**Экранная заставка**

**Постановка задачи**

Прежде всего, экранная заставка занимает весь экран, не имеет системной строки, главного меню и обрамления главного окна. Да и окно у экранной заставки, как правило, одно. Еще одно различие - файл с экранной заставкой имеет расширение *\*.scr* (от англ. *screen* - экран) а не *\*.exe*, как программа. Тем не менее, экранная заставка - та же программа.

Подумаем, каким образом мы можем реализовать такую заставку в виде часов. Ответ напрашивается сам собой: установить в центре экрана обычную панель, на которой в свойстве Caption выводить текущее время. Однако, это скучно. Пусть эта панель еще и перемещается по экрану по случайной траектории! Чтобы реализовать такой проект, нам придется изучить пару вещей: системный таймер TTimer и функцию-генератор случайных чисел Random().

**Реализация проекта**

**Алгоритм** *- набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное число действий (из Википедии).*

Но наш проект достаточно прост, поэтому алгоритма, как такового, нет. Начнем проект, при необходимости разрабатывая алгоритм дальнейших действий по ходу дела. Откройте **Lazarus** с новым проектом, свойство Name главной формы сразу переименуйте в fMain. Сохраните проект в папку **06** под именем **MyClock**, а модуль главной формы назовите Main.

Теперь займемся окном. Поскольку у окна заставки не должно быть системной строки, свойство BorderStyle установим в bsNone. А так как это окно должно занимать весь экран, свойство WindowState переведем в wsMaximized. Свойство Color установим в clBlack, чтобы форма стала черной. И ещё: свойство формы Cursor переведем в crNone, чтобы в работающей заставке не было видно указателя мыши (зачем она там?).

Пойдем далее. Установим на форму метку TLabel. Не обращайте внимания, что на черном фоне её пока не видно на форме. Изменим у неё некоторые свойства:

* Name = lClock (так мы будем обращаться к метке в программном коде; l - сокращение от label - метка, Clock - часы)
* Caption = 00:00:00
* Font = Times New Roman, жирный курсив, 48 пикселей. Цвет шрифта выберите желтый.
* ParentColor = True (это заставит метку всегда сливаться с формой, а цвет шрифта будет контрастировать с общим фоном)

Если вы забыли, как настроить свойства шрифта, то напомню: нужно выделить свойство Font, затем нажать на кнопку с тремя точками в правой части свойства, в **Инспекторе объектов**:



Откроется окно, где вы сможете выбрать шрифт, начертание, размер и цвет шрифта. Можете выбрать свои цвета для шрифта и панели, главное, чтобы они контрастировали друг с другом, и хорошо сочетались с черным цветом формы, который мы установим позже.

В результате выполненных действий у вас должно получиться примерно такое окно:

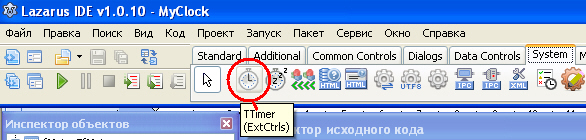


Не волнуйтесь по поводу того, что окно пока маленькое, а системная строка и обрамления окна есть - когда программа будет работать, окно займет весь экран, если вы не напутали с параметрами формы.

**Компонент TTimer**

Пришла очередь таймера. Он нам нужен, чтобы заставить надпись метки меняться - показывать текущее время. Кроме того, благодаря этому компоненту мы сможем обеспечить метке непрерывное движение.

Таймер TTimer находится на вкладке **System Палитры компонентов**:



Компонент этот **невизуальный** - невидимый для пользователя, поэтому его можно установить в любое место на форме, лишь бы он вам не мешал при проектировании формы, не перекрывал другие компоненты. Компонент будет вам виден, несмотря на черный цвет формы.

У компонента есть всего по одному свойству и событию, которые нам могут понадобиться:

Interval - это свойство устанавливает интервал времени, когда таймер сработает. По умолчанию, он равен 1000, что соответствует 1 секунде. Это нам и нужно, поэтому оставим значение свойства без изменений. А на будущее запомните: если вам потребуется установить другое время, то умножайте требуемое количество секунд на 1000, и получите то, что нужно.

Далее, переходим в **Инспекторе объектов** на вкладку **События**. Нас интересует только одно событие:

OnTimer - событие срабатывает всякий раз, когда заканчивается установленный интервал. Сгенерируйте код для этого события так же, как ранее генерировали событие OnClick для кнопок, мы заполним его чуть позже.

Вот теперь нам требуется продумать алгоритм действий - как мы будем двигать надпись. Идея такова: сгенерируем случайное число от 0 до 3, получится 4 варианта. Ведь у нас есть 4 направления, куда двигаться: вверх, вниз, влево или вправо. В зависимости от полученного случайного числа будем выбирать и направление движения. Двигать будем, скажем, на 50 пикселей влево-вправо, или на 25 пикселей вверх-вниз. А как двигать? Да очень просто! Left - это расстояние в пикселях от левого края формы до компонента. Допустим, у метки Left равен 100. То есть, от края формы до метки 100 пикселей. Если мы прибавим еще 50, то тем самым, сдвинем метку вправо, а если наоборот, отнимем, то сдвинем влево. Для движения вверх-вниз используем свойство Top - расстояние от верхней части формы до компонента. Движения будут аналогичные, но на 25 пикселей.

Если получим направление 0, то двигать будем вправо: прибавлять к свойству Left метки 50 пикселей. Если направление 1, то двигаем влево, отнимая 50 пикселей от свойства Left. Если направление 2, то двигаем вниз: прибавляем 25 пикселей к свойству Top метки. Ну и если направление 3, то двигаем вверх, прибавляя 25 пикселей к свойству Top.

Причем может случиться и так, что надпись окажется за пределами окна. В этом случае нам нужно будет вернуть её обратно в эти пределы. Сделаем это следующим образом: если метка ушла влево, то свойству Left просто присвоим значение 0 - метка вернется в пределы формы и окажется прижатой к левому краю.

Если метка ушла за правый край, то присвоим свойству Left ширину формы минус ширину метки, таким образом, метка окажется прижатой к правой части формы и будет в её пределах.

Точно таким же образом будем возвращать метку, если она уйдет вверх или вниз за пределы формы.

Значит, нам потребуется только одна целочисленная переменная, в которую мы будем генерировать направление - 4 позиции. Для этого используем функцию Random(), которую ещё называют **генератором случайных чисел**.

Random() - функция возвращает случайное число (англ. *random* - случайный) от 0 до указанного количество чисел минус один. Синтаксис очень простой:

Random(<количество чисел>);

Функция вернет случайное число от 0 до <количество чисел> - 1. То есть, в случае

i:= Random(4);

в переменную i попадет случайное число от 0 до 3. Этим мы и воспользуемся. А чтобы не тратить лишнюю память, переменную i объявим с типом byte.

Несмотря на амбициозную идею, код события OnTimer будет довольно прост:

procedure TfMain.Timer1Timer(Sender: TObject);

var

i: byte; //для получения случайного числа

begin

//первым делом меняем надпись метки:

lClock.Caption:= TimeToStr(Now);

//получаем случайное направление:

i:= Random(4);

//теперь в зависимости от направления двигаем метку:

case i of

0: lClock.Left:= lClock.Left + 50; //вправо

1: lClock.Left:= lClock.Left - 50; //влево

2: lClock.Top:= lClock.Top + 50; //вверх

3: lClock.Top:= lClock.Top - 50; //вниз

end;

//теперь проверяем: не вышла ли метка за пределы формы?

//если вышла - возвращаем её обратно

//если ушла влево:

if lClock.Left < 0 then lClock.Left:= 0;

//если ушла вверх:

if lClock.Top < 0 then lClock.Top:= 0;

//если ушла вправо:

if (lClock.Left + lClock.Width) > fMain.Width then

lClock.Left:= fMain.Width - lClock.Width;

//если ушла вниз:

if (lClock.Top + lClock.Height) > fMain.Height then

lClock.Top:= fMain.Height - lClock.Height;

end;

Вот и весь код! Вам может оказаться непонятной строка

lClock.Caption:= TimeToStr(Now);

Таким образом, мы присваиваем свойству Caption метки lClock текущее время. Работе с датой и временем у нас будет посвящена следующая лекция, так что пока на этом не зацикливайтесь. Весь остальной код должен вам быть понятен.

Собственно, проверить работу заставки можно уже сейчас, только не забудьте предварительно сохранить изменения. Чтобы закрыть окно, у которого нет для этого кнопок и меню, воспользуйтесь стандартными кнопками Windows **<Alt + F4>**.

Но заставка на этом еще не закончена. Прежде всего, из-за включенной отладочной информации программа имеет большой размер, почти 15 мегабайт! Кроме того, мы получили исполняемый exe-файл, а нам нужна заставка с расширением *\*.scr*. Исправляем эти недостатки:

1. Выбираем команду меню **<Проект -> Параметры проекта>**.
2. В разделе **<Параметры компилятора>** выбираем подраздел **<Пути>**.
3. В поле **<Имя исполнимого файла (-о)>** вместо **MyClock** указываем **MyClock.scr** (то есть, добавляем нужное расширение).
4. Отключаем флажок **<Применять соглашения по именованию>**, чтобы не было конфликта расширений.
5. В разделе **<Параметры компилятора>** переходим на подраздел **<Компоновка>**.
6. Отключаем флажок **<Генерировать отладочную информацию для GDB>**, чтобы уменьшить размер полученного файла.
7. Сохраняем проект, и заново его запускаем. В результате в папке **11-01** получаем требуемый файл **MyClock.scr** размером чуть больше 1,5 мегабайта.

Если вас устраивает, что для закрытия заставки приходится нажимать **<Alt + F4>**, то ничего больше делать не нужно. Но мне бы хотелось, чтобы эту заставку можно было закрыть кнопкой **<Esc>** (код которой 27 в таблице символов ANSI), как любую другую нормальную заставку. А для этого придется самую малость доработать проект.

Вернитесь в **Редактор форм**, и выделите саму форму. В **Инспекторе объектов** перейдите на закладку **События** и сгенерируйте событие формы OnKeyPress, которое срабатывает при нажатии на любую кнопку:



Его код совсем прост:

procedure TfMain.FormKeyPress(Sender: TObject; var Key: char);

begin

//если нажали Esc, то выходим:

if Key = #27 then Close;

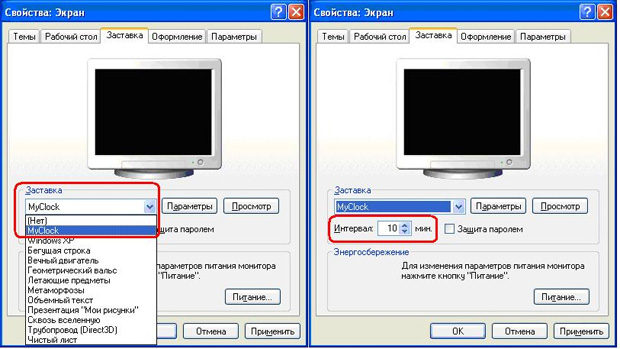
end;

Сохраните проект и снова его запустите. Заставка теперь должна закрываться при нажатии клавиши **<Esc>**. **Lazarus** нам больше не нужен, можете его закрыть.

**Установка заставки в Windows**

Большинство из вас наверняка знает, как полученную заставку установить в своей Windows. Для тех, кто этого все-таки не знает, расскажу подробно:

1. Найдите файл **MyClock.scr**, он должен быть в папке **11-01**, где вы храните проекты.
2. Скопируйте этот файл в папку **C:\Windows**
3. Щелкните правой кнопкой по свободному от окон месту **Рабочего стола** и выберите команду "**Свойства**".
4. В открывшемся окне "**Свойства: Экран**" перейдите на вкладку "**Заставка**".
5. Там выберите нашу заставку и требуемый интервал её появления, после чего нажмите **<OK>**:



Если пользователь в течение указанного интервала не будет двигать мышью и не нажмет никакой клавиши, то появится наша заставка. На сегодня это всё.

**Дата и время**

**Тип TDateTime**

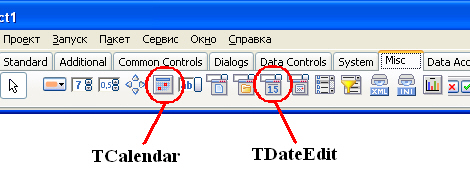
Для работы с датой и временем в Lazarus предусмотрен тип TDateTime. Это универсальный тип, он позволяет одновременно определить и дату, и время. TDateTime занимает 8 байт оперативной памяти и фактически, является вещественным числом (TDateTime = Double): целая часть этого числа содержит дату, а десятичная - время. Можно также для работы с датой и временем использовать типы соответственно, TDate и TTime, но это просто синонимы того же типа TDateTime.

В справочникe Lazarus говорится, что минимальное возможное значения даты-времени равно -693593.0, а максимальное +2958465.99999. Целая часть такого числа обозначает количество суток, а дробная - часть суток, прошедших с 0 часов. Прибавление или вычитание единицы из целой части числа равносильно прибавлению или вычитанию одного дня. Число -693593 соответствует дате 01.01.0001, а число +2958465 соответствует дате 31.12.9999, в этом диапазоне вы и сможете работать с датами в Lazarus.

**Компоненты для работы с датой-временем**

В Lazarus для ввода и вывода даты-времени имеются специальные компоненты. Знакомиться с этими компонентами будем сразу на примерах. Загрузите Lazarus с новым проектом, сразу же сохраните его в папку **07**.

Компоненты для работы с датой и временем располагаются на вкладке **Misc Палитры компонентов**. Вообще, очень интересные компоненты находятся на этой вкладке, и с некоторыми из них мы обязательно еще познакомимся. Сейчас же нас интересуют всего два компонента - календарь TCalendar и строка редактирования даты TDateEdit:

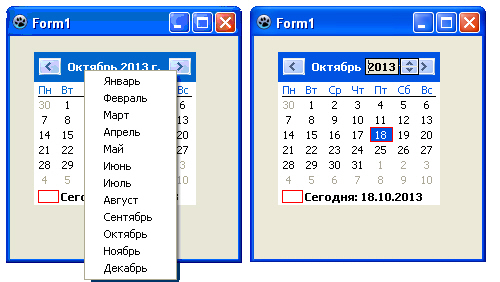


Правда, нужно заметить, компоненты эти еще сырые: некоторые свойства есть, но не работают, некоторые работают лишь частично, некоторых и вовсе нет. Так, в платном аналоге Lazarus - Delphi в компоненте TDateEdit есть возможность вводить как дату, так и время. В Lazarus можно вводить только дату. Для ввода времени, вероятно, придется использовать обычный TMaskEdit.

Поскольку Lazarus - проект новый, будем надеяться, что эти недоработки в будущем будут исправлены.

**TCalendar**

Вначале познакомимся с компонентом TCalendar. Установите его на форму, и он сразу примет вид календаря, причем текущая дата будет выделена. Выделение будет как красной окантовкой, так в календаре выделяется текущая дата, так и синей заливкой - так в календаре выделяется выбранная дата. А поскольку выбрать можно любую дату, то они не всегда совпадают. Однако из формы управлять календарем не очень удобно. Сохраните проект и запустите его. В рабочем приложении календарем можно управлять: щелкая по синим стрелкам в левой правой частях заголовка календаря можно листать месяцы, щелкнув по названию месяца можно открыть список, в котором выбрать другой месяц, щелкнув по году, можно выбрать или ввести вручную другой год.



Закройте запущенную программу и вернитесь в проект. Познакомимся с некоторыми полезными свойствами компонента TCalendar.

|  |  |
| --- | --- |
| Align | - выравнивание компонента относительно формы (или другого контейнера). С этим свойством вы уже знакомы, но в случае с календарем применять его не имеет смысла - если еще можно прижать календарь к одной из сторон, то изменить его размеры все равно не удастся. |
| BorderSpacing | - раскрывающееся сложное свойство. По идее, оно должно настраивать параметры обрамления: ширину обрамления вокруг календаря или по одной из его сторон, но, увы, это свойство тоже не работает - как я не пытался что-то изменить, вид календарь остался прежним. |
| Date и DateTime | - установленная в календаре дата. При изменении одного из этих свойств, изменится и другое, но есть и отличия: Date имеет тип String, а DateTime - TDateTime. Таким образом, если вам нужно получить дату в виде строки, обращайтесь к Date. Если надо работать с датой в форме TDateTime, а это часто бывает нужно, например, для расчета дат, то обращайтесь к TDateTime. Присвоить календарю новую дату можно как через первое, так и через второе свойство, но если вы будете менять дату с помощью Date, вы должны быть уверены, что дата в строке написана правильно. |
| DisplaySettings | - свойство должно управлять видом календаря, но тоже почему-то не изменяет его. Имеет следующие подсвойства, разрешающие (при True) или запрещающие (при False) различные действия:   * dsNoMonthChange - Не разрешать изменять месяц * dsShowDayNames - Показывать имя дня недели * dsShowHeadings - Показывать заголовок * dsShowWeekNumbers - Показывать номер недели, в году их 52 (это подсвойство установить можно). * dsStartMonday - Начинать неделю с Понедельника (в английских и американских календарях неделю принято начинать с Воскресенья). |

Перейдем к событиям компонента (переключитесь на вкладку **События** в **Инспекторе объектов**). Здесь нам могут понадобиться следующие события:

* OnChange - событие наступает при любом изменении в компоненте: изменили год, месяц или день.
* OnClick - щелкнули по компоненту мышью.
* OnDayChanged - изменился день.
* OnDblClick - дважды щелкнули по компоненту мышью.
* OnMonthChanged - изменился месяц.
* OnYearChanged - изменился год.

**TDateEdit**

Теперь изучим компонент TDateEdit. Установите его на форму, где-нибудь под календарем. Как видите, фактически компонент состоит из двух компонентов: поля для редактирования даты и кнопки, при нажатии на которую появляется календарь:

Компонент TDateEdit

Теперь перейдите на вкладку **Свойства** в **Инспекторе объектов**. У компонента имеются следующие полезные свойства:

|  |  |
| --- | --- |
| ButtonOnlyWhenFocused | - Показывать кнопку выбора даты только когда фокус ввода находится на компоненте. По умолчанию False, то есть, кнопку видно всегда. Честно говоря, не вижу смысла прятать эту кнопку. |
| ButtonWidth | - ширина кнопки. По умолчанию равна 23, но можно и изменить. Правда, опять непонятно зачем. |
| CalendarDisplaySettings | - аналогично свойству DisplaySettings календаря, и предназначено для того же: для изменения внешнего вида календаря, который появляется при нажатии на кнопку. |
| CancelCaption | - заголовок для кнопки Отмена в диалоговом окне **Календарь**. Надо полагать, речь идет о привязанном к компоненту диалоге TCalendarDialog (диалоги мы будем изучать в другой лекции). Мне лично не удалось найти, как диалог привязать к компоненту, поиски информации в Интернете также были безуспешны. Будем надеяться, в следующих версиях Lazarus этот компонент заработает, как надо. |
| DateOrder | - формат вывода даты. При выборе формата, отличного от doNone (по умолчанию), появляется маска даты. Может иметь следующие значения:   * doDMY - последовательность даты: День, Месяц, Год. * doMDY - последовательность даты: Месяц, День, Год. * doNone - последовательность даты соответствует системной. Для русской Windows это обычно День, Месяц, Год. Маски в этом формате не предусмотрено, если пользователь вводит дату сам, не пользуясь выбором в календаре, то легко может ошибиться. Поэтому рекомендую использовать значение doDMY, соответствующее русскому представлению даты. * doYMd - последовательность даты: Год, Месяц, День. |
| DefaultToday | - если True, то в работающей программе компонент по умолчанию отображает текущую дату, если False, то не отображает ничего. |
| OKCaption | - заголовок для кнопки **ОК** в диалоговом окне **Календарь**. Проблемы те же, что и у CancelCaption. |

В **Инспекторе объектов** почему то не отображаются следующие необходимые свойства:

|  |  |
| --- | --- |
| Date | - имеет тип TDateTime, содержит выбранную или введенную пользователем дату. |
| Text | - имеет тип TCaption, который совместим с обычной строкой. Тут выбранная или введенная дата хранится в виде строки. |

Этими свойствами часто придется пользоваться, чтобы узнать, какую же дату выбрал или ввел пользователь, так что запомните их.

Основные события, которые вам могут понадобиться, это OnChange (при изменении), OnClick (щелкнули мышью), OnKeyPress (нажали любую клавишу). События эти работают так же, как в других, уже известных вам компонентах, так что задерживаться тут мы не будем.

**Стандартные функции для работы с датой и временем**

Ознакомимся с основными функциями, которые помогут нам работать с датой и временем. Для выполнения примеров советую установить на форму простую кнопку, и сгенерировать для нее событие OnClick, в котором и будете пробовать предлагаемые примеры. Итак, начнем.

Date - Функция возвращает текущую дату в формате TDateTime. Часть числа после запятой (временная часть) будет равна нулю. Пример применения:

var dt: TDateTime;

begin

dt:= Date; //получили в dt текущую дату

…

Time - Функция возвращает текущее время в формате TDateTime. Часть числа перед запятой (часть даты) будет равна нулю. Пример применения аналогичен функции Date.

Now - Функция возвращает текущие дату и время в формате TDateTime. Обе части числа будут заполнены значениями. Пример применения аналогичен совместным функциям Date и Time.

DateTimeToStr() - Функция принимает в качестве параметра дату и время в формате TDateTime и возвращает эти данные в виде строки. Синтаксис функции:

function DateTimeToStr(DateTime: TDateTime): String

Пример вывода текущих даты времени на экран:

begin

ShowMessage(DateTimeToStr(Now));

В результате на экран выйдет сообщение с текущими датой и временем, например:

**19.10.2013 9:25:57**

DateToStr() - Функция принимает в качестве параметра дату в формате TDateTime и выводит её в виде строки. Время при этом не выводится. Применение функции аналогично DateTimeToStr.

TimeToStr() - Функция принимает в качестве параметра время в формате TDateTime и выводит его в виде строки. Дата при этом не выводится. Применение функции аналогично DateTimeToStr.

Обратите внимание, что форматы вывода даты-времени зависят от того, какая Windows у вас установлена, и какие правила действуют в вашей стране. Так, указанное сообщение сформировалось по правилам России, для русской Windows. А вот в США, например, другие правила. Там эта строка выглядела бы так:

**2013.10.19 9:25:57AM**

Все эти правила зависят от системных переменных ShortDateFormat (краткий формат даты), LongDateFormat (длинный формат даты), ShortTimeFormat (краткий формат времени) и LongTimeFormat (длинный формат времени). Объявлять эти переменные не нужно, они уже присутствуют в системе и содержат строки с соответствующими форматами (о форматах мы поговорим ниже). Так, для русской Windows краткий формат дат установлен, как dd.MM.yyyy (то есть, день с ведущим нулем, месяц и год в четырех цифрах). Краткий формат времени будет h:nn (час без ведущего нуля и минуты с ведущим нулем), а длинный - h:nn:ss (час без ведущего нуля, минуты и секунды с ведущим нулем). При желании, эти системные переменные можно изменить прямо в программе, например:

begin

ShortDateFormat:= 'dd.mm.yy';

В результате, год будет выводиться в двух цифрах, а не в четырех, т.е. вместо "2013" будет "13". Однако я не рекомендую это делать. Когда вам понадобится (а это обязательно произойдет) вывести дату и время в желаемом вам, или вашему заказчику, виде, то лучше воспользоваться функциями, реализующими форматный вывод, речь о которых пойдет ниже. Тем более что это гарантирует правильный вывод даты-времени на любой Windows, в любой стране.

FormatDateTime() - функция выводит заданную дату и время в требуемом вам формате, который не зависит от того, какая Windows (русская или английская) у вас установлена. Синтаксис функции следующий:

function FormatDateTime(const Format: String; DateTime: TDateTime): String;

В функцию следует передать два параметра: строку с требуемым форматом, и дату-время (или только дату, или только время) в формате TDateTime. Функция вернет эти данные в указанном формате. Например:

var

s: String;

begin

s:= FormatDateTime('dd mmmm yyyy - hh:nn:ss', Now);

В результате в строковой переменной s окажется строка типа "21 Октябрь 2013 - 11:41:56". Но это в теории. На практике вместо русских букв выходят знаки "?" - сказываются проблемы совместимости Lazarus с различными кодировками Windows. Что-ж, решение есть и из этой ситуации: нужно конвертировать полученную строку функцией SysToUTF8(), которая получает ANSI-строку, и гарантированно возвращает формат UTF8, в котором, как мы знаем, и работает Lazarus:

s:= SysToUTF8(FormatDateTime('dd mmmm yyyy - hh:nn:ss', Now));

Указанный программистом формат может содержать символы, которые представлены в таблице ниже:

|  |  |
| --- | --- |
| **Символы** | **Описание** |
| c | Отображает короткий формат даты и длинный формат времени, то есть, в русской Windows это будет в виде "дд.мм.гггг чч:мм:сс", например: "21.10.2013 11:54:56". |
| d | Отображает день месяца (число) без ведущего нуля (1-31) |
| dd | Отображает день месяца (число) с ведущим нулем (01-31) |
| ddd | День недели в сокращенном виде, например: Пн, Вт для русских Windows. |
| dddd | День недели в полном виде, например, понедельник, вторник. |
| ddddd | Короткий формат даты, например: "21.10.2013" |
| dddddd | Длинный формат даты, например: "21 Октябрь 2013 г." - для вывода даты в документ или на экран самое то. |
| m | Номер месяца без ведущего нуля (1-12). |
| mm | Номер месяца с ведущим нулем (01-12). |
| mmm | Краткое название месяца, например: "окт". |
| mmmm | Полное название месяца, например: "Октябрь". |
| y или yy | Год из двух цифр, например: "13" |
| yyyy | Год из четырех цифр, например: "2013" |
| h | Час без ведущего нуля (0-23) |
| hh | Час с ведущим нулем (00-23) |
| n | Минуты без ведущего нуля (0-59) |
| nn | Минуты с ведущим нулем (00-59) |
| s | Секунды без ведущего нуля (0-59) |
| ss | Секунды с ведущим нулем (00-59) |
| z | Вывод миллисекунд (0-999). Например, формат "hh:nn:ss:z" выведет время примерно так: "12:36:01:745" |
| t | Короткий формат времени, например: "12:16" |
| tt | Длинный формат времени, например: "12:16:53" |
| am/pm | Время будет выводиться так, как принято на западе: 12-часовое представление, после которого указывается am (до полудня) или pm (после полудня). Эти символы добавляются к требуемому формату, например, формат "t am/pm" выведет время в виде "12:21 pm" |
| a/p | Аналогично предыдущему, но вместо am (до полудня) или pm (после полудня) будут выходить a или p. |

DateTimeToString() - процедура, которая преобразует указанные дату-время в строку с использованием указанного формата, и результат помещает в строковую переменную, которая указывается, как параметр по ссылке. Указанный формат использует символы из [таблицы 12.1](http://intuit.valrkl.ru/course-1265/#ID.12.table.12.1), как и FormatDateTime().

Синтаксис:

procedure DateTimeToString(var Result: String; const Format: String; DateTime: TDateTime);

Пример применения:

var

s: String;

begin

DateTimeToString(s, 'dddddd', Now);

ShowMessage(SysToUTF8(s)); //тут необходимо преобразовать строку в UTF8

Как видите, для корректного вывода русского текста нам пришлось преобразовать его из ANSI в UTF8 прямо внутри оператора ShowMessage(), так как в процедуре DateTimeToString использовать SysToUTF8() невозможно. Собственно, можно было вначале преобразовать строку, а уж затем вывести ее:

…

DateTimeToString(s, 'dddddd', Now);

s:= SysToUTF8(s);

ShowMessage(s);

Но так пришлось бы делать лишнюю строку кода.

DayOfWeek() - функция принимает в качестве параметра дату, и возвращает номер дня недели этой даты в виде целого числа. Причем делает это она так, как принято на западе: первым днем недели считается воскресение, последним - суббота. Так что понедельник будет вторым днем недели. Синтаксис:

function DateOfWeek(Date: TDateTime): Integer;

Если же вам потребуется вывести день недели словом, а не числом, то можно поступить, например, так:

procedure TForm1.Button1Click(Sender: TObject);

var

s: String;

begin

case DayOfWeek(Now) of

1: s:= 'Воскресение';

2: s:= 'Понедельник';

3: s:= 'Вторник';

4: s:= 'Среда';

5: s:= 'Четверг';

6: s:= 'Пятница';

7: s:= 'Суббота';

end; //case

ShowMessage('Сегодня ' + FormatDateTime('ddddd', Now) + ', ' + s);

end;

Вначале с помощью case мы получили в s требуемый день недели. Затем мы вывели сообщение, типа "Сегодня 21.10.2013, Понедельник".

DecodeDate() - очень полезная процедура. Она разбивает указанную дату отдельно на год, месяц и число, возвращая эти данные в виде целых чисел. Синтаксис:

procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);

Пример применения:

procedure TForm1.Button1Click(Sender: TObject);

var

Year, Month, Day: Word;

s: String;

begin

DecodeDate(Now, Year, Month, Day);

ShowMessage('Год - ' + IntToStr(Year) + #13 +

'Месяц - ' + IntToStr(Month) + #13 +

'День - ' + IntToStr(Day));

end;

В данном примере мы получили в виде чисел год, месяц и число текущей даты в соответствующие переменные. Затем вывели их на экран, преобразовав с помощью IntToStr() в строки, каждое данное на отдельной строке.

EncodeDate() - функция, которая является противоположностью DecodeDate. Она напротив, собирает дату из отдельных чисел, и возвращает её, преобразовав в формат TDateTime. Синтаксис:

function EncodeDate(Year, Month, Day: Word): TDateTime;

Пример применения:

procedure TForm1.Button1Click(Sender: TObject);

var

Year, Month, Day: Word; //для указания года, месяца и дня

dt: TDateTime; //для получения даты

s: String;

begin

//указываем год, число и день:

Year:= 1999;

Month:= 3;

Day:= 9;

//получаем дату:

dt:= EncodeDate(Year, Month, Day);

//теперь выводим ее:

ShowMessage('Указали ' + SysToUTF8(FormatDateTime('dddddd', dt)));

end;

Комментарии достаточно подробны, чтобы вы разобрались с кодом. В результате выполнения примера будет выведено сообщение "Указали 9 Март 1999 г.". Вы можете указать и другие год, число и день, только помните, что указывать их нужно правильно. Например, нельзя в Феврале указывать 30 дней, а в Апреле 31. И еще: если вы попытаетесь использовать дату, меньшую чем 01.01.0001, или большую чем 31.12.9999, то воспользоваться этими функциями вам не удастся, программа выведет ошибку.

DecodeTime() - Процедура разбивает указанное время отдельно на часы, минуты, секунды и миллисекунды, возвращая эти данные в виде целых чисел. Синтаксис:

procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);

Поскольку процедура работает аналогично DecodeDate, приводить примеры мы не будем.

EncodeTime() - а эта функция работает аналогично EncodeDate(), и собирает время из отдельных чисел, возвращая его в формате TDateTime. Синтаксис:

function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;

IsLeapYear() - функция, которая иногда бывает нужна. Она определяет, был ли год високосным, или нет. Год указывается в виде целого числа. Функция возвращает True, если указанный год високосный, и False в противном случае. Синтаксис:

function IsLeapYear(Year: Word): Boolean;

Пример применения:

procedure TForm1.Button1Click(Sender: TObject);

var

Year: Word;

begin

//указываем год:

Year:= 1917;

//високосный или нет?:

if IsLeapYear(Year) then ShowMessage('Указан високосный год')

else ShowMessage('Указан невисокосный год');

end;

Напоследок остались функции преобразования даты-времени из строки в TDateTime, они тоже очень часто бывают нужны.

StrToDateTime() - функция преобразует указанные в виде строки дату и время в формат TDateTime. Синтаксис:

function StrToDateTime(const S: string): TDateTime;

Пример:

procedure TForm1.Button1Click(Sender: TObject);

var

dt: TDateTime;

begin

dt:= StrToDateTime('21.10.2013 14:25:30');

ShowMessage('Указали ' + FormatDateTime('c', dt));

end;

Обратите внимание вот на что. В этой функции строку нужно составлять правильно, в зависимости от языка вашей Windows. Дата должна указываться в кратком формате, от времени дата отделяется пробелом. Сепаратором (разделителем частей даты) может быть как точка - в русской Windows, так и слеш "/" в английской, это зависит от переменной DateSeparator. Три части даты (y-год, m-месяц и d-день) могут быть в виде y/m/d, m/d/y, d/m/y - порядок этих частей зависит от переменной ShortDateFormat.

StrToDate() - функция преобразует указанную в виде строки дату в формат TDateTime, время при этом не указывается. Синтаксис:

function StrToDate(const S: string): TDateTime;

StrToTime() - функция преобразует указанное в виде строки время в формат TDateTime, дата при этом не указывается. Синтаксис:

function StrToDate(const S: string): TDateTime;