Расширение возможностей среды IDE JBuilder с помощью Opentools. Часть 2

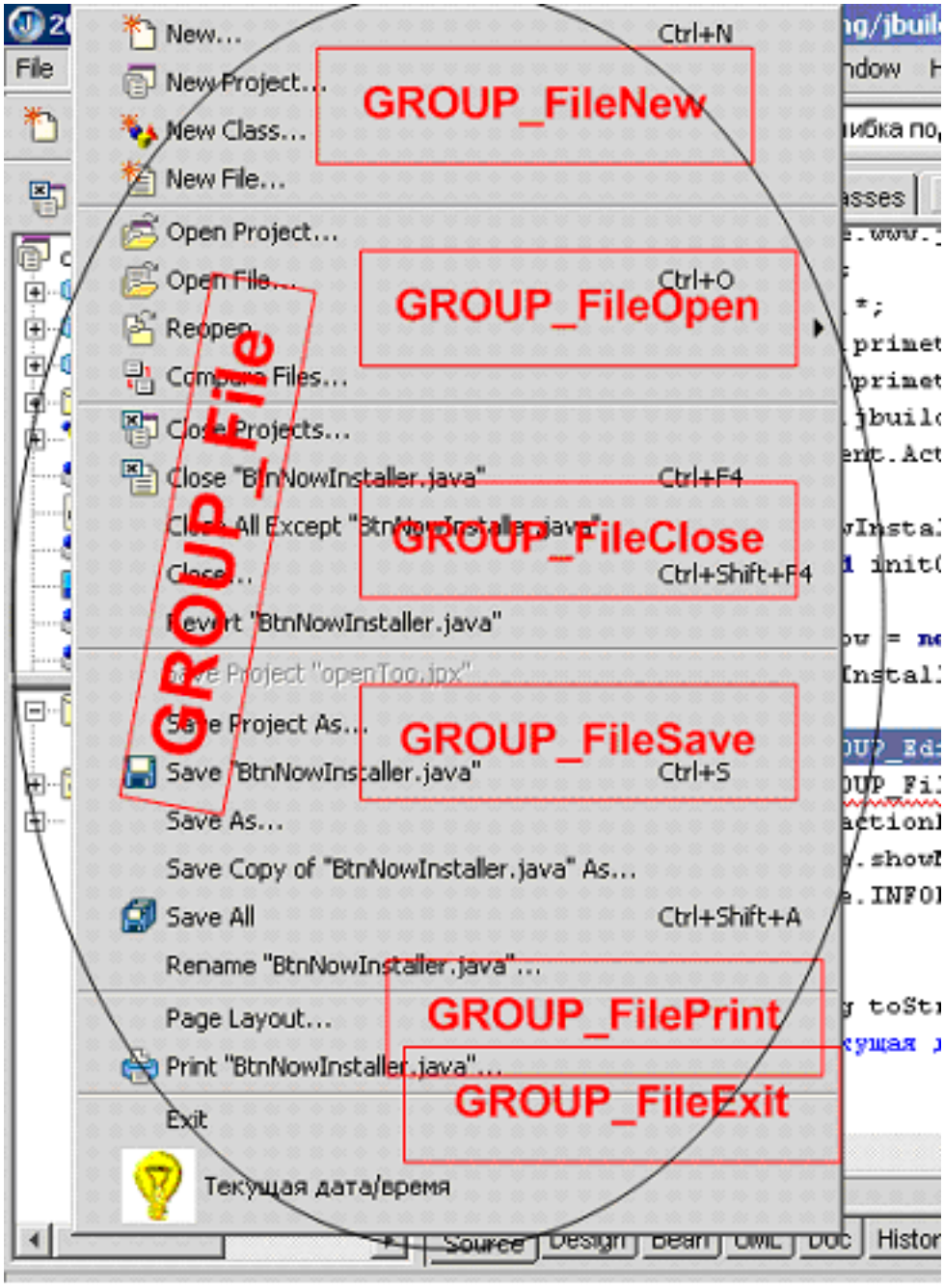
Николай Жишкевич

1. Взаимодействие с меню и панелями кнопок

В данной статье продолжается рассказ начатый в предыдущей части материала, как я и обещал сегодня мы сосредоточимся на создании панелей кнопок, настройки меню и даже созданию собственных мастеров.

Как я уже говорил в предыдущем материале, каждому компоненту среды JBuilder соответствует набор классов. Так для доступа к меню следует использовать класс `JBuilderMenu`.

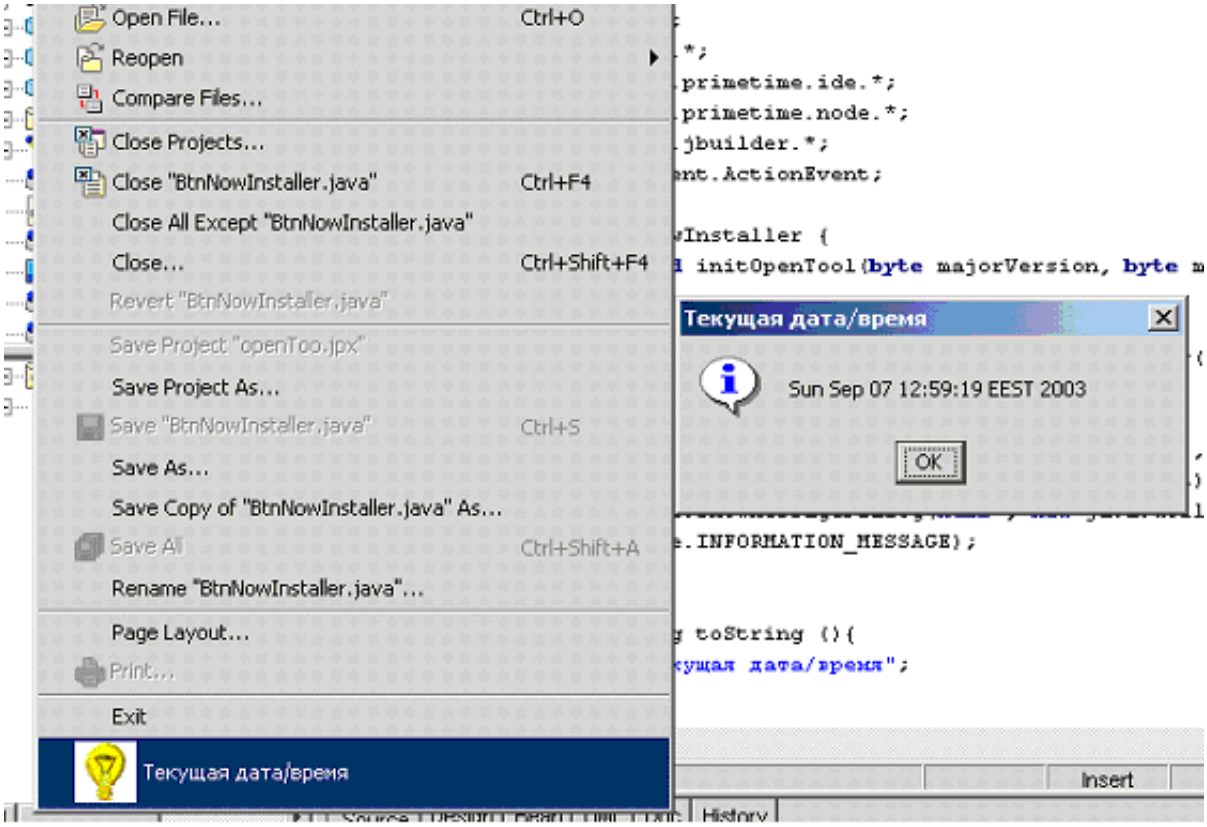
Для иллюстрации я создам пример Opentool, который будет вызываться при нажатии на кнопку меню и будет выводить окно сообщения с информацией о текущей дате/времени. В приведенном ниже примере я обращаюсь к классу `JBuilderMenu`, внутри данного класса есть множество открытых (public) статических полей, которые дают доступ к различным меню JBuilder. Например для доступа к меню "файл" (file) следует использовать поле GROUP_File, а для доступа к меню "редактирование" (edit) необходимо обратиться к полю GROUP_Edit, и так далее. Разумеется, что доступ может осуществляться не только к меню в целом, но и его подчастям. Например, в меню "файл" есть группы "новое", "сохранить", "закрыть" и другие. И каждой из этих групп соответствует определенное поле класса `JBuilderMenu`.



Сейчас я приведу пример кода, который добавляет новую кнопку в меню File, и при нажатии на которую появляется окно сообщения с текущим временем и датой.

```
package com.javable.www.jbuilder.opentool;
import com.borland.jbuilder.*;

import java.awt.event.*;

import javax.swing.*;


public class BtnNowInstaller
{
    public static void initOpentool(byte majorVersion,
                                    byte minorVersion)
    {

        ImageIcon ico_now = new ImageIcon(new BtnNowInstaller().
                getClass().getClassLoader().getResource(
                                        "now-pic.PNG"));
        JBuilderMenu.GROUP_File.add(new AbstractAction(
                                            "Now",
                                            ico_now) {
            public void actionPerformed(ActionEvent actionEvent)
            {
                JOptionPane.showMessageDialog(
                        null,
                        new java.util.Date(),
                        "Текущая дата/время",
                        JOptionPane.INFORMATION_MESSAGE);
            }

            public String toString()
            {

                return "Текущая дата/время";
            }
        });
    }
}
```

А вот и картинка, которая демонстрирует результат выполнения данного фрагмента кода, незабудьте только создать архив с описателем Opentool как "ui".



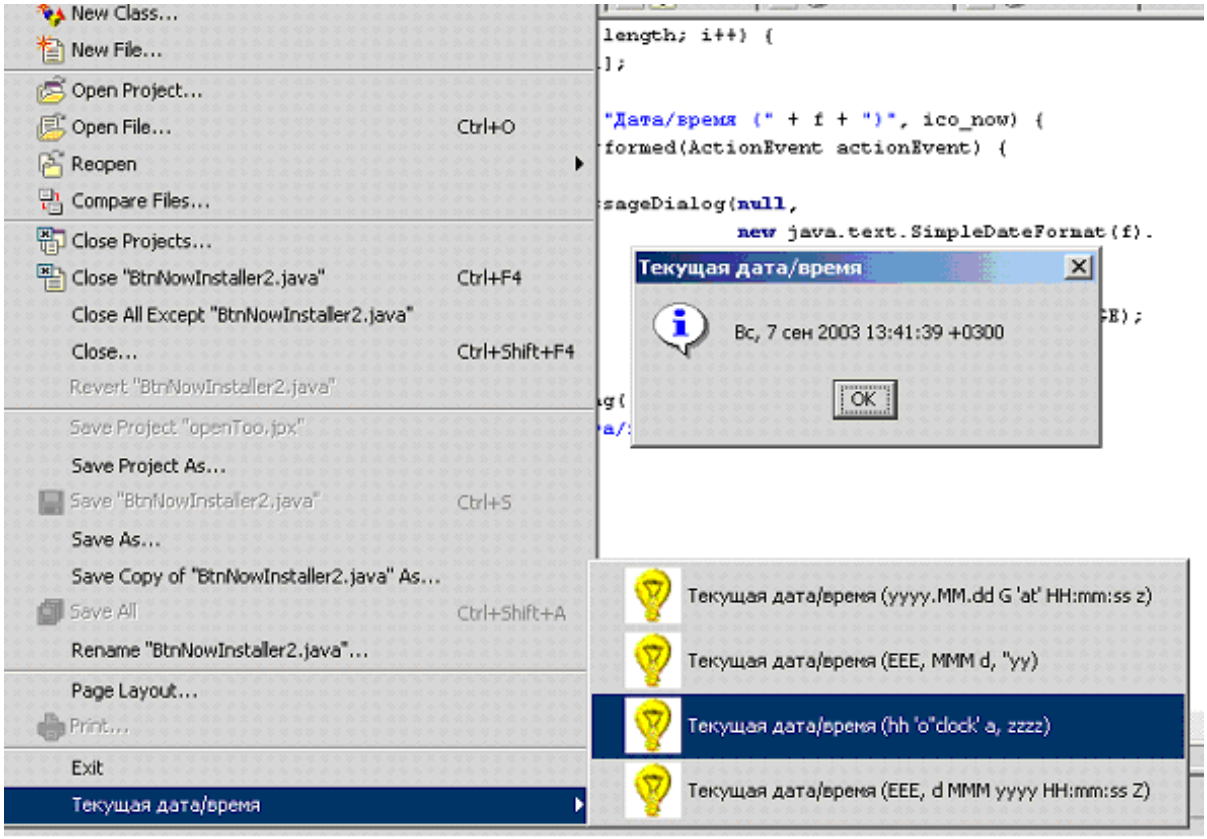
Разумеется, что часто бывает необходимо создать новое меню, наполнить его соответствующими командами. Давайте ниже мы проиллюстрируем подобный подход. Кроме уже знакомого нам класса `JBuilderMenu` нам потребуется также классы `Action`, который описывает одну команду или пункт меню, а также новый для нас класс `com.borland.primetime.actions.ActionGroup`, данный класс представляет собой контейнер, в котором может размещаться множество объектов Action. Для добавления и удаления существуют специальные методы "add", "remove". Итак, для иллюстрации будет создано новое подменю, содержащее набор пунктов для вывода даты/времени, отличия будут в используемом формате представления в виде строки. Техническое преобразования объекта `java.util.Date` в `String` (форматирование) будет выполняться с помощью класса `SimpleDateFormat`.

Обращаю дополнительно ваше внимание на очень важный метод в классе `com.borland.primetime.actions.ActionGroup`, это метод `setPopup (Boolean b)`, с помощью данного метода можно управлять стилем группы команд,

существует вариант, что все команды в этой группе будут расположены друг за другом на уровне верхнего меню, а если же свойство `isPopup` установлено в true, то группа команд будет реализована в виде подменю.

```
package com.javable.www.jbuilder.opentool;
import com.borland.jbuilder.*;

import com.borland.primetime.actions.*;

import java.awt.event.*;

import javax.swing.*;


public class BtnNowInstaller2
{
    public static void initOpentool(byte majorVersion,
                                     byte minorVersion)
    {

        ImageIcon ico_now = new ImageIcon(new BtnNowInstaller2().
          getClass().getClassLoader().getResource(
                                       "now-pic.PNG"));
        ActionGroup ag = new ActionGroup(
                                 "Текущая дата/время",
                                 'c',
                                 "Просмотр текущей даты/времени в
                                 различных форматах представления");
        String[] fmtg = new String[]
        {
            "yyyy.MM.dd G 'at' HH:mm:ss z",
            "EEE, MMM d, ''yy",
            "hh 'o''clock' a, zzzz",
            "EEE, d MMM yyyy HH:mm:ss Z"
        };

        for (int i = 0; i < fmtg.length; i++)
        {

            final String f = fmtg[i];
```

```
            ag.add(new AbstractAction(
                           "Дата/время (" + f +
                               ")", ico_now) {
                public void actionPerformed(ActionEvent actionEvent)
                {
                    JOptionPane.showMessageDialog(
                            null,
                            new java.text.SimpleDateFormat(
                                    f).format(new java.util.Date()),
                            "Текущая дата/время",
                            JOptionPane.INFORMATION_MESSAGE);
                }

                public String toString()
                {

                    return "Текущая дата/время (" + f +
                           ")";
                }
            });
        }

        ag.setPopup(true); // самое главное указать, что данный набор действий

        // должен быть реализован в виде подменю
        JBuilderMenu.GROUP_File.add(ag);
    }
}
```



2. Работа с панелями инструментов

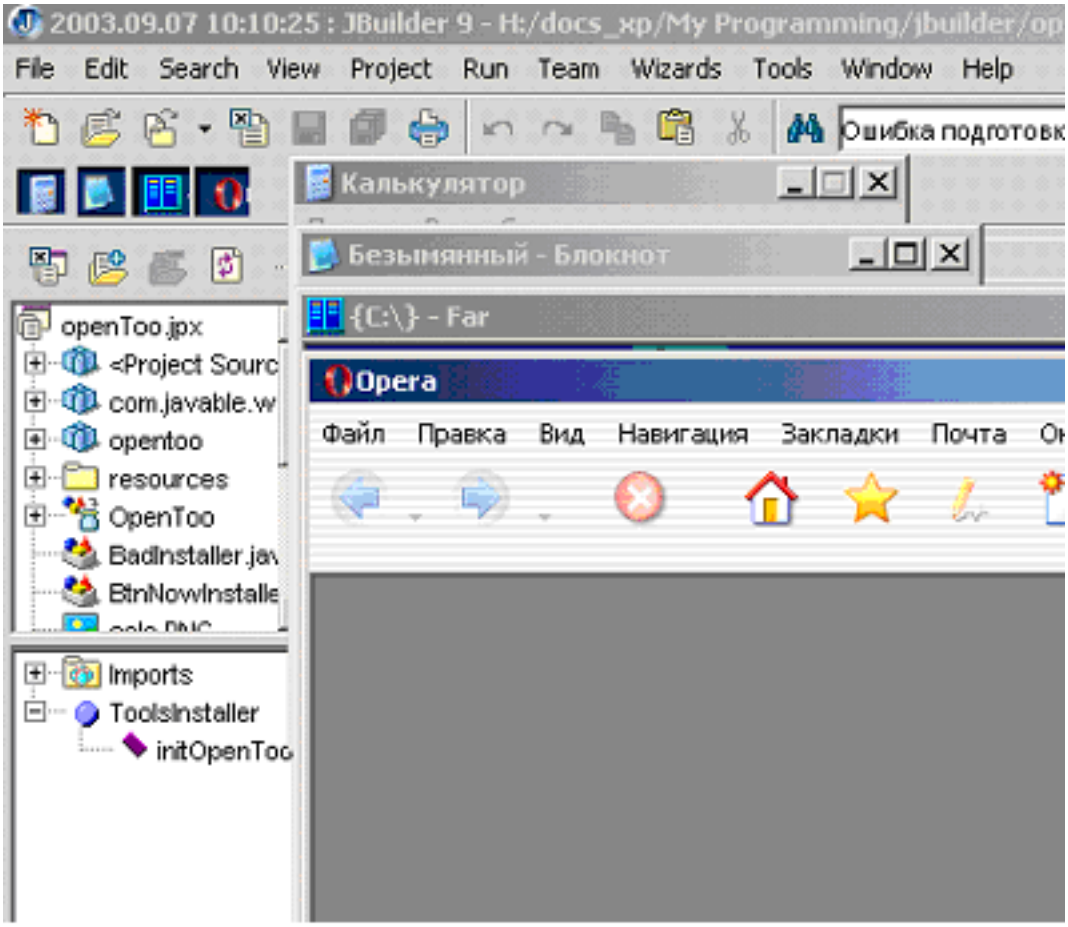
Если говорить честно, то схема работы с панелями инструментов практически не отличается от схемы создания новых пунктов меню. Что, в принципе говоря, и логично, ведь кнопки на панели инструментов являются аналогами или синонимами команд, которые вызываются из меню.

Пожалуй, задача, на примере которой мы рассмотрим механизмы управления панелями инструментов, будет не слишком оригинальна. Я создам Opentool который будет служить для запуска набора приложений, например калькулятор, paint и любые другие программы. Что касательно хранения информации о командах, то я поступлю очень некрасиво и не оригинально, жестко в исходных кодах программы задав комбинации значений для каждой из команд, каждая из которых будет состоять их символического изображения, названия команды и, разумеется, строка запуска. Таким

образом, этот Opentool будет функционально идентичен стандартному способу создания внешних команд с помощью меню "tools->configure tools".

```
package com.javable.www.jbuilder.opentool;
import com.borland.primetime.actions.*;
import com.borland.primetime.ide.*;

import java.awt.event.*;

import java.io.*;

import javax.swing.*;


public class ToolsInstaller
{
    public static void initOpentool(byte majorVersion,
                                    byte minorVersion)
    {

        String[][] parms =
        {
            { "calculator", "calc", "calc.PNG" },
            { "notepad", "notepad", "notepad.PNG" },

            {
                "far",
                "\"E:\\Program Files\\Far\\Far.exe\"",
                "far.PNG"
            },

            {
                "opera",
                "\"E:\\Program Files\\Opera7\\opera.exe\"",
                "opera.PNG"
            }
        };
        ActionGroup ag = new ActionGroup(
                                "Внешние команды",
                                'c',
```

```
                                "Запуск самых различных внешних команд");

        for (int i = 0; i < parms.length; i++)
        {

            final String command = parms[i][1];
            final String command_name = parms[i][0];
            final ImageIcon ico_comm = new ImageIcon(new ToolsInstaller().
            getClass().getClassLoader().getResource(
                                       parms[i][2]));
            ag.add(new AbstractAction(
                           command_name, ico_comm) {
                public void actionPerformed(ActionEvent actionEvent)
                {

                    try
                    {
                        Runtime.getRuntime().exec(
                                command);
                    }
                    catch (IOException ex)
                    {
                        JOptionPane.showMessageDialog(
                                null, ex);
                    }
                }

                public String toString()
                {

                    return command_name;
                }
            });
        }

        Browser.addToolBarGroup(ag);
    }
}
```

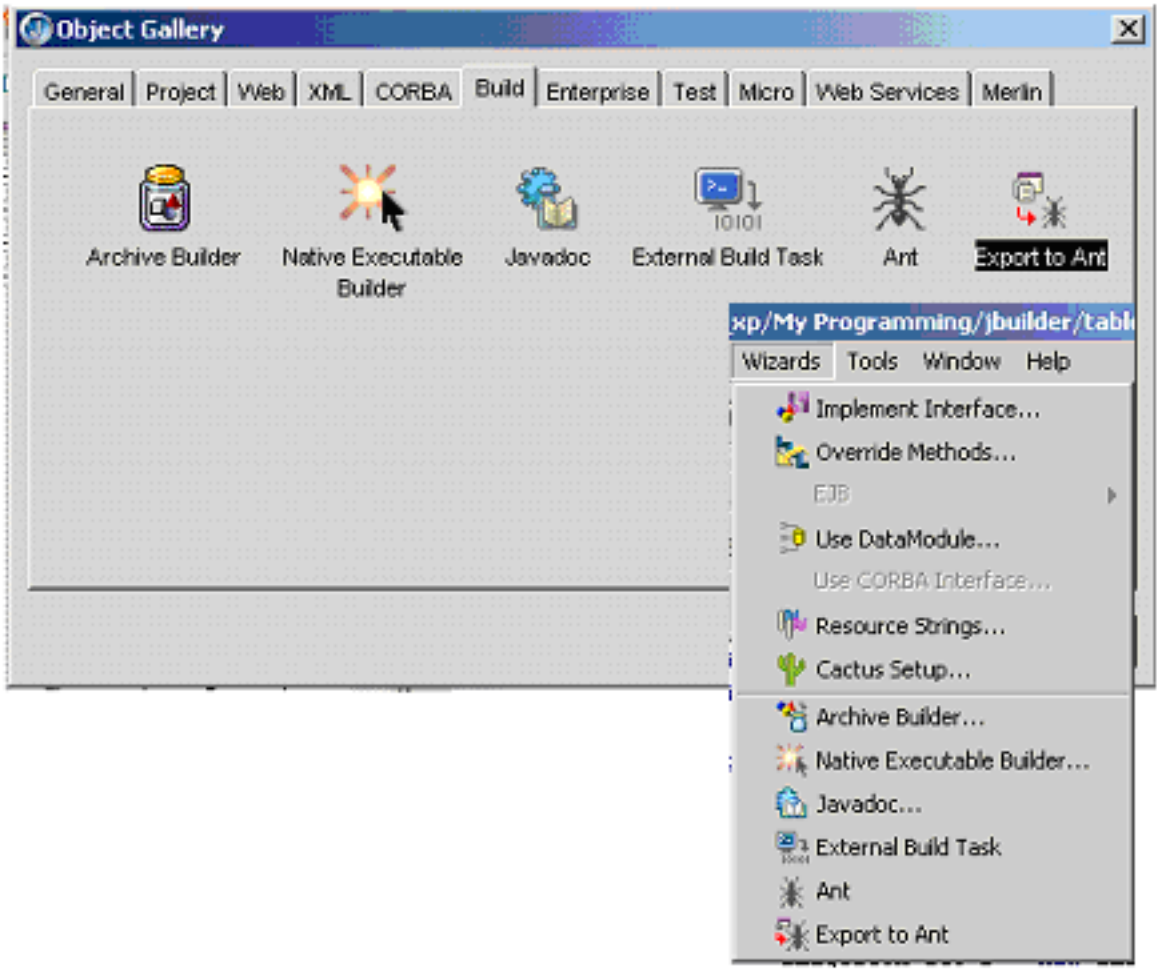


В качестве небольшого домашнего упражнения предлагаю вам модифицировать исходный код примера, добавив в него возможность запуска explorer или любой другой вашей любимой оболочки в том каталоге, который содержит активный проект.

3. Мастера и волшебники

Следующий пример, который мы рассмотрим, будет служить для добавления в среду JBuilder нового мастера, который будет решать очень важную и полезную задачу — с помощью его можно будет легко создать строку формата преобразования между объектом `java.util.Date` и строкой String для последующего использования в рамках класса `SimpleDateFormat` (с ним мы уже довольно неплохо поработали сегодня).

Давайте кратко рассмотрим те классы и интерфейсы, которые нам потребуются для этого примера. Прежде всего, нам будет нужен класс `com.borland.primetime.wizard.WizardManager`. В данном классе есть только два метода, причем оба являются статическими, и к тому же для нас интерес представляет пока только один метод. Этот метод называется `registerWizardAction` и именно он служит для регистрации мастеров. Стоит сказать, что создаваемые нами мастера могут располагаться либо в подменю "Wizards" главного меню среды JBuilder, либо как вариант мастера будут вызываться при создании/добавлении в проект определенного вида ресурса. Например, если вы обратите внимание на закладку Build появляющегося диалогового окна при выборе "file->new" и пункты меню "Wizards", то можно заметить соответствие. Таким образом можно сделать вывод, что существуют мастера двух видов: первый вид мастеров вызывается с помощью меню "wizards", а второй через галерею создания нового объекта. Разумеется, что существуют волшебники, которые сочетают эти два подхода, а отсюда следует вывод, что различия в коде для обоих вариантов будут не велики. И действительно немного позже мы убедимся, что все различие можно свести всего к одной строчке кода, и это правда.



Как я сказал, регистрация нового мастера выполняется с помощью вызова `WizardManager.registerWizardAction`. В качестве параметров для данного метода следует передать некоторый объект класс которого расширяет или является потомком класса `com.borland.primetime.wizard.WizardAction`. В данном классе определен ряд методов для описания характеристик волшебника.

Давайте, прежде всего, обратим внимание на конструктор для данного класса. Всего существует 7 версий конструктора, которые отличаются друг от друга количеством явно указываемых параметров. Мы рассмотрим наиболее полную версию конструктора.

```
WizardAction(java.lang.String shortText,
```

```
      char mnemonic, java.lang.String longText,
      javax.swing.Icon smallIcon, javax.swing.Icon largeIcon,
      boolean galleryWizard, java.lang.String category)
```

В качестве параметров здесь выступает следующий набор величин:

- shortText — короткое текстовое название нашего волшебника;
- mnemonic — переменная типа char задает мнемонику для данной команды;
- longText — длинное название мастера;
- smallIcon — ссылка на объект иконки размером 16*16 пикселей;
- largeIcon — ссылка на объект иконки размером 32*32 пикселей;
- galleryWizard — обычная булевская переменная, от значения которой зависит каким именно будет наш мастер, или он будет размещен в галерее создания новых объектов либо в меню Wizards;
- category — название категории к которой будет отнесен мастер, например если мы указываем что мастер должен быть добавлен в галерею, то название категории будет являться названием закладки диалога создания нового объекта.

А теперь давайте рассмотрим метод класса, который представляет для нас самый большой интерес.

`protected abstract Wizard createWizard()` — при создании собственной реализации мастера следует обязательно перекрыть этот метод. Данный метод должен выполнять всю основную работу по созданию интерфейса набора панелей или более точно закладок мастера. На каждой из закладок (страниц) мастера следует расположить элементы управления, а также создать обработчики событий, вызывающиеся при переходе от одной страницы к другой.

Итак, мы уже знаем, что результатом своей работы метод `createWizard` должен вернуть объект, который поддерживает интерфейс `com.borland.primetime.wizard.Wizard`.

Как я уже упоминал раньше, интерфейс `Wizard` логически представляет собой набор страниц, через которые необходимо пройти. Класс, реализующий данный интерфейс, представляет собой хранилище и менеджер страниц, каждая из которых представляется классом, реализующим интерфейс `com.borland.primetime.wizard.WizardPage`. Данный интерфейс содержит всего три метода:

- `public javax.swing.JComponent getPageComponent(WizardHost host)` — данный метод возвращает GUI компонент представляющий текущую страницу.
- `public void activated(WizardHost host)` — данный метод вызывается при переходе в мастере к текущей странице.
- `public void deactivated()` — как вы догадались, данный метод вызывается в том случае, когда пользователь покидает текущую страницу.

А теперь внимание, вы уже заметили, что двум первым методам в качестве параметра передается объект типа `WizardHost`. На самом деле это интерфейс класса, совокупность методов которого служит для управления сеансом пользователя с мастером, т.е. вызывая соответствующие методы можно запрещать использовать те или иные кнопки для перехода к иным страницам мастера. Например, для того чтобы запретить выполнить переход к предыдущей странице можно воспользоваться методом:`public void setBackEnabled(boolean enabled)`.

А метод `public void setFinishEnabled(boolean enabled)` – может запрещать или разрешать действие кнопки "Finish" для текущей страницы.

Теперь давайте соберем вместе все что мы знаем и постараемся выработать стратегию создания мастера.

1. Необходимо зарегистрировать класс который будет представлять собой `WizardAction`, делается это с помощью вызова статического метода: `WizardManager.registerWizardAction(WizardAction)`
2. В качестве входного параметра для метода `registerWizardAction` следует использовать класс расширяющий собой `WizardAction` и соответственно имеющий метод protected Wizard `createWizard()`. Результатом исполнения данного метода должен быть класс, который реализует интерфейс `Wizard`. Однако для удобства я рекомендую создать предка для вспомогательного класса `com.borland.primetime.wizard.BasicWizard`, данный класс уже реализует ранее упомянутый интерфейс `Wizard`, выполняя ряд рутинной работы, по хранению массива страниц мастера и реакцию по умолчанию на переход между этими страницами. Хотя ничего не мешает нам просто перекрыть нужные методы, реализуя уникальность и специфику именно нашего мастера.
3. У нас в наличии должен быть набор классов реализующих GUI для каждой

страницы, все они должны быть производными от класса `javax.swing.JComponent`.

4. Каждый из классов, о которых я упоминал в шаге 3, должен иметь "упаковку" в виде класса реализующего интерфейс `WizardPage` и метод определенный в данном интерфейсе `public javax.swing.JComponent getPageComponent(WizardHost host)` должен быть реализован так, чтобы возвращать экземпляр "обертываемого" класса. Плюс того, как я уже говорил, реализуя методы данного интерфейса можно выполнить настройку реакции приложения на переход от одной страницы к другой, вплоть до того как будет нажата кнопка "Finish" и работа мастера должна быть завершена.
5. Теперь мы вернемся к шагу 2, и на стадии создания нашего класса, который производен от `BasicWizard`, следует вызвать метод `addWizardPage` для каждой из страниц нашего мастера, необходимая информация о способах создания страниц определена на шагах 3 и 4.
6. А теперь нам следует решить, где сосредоточить объем кода выполняющего работу мастера, ведь то что мы делали до сих пор, это было определение и связывание в единое целое страниц на которых пользователь вводил по шагам необходимые данные для успешной завершения работы мастера. В общем случае у нас два пути либо реализовать набор действий в классе `WizardPage` либо, и я этим приемом всегда пользуюсь, перекрыть метод `protected void finish()` нам уже очень хорошо знакомого класса `BasicWizard`.

Разумеется, что приведенная схема очень примитивна, так я не упомянул о специальных методах, которые присутствуют в классе `BasicWizard` и служат для управления направлением перехода пользователя по страницам мастера. Но, тем не менее, информация вполне применима, и давайте в качестве несложной демонстрации я сейчас создам код мастера. Работа которую будет выполнять наш мастер не слишком сложна, он должен будет служить для формирования строк форматирования даты/времени. Как известно, для того чтобы выполнять преобразование строки в тип `java.util.Date` и в обратном направлении служит класс `SimpleDateFormat`. В качестве параметров конструктора которому необходимо будет передать строку со специальными метасимволами, обозначающими те или иные части даты/времени. Что же давайте, попробуем упростить эту и так очень простую задачу.

```
package com.javable.www.columns.dbtables;
```

```
import com.borland.primetime.wizard.*;
import com.borland.primetime.wizard.WizardHost;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.JComponent;


public class XDFormatter
    extends JPanel
    implements WizardPage
{

    BorderLayout borderLayout1 = new BorderLayout();
    JScrollPane jScrollPane1 = new JScrollPane();
    JTextPane txtHint = new JTextPane();
    JButton btnTryFormat = new JButton();
    JPanel jPanel1 = new JPanel();
    JLabel labMes1 = new JLabel();
    GridLayout gridLayout1 = new GridLayout();
    JTextField txtFormat = new JTextField();

    public XDFormatter()
    {

        try
        {
            jbInit();
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }

    void jbInit()
         throws Exception
    {
```

```
        gridLayout1.setColumns(1);
        gridLayout1.setRows(0);
        jPanel1.setLayout(gridLayout1);
        labMes1.setText(
                "Используйте подсказки для конструирования формата даты времени");
        this.setLayout(borderLayout1);
        txtHint.setToolTipText("");
        txtHint.setText("");
        btnTryFormat.setText("Попробовать");
        btnTryFormat.addActionListener(new XDFormatter_btnTryFormat_actionAdapter(
                                                this));
        txtFormat.setText("");
        this.add(jScrollPane1,
                 BorderLayout.CENTER);
        jScrollPane1.getViewport().add(txtHint,
                                       null);
        this.add(btnTryFormat, BorderLayout.SOUTH);
        this.add(jPanel1, BorderLayout.NORTH);
        jPanel1.add(labMes1, null);
        jPanel1.add(txtFormat, null);

        java.net.URL ul = this.getClass().getClassLoader()
            .getResource("hint.html");
        txtHint.setPage(ul);
        txtHint.setEditable(false);
        setSize(400, 250);
    }

    void btnTryFormat_actionPerformed(ActionEvent e)
    {

        try
        {

            java.text.SimpleDateFormat sdf = new java.text.SimpleDateFormat(txtFormat.g
            String ft = sdf.format(new java.util.Date());
            JOptionPane.showMessageDialog(this,
                                          ft,
                                          "Результат преобразования",
                                          JOptionPane.INFORMATION_MESSAGE);
```

```
        }
        catch (Exception ex)
        {
            JOptionPane.showMessageDialog(this,
                                          ex +
                                          "\n" +
                                          ex.getMessage(),
                                          "Ошибка преобразования",
                                          JOptionPane.ERROR_MESSAGE);
        }
    }

    public void deactivated()
    {
    }

    public void activated(WizardHost wizardHost)
    {
    }

    public JComponent getPageComponent(WizardHost wizardHost)
    {

        return this;
    }

    public String getFormat()
    {

        return txtFormat.getText();
    }
}

class XDFormatter_btnTryFormat_actionAdapter
    implements java.awt.event.ActionListener
{

    XDFormatter adaptee;

    XDFormatter_btnTryFormat_actionAdapter(XDFormatter adaptee)
```

```
    {
        this.adaptee = adaptee;
    }

    public void actionPerformed(ActionEvent e)
    {
        adaptee.btnTryFormat_actionPerformed(e);
    }
}
package com.javable.www.columns.dbtables;

import com.borland.primetime.wizard.*;

import javax.swing.*;


public class XDFWizard
    extends BasicWizard
{

    XDFormatter wiz;

    public XDFWizard()
    {
        super();
        super.addWizardPage(
                wiz = new XDFormatter());
    }

    public void finish()
    {
        JOptionPane.showMessageDialog(null,
                                      "Мастер завершил подготовительную работу,
   строка формата была скопирована в буфер обмена");

        String rez = wiz.getFormat();
        java.awt.Toolkit.getDefaultToolkit().getSystemClipboard()
                .setContents(new java.awt.datatransfer.StringSelection(
                                    rez), null);
    }
```

```
    public String getWizardTitle()
    {

        return "XDF — Easy Create Date/Time Format";
    }
}
package com.javable.www.columns.dbtables;
import com.borland.primetime.wizard.*;

import java.net.*;

import javax.swing.*;


public class DFFormatterInstaller
{
    public static void initOpentool(byte majorVersion,
                                    byte minorVersion)
    {

        URL ul_s = DFFormatterInstaller.class.getClass()
                         .getClassLoader().getSystemResource(
                          "df-wizard-pic-small.PNG");
        ImageIcon ico_s = new ImageIcon(ul_s);
        WizardManager.registerWizardAction(new WizardAction(
                                "Simple XDate/Time Format Wizard (Merlin project)",
                                'x',
                                "You can create Date/Time conversion strings
                                with aid of this wizard",
                                                ico_s,
                                                ico_s,
                                                false) {
            protected Wizard createWizard()
            {

                return new XDFWizard();
            }
        });
        System.out.println(
```

```
                "Success Create XDF Wizard (Merlin Project)");
    }
}
```

Ниже я привожу текст файла манифеста для созданного нами архива Opentool.

```
Opentools-Wizard:  com.javable.www.columns.dbtables.DFFormatterInstaller
```

Да, кстати, чуть не забыл. Есть маленькие подводные камни в случае, когда вы пытаетесь одновременно в одном файле манифеста поместить описание нескольких однотипных Opentool. Если вы бы захотели выполнить инсталляцию мастеров Wiz1, Wiz2, Wiz3, и написали бы следующий фрагмент текста.

```
Opentools-Wizard:  FooPackage.Wiz1
Opentools-Wizard:  FooPackage.Wiz2
Opentools-Wizard:  FooPackage.Wiz3
```

То в результате сборки файла архива "*.jar" из всех ссылок осталась только одна, последняя. Для недопущения подобной ситуации рекомендуется использовать следующий формат:

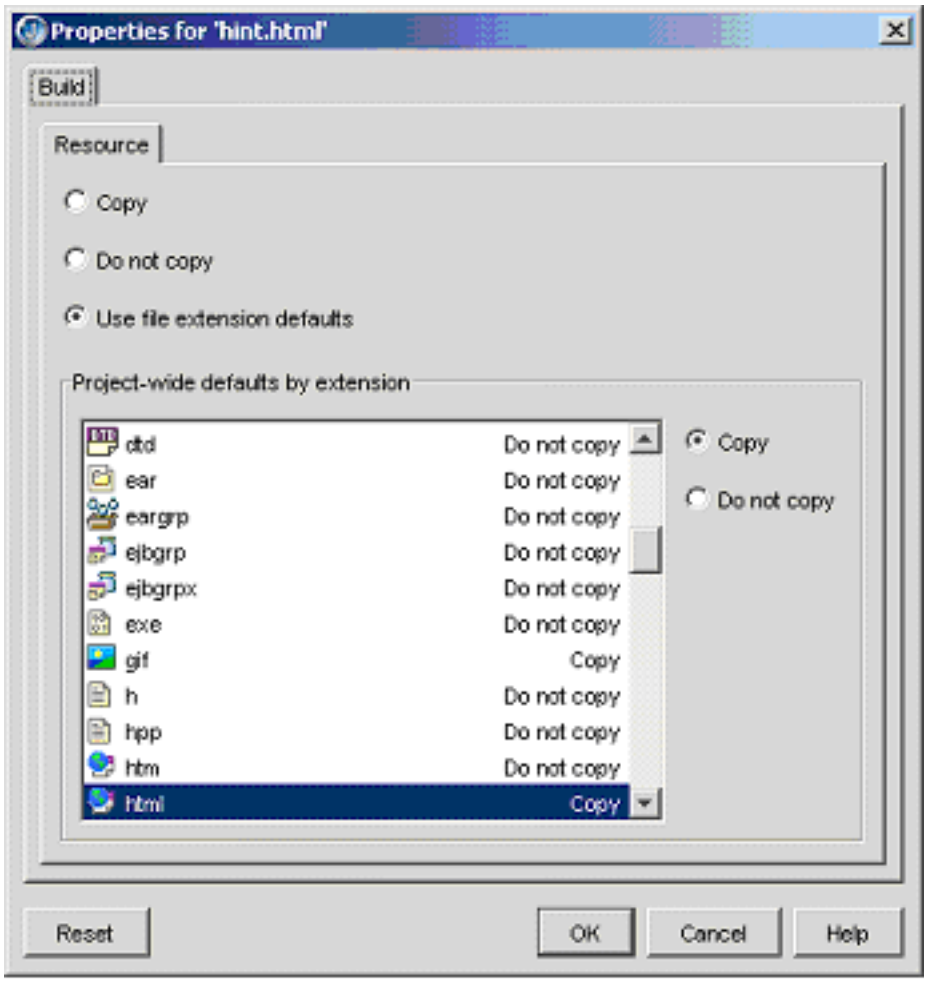
```
Opentools-Wizard:  FooPackage.Wiz1
  FooPackage.Wiz2
  FooPackage.Wiz3
```

Да, кстати, и не забывайте про обязательную пустую строку в конце всего содержимого файла. Для того чтобы пример успешно заработал вам необходимо будет самостоятельно выполнить еще ряд шагов. Прежде всего, необходимо будет создать картинку, которая будет представлять наш Opentool, для меню "wizards". Не забудьте добавить ее в проект так, что при последующем вызове получить корректный URL изображения.

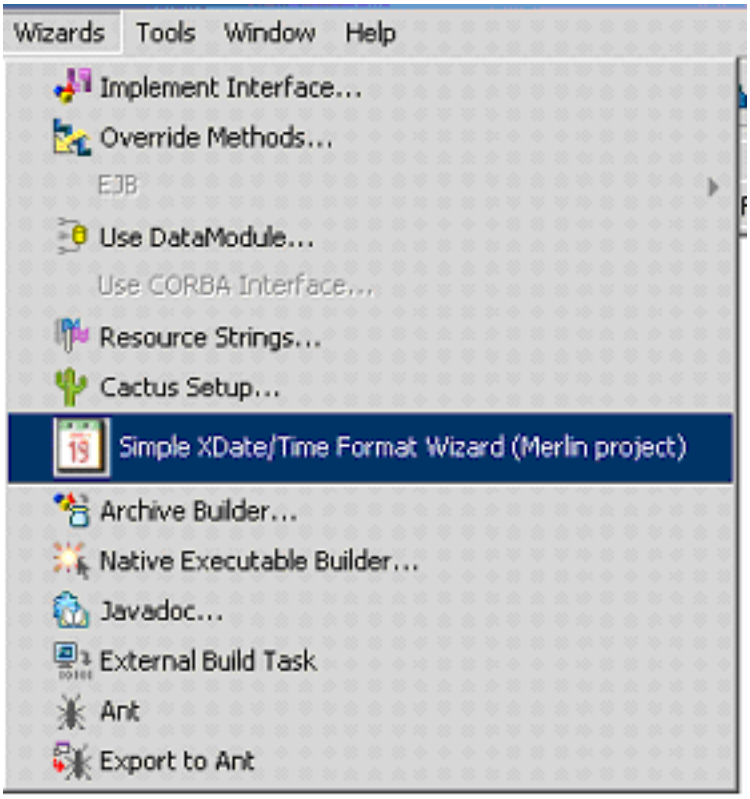
```
URL ul_s = DFFormatterInstaller.class.getClass().
getClassLoader().getSystemResource("df-wizard-pic-small.PNG");
```

Но это еще не все, как вы думаете откуда я взял текст документации по символам метаподстановки. Разумеется из справочной системы. Вам будет необходимо зайти в каталог где у вас установлена справка к java и найти там файл "SimpleDateFormat.html" (в крайнем случае можно воспользоваться следующим приемом: откройте справку JBuilder, найдите раздел по классу `SimpleDateFormat` и внизу в строке состояния посмотрите на URL открытого файла справки, ну а дальше дело практики). Получив в свое распоряжение файл справки, вам будет необходимо добавить его к своему проекту, так чтобы он был помещен в архив вашего Opentool. Я этот файл

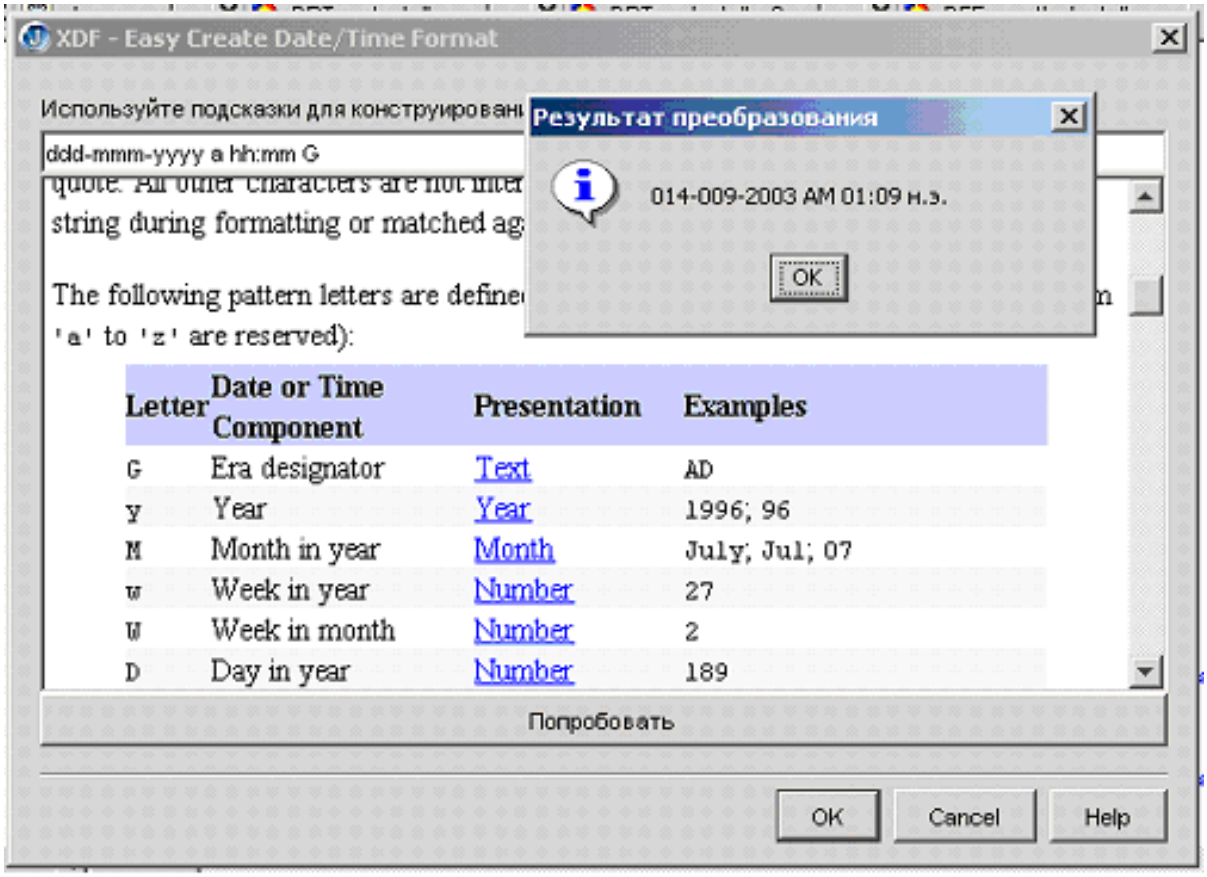
переименовал в "hint.html" вырезал из него части не представляющие для меня интереса, и вместе с файлом css-стилей добавил к проекту. Да кстати, по умолчанию файлы с расширением html не помещаются в архивы при сборке проекта, это поведение следует переопределить с помощью окна свойств для данного файла, свойства следует вызывать по контекстному меню для соответствующего узла проекта.

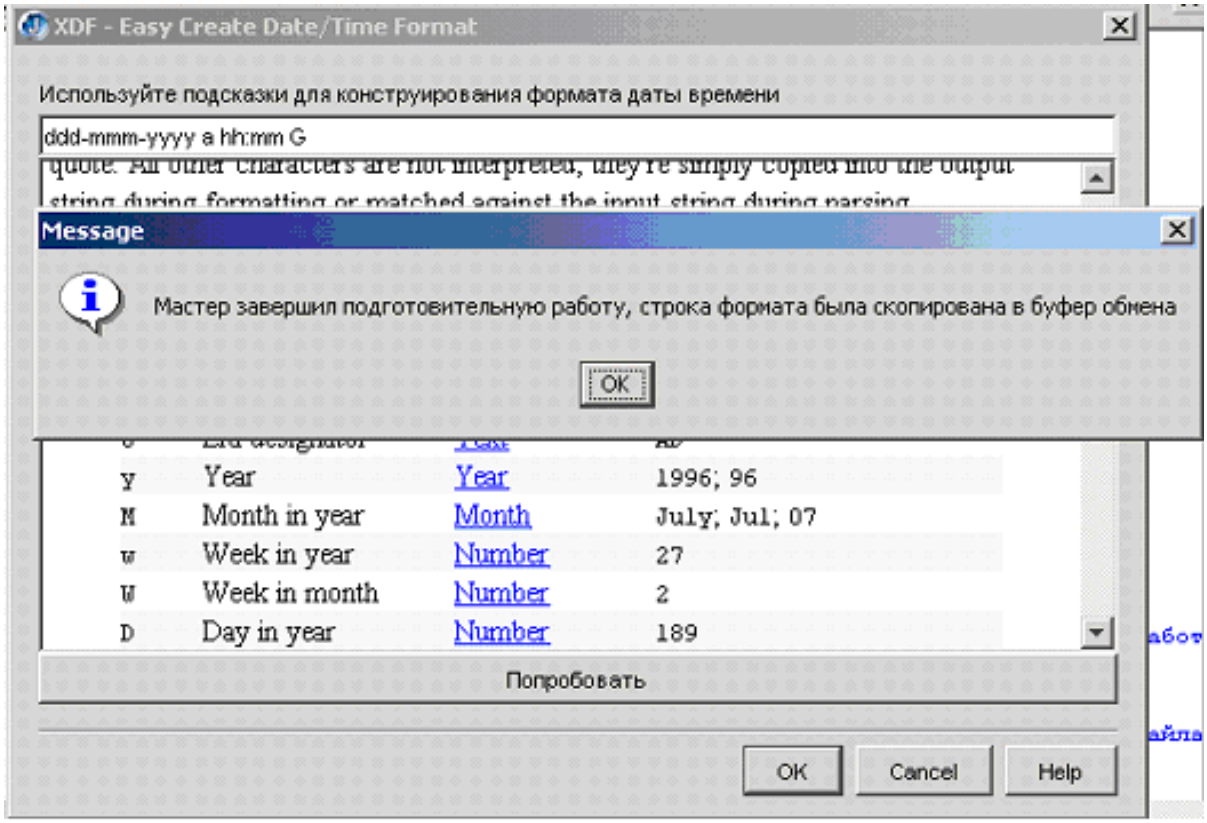


Ну вроде все, теперь можно полюбоваться примером работы нашего проекта. Ниже по порядку я привожу изображение нового пункта меню, который появился после инсталляции нашего opntool.



Далее я привожу изображения с появившимся окном мастера и примеры его работы.





4. Заключение

Конечно, все, что мне удалось сегодня рассказать — это всего малая доля все возможностей, которые представляют собой Opentool. Сегодня мы продолжили знакомство с основными концепциями при разработке Opentool. Мы попробовали взаимодействовать с объектной моделью, управляли наборами панелей кнопок и меню. И даже создали собственный не очень сложный, но все-таки мастер. А закончу я свое повествование на несколько философской ноте, что не стоит делать те Opentool, которыми не будешь пользоваться сам, ведь и другие тогда наверняка ими не заинтересуются.

5. Ресурсы

Увы, но единственное что я могу порекомендовать для чтения это справочная система

JBuilder раздел “Opentools Documentation”. Хочется верить что, начиная с этой статьи, в сети станет несколько больше руководств по разработке Opentools.