Winform分页控件

使用说明书

V0.1

序号	修改人	修改日期	修改后版本	修改说明
1	伍华聪	2012-6-25	V0.1	初稿

目 录

1. 引言

1.1. 背景

在 Winform 程序开发中，分页是永恒的话题，因为需要显示的数据总是很多很多，分页展示在程序性能和数据查看感官方面得到很好的平衡，是一种良好的编程习惯和 UI 设计。Winform 中的分页控件可能没有 Asp.net 世界中的分页控件那么丰富多彩，不过也有不少的分页控件可以采用，各个人的可能都有一些不同的东西，一些好的东西。就我而言，我希望控件能够尽可能的多一些功能，耦合性低一些，例如我不想是基于存储过程的，因为我很多程序需要使用 Access 作为数据库，一般来说，我还希望有导出 Excel 数据的功能，还有打印预览功能，由于我的数据源表头，如实体类集合、表格内容绑定的时候，表头是英文的，我需要变为中文的，其他的功能有则更好。一句话：良好封装、功能丰富，使用简便。

基于这个思想，我在 Winform 开发框架中引入一个 Winform 分页控件模块，这个控件能满足我绝大多数的分页应用及界面要求。该控件已经在我的共享软件中大量使用并得到升华，如 Winform 开发框架、WCF 开发框架、酒店管理系统、备件仓库管理系统、送水管理系统软件、病人资料管理软件，以及其他为单位或个人定做的各种业务系统等等。

在软件开发过程中，为了节省开发时间，提高开发效率，统一用户处理界面，尽可能使用成熟、功能强大的分页控件，这款 Winform 环境下的分页控件，集成了数据分页、内容提示、数据打印、数据导出、表头中文转义等很多功能，由于集成性很好，省却很多功夫，专注客户的业务及变化即可，否则一项表头的中文转换就够呛，还不说数据的分页，由于整合性、一致性、稳定性等特点，客户使用感觉比较好。

1.2. 编写目的

本文档主要介绍《Winform 分页控件》的特性以及如何在 Winform 业务系统中进行使用。

1.3. 参考资料

序号	名称	版本/日期	来源

1	《Winform 开发框架-架构设计说明书.doc》		内部
2			内部
3			内部
4			内部

1.4. 术语与缩写

1 在本文件中出现的“系统”一词，除非特别说明，均适用于《WCF开发框架》、《Winform开发框架》。

2 在本文安装.NET框架中，除非特别说明，均指.NET 2.0框架。

2. Winform 分页控件

2.1. 控件特性

本 Winform 分页控件具有下面几个特点：

1）本分页控件主要是实现 Winform 下方便快捷、功能集成、高效友好的数据分页，分页控件和具体的数据库无关，支持各种数据库的分页解决方案，如 Access、Sqlite、SqlServer、MySql、Oracle 等等。

2）支持文件复制或者对压缩包进行解压，适合更多琐碎程序集的整体升级。

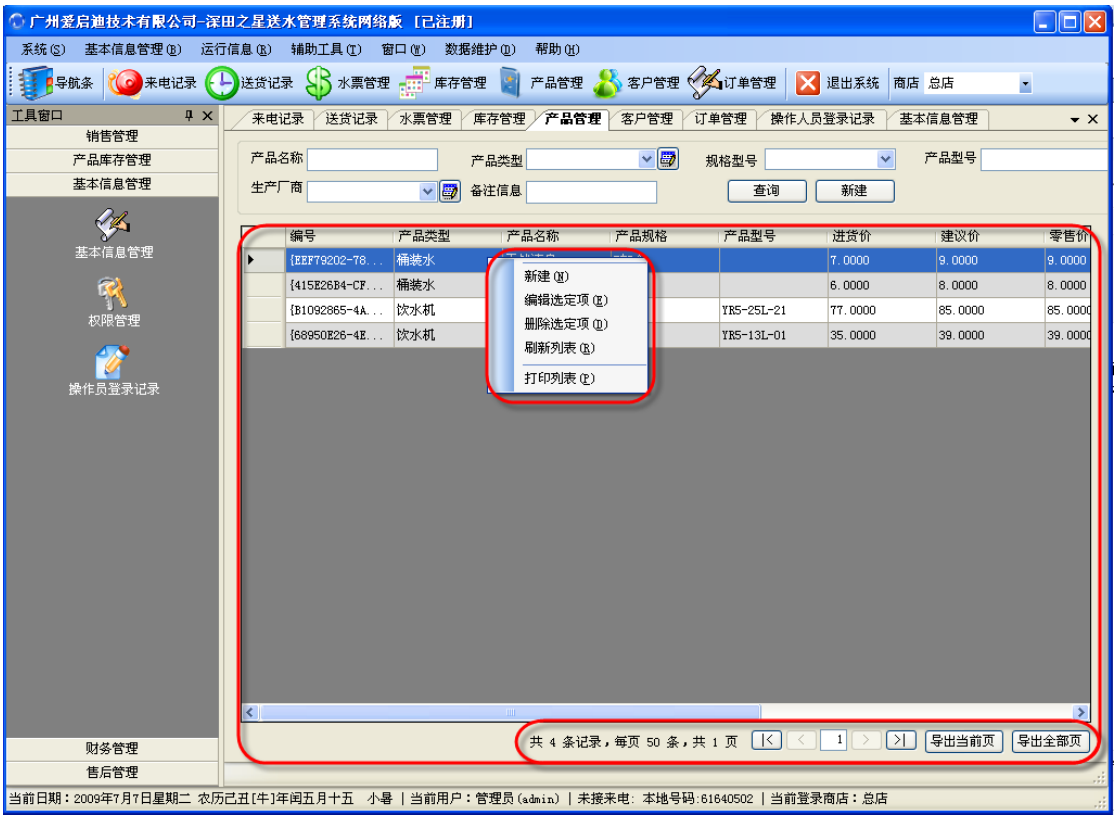
3）控件集成除了基本的分页外，还有其他功能，如导出当前页、导出全部页、打印列表、以及相关功能操作的菜单（只要实现了相关的接口，则呈现相同的菜单）。

4）另外还有一些小地方，也是很常用关键的地方，就是间隔行的颜色变化设置，表头的中文化，表字段顺序可调整，行提示内容，是否显示行号、页面显示记录大小等基本界面功能。

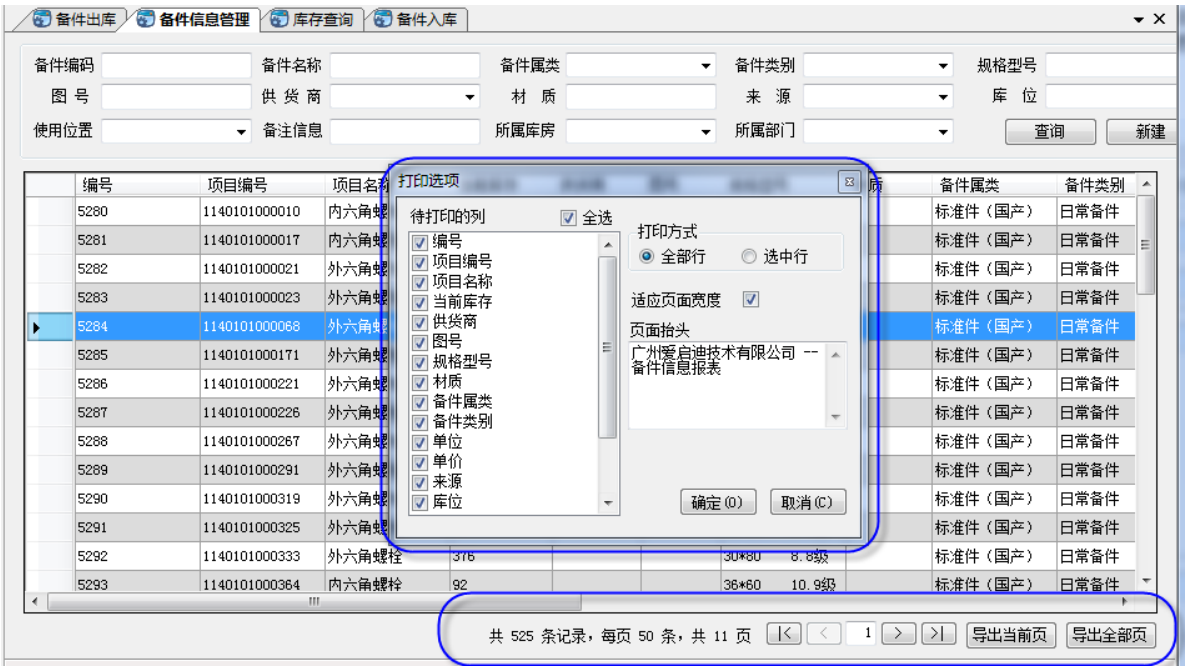
5）支持表头勾选操作，方便一些特殊界面处理，需要勾选而不是鼠标框选操作。

2.2. 使用效果介绍

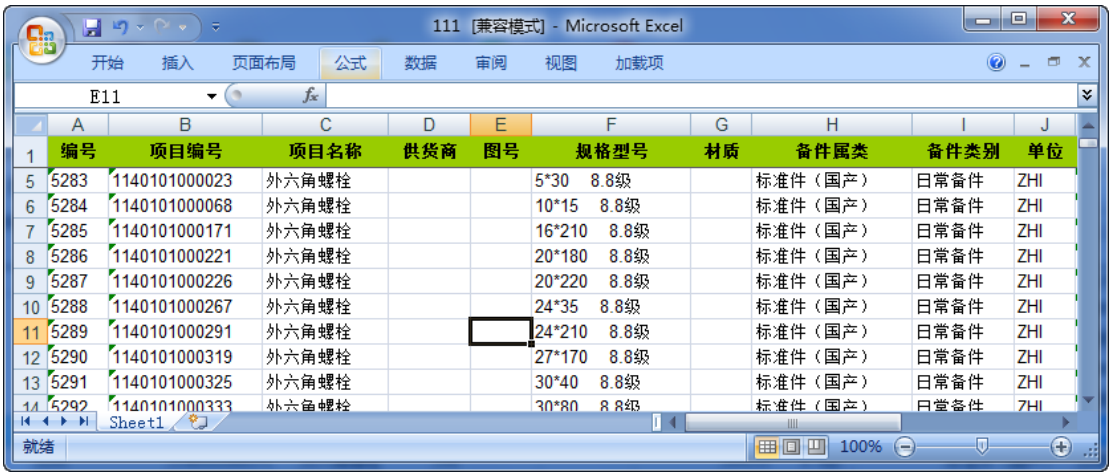
1） 控件使用整体界面截图。



2） 分页控件支持选定指定的列内容进行打印操作。

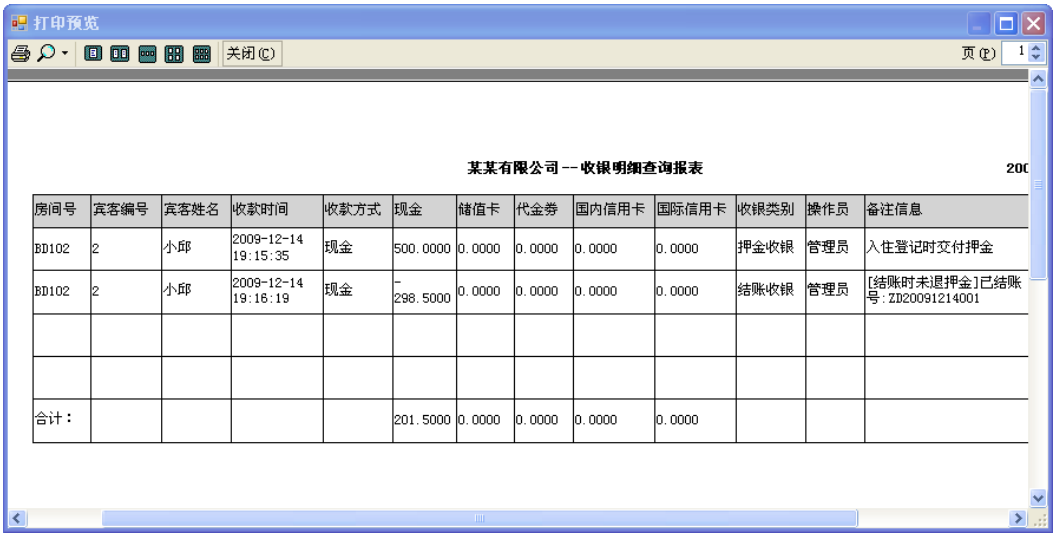


3） 导出 Excel 是基本的功能，本控件支持当前页导出，全部页导出两种模式。导出的 Excel 数据也还是比较好看的，不是一般的格式哦。该分页控件整合了优秀的 Aspose.Cell 控件来进行 Excel 数据的导出，速度非常快，而且默认表头冻结，非常方便。



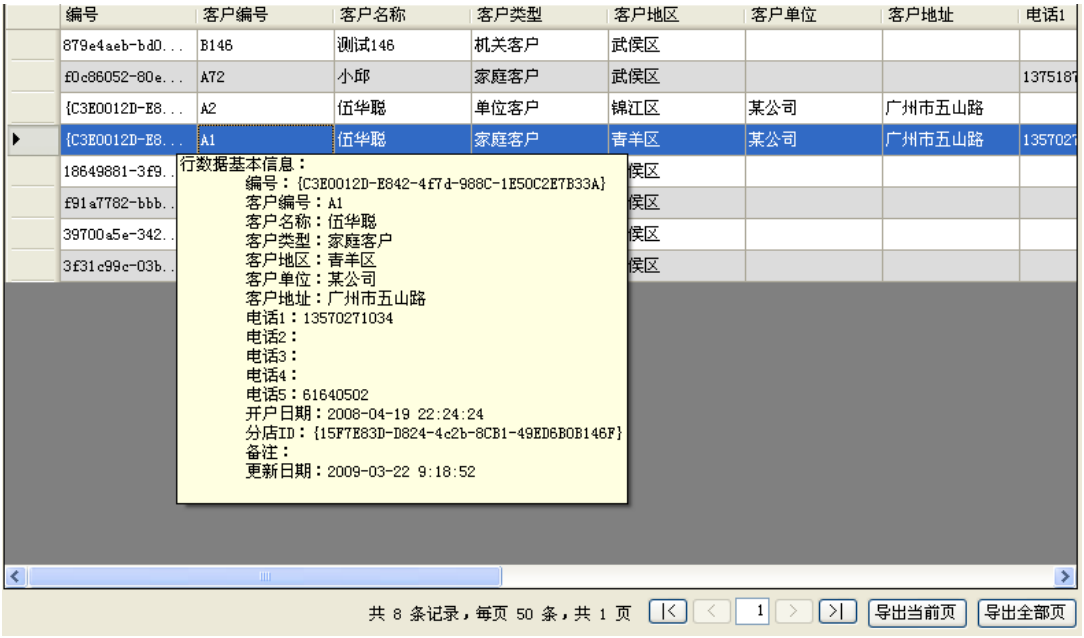
	A	B	C	D	E	F	G	H	I	J
1	编号	项目编号	项目名称	供货商	图号	规格型号	材质	备件属类	备件类别	单位
5	5283	1140101000023	外六角螺栓			5*30 8.8级		标准件（国产）	日常备件	ZHI
6	5284	1140101000068	外六角螺栓			10*15 8.8级		标准件（国产）	日常备件	ZHI
7	5285	1140101000171	外六角螺栓			16*210 8.8级		标准件（国产）	日常备件	ZHI
8	5286	1140101000221	外六角螺栓			20*180 8.8级		标准件（国产）	日常备件	ZHI
9	5287	1140101000226	外六角螺栓			20*220 8.8级		标准件（国产）	日常备件	ZHI
10	5288	1140101000267	外六角螺栓			24*35 8.8级		标准件（国产）	日常备件	ZHI
11	5289	1140101000291	外六角螺栓			24*210 8.8级		标准件（国产）	日常备件	ZHI
12	5290	1140101000319	外六角螺栓			27*170 8.8级		标准件（国产）	日常备件	ZHI
13	5291	1140101000325	外六角螺栓			30*40 8.8级		标准件（国产）	日常备件	ZHI
14	5292	1140101000333	外六角螺栓			30*80 8.8级		标准件（国产）	日常备件	ZHI

4） 报表打印界面。控件一项功能，也是集实用功能之所成，打印当前列表内容，如下图所示，该内容会保存用户在每个列表数据中的信息，打印不同的表头内容，如下图所示。总体该功能上就是我们一般报表所需要的功能。其中报表打印预览可以设置报表标题，打印的列也可以设定，有一些字段的汇总功能，而且这样的报表基本上不需要额外的代码就能实现（相对分页控件来说）。



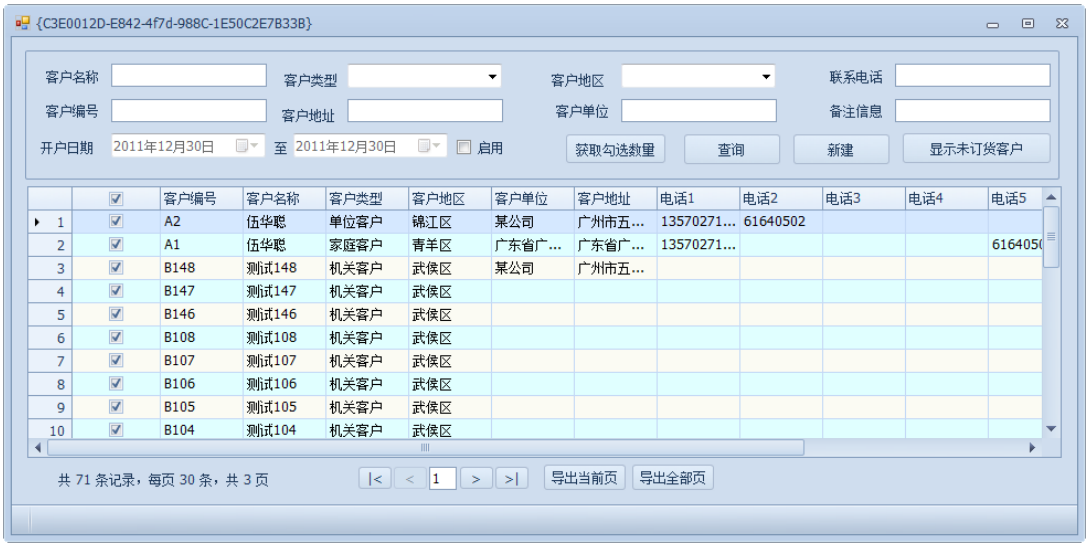
房间号	宾客编号	宾客姓名	收款时间	收款方式	现金	储值卡	代金券	国内信用卡	国际信用卡	收银类别	操作员	备注信息
BD102	2	小邱	2009-12-14 19:15:35	现金	500.0000	0.0000	0.0000	0.0000	0.0000	押金收银	管理员	入住登记时交付押金
BD102	2	小邱	2009-12-14 19:16:19	现金	-298.5000	0.0000	0.0000	0.0000	0.0000	结账收银	管理员	[结账时未退押金]已结账号:ZD20091214001
合计：					201.5000	0.0000	0.0000	0.0000	0.0000			

5） 列内容自动提示。由于我们设置了中文表头，另外一项便利的功能就是，当鼠标停放在某一行的时候，出现改行内容的信息提示，这样可以方便用户了解一些详细的信息，如下所示。

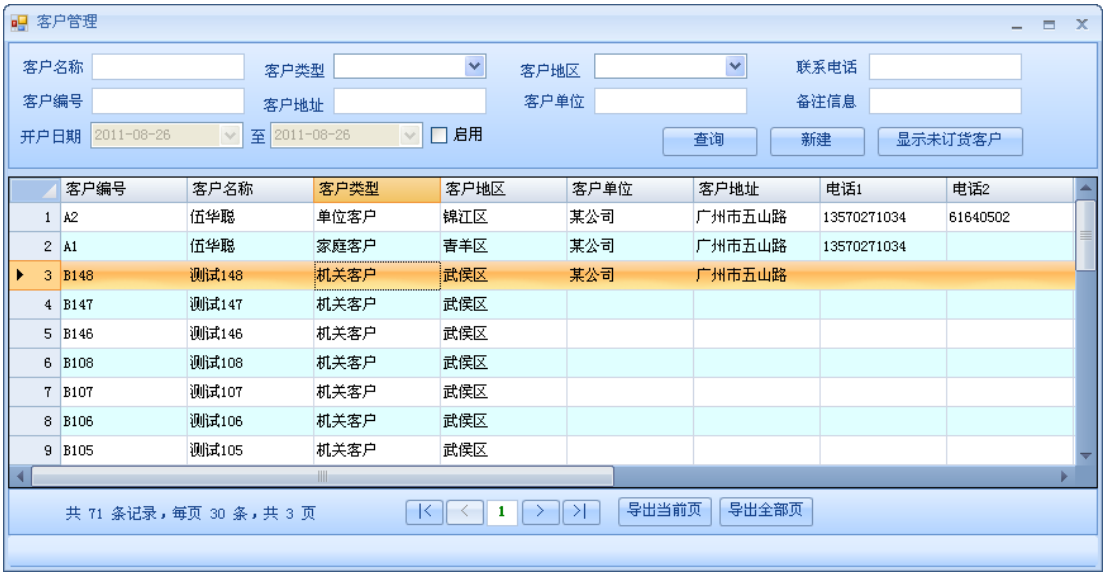


6） 支持表头全选操作。表头全选操作只需要一行代码即可实现。

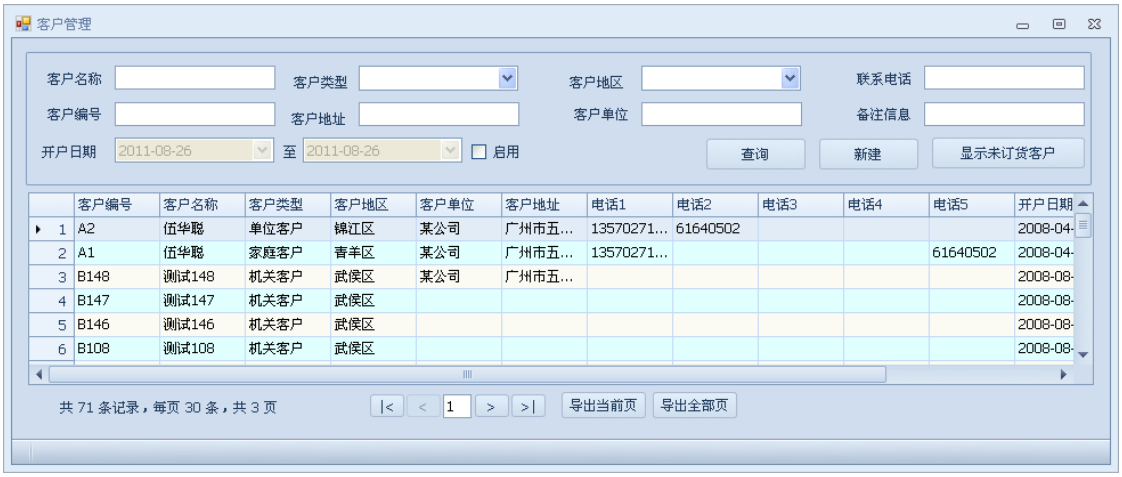
```
this.winGridViewPager1.ShowCheckBox = true;
```



7） 支持基于 dotNetbar 控件界面的分页。由于本人随笔《Winform 开发框架之 Office Ribbon 界面》 引入了基于 DotNetbar 的 Winform 开发框架，因此在此基础上利用 DotNetBar 的界面效果特性，修改了原来的分页控件，提供基于 DotNetbar 控件效果的分页控件支持，效果如下所示。



8） 分页控件除了支持传统效果、DotNetBar 效果外，还是支持目前应用很广泛的 DevExpress 界面控件，这几种模式的控件使用属性及方法 99%以上是一致的。



2.3. 控件使用

2.3.1. 添加到 Visual Studio 的工具箱

1）在 Visual Studio 开发环境的工具箱中，添加一个分页控件（可以其他名称）的项目，然后选择 WHC.Pager.WinControl.dll（DevExpress 版本的为 WHC.Pager.WinControlDx.dll）文件，导入分页控件的工具箱图标，如下所示。

其中 WinGridViewPager 和 WinGridView 两个控件就是我们常用到的分页控件，两者用法几乎一致，界面效果不同在于 WinGridViewPager 具有分页工具条，而 WinGridView 是提供显示所有内容，没有工具条。

2）拖动到指定的窗体中。

3）在代码引用相关的代码实现动态调用。

2.3.2. 控件相关菜单

在窗体加载实现中添加分页控件的实现代码，以 On 开始的是相关操作的实现事件，如 OnPageChanged 表示分页控件页面发生变化的时候，需要实现的事件处理（这个是必须的），另外 AppendedMenu 是可以在分页控件自带菜单上增加的菜单，如下所示。注意，只要实现相关的事件处理，那么对应的上下文菜单将会出现，默认几个标准的上下文菜单如下所示，包含新建、编辑选定项、删除选定项、打印列表、刷新列表菜单。

注意，为了使分页控件能够显示总数，并记住当前的分页，那么在 OnPageChanged 实现中需要修改分页控件的这两个属性。

默认菜单需要实现相关的事件，才会显示出来，如新建菜单，需要实现事件 OnAddNew 的事件处理，“编辑选定项”菜单需要实现 OnEditSelected 事件处理，删除菜单需要实现 OnDeleteSelected 事件处理。

C#代码

```
this.winGridViewPager1.OnAddNew += new EventHandler(winGridViewPager1_OnAddNew);
this.winGridViewPager1.OnEditSelected += new EventHandler(winGridViewPager1_OnEditSelected);
this.winGridViewPager1.OnDeleteSelected += new EventHandler(winGridViewPager1_OnDeleteSelect
ed);
```

如果需要添加自己的菜单，则指定 AppendedMenu 对象即可。

标准的分页控件初始化代码如下所示：

C#代码

```
    public FrmCustomer()
    {
    InitializeComponent();

    if (!this.DesignMode)
    {
    this.winGridViewPager1.OnPageChanged += new EventHandler(winGridViewPager1_OnPageChanged);
    this.winGridViewPager1.OnStartExport += new EventHandler(winGridViewPager1_OnStartExport);
  this.winGridViewPager1.OnEditSelected += new EventHandler(winGridViewPager1_OnEditSelected);
    this.winGridViewPager1.OnDeleteSelected +=
       new EventHandler(winGridViewPager1_OnDeleteSelected);
        this.winGridViewPager1.OnRefresh += new EventHandler(winGridViewPager1_OnRefresh);
        this.winGridViewPager1.OnAddNew += new EventHandler(winGridViewPager1_OnAddNew);
        this.winGridViewPager1.AppendedMenu = this.contextMenuStrip1;
        this.winGridViewPager1.ShowLineNumber = true;//显示行号
        this.winGridViewPager1.PagerInfo.PageSize = 30;//页面大小
        this.winGridViewPager1.EventRowBackColor = Color.LightCyan;//间隔颜色
        this.winGridViewPager1.ShowCheckBox = true;//全选操作
    }
       }
```

2.3.3. 分页控件表头中文转义及显示字段控制

控件表头的中文转义是通过函数 `AddColumnAlias` 为字段添加别名进行转义的，如下所示。

C#代码

```
private void BindData()
{
    #region 添加别名解析
    this.winGridViewPager1.AddColumnAlias("ID", "编号");
    this.winGridViewPager1.AddColumnAlias("Number", "客户编号");
    this.winGridViewPager1.AddColumnAlias("Name", "客户名称");
    this.winGridViewPager1.AddColumnAlias("Type", "客户类型");
    this.winGridViewPager1.AddColumnAlias("Area", "客户地区");
    this.winGridViewPager1.AddColumnAlias("Company", "客户单位");
    this.winGridViewPager1.AddColumnAlias("Address", "客户地址");
    this.winGridViewPager1.AddColumnAlias("Telephone1", "电话 1");
    this.winGridViewPager1.AddColumnAlias("Telephone2", "电话 2");
    this.winGridViewPager1.AddColumnAlias("Telephone3", "电话 3");
    this.winGridViewPager1.AddColumnAlias("Telephone4", "电话 4");
    this.winGridViewPager1.AddColumnAlias("Telephone5", "电话 5");
    this.winGridViewPager1.AddColumnAlias("CreateDate", "开户日期");
    this.winGridViewPager1.AddColumnAlias("Shop_ID", "分店 ID");
    this.winGridViewPager1.AddColumnAlias("Note", "备注");
    this.winGridViewPager1.AddColumnAlias("LastUpdated", "更新日期");
    #endregion

    string where = GetSearchSql();
    this.winGridViewPager1.DataSource = BLLFactory<Customer>.Instance.Find(where, this.winGr
idViewPager1.PagerInfo);
    this.winGridViewPager1.dataGridView1.Refresh();
}
```

上面默认的操作时显示数据源里面所有的字段信息，如果需要控制显示的字段内容及顺序，那么使用 DisplayColumns 属性即可。这个属性指定字段的顺序就是控件表头字段的显示顺序。

C#代码

```
this.winGridViewPager1.DisplayColumns =
    "ID,ItemNo,ItemName,Manufacture,MapNo,Specification,StockQuantity,AlarmQuantity,WareHouse";
```

控件报表标题如下所示。

C#代码

```
this.winGridViewPager1.PrintTitle = Portal.gc.gAppUnit + " -- " + "备件信息报表";
```

2.3.4. 每页显示记录大小

分页控件可以通过属性设置每页显示多少条记录，如下代码所示：

C#代码

```
this.winGridViewPager1.PagerInfo.PageSize = 30;//页面大小
this.winGridViewPager1.ShowLineNumber = true;//显示行号
this.winGridViewPager1.EventRowBackColor = Color.LightCyan;//间隔颜色
```

2.3.5. 表头全选操作

表头全选操作，只需要设置一个属性 ShowCheckBox=True 即可显示。各个版本（传统、DotNetBar、DevExpress 版本均一致）。

C#代码

```
this.winGridViewPager1.ShowCheckBox = true;
```

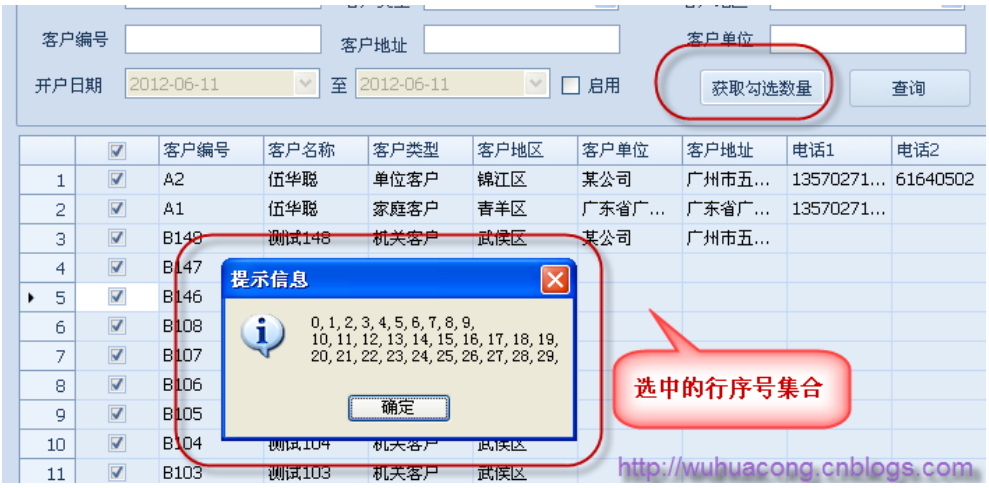
这样的简单调用就能实现控件表头全选的操作，不需要另外其他额外的操作，是不是很好，简洁呢？这其实就是我追求的分页控件表头全选的最优方法了。

最后，您可能还有一个疑问，就是我们全选或者部分选择，我要知道我选择的是那些行，该怎么操作呢？放心，我这个分页控件也已经为你考虑了，呵呵。

通过调用下面的代码即实现。

C#代码

```
private void btnGetCheckedRows_Click(object sender, EventArgs e)
{
    List<int> list = this.winGridViewPager1.GetCheckedRows();
    StringBuilder sb = new StringBuilder();
    int i = 1;
    foreach (int rowindex in list)
    {
        sb.Append(rowindex.ToString() + ",");
        if (i++ % 10 == 0)
        {
            sb.Append("\r\n");
        }
    }
    MessageUtil.ShowTips(sb.ToString());
}
```



2.3.6. 简单数据绑定

使用简单操作加载数据方式绑定分页控件，而不是基于我 Database2Sharp 自动生成的整个 EnterpriseLibary 架构的实现方式。

C#代码

```
private void BindData()
{
    #region 添加别名解析
    this.winGridViewPager1.AddColumnAlias("ID", "编号");
    this.winGridViewPager1.AddColumnAlias("Number", "客户编号");
    this.winGridViewPager1.AddColumnAlias("Name", "客户名称");
    this.winGridViewPager1.AddColumnAlias("Type", "客户类型");
    this.winGridViewPager1.AddColumnAlias("Area", "客户地区");
    this.winGridViewPager1.AddColumnAlias("Company", "客户单位");
    this.winGridViewPager1.AddColumnAlias("Address", "客户地址");
    this.winGridViewPager1.AddColumnAlias("Telephone1", "电话 1");
    this.winGridViewPager1.AddColumnAlias("Telephone2", "电话 2");
    this.winGridViewPager1.AddColumnAlias("Telephone3", "电话 3");
    this.winGridViewPager1.AddColumnAlias("Telephone4", "电话 4");
    this.winGridViewPager1.AddColumnAlias("Telephone5", "电话 5");
    this.winGridViewPager1.AddColumnAlias("CreateDate", "开户日期");
    this.winGridViewPager1.AddColumnAlias("Shop_ID", "分店 ID");
    this.winGridViewPager1.AddColumnAlias("Note", "备注");
    this.winGridViewPager1.AddColumnAlias("LastUpdated", "更新日期");
    #endregion

    string where = GetSearchSql();
    if (this.radNormal.Checked) //采用框架加载
    {
        this.winGridViewPager1.DataSource =
            BLLFactory<Customer>.Instance.Find(where, this.winGridViewPager1.PagerInfo);
    }
    else
    {
        //直接加载数据
        DataTable dt = DirectLoadData(where, this.winGridViewPager1.PagerInfo);
        this.winGridViewPager1.DataSource = dt.DefaultView;
    }

    this.winGridViewPager1.dataGridView1.Refresh();
}
```

C#代码

```
/// <summary>
/// 通过自己组装分页语句
/// </summary>
private DataTable DirectLoadData(string where, PagerInfo pagerInfo)
{
    DataTable dt = null;
    PagerHelper helper = new PagerHelper("All_Customer", "*", "ID", pagerInfo.PageSize, pagerInfo.CurrenetPageIndex, true, where);
    string countSql = helper.GetPagingSql(DatabaseType.SqlServer, true);
    string dataSql = helper.GetPagingSql(DatabaseType.SqlServer, false);

    string value = SqlValueList(countSql);
    pagerInfo.RecordCount = Convert.ToInt32(value);

    dt = SqlTable(dataSql);
    return dt;
}
```

2.3.7. 如何自定义数据列表的宽度。

传统界面和 DotNetBar 控件的，通过 dataGridView1.DataBindingComplete 的事件处理，实现自定义宽度设置；而 DevExpress 控件样式的分页控件，则通过 DataSourceChanged 事件进行处理。

C#代码

```
public partial class FrmItemDetail : BaseDock
{
    public FrmItemDetail()
    {
        InitializeComponent();
        InitDictItem();

        this.winGridViewPager1.OnPageChanged +=
                        new EventHandler(winGridViewPager1_OnPageChanged);
        this.winGridViewPager1.OnStartExport += new EventHandler(winGridViewPager1_OnStartExport);
        this.winGridViewPager1.OnEditSelected += new EventHandler(winGridViewPager1_OnEditSelected);
        this.winGridViewPager1.OnAddNew += new EventHandler(winGridViewPager1_OnAddNew);
        this.winGridViewPager1.OnDeleteSelected +=
                        new EventHandler(winGridViewPager1_OnDeleteSelected);
        this.winGridViewPager1.OnRefresh += new EventHandler(winGridViewPager1_OnRefresh);
        this.winGridViewPager1.AppendedMenu = this.contextMenuStrip1;
        this.winGridViewPager1.ShowLineNumber = true;
        this.winGridViewPager1.dataGridView1.DataBindingComplete +=
                    new DataGridViewBindingCompleteEventHandler(dataGridView1_DataBindingComplete);
    }

    /// <summary>
    /// 绑定数据后，分配各列的宽度
    /// </summary>
    void dataGridView1_DataBindingComplete(object sender, DataGridViewBindingCompleteEventArgs e)
    {
        if (this.winGridViewPager1.dataGridView1.Columns.Count > 0)
        {
            this.winGridViewPager1.dataGridView1.Columns["ID"].Width = 100;
            this.winGridViewPager1.dataGridView1.Columns["ItemNo"].Width = 100;
            this.winGridViewPager1.dataGridView1.Columns["ItemName"].Width = 100;
            this.winGridViewPager1.dataGridView1.Columns["Manufacture"].Width = 100;
            this.winGridViewPager1.dataGridView1.Columns["MapNo"].Width = 100;
            this.winGridViewPager1.dataGridView1.Columns["Specification"].Width = 100;
        }
    }
```

如果是 DevExpress 控件样式的的分页控件，要实现自定义宽度，需要下面的操作。

C#代码

```
            this.winGridViewPager1.BestFitColumnWith = false;
            this.winGridViewPager1.gridView1.DataSourceChanged +=
                                        new EventHandler(gridView1_DataSourceChanged);
        }

        private void gridView1_DataSourceChanged(object sender, EventArgs e)
        {
            if (this.winGridViewPager1.gridView1.Columns.Count > 0
                        && this.winGridViewPager1.gridView1.RowCount > 0)
            {
                this.winGridViewPager1.gridView1.Columns["ItemNo"].Width = 140;
                this.winGridViewPager1.gridView1.Columns["ItemBigType"].Width = 140;
                this.winGridViewPager1.gridView1.Columns["WareHouse"].Width = 140;
            }
        }
```

2.3.8. 如何实现多表关联的分页数据查询

PagerHelper 是分页控件里面内置的一个用于生成多种数据库不同的分页语句的类库。

C#代码

```
public DataTable SearchParkUser(string condition, PagerInfo pagerInfo)
{
    DataTable dt = null;
    string selectSql = string.Format(@"select u.*, p.current_space_number, to_char(p.sample_time, 'yyyy-mm-dd
hh24:mi:ss') as sample_time from device_user u
    left join device_parking p on u.park_id = p.park_id order by u.id desc ");
    condition += string.Format(" AND status=0 ");

    PagerHelper helper = new PagerHelper(selectSql, "*", "id",
                        pagerInfo.PageSize, pagerInfo.CurrenetPageIndex, true, condition);
    string countSql = helper.GetPagingSql(DatabaseType.Oracle, true);
    string dataSql = helper.GetPagingSql(DatabaseType.Oracle, false);

    string value = SqlValueList(countSql);
    pagerInfo.RecordCount = Convert.ToInt32(value);
    dt = SqlTable(dataSql);

    return dt;
}
```

2.3.9. 如何实现附加语句查询

SearchCondition 控件是公用类库里面的对象，功能是快速构造相关的查询语句，支持多种数据库语句生成。

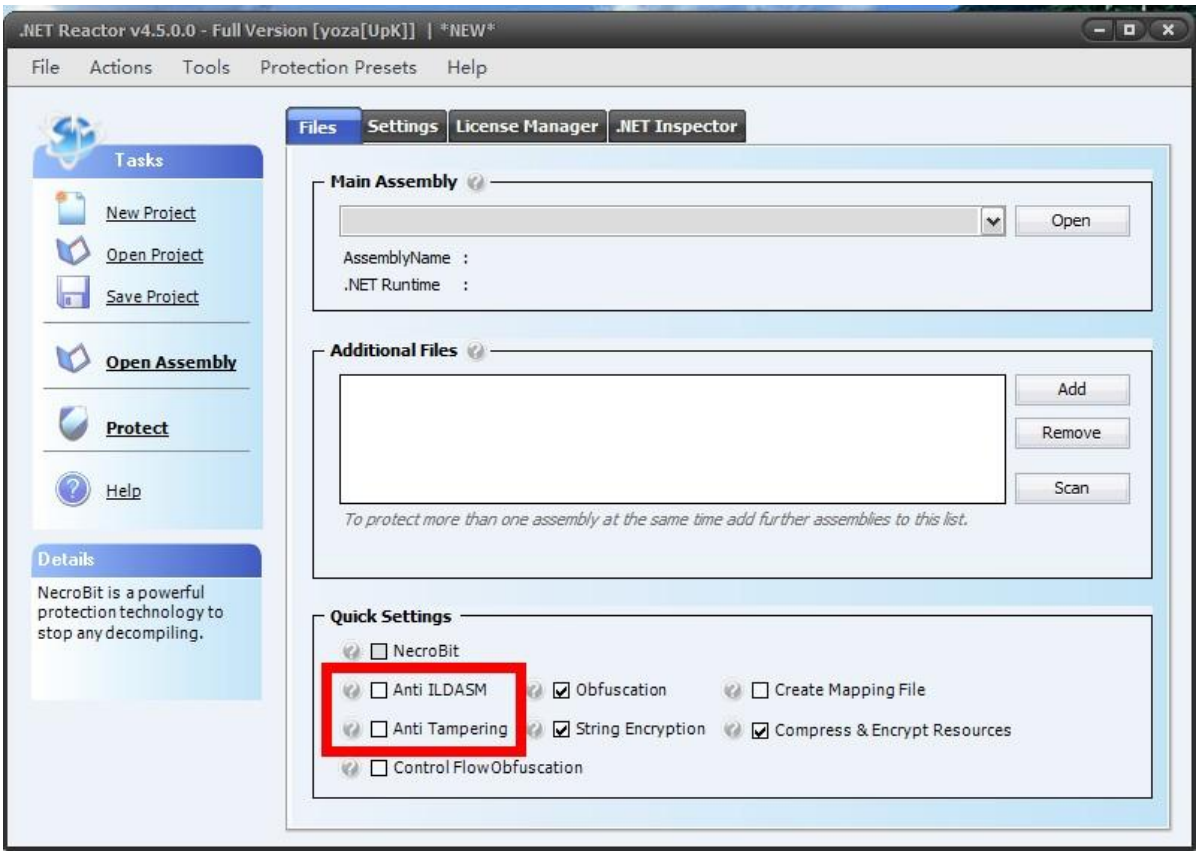
C#代码

```
private string GetSearchSql()
{
    SearchCondition condition = new SearchCondition();
    condition.AddCondition("COMPANY_CODE", this.txtCompanyCode.Text, SqlOperator.Like)
        .AddCondition("COMPANY_NAME", this.txtCompanyName.Text, SqlOperator.Like)
        .AddCondition("LICENSE", this.txtLicense.Text, SqlOperator.Like)
        .AddCondition("PARK_NAME", this.txtParkName.Text, SqlOperator.Like)
        .AddCondition("PARK_ADDR", this.txtParkAddr.Text, SqlOperator.Like)
        .AddCondition("STATUS", this.txtStatus.Text, SqlOperator.Equal)
        .AddCondition("AREA_COUNTRY", this.txtAreaCountry.Text, SqlOperator.Like);
    string where = condition.BuildConditionSql(DatabaseType.Oracle);
    if (this.chkSpaceUp.Checked)
    {
        where += " AND CURRENT_SPACE_NUMBER >= PARK_SPACE_NUMBER";
    }
    if (this.chkLatLong.Checked)
    {
        where += " AND (LATITUDE is null or LATITUDE=0)";
    }
    if (chkSpaceNumber.Checked)
    {
        where += " AND CURRENT_SPACE_NUMBER is not null ";
    }
    if (this.chkHourOnline.Checked)
    {
        where += " AND SPACE_LAST_UPDATE >= sysdate - 1/24";
    }
    if (this.chkContact.Checked)
    {
        where += " AND (CONTACT is null OR CONTACT_TELEPHONE is null)";
    }

    return where.Replace("Where", "");
}
```

2.4. 分页控件使用注意事项

特别说明：如果在 VS2010 中打开含有分页控件的窗体，出现错误，请大家使用 DotNetReactor 混淆的时候，不要勾选 Anti ILDASM 项目，或者不要使用混淆也可以。



在实际项目使用过程中，发现 VS2010 和 VS2008 不同，如果不能 ILDASM，那么会出现异常，正常打包项目好像都会受影响的，不能刷新 dll 的依赖项，因此对于一些项目控件，尽量不用勾选上面的那两个 Anti 项目。