать группу тестеров среди российских пользователей. Для этого раздай им бесплатные лицензии, они всё равно не заплатят (в России ещё не привыкли платить за программы). Чем больше будет ошибок в твоей программе, тем хуже будут к тебе относиться. Я в своё время очень сильно запустил своё детище и ко мне потеряли интерес все пользователи. После этого мне понадобилось два дня, чтобы исправить все ошибки и пол года, чтобы снова завоевать доверие. Так что выводы делай сам.

Я постарался дать тебе самые основные (на мой взгляд) правила написания программы. Не думай, что что-то из этого можно не учитывать, в нашем деле важно всё, особенно при работе с иностранными клиентами. Так что выучи наизусть всё, что я тебе рассказал.

Где размещать

Твоя программа готова и теперь её надо где-то разместить, чтобы потенциальные клиенты могли скачать твоё творение. Для этого опять же полно серверов бесплатно предоставляющих место для твоих страничек. Конечно же, это не самый лучший шаг, потому что в цивилизованных странах не очень любят пользоваться такими страницами. Но для тебя этого будет достаточно. Ты всё равно сможешь заработать свои кровные даже при использовании бесплатного хостинга.

Советую выбирать тебе сервер за границей, чтобы ничего в его адресе не указывало на происхождение твоей программы. Желательно, чтобы никто не знал, что ты русский. Твой почтовый ящик тоже должен быть не в зоне RU. И я надеюсь, что твоя страничка сделана без ошибок в английском языке. Если ошибки есть, то твои шансы заработать падают на 99%.

Если не можешь писать на английском, то сделай перевод с помощью любого электронного переводчика и отправляйся сюда: <http://members.home.net/djosborne1/>. Здесь тебе за небольшую плату исправят все ошибки. Воспользуйся этим, не жалея сотню долларов. Потрать их и ты уже через месяц сможешь ощутить живую прибыль, если твоя программа хоть чего-то стоит.

Я немного отклонился от темы. Вернёмся к нашим серверам. Вот тебе несколько адресов, которые тебе помогут.

<http://www.crosswinds.net/> - хороший сервер, но до него очень трудно добиться, чтобы закачать файлы. Скорость загрузки ужасная, меньше 1 кило в секунду. Зато скорость скачивания моментальная.

<http://www.webjump.com/> - добиться легко, скорость загрузки хорошая, но вешают большой банер. Есть специальная поддержка программистов распространяющих Shareware программы, таких как ты.

<http://www.freervers.com/> - Дают мало места. Я им не пользовался и больше ничего сказать не могу.

<http://www.tripod.com/> - говорят хороший, сам не пробовал, поэтому утверждать не буду.

<http://www.virtualave.net/> - третий сорт не уродство. Полно возможностей и достаточно много места.

Как рекламировать

После того, как ты выложил свою программу в глобальной сети, её надо начать рекламировать. Я тебе советую зайти на следующий адрес: <http://download.cnet.com/>. Зайди туда и жми на ссылку *Submint file*. Перед тобой появится простая форма для описания программы, которую надо заполнить (Рисунок 24.4.1).

CNET.com - Downloads - Submissions - Submitting a new program or update - Microsoft Internet Explorer

Address http://download.cnet.com/downloads/submissions/0-1562081.htm?tag=ds

Completed forms will be reviewed by our staff. CNET Download.com reserves the right to decline submissions for any reason.

Title

Version

Status
☐ New submission
☐ Update of a title already on CNET Download.com

File size
 (in bytes)

Category
 Select one category:

License

Code license type (Linux/Unix only)

Price
 (If you are submitting freeware, leave this field blank.)
☐ Shareware registration fee ☐ Full version retail price

Release date
 mm dd yyyy

Download link(s)
 Please list one URL per line, followed by a hard return. The URL must link to a downloadable file.
 (Example: <http://www.cnet.com/downloads/0-1562081.htm>)

Formoset

Рис 24.4.1. Форма регистрации программы на www.download.com

Заполни эту форму сведениями о своей программе и через некоторое время на этом сервере будет вывешено её описание. На момент написания книги, время обновления сведений сильно увеличилось (до 2-х месяцев), а моментальное обновление доступно за деньги. Если есть лишние деньги, то можешь потратить их на обновление информации. Я уже говорил, что скорость в нашем деле очень важна, но и про качество забывать не надо.

Только на download.com твою программу будут скачивать по 100 копий в день. Только не думай, что на этом сервере будет располагаться твоя программа. Её будут качать с твоей страницы, а download.com хранит только описание и ссылку на твой файл.

Если ты всё правильно сделал и программа оказалась на download.com, то через некоторое время другие архивы программ будут сами делать ссылку на твоё детище. Так что можно не напрягать свои мозги регистрацией на всех серверах подряд. Тем более, что только download.com даёт ощутимый трафик, остальные смогут дать только 5-10 скачиваний в день (по моему личному наблюдению).

Как получать деньги

Твоя программа готова, выложена в интернете и её качают. Возможно, что её скоро захотят купить. Ты должен заранее подготовиться к этому моменту. Сам ты не сможешь получать деньги от клиентов. Как же тогда решить эту проблему? Есть несколько серверов, которые предоставляют услуги по оплате счетов через интернет, почту и даже телефон. Ты просто регистрируешься у них, и они сами будут получать деньги с твоих клиентов. Берут они за это немного, всего 9-15 процентов. Не надо жадничать, это действительно очень мало. Воспользуйся этими услугами.

Я могу посоветовать тебе использовать www.regnow.com. Он очень удобен, а главное прост в использовании. После регистрации тебе приходит письмо, в котором находятся логин и пароль доступа. Когда ты войдёшь в свою зону, ты сможешь легко разобраться с работой сервера. Просто попробуй, там всё понятно, если ты хоть немного понимаешь английский язык или умеешь пользоваться переводчиками.

24.5. Создание программ маленького размера

Очень часто нам приходится задумываться о написании программ маленького размера. Если при написании офисных приложений мы можем забыть про оптимизацию размера, то для программ постоянно находящихся в памяти размер кода критичен. Допустим, что тебе надо написать маленькую утилиту – будильник. Эта программа должна постоянно находиться в памяти и следить за временем. Я думаю, что будет неприятно, если программа будет занимать мегабайт в оперативной памяти на какой-то будильник.

Программы созданные Delphi получаются достаточно большого размера. С чем это связано? А с тем, что Delphi является объектным языком. В нем каждый элемент выглядит как объект, который обладает своими свойствами, методами и событиями. Любой объект вполне автономен и может работать без твоего ведома. Это значит, что тебе нужно только подключить его к своей форме, изменить нужные свойства и все готово. После этого все будет работать без какого-либо внешнего вмешательства.

Но в этом есть и свои недостатки. В объектах реализовано большинство возможных действий, которые ты можешь производить с ним. Но реально, в любой программе мы пользуемся двумя-тремя из всех этих свойств. Все остальное для программы лишний груз, который никому не нужен.

Но как же тогда создать компактный код, чтобы мой будильник занимал минимум места на винте и как можно меньше занимал памяти? У тебя есть несколько вариантов:

1. Не использовать визуальную библиотеку VCL (для любителей Visual C++ это библиотека MFC), которая упрощает программирование. В этом случае придется все делать вручную и работать только с WinAPI. Код в этом случае получается очень маленьким и быстрым. Но тут ты лишаешься визуальности и можешь ощутить все неудобства программирования на чистом WinAPI.

2. Сжимать программы с помощью компрессоров. Такой код сжимается в несколько раз и программа с использованием VCL может превратиться из 300 кило в 30-50. Главное преимущество тут в том, что ты не лишаешься возможностей объектного программирования и можешь спокойно забыть про неудобства WinAPI.

Здесь я постараюсь, как можно подробнее рассмотреть оба этих метода.

Программирование на WinAPI

Если ты хочешь создать программу маленького размера, то ты должен забыть про все удобства. Ты не сможешь подключать визуальные формы или другие примочки написанные фирмой Barland для упрощения жизни программиста. Только API функции и ничего больше.

Для того, чтобы создать маленькую программу в Delphi, нужно создать новый проект и зайти в менеджер проектов (меню *View->Project Manager*). Здесь нужно удалить все формы, чтобы остался только файл самого проекта (по умолчанию его имя Project1.exe). Никаких модулей в проекте не должно быть.

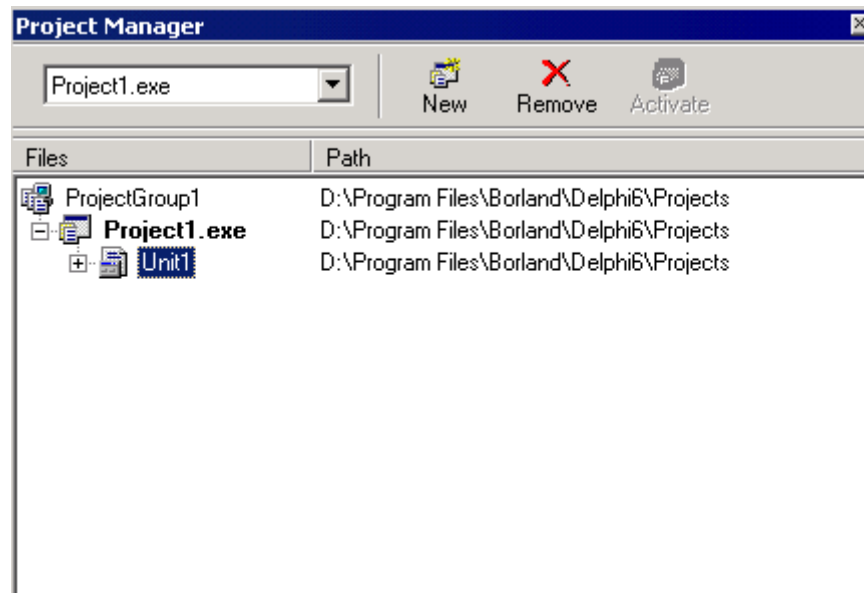


Рисунок 24.5.1. Project Manager

Теперь щелкни правой кнопкой по имени проекта и выбери из появившегося меню пункт *View Source* (или из главного меню *Project* выбери пункт меню *View Source*). В редакторе кода откроется для редактирования файл проекта *Project1.dpr*. Если ты уже удалил все модули, то его содержимое должно быть таким:

```

program Project1;

uses
  Forms;

{$R *.res}

begin
  Application.Initialize;
  Application.Run;
end.

```

Я удалил все визуальные формы и теперь могу скомпилировать абсолютно пустой проект. Я решил попробовать сделать это. После компиляции я выбрал из меню *Project* пункт *Information for Project1*. Передо мной появилось окно с информацией о проекте. Моё окно ты можешь увидеть на рисунке 24.5.2.

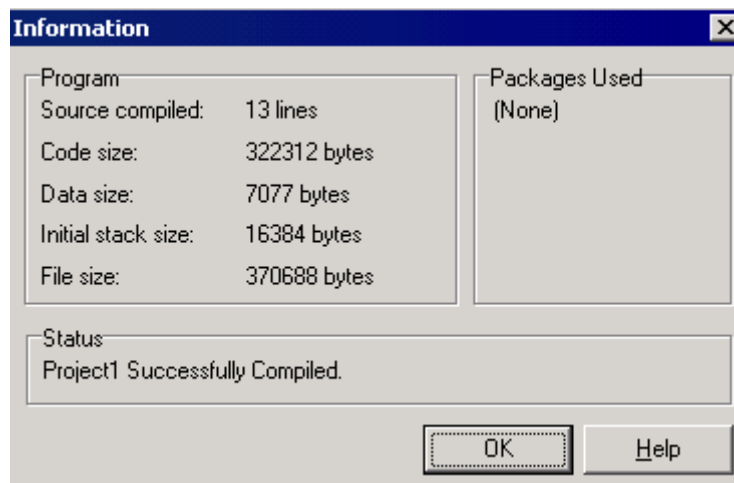


Рисунок 24.5.2. Окно информации о проекте

В правой части окна должны быть описаны используемые пакеты. Мы все удалили, значит там точно должна красоваться надпись *None*. А вот с левой стороны должна быть описана информация о скомпилированном коде. Самая последняя строка показывает размер файла и у меня он равен 370688 байт. Ничего себе пельмень!!! Мы же ничего еще не писали. Откуда же тогда такой большой код?

Давай разберем, что осталось в нашем проекте, чтобы обрезать все, что еще не обрезано. Сразу обрати внимание, что в разделе **uses** подключен модуль *Forms*. Это объектный модуль, написанный дядей Борманом, а значит, его использовать нельзя, потому что именно он увеличивает размер нашей программы. Между **begin** и **end** используется объект *Application*. Его тоже использовать нельзя, потому что это объект.

Все накладки большого кода, даже у пустой программы, как раз и связаны с объектом *Application*, который объявлен в модуле *Forms*. Хотя мы используем только два метода *Initialize* и *Run*, при компиляции в запускной файл попадает весь объект *TApplication*, а он состоит из сотен, а может и тысяч строчек кода.

Чтобы избавиться от накладных расходов нужно заменить модуль *Forms* на *Windows*. Второй – это модуль который описывает только WinAPI и не связан с объектами Delphi. Его подключение обязательно, иначе мы не сможем вызвать ни одной функции из набора WinAPI. А между **begin** и **end** вообще все можно удалять. Самый минимальный код проги будет выглядеть так:

```

program Project1;

uses Windows;

Begin

end.

```

Снова откомпилируй проект. Зайди в окно информации и посмотри на размер получившегося файла. У меня получилось 8192 байта (смотри рисунок 24.5.3). Вот это уже по человечески.

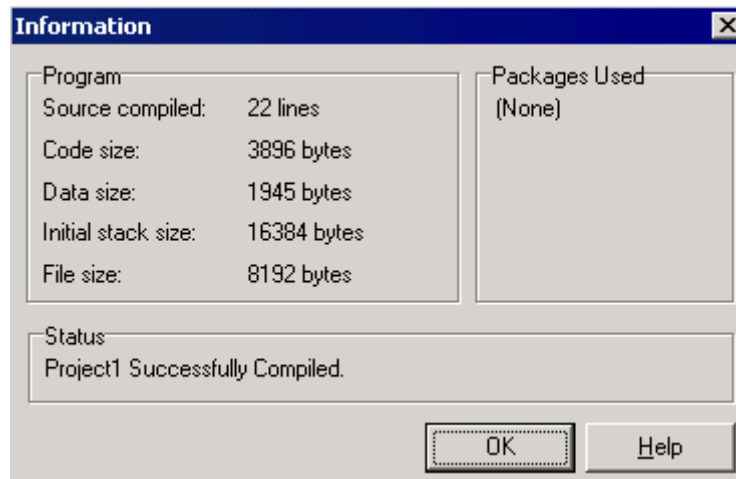


Рисунок 24.5.3. Информации о проекте

Заготовка минимальной программы с использованием WinAPI готова. Теперь ты можешь смело добавлять свой код. Мне только надо объяснить тебе какие модули можно подключать к своему проекту в раздел **uses**. Тут все очень просто и не займет много времени.

Если при установке **Delphi** ты не отключал копирование исходников библиотек, то перейди в директорию, куда ты установил Delphi. Здесь перейди в папку *Source*, затем в *Rtl* и наконец *Win*. Вот здесь расположены исходники модулей, в которых описаны все API функции Windows. Именно эти модули ты можешь подключать к своим проектам, если хочешь получить маленький код. Если ты подключишь что-то другое, то я уже не гарантирую тебе минимум размера твоей программы (хотя есть и исключения).

Сразу же пример. Если ты хочешь, чтобы в твоей программе были возможности работы с сетью, то тебе нужно подключить к нему библиотеку сокетов. Среди модулей WinAPI есть файл с именем *winsock.pas*. Значит, ты должен в раздел **uses** написать *winsock* (расширение писать не надо) и твоя программа сможет работать с сетью.

Пока что я описал минимальный проект, в который можно добавлять свой код. Но код, который ты вставишь, выполниться один раз и программа выгрузится из памяти. А что если тебе надо, чтобы твоя программа постоянно висела в памяти и что-то делала? Для этого используй следующий шаблон для своих программ:

```

program Project1;

uses Windows;

var
  Msg: TMsg;
begin

  //Сюда можешь добавлять свой код

  // Дальше идет код, который заставит прогу висеть в
  // памяти вечно и не будет сильно грузить систему.
  while GetMessage( Msg, HInstance, 0, 0) do
    begin
      TranslateMessage(msg);
      DispatchMessage(msg);
    end;
end.

```

Сжатие программ

Если тебя не устраивает программирование на чистом WinAPI, то твоим лучшим другом должна стать какая-нибудь программа для сжатия размера файлов. Я очень люблю ASPack, которую ты можешь забрать по адресу www.cydsoft.com/vr-online/download.htm или на компакт диске в директории *Programs* (файл установки называется ASPack.exe). Она прекрасно сжимает запускные файлы .exe и динамические библиотеки .dll.

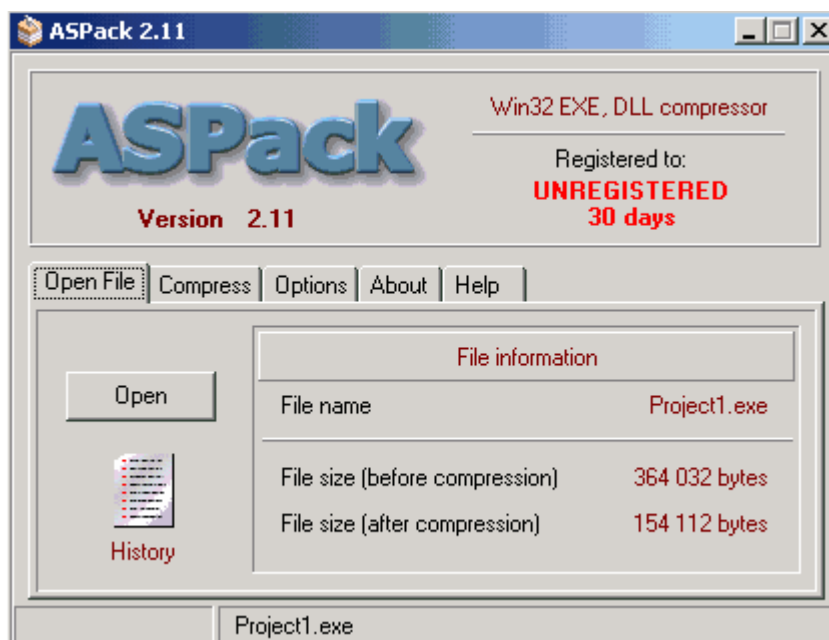


Рисунок 24.5.4. Главное окно ASPack

Я не буду объяснять установку ASPack, потому что там абсолютно ничего сложного нет. Только одно нажатие на кнопке *Next* и все готово. Теперь запусти установленную программу и ты увидишь окно, как на рисунке 24.5.4. Главное окно состоит из нескольких закладок:

1. Open File.
2. Compress.
3. Options.
4. About.
5. Help.

На закладке *Open File* есть только одна кнопка - *Open*. Нажми на нее и выбери файл, который ты хочешь сжать. Как только ты выберешь файл, программа перескочит на закладку *Compress* и начнет сжатие (рисунок 24.5.5).

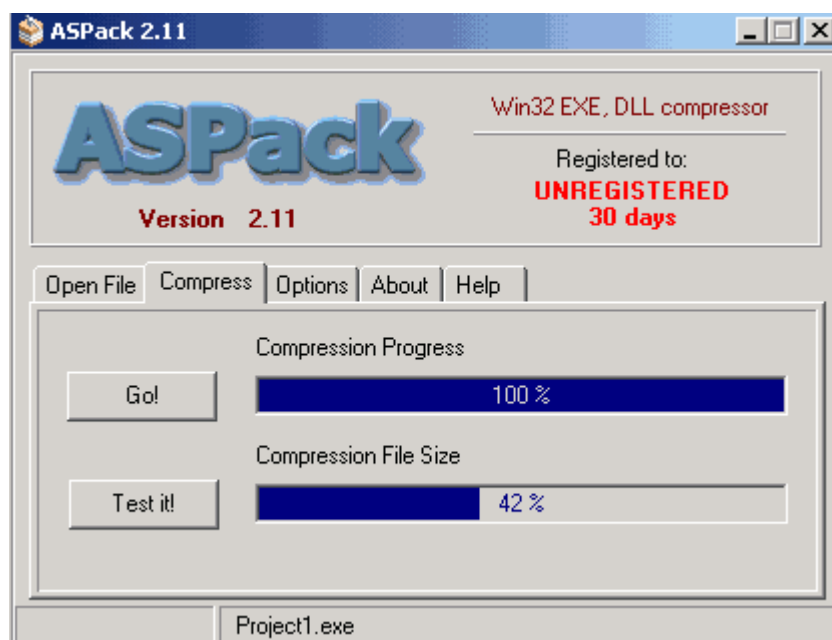


Рисунок 24.5.5. Сжатие файла

Сжатый файл сразу перезаписывает существующий, а старая несжатая версия сохраняется на всякий случай под тем же именем, только с расширением *bak*. Настроек у ASPack не так уж много (рисунок 24.5.6), и с ними ты сможешь разобраться без меня. Уж лучше я расскажу тебе, как все работает.



Рисунок 24.5.6. Настройки ASPack

Давай разберемся, как работает сжатие. Сначала весь код программы сжимается архиватором. Если ты думаешь, что он какой-то навороченный, то сильно ошибаешься. Для сжатия используется простой архиватор типа WinZIP, только оптимизированный для сжатия байт-кода. После этого, в конец сжатого кода добавляется код разархиватора,

который будет все это дело разжимать обратно. И в самом конце ASPack изменяет заголовок запускового файла так, чтобы при старте, сначала запускался разархиватор.

Теперь, когда ты запустишь сжатую программу, сначала запустится разархиватор, который разожмет байт-код программы и аккуратно выложит его в памяти машины. Как только этот процесс пройдет, разархиватор передаст управление твоей программе.

Некоторые считают, что из-за расходов на распаковку программа будет работать медленней!!! Я бы не сказал, что ты это заметишь. Даже если и будут какие-то потери, то они будут неощутимы. Это потому что архивация хорошо оптимизирована под байт-код. К тому же, размер программы уменьшится, а значит, она загрузится в память намного быстрее. В результате потери на скорости неощутимы даже с секундомером.

Конечно же, программирование на WinAPI слишком сложный процесс, но программы получаются очень маленького размера, и никакой архиватор не сможет сжать байт-код до такой степени. А если еще и сжать программу написанную на WinAPI, то ее размеры будут меньше некуда.

При нормальном программировании с использованием всех навороченных возможностей типа визуальности и объектного программирования код получается большим, но его можно сжать на 60-70% спец архиватором. К тому же такой код писать намного легче и быстрее.

Ещё одно «за» использование сжатие - сжатый код труднее взломать, потому что не каждый disassembler сможет прочитать упакованные команды. Так что, помимо уменьшения размера ты получаешь защиту способную отпугнуть большинство взломщиков. Конечно же, профессионала не отпугнешь даже этим, но взломщик средней руки не будет мучиться со сжатым байт-кодом.

24.6. Оптимизация скорости выполнения программы

Вся наша жизнь это борьба с тормозами и нехваткой времени. Каждый день мы тратим по несколько часов на оптимизацию. Каждый из нас старается оптимизировать все, что попадает под руку. А ты уверен, что ты это делаешь правильно? Может быть, есть возможность что-то сделать еще лучше?

Я понимаю, что все сейчас зажрались и выполняют свои обязанности лениво и не принужденно. Лично я до такой степени привык, что за меня все делает железный друг, что даже забыл, как выглядит шариковая ручка. Недавно мне пришлось писать заявление на отпуск на простой бумаге, так я забыл, как пишется буква "ю". Пришлось подглядывать, как она выглядит на клавиатуре :).

Даже для того, чтобы написать текст из двух строк, мы включаем свой компьютер и загружаем MS Word, тратя на это драгоценное время. А может легче было бы написать этот текст вручную? Я тебя понимаю - не солидно!!!

Программисты - так это вообще полное бесстыдство, тра-та-та (далее все вырезано цензурой). Если они считают, что раз их творение (в виде исходника) никто не увидит, то можно писать что угодно? Так это они ошибаются. С этой точки зрения программы с открытым исходником в большом преимуществе, потому что они намного чище и быстрее. Создавая код, мы ленимся его оптимизировать не только с точки зрения размера, но и с точки зрения скорости. Глядя на такие творения, хочется выругаться матом, так ведь опять цензура обрежет.

Хакеры далеко не ушли. Если раньше, глядя на программиста или хакера создавался образ прокуренного, заросшего и немытого молодого человека, то сейчас это цифровое существо, залитое пивом Балтика по самые уши за которого все выполняют машины. Тебе медсестра в поликлинике не говорила, что у тебя вместо крови одно только пиво льется? Не, я ничего против пива не имею, я и сам его люблю.

Все это деградация по методу MS!!! Мы берем в руки мышку и начинаем тыкать ей где попало, забывая про клавиатуру и горячие клавиши. Я считаю, что надо бороться с этим. В последнее время меня самого посещает такая лень, что я убираю клавиатуру, запускаю экранную клавиатуру и начинаю работать только мышкой. Осталось только покрыть мое тело шерстью и посадить в клетку к таким же ленивым шимпанзе.

Не надо тратить большие деньги на апгрейд компьютера!!! Начните лучше апгрейд с себя. Давайте, оптимизируем свою работу и то, что мы делаем.

В своем зародыше, эта часть книги задумывалась как рассказ об оптимизации кода программ, но в последствии я перенёс в неё свой труд, который можно найти и в инете на моём сайте, потому что оптимизировать надо всё. Я буду говорить про теорию оптимизации, а ее законы действуют везде. По тем же законам ты можешь оптимизировать свой распорядок дня, чтобы успевать все сделать, и свою ОС, чтобы она работала быстрее. Но основа все же будет относиться к коду программ.

Как всегда я постараюсь давать как можно больше реальных примеров, чтобы ты смог убедиться в том, что тебе не вешают очередную лапшу на уши, и ты мог применить все сказанное на практике.

Начну я с законов, которые работают не только в программировании, но и в реальной жизни. Ну а напоследок я оставлю только то, что может пригодиться только при оптимизации кода.

ЗАКОН №1

Оптимизировать можно все. Даже там, где тебе кажется, что все и так работает быстро, можно сделать еще быстрее.

Это действительно так. И этот закон очень сильно проявляется в кодинге. Идеального кода не существует. Даже простую операцию сложения 2+2, тоже можно оптимизировать. Чтобы достичь максимального результата, нужно действовать последовательно и желательно в том порядке, в котором я буду описывать.

ЗАКОН №2

Первое с чего нужно начинать - это с поиска самых слабых и тормознутых мест. Зачем начинать оптимизацию с того, что и так работает достаточно быстро. Если ты будешь оптимизировать сильные места, то можешь нарваться на неожиданные конфликты.

Тут же я вспоминаю пример из своей собственной жизни. Где-то в 1995-96-м году меня посетила одна невероятная идея - написать собственную игру в стиле Doom. Я не собирался ее делать коммерческой, а хотел только потренировать свои мозги на сообразительность. Четыре месяца невероятного труда, и нечто похожее на движок уже было готово. Я создал один голый уровень, по которому можно было перемещаться, и с чувством гордости побежал по коридорам.

Никаких монстров, дверей и атрибутки на нем не было, а тормоза ощущались достаточно значительные. Тут я представил себе, что будет, если добавить монстров и атрибуты, да еще и наделить все это AI.... Вот тут чувство достоинства поникло. Кому нужен движок, который при разрешении 320x200 (тогда это было круто) в голом виде тормозит со страшной силой? Вот именно....

Понятное дело, что мой виртуальный мир нужно было оптимизировать. Целый месяц я бился над кодом и вылизывал каждый оператор моего движка. Результат - мир стал прорисовываться на 10% быстрее, но тормоза не исчезли. И тут я увидел самое слабое место - вывод на экран. Мой движок просчитывал сцены достаточно быстро, а пробойной тормозов был именно вывод изображения. Тогда еще не было шины AGP и я

использовал простую PCI видюху от S3 с 1 мегом памяти. Пару часов колдовства, и я выжал из PCI все возможное. Откомпилировав движок, я снова загрузился в свой виртуальный мир. Одно нажатие клавиши вперед и я очутился у противоположной стены. Никаких тормозов, сумасшедшая скорость просчета и моментальный вывод на экран.

Как видишь, моя ошибка была в том, что я неправильно определил слабое место своего движка. Я месяц потратил на оптимизацию математики и что в результате? Мизерные 10% прироста в производительности. Но когда я реально нашел слабое звено, то смог повысить производительность в несколько раз.

Слабые места компьютера

Меня поражают люди, которые гонятся за мегагерцами процессора и сидят на доисторической видюхе от S3, винте на 5400 оборотов и с 32 мегами памяти. Посмотри в корпус своего железного друга и оцени его содержимое. Если ты увидел, что памяти у тебя не более 32 мегов, то встань и громко произнеси: "Уважаемый DIMM, команда выбрала вас. Вы сегодня самое слабое звено и должны покинуть мой компьютер" :). После этого, покупаешь себе 128, а лучше 256 мегов памяти и наслаждаешься ускорением работы Delphi, Photoshop и других тяжелых программ.

В данном случае, наращивание мегагерцов у процессора даст более маленький прирост в скорости. Если ты используешь тяжелые приложения при нехватке памяти, то процессор начинает тратить слишком много времени на загрузку и выгрузку данных. Ну а если в твоём железном друге достаточно оперативки, то процессор уже занимается только расчетами и не расходуется по лишним загрузкам-выгрузкам.

То же самое с видюхой. Если она у тебя слабенькая, то процессор будет просчитывать сцены быстрее, чем они будут выводиться на экран. А это грозит простоями и минимальным приростом производительности.

ЗАКОН №3

Следующим шагом ты должен разобрать все операции по косточкам и выяснить, где происходят регулярно повторяющиеся операции. Начинать оптимизацию нужно именно с них.

Опять начнем рассмотрение этого закона с кодирования. Допустим, что у тебя есть следующий код (я приведу просто логику, а не реальную программу):

1. $A := A * 2;$
2. $B := 1;$
3. $X := X + B;$
4. $B := B + 1;$
5. Если $B < 100$ то перейти на шаг 3;

Любой программист скажет, что здесь слабым местом является первая строка, потому что там используется умножение. Это действительно так. Умножение всегда выполняется дольше, и если заменить его на сложение ($A := A + A$) или еще лучше на сдвиг, то ты выиграешь пару тактов процессорного времени. Но это только пару тактов и для процессора это будет незаметно.

Теперь посмотри еще раз на наш код. Больше ничего не видишь? А я вижу. В этом коде используется цикл: "Пока $B < 100$ будет выполняться операция $X := X + B$ ". Это значит, что процессору придется выполнить 100 переходов с шага №5 на шаг №3. А это уже не мало. Как же можно здесь что-то оптимизировать? Очень легко. Здесь у нас внутри цикла выполняется две строки 3 и 4. А что если мы внутри цикла размножим их 2 раза:

2. $B := 1;$

3. $X:=X+B$;

4. $B:=B+1$;

5. $X:=X+B$;

6. $B:=B+1$;

7. Если $B < 50$ то перейти на шаг 3;

Здесь я разложил цикл на более маленький. Вторую и третью операцию я повторил два раза. Это значит, что за один проход цикла я выполню два раза строки 2 и 3 и только после этого перейду на строку 1, чтобы повторить операцию. Такой цикл уже нужно повторить только 50 раз (потому что за один раз выполняется два действия). Это значит, что я сэкономил 50 операций переходов. Нехило? А это уже несколько сотен тактов процессорного времени.

А что если внутри цикла написать строки 2 и 3 десять раз. Это значит, что за один проход цикла строки 2 и 3 будут вычисляться 10 раз и мне понадобится повторить такой цикл только 10 раз, чтобы получить в результате 100. А это уже экономия 90 операций переходов.

Недостаток этого подхода - увеличился код моей программы, зато повысилась скорость и очень значительно. Этот подход очень хорош, но им не стоит злоупотреблять. С одной стороны увеличивается скорость, а с другой увеличивается размер. А большой размер это враг любой программы.

В жизни таких примеров намного больше. Любую циклическую операцию можно оптимизировать. Хочешь пример? Пожалуйста. Допустим у твоего провайдера интернета есть несколько телефонов доступа. Ты каждый день перезваниваешь на каждый из них в ожидании найти свободный. Начинаящий тут же скажет, что провайдер обязан оптимизировать свои пулы модемов в один, чтобы не надо было трезвонить по всем номерам сразу. Но продвинутый должен знать, что не у каждого хорошая связь с любой станцией города. Поэтому провайдеры держат пулы на разных станциях, чтобы ты мог выбрать тот с которым у тебя лучший коннект. Тогда что же оптимизировать? Очень просто - поставь прогу дозвонщик (таких сейчас полно в интернете) которая сама будет перебирать номера телефонов.

А теперь другой пример - тебе досталась карточка на 1 час какого-то нового провайдера. Заносить ее в прогу-дозвон не имеет смысла, потому что ты можешь больше никогда не позвонить ему. Из-за этой одноразовой операции тебе придется перенастраивать свой дозвонщик на нового провайдера и потом обратно, а выигрыш практически нулевой, потому что пока ты меняешь настройки уже можно было дозвониться.

ЗАКОН №4.

(этот закон как бы расширение предыдущего). Оптимизировать одноразовые операции - это только потеря времени. Сто раз подумай, прежде чем начать мучиться с редкими операциями.

Пол годика назад я прочитал рассказ в интернете "Записки жены программиста" (<http://www.exler.ru/novels/wife.htm>). Очень даже не кислый и жизненный рассказ. Когда я его читал, у меня было ощущение, что его написала моя жена :). Слава "Красной Шапочке", что она на такую подлость не способна. Так вот там была такая ситуация:

"Очаровашка выходит замуж за программера и им надо разослать приглашения на свадьбу. Вместо того, чтобы набрать их на печатной машинке, программер кричит, что он крутой и пишет специальную прогу. Написание проги заняло один день и столько же ее отладка".

Главная ошибка - неправильная оптимизация своего труда. Легче набрать шаблон в любом текстовом редакторе и потом только менять фамилии приглашенных на этот траурный день (это я сужу по себе :)). Но даже если нет текстового редактора, писать прогу действительно нет смысла. Затраты большие а пользоваться ей будешь только один раз. Здесь действительно легче будет даже набрать на печатной машинке.

Получается, что одноразовые операции оптимизировать нет смысла. Затраты тут себя не окупают, поэтому не стоит тратить свои нервы на этот бессмысленный труд.

В самом начале этой части я раскритиковал тебя как лентяя, который ленится что-то делать. Так вот именно здесь ты можешь проявлять все свои ленивые качества в полном объеме. В данном случае крутым считается не тот, кто целый день промучился и ничего не добился, а тот, кто выполнил свою работу наиболее эффективно. И эти две вещи путать нельзя.

ЗАКОН №5

Нужно знать внутренности компьютера и принципы его работы. Чем лучше ты знаешь, каким образом компьютер будет выполнять твой код, тем лучше ты сможешь его оптимизировать.

Это последний закон, который я хотел бы тебе сказать, и относится он только к кодингу. Тут трудно привести полный набор готовых решений, но некоторые приемы я постараюсь описать.

1. Старайся поменьше использовать вычисления с плавающей запятой. Любые операции с целыми числами выполняются в несколько раз быстрее.

2. Операции умножения и тем более деления так же выполняются достаточно долго. Если тебе нужно умножить како-то число на 3, то для проца будет легче три раза сложить одно и то же число, выполнить умножение.

А как же тогда экономить на делении? Вот тут нужно знать математику. У проца есть такая вещь как сдвиг. Ты должен знать, что процессор думает с помощью нулей и единиц. Это значит, что числа в нем хранятся в двоичной системе. Например, число 198 для процессора будет выглядеть как 11000110. Теперь посмотрим, как работают операции сдвига.

Сдвиг вправо: Если сдвинуть число 11000110 вправо на одну позицию, то последняя цифра исчезнет и останется только 1100011. Теперь загони это число в калькулятор и переведи его в десятичную систему. Твой результат должен быть 99. Как видишь - это ровно половина числа 198. Вывод - когда ты сдвигаешь число вправо на одну позицию, то ты делишь его на 2.

Сдвиг влево: Возьмем то же самое число 11000110. Если сдвинуть его влево на одну позицию, то с правой стороны освободится место, которое сразу заполняется нулем 110001100. Теперь переведи это число в десятичную систему. У тебя должно получится 396. Ни что не напоминает тебе? Это 198 умноженное на 2.

Вывод: Когда ты сдвигаешь число вправо, то ты делишь его на 2. Когда сдвигаешь влево, то умножаешь его на 2. Так что используй это свойство сдвигов везде, где это возможно, потому что сдвиги работают в десятки раз быстрее.

3. Когда создаешь процедуры, то не пичкай их большим количеством входных параметров. Перед каждым вызовом процедуры ее параметры поднимаются в специальную область памяти (стек), а после входа изымаются оттуда. Чем больше параметров, тем больше расходы на общение со стеком.

Тут же нужно сказать, что ты должен быть аккуратным и с самими параметрами. Не вздумай пересылать процедурам переменные, которые могут содержать данные большого объема в чистом виде. Лучше передай адрес ячейки памяти, где хранятся данные, а внутри процедуры работай с этим адресом. Ты представь себе ситуацию, когда тебе нужно передать текст размером одного тома "Войны и мир".... Перед входом в процедуру, программа попытается вогнать все это в стек. Если ты не схватишь его переполнение, то тормоза ощутишь значительные.

4. В самых критичных моментах (как, например вывод на экран) можно воспользоваться языком Assembler. Даже встроенный в Delphi или C++ ассемблер на много быстрее. Ну а если скорость в каком-то месте уж слишком критична, то код ассемблера можно вынести в отдельный модуль. Там его нужно откомпилировать с помощью компиляторов TASM или MASM, и подключить к своей проге.

Ассемблер достаточно быстрая и компактная вещь, но писать достаточно большой проект только на нем это чистый геморрой. Поэтому я не советую им увлекаться, и используй его только в самых критичных для скорости местах.

Если ты прочитал все внимательно, то можешь считать, что с основами оптимизации ты уже знаком. Но это только основы и тут есть куда развиваться. Я бы мог рассказать больше, но не вижу особого смысла, потому что оптимизация - это процесс творческий и в каждой отдельной ситуации можно подойти с разных сторон. И все же, те законы, которые я сегодня описал, действуют в 99,9% случаев.

Если ты хочешь познать теорию оптимизации более глубоко, то тебе нужно больше изучать принципы работы процессора и операционных систем. Главное, что законы ты уже знаешь, а остальное пройдет со временем и навыками.