15 业务码

诸如营运、生产之类的企业当中，会流转、处理大批量、各种类型的单证、产品，在原始的手工处理场景下，为了识别、区分、归档这些各类、有序的纸质单证和产品，一般都会为它们打上（标记）流水号，一个流水号就是这个单证、产品的身份证和识别码。

由于流水号是提供给人工处理的识别码，所以包含了方便人工识别的信息，比如会有规律地夹杂进单证类型、制作部门、产生单证的年月日等。在基于信息系统支撑下的企业管理中，流水号并没有被废弃，反而是人机交互的信息桥梁和便捷手段。虽然现今有了条码识别和射频技术，可以提高系统的自动化程度，但流水号仍然没有被替代，这是因为在大多数业务场景下，还没有完全做到全自动化的作业流程，这个事实所决定的。因此，流水号仍然在现有的应用系统中被大量使用着。

流水号的生成是有一定规律的，为了实现快速开发、减轻开发负担的目的，Phenix丶为此做了抽象和提炼，实现了流水号的生成规则配置、自动生成、自动赋值等功能，并提出了“业务码”（BusinessCode）的概念。

“业务码”（BusinessCode），泛指可以通过事先配置的业务码格式（BusinessCodeFormat），自动为指定业务对象打上的标识码。至于“流水号”，在 Phenix丶中仅指“业务码”中按一定规律自动生成的序列号。从理论上讲，一个“业务码”中可以存在多个“流水号”。

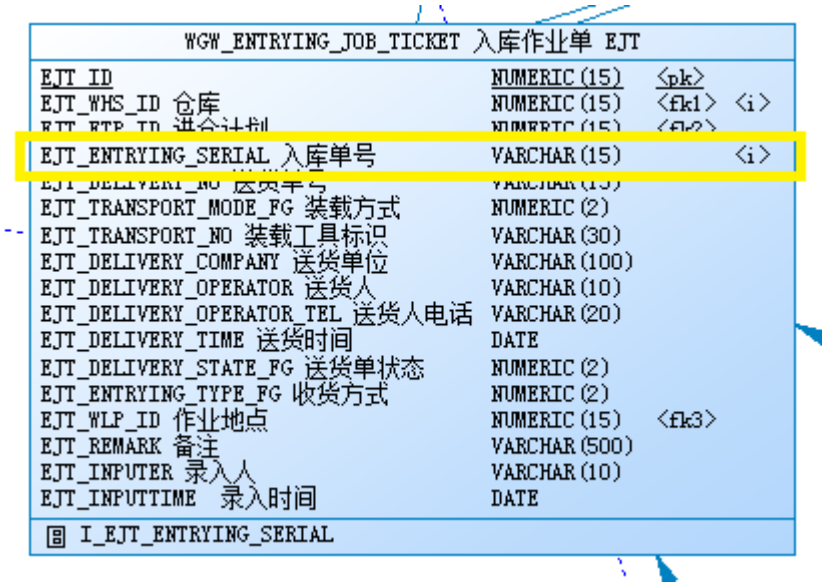
在应用系统的设计中，我们要将主键（PrimaryKey）、条码（BarCode）、业务码（BusinessCode）从物理上就区别开来，也即在表单设计中，分别用字段来存储它们。个中缘由，在于它们各自有各自的用途：

- 主键，仅用于标识记录的唯一性和支撑数据结构的关联性；从优雅性设计上考虑，建议不要赋予其他的含义；为了效率考虑，建议用数字来存储；为了规范化开发考虑，字段命名以“_ID”为后缀，字段类型为 NUMERIC(15)，这样可以由 Phenix丶自动赋值（Int64 类型主键值生成器 Phenix.Core.Data.Sequence.Value）；
- 条码，包括一维条码、二维条码、多维条码、射频技术等，以机器识别为目的，条码标准和规格纷繁复杂，往往不会兼顾人工识别；与实体的对应关系是一对一还是一对多，由业务场景决定；
- 业务码，以人工识别为目的，将业务对象的特性信息浓缩到约定（容易记忆的）编码规则的一行字符串中，与实体的对应关系往往是一对一，且由编码规律决定；为了规范化开发考虑，字段命名以“SERIAL”为后缀，可自动纳入 Phenix丶的业务码管理；

15.1 业务码表字段的命名规则

首先，在表结构设计中，我们要遵守 Phenix丶在 Phenix.Core.Mapping.CodingStandards 类中约定

的表字段命名规则：



上述示例，是按照 Phenix 丶在 Phenix.Core.Mapping.CodingStandards 类中提供的如下标准字段命名格式来命名“入库单号”的：

属性	说明	备注
DefaultBusinessCodeColumnName	缺省“业务码”字段名	缺省值：字段名后缀为“SERIAL”

当表字段是以上述格式命名的，则这个字段必定是业务码，并强制纳入到 Phenix 丶的“业务码”服务的管理范围内。

15.2 标识业务类属性为业务码

如果按照上述规则命名表字段，那么通过 Phenix 丶的 Addin 工具在初始化/编辑业务类时（见“03.Addin 工具使用方法”的“初始化/编辑业务类”章节），就可以自动为业务类上对应字段的 Phenix.Core.Mapping.FieldAttribute 标签设置上 IsBusinessCodeColumn = true，生成出正确的映射关系代码：

```
        /// <summary>
        /// 入库单号
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<string> EntryingSerialProperty =
RegisterProperty<string>(c => c.EntryingSerial);
        [Phenix.Core.Mapping.Field(FriendlyName = "入库单号", Alias = "EJT_ENTRYING_SERIAL", TableName
= "WGW_ENTRYING_JOB_TICKET", ColumnName = "EJT_ENTRYING_SERIAL", NeedUpdate = true, IsBusinessCodeColumn
```

```
= true)]
        private string _entryingSerial;
        /// <summary>
        /// 入库单号
        /// </summary>
        [System.ComponentModel.DisplayName("入库单号")]
        public string EntryingSerial
        {
            get { return GetProperty(EntryingSerialProperty, _entryingSerial); }
            set { SetProperty(EntryingSerialProperty, ref _entryingSerial, value); }
        }
```

当然，如果字段命名不符合规范，这也问题不大，就是要多做一步工作，在相应的业务类字段 Phenix.Core.Mapping.FieldAttribute 标签上自行设置 IsBusinessCodeColumn = true 即可：

属性	说明	备注
IsBusinessCodeColumn	指示该字段是业务码	缺省为 false；当 ColumnName 包含 Phenix.Core.Mapping.CodingStandards.DefaultBusinessCodeColumnName 时必定是业务代码；
BusinessCodeName	业务码名称	缺省为所属类名.字段名
BusinessCodeDefaultFormat	业务码缺省格式	
BusinessCodeCriteriaPropertyName	业务码条件属性名称	

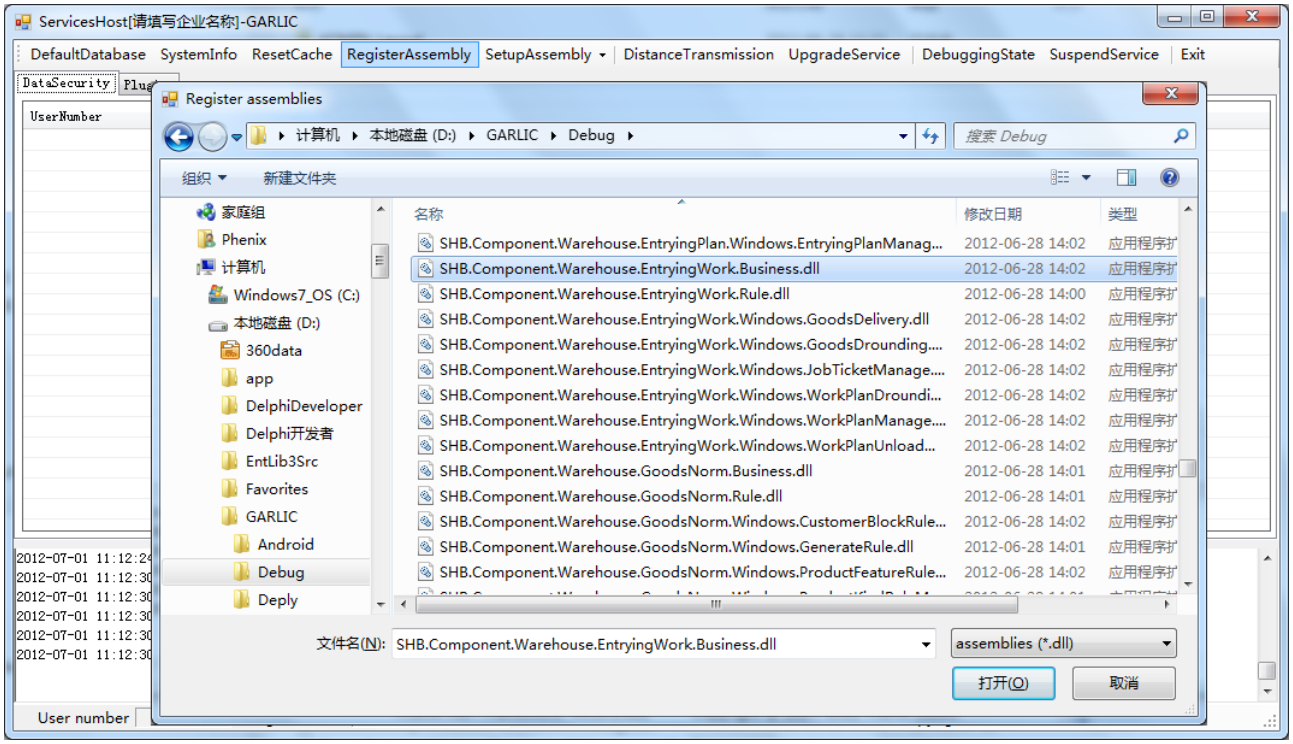
上述清单中，除了 IsBusinessCodeColumn 属性外，还有三个属性与业务码有关，待下文一一讲解。

业务码的字段必定是水印字段，也就是说，它只允许 New 业务对象在提交之前编辑，一旦提交之后就不允许编辑了。

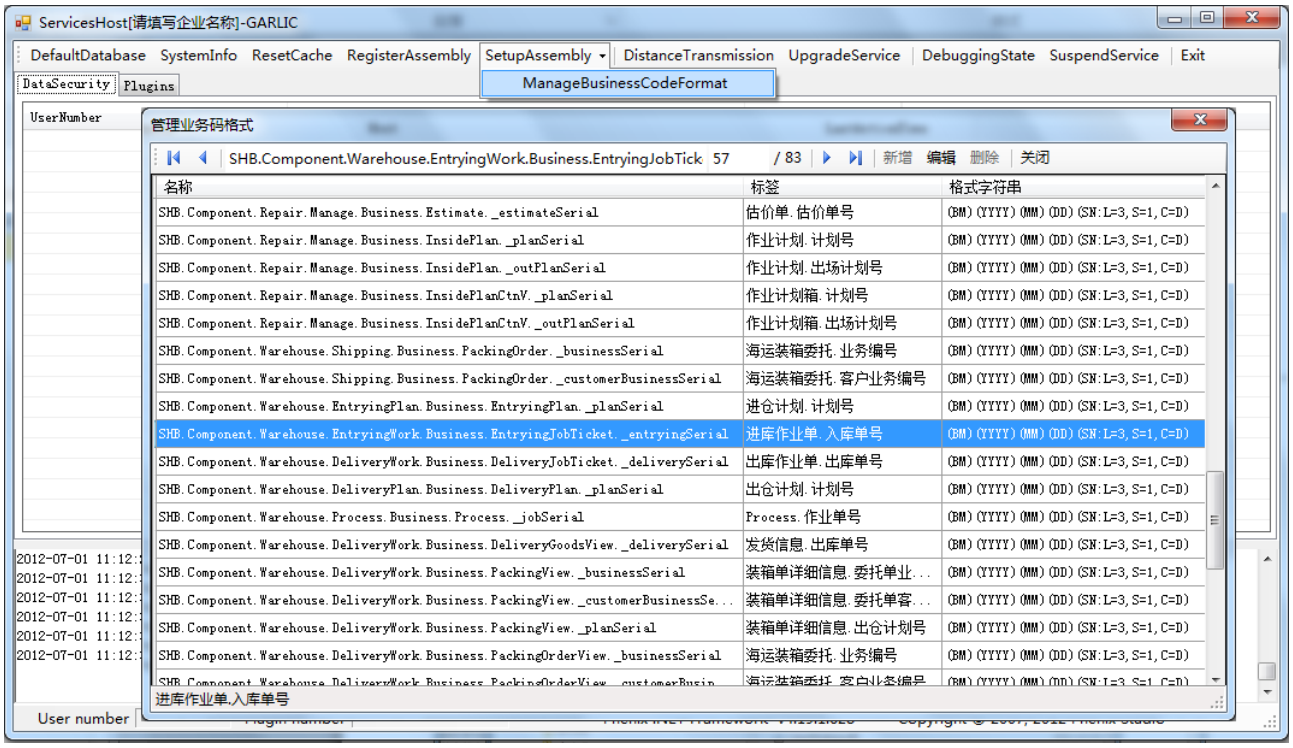
15.3 配置业务码格式

前文中 Phenix.Core.Mapping.FieldAttribute 标签的 BusinessCodeDefaultFormat 属性，是为业务码定义缺省的业务码格式，在注册程序集的时候，会被自动写入配置库中。而只有当业务类被注册进配置库中，我们才能在产品的实施阶段，根据客户需要，配置出个性化的业务码格式。

业务类的注册方法，是通过在 Phenix.Services.Host.exe 上的 RegisterAssembly 菜单来实现的：



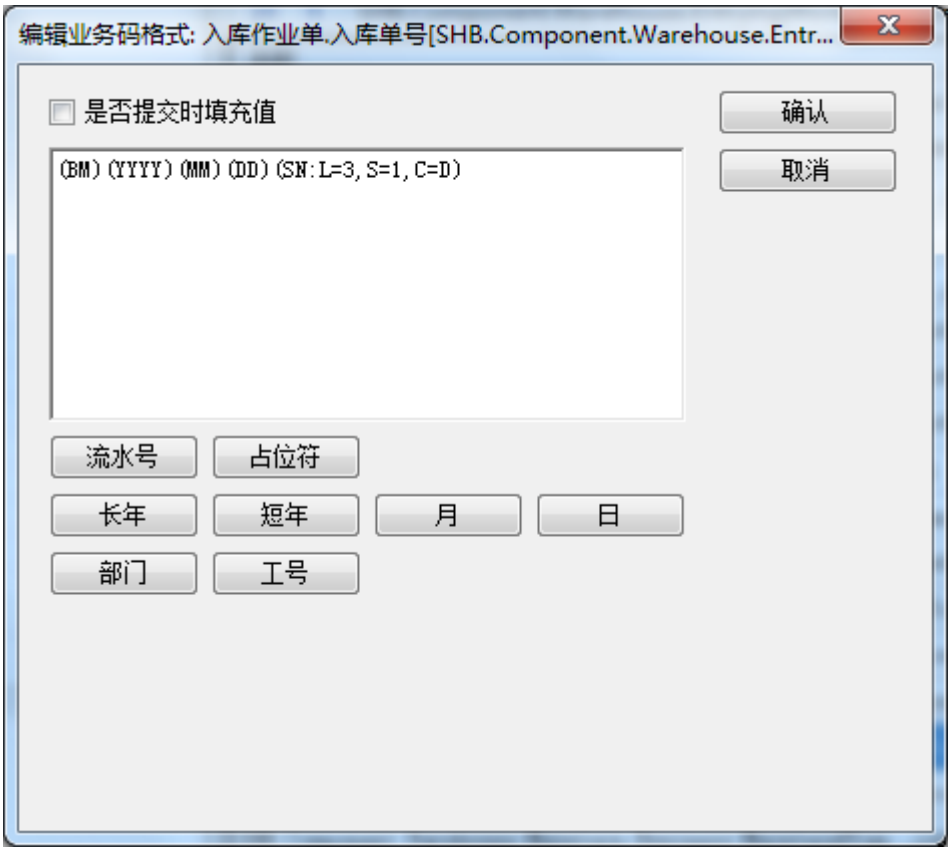
注册成功后，可以通过其 ManageBusinessCodeFormat 菜单弹出“管理业务码格式”对话框：



也可以调用以下函数（这段代码可嵌入到应用系统）来启动这个配置工具：

```
Phenix.Core.Data.Setup.BusinessCodeFormatManageDialog.Execute();
```

在配置工具界面中，列出了应用系统中被注册的全部业务码格式，可选择（也可以通过名称查询到）需要编辑的业务码格式记录，双击后弹出它的编辑对话框：



你可以手工编辑业务码格式，也可以通过界面上的辅助功能按钮进行填充。

业务码格式中，各格式项由大写字符、冒号、逗号、等于号和数字组成，并用小括号圈定。如果在格式项之间夹入其他不能识别的字符，则保留到实际生成的业务码字符串中。

15.3.1 “流水号”格式项

“流水号”是按照指定规则要求（见下文中的流水号参数）自动生成的一组数值序列。每次获取它，都会得到一个较之前一次获取数值递增 1 的新数值，并被转换成指定长度的字符串。

在上述编辑界面中，点击“流水号”功能按钮：

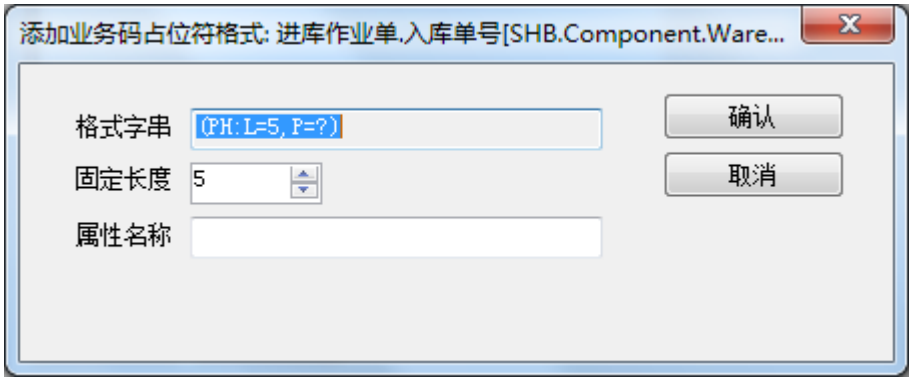
“流水号”格式项以“SN：”为标识，其余为生成流水号的参数：

参数	取值范围	说明	备注
L=	整型数值	固定长度标记	缺省为 3；流水号的字符串长度；不够长度则凑足，方法为左侧填充足够数量的“0”（右对齐）；超长则截掉左侧多余的字符；
S=	整型数值	起始值标记	缺省为 1；流水号为一组递增的数值序列，本参数值为这个序列的第一个数值；
G=	NO	分组模式：无	缺省为 NO；流水号可以按照部门或者工号独立维持一组序列；当参数值为非“NO”时，在业务码里，应该有对应的部门(BM)、工号(GH)格式项；
	BM	分组模式：部门	
	GH	分组模式：工号	
C=	D	重置周期：日	缺省为 D；流水号序列重置的循环周期，也就是申明间隔多长时间重启一个新的序列；在业务码里，应该有对应的年(YYYY)/(YY)、月(MM)、日(DD)格式项；
	M	重置周期：月	
	Y	重置周期：年	

15.3.2 “占位符”格式项

“占位符”是在业务码的指定位置（占位）上按照指定规则要求（见下文中的占位符参数）自动填充其所在业务对象指定属性的值（以字符串形式）内容。

在上述编辑界面上，点击“占位符”功能按钮：



“占位符”格式项以“PH：”为标识，其余为占位符的参数：

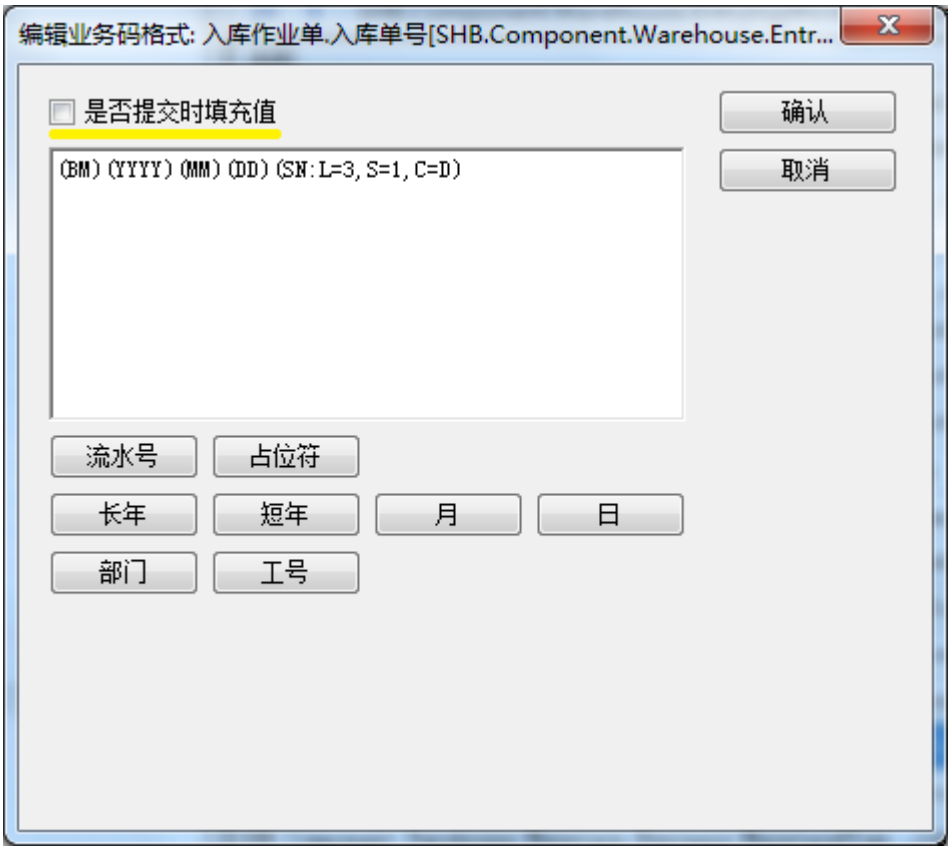
参数	取值范围	说明	备注
L=	整型数值	固定长度标记	缺省为 5；占位符的字符串长度；不够长度则凑足，方法为右侧填充足够数量的“ ”（左对齐）；超长则截掉右侧多余的字符；
P=	属性名	所在业务类的属性名	

15.3.3 其他格式项

格式项	说明	备注
(YYYY)	长年	取值：DateTime.Now.ToString("yyyy")
(YY)	短年	取值：DateTime.Now.ToString("yy")
(MM)	月	取值：DateTime.Now.ToString("MM")
(DD)	日	取值：DateTime.Now.ToString("dd")
(BM)	部门	取值：Phenix.Business.Security.User.Identity.Department.Code
(GH)	工号	取值：Phenix.Business.Security.User.Identity.UserNumber

15.3.4 是否提交时填充值

在上述编辑界面中，左上角有个选项：



这是控制业务码在 New 业务对象生命周期中的哪个节点生成并被填充到业务对象的相应属性中。

业务码生成和填充的时机，是根据业务场景的需要：

时机	优点	缺陷

New 业务对象在初始化时填充	可在提交之前看到业务码	在提交之前删除（废弃）New 业务对象的话，业务码中流水号序列会缺档
New 业务对象在提交时填充	业务码中流水号序列不可能出现缺档	在提交之前看不到业务码，Phenix 仅用“*”临时填充

15.4 共享业务码格式及其流水号序列

缺省情况下，被标识为业务码的属性将拥有自己独享的业务码格式及其流水号序列。因为业务码名称是业务码格式的唯一标识，只要在 Phenix.Core.Mapping.FieldAttribute 上未显式申明业务码名称，都是以“所属类名.字段名”为缺省的业务码名称，以避免重复，比如：

```
SHB.Component.Warehouse.EntryingWork.Business.EntryingJobTicket._entryingSerial
```

但是，有些应用系统的用户却希望不同的业务场景能共享到一个业务码格式及其流水号序列。为此，我们可以显式申明业务码名称，比如：

```
        /// <summary>
        /// 入库单号
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<string> EntryingSerialProperty =
RegisterProperty<string>(c => c.EntryingSerial);
        [Phenix.Core.Mapping.Field(FriendlyName = "入库单号", Alias = "EJT_ENTRYING_SERIAL", TableName
= "WGW_ENTRYING_JOB_TICKET", ColumnName = "EJT_ENTRYING_SERIAL", NeedUpdate = true, BusinessCodeName =
"WarehouseWorkSerial")]
        private string _entryingSerial;
        /// <summary>
        /// 入库单号
        /// </summary>
        [System.ComponentModel.DisplayName("入库单号")]
        public string EntryingSerial
        {
            get { return GetProperty(EntryingSerialProperty, _entryingSerial); }
            set { SetProperty(EntryingSerialProperty, ref _entryingSerial, value); }
        }
```

注册到配置库中的业务码名称就是：

```
WarehouseWorkSerial
```

只要其他业务类上的业务码属性也显式申明这相同的业务码名称，这些业务对象就能共享到相同的业务码格式及其流水号序列。

15.5 动态业务码格式

前文中介绍的方法，仅支持一个业务类的业务码属性对应一个业务码格式。但是，如果希望能根据业务对象中的其他某个属性值的不同而对应到不同的业务码格式（比如“入库单号”能根据不同的“入库计划类型”而产生不同格式的业务码），这如何实现呢？

Phenix.Core.Mapping.FieldAttribute 标签的 BusinessCodeCriteriaPropertyName 属性就是为此而设的：

```
        /// <summary>
        /// 入库单号
        /// </summary>
        public static readonly Phenix.Business.PropertyInfo<string> EntryingSerialProperty =
RegisterProperty<string>(c => c.EntryingSerial);
        [Phenix.Core.Mapping.Field(FriendlyName = "入库单号", Alias = "EJT_ENTRYING_SERIAL", TableName
= "WGW_ENTRYING_JOB_TICKET", ColumnName = "EJT_ENTRYING_SERIAL", NeedUpdate = true,
BusinessCodeCriteriaPropertyName = "EntryingPlanClass")]
        private string _entryingSerial;
        /// <summary>
        /// 入库单号
        /// </summary>
        [System.ComponentModel.DisplayName("入库单号")]
        public string EntryingSerial
        {
            get { return GetProperty(EntryingSerialProperty, _entryingSerial); }
            set { SetProperty(EntryingSerialProperty, ref _entryingSerial, value); }
        }
```

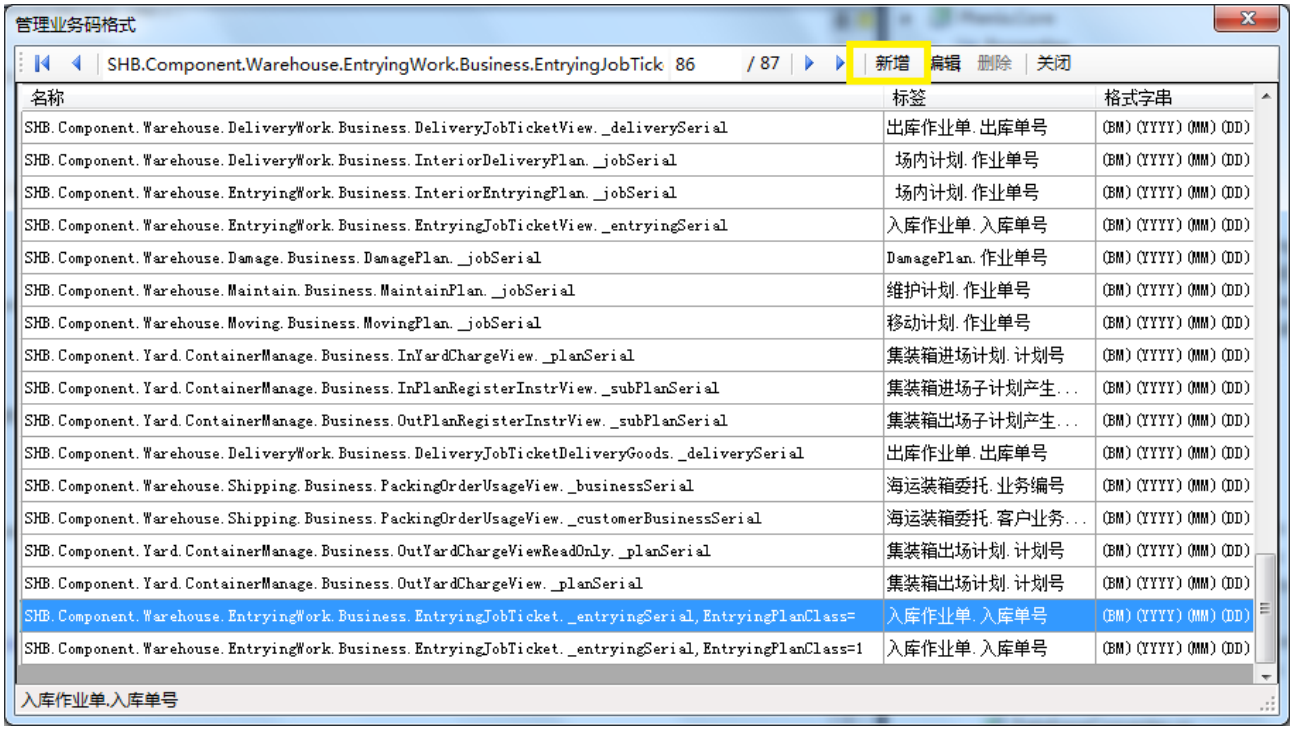
当我们注册（见前文）了这个业务类后，可以在“管理业务码格式”界面中找到这个业务码格式，其名称被标识为：

```
SHB.Component.Warehouse.EntryingWork.Business.EntryingJobTicket._entryingSerial,EntryingPlanCla
ss=
```

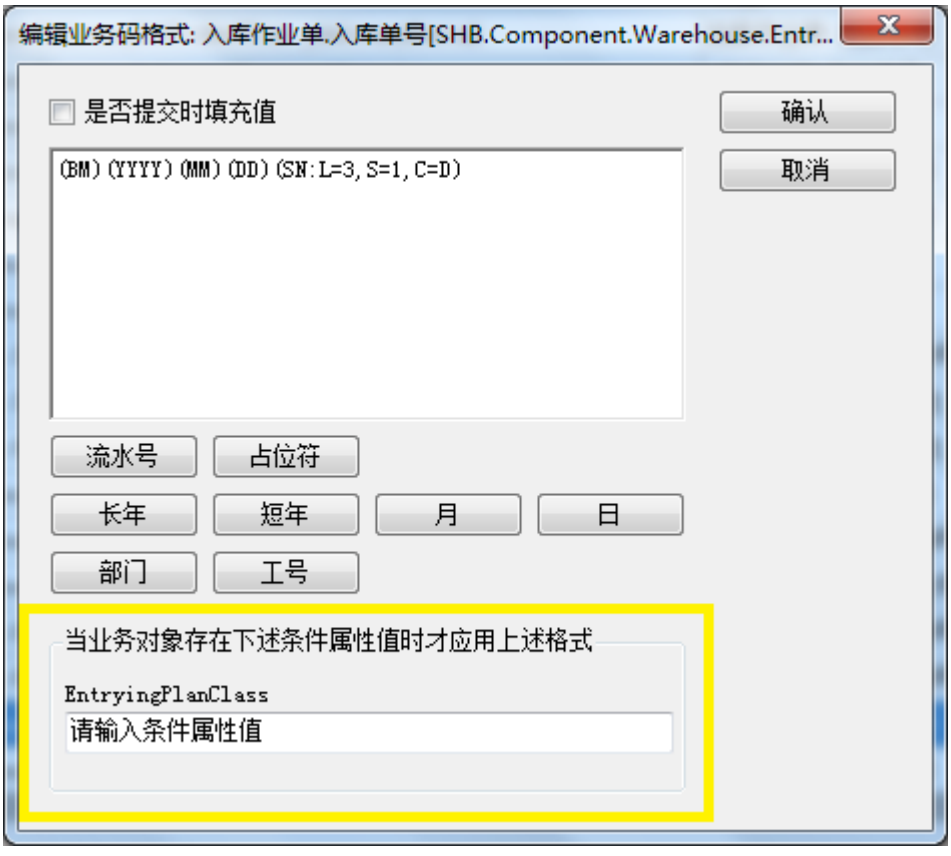
如果定位它并直接点击“编辑”功能按钮（或双击它），则弹出的是和普通的业务码一样的配置界面，

但此处是用于缺省业务码的配置。

如果定位它并点击“新增”功能按钮（此时按钮已亮），则弹出的配置界面，就是用于新增一个格式（将以输入的条件属性值作为业务码规则名称）：



点击“新增”功能按钮，进入它的编辑界面，黄色框内为区别于普通业务码格式的配置项：

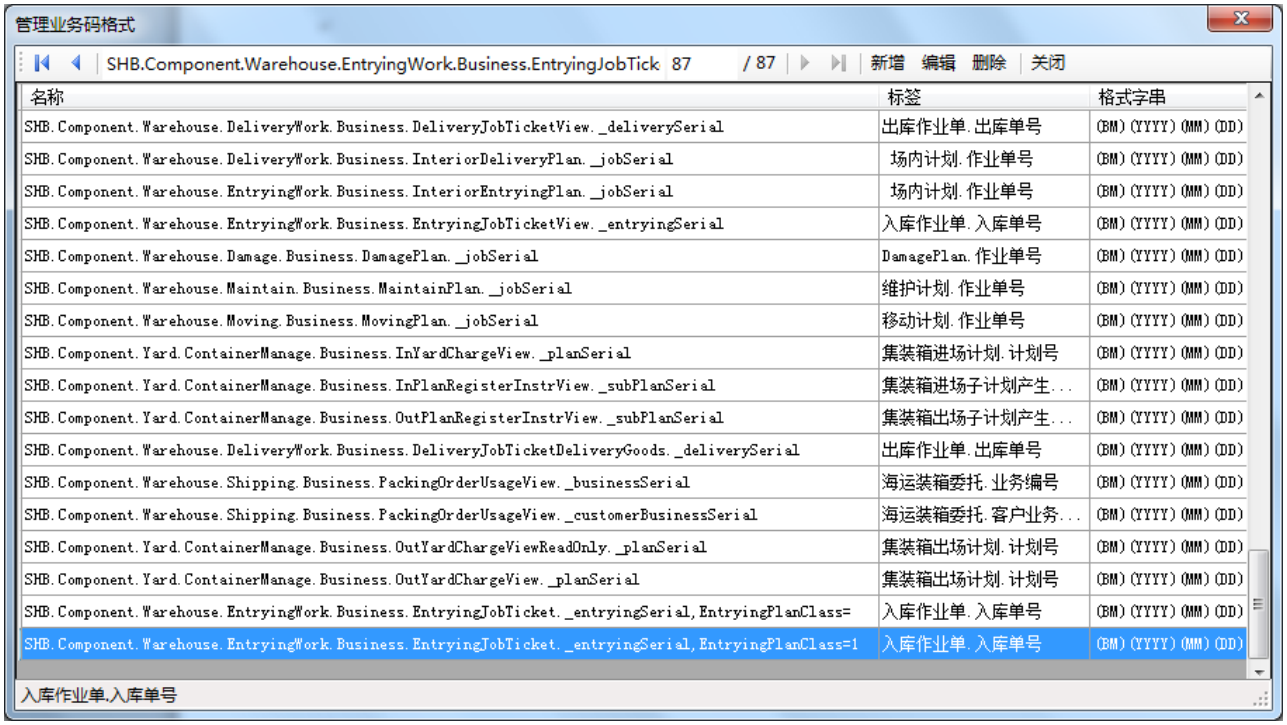


当输入条件属性值，并设置完新的业务码格式：

点击确认后，Phenix丶会将它保存为一个新的业务码格式，业务码名称为：

SHB.Component.Warehouse.EntryingWork.Business.EntryingJobTicket._entryingSerial,EntryingPlanClass=1

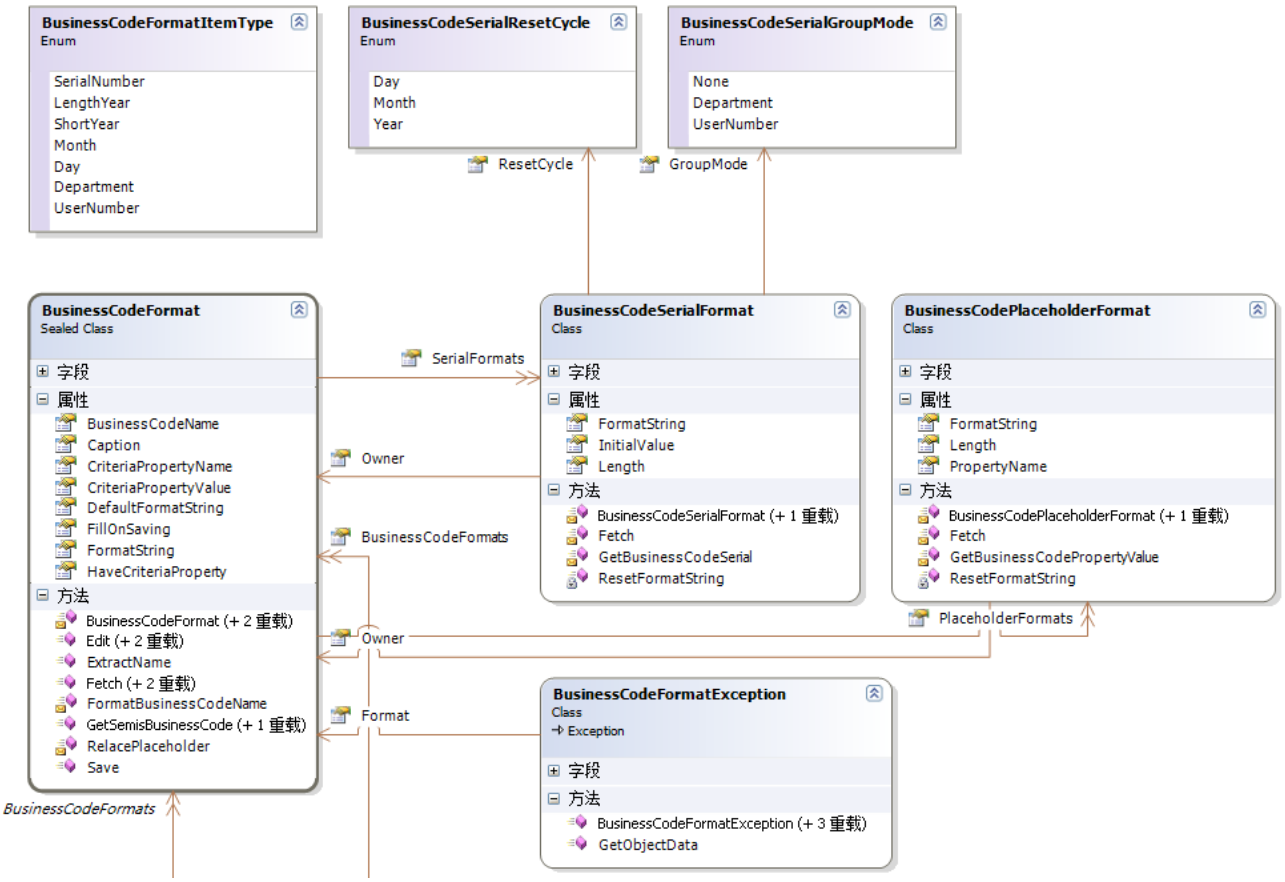
可见条件属性名和条件属性值都拼到了业务码名称中：



需提请注意的是，“入库计划类型”被设计成枚举类型，而枚举类型在 Phenix 内部都是被转换为整型来处理的，Phenix.Services.Host.exe 提供的配置工具并不操作业务类，所以在配置业务码格式的时候，要求输入的是枚举的 Flag 值。另外，在这个配置界面上，业务码条件属性值、占位符的属性名，都没有提示和约束。如果要实现更加人性化的交互界面，也是可以做到的，请见下文。

15.6 直接操作业务码

Phenix 在业务对象上所做的业务码功能，底层都是通过 BusinessCodeFormat 类来实现的，你也可以通过它来直接操作业务码。



15.6.1 New 业务码格式

```
/// <summary>
/// 初始化
/// </summary>
/// <param name="businessCodeName">业务码名称</param>
/// <param name="caption">标签</param>
/// <param name="formatString">格式字符串</param>
/// <param name="fillOnSaving">是否提交时填充值</param>
public BusinessCodeFormat(string businessCodeName, string caption, string formatString, bool fillOnSaving)
```

15.6.2 Fetch 业务码格式

```
/// <summary>
/// 构建业务码
/// </summary>
/// <param name="businessCodeName">业务码名称</param>
public static BusinessCodeFormat Fetch(string businessCodeName)
```

```
/// <summary>
/// 构建业务码
/// </summary>
/// <param name="propertyInfo">属性信息</param>
public static BusinessCodeFormat Fetch(IPropertyInfo propertyInfo)

/// <summary>
/// 构建业务码
/// </summary>
/// <param name="obj">业务对象</param>
/// <param name="propertyInfo">属性信息</param>
public static BusinessCodeFormat Fetch(object obj, IPropertyInfo propertyInfo)
```

15.6.3 Edit 业务码格式

```
/// <summary>
/// 编辑
/// </summary>
public bool Edit()

/// <summary>
/// 编辑
/// </summary>
public bool Edit(KeyCaptionCollection criteriaPropertySelectValues)

/// <summary>
/// 编辑
/// </summary>
public bool Edit(EnumKeyCaptionCollection criteriaPropertySelectValues)

/// <summary>
/// 编辑
/// </summary>
public bool Edit<T>()
  where T : IEntity

/// <summary>
/// 编辑
/// </summary>
public bool Edit<T>(KeyCaptionCollection criteriaPropertySelectValues)
  where T : IEntity
```

```
/// <summary>
/// 编辑
/// </summary>
public bool Edit<T>(EnumKeyCaptionCollection criteriaPropertySelectValues)
  where T : IEntity
```

15.6.4 Save 业务码格式

```
/// <summary>
/// 保存
/// </summary>
public void Save()
```

15.6.5 Get 业务码

Phenix丶的业务码组件是为业务类的业务码属性值的自动填充配套使用的，特别是在业务码格式的配置上，如果：

- 含“占位符”格式项
- 属于动态业务码格式

就不能通过下述函数直接获取到实际的业务码。

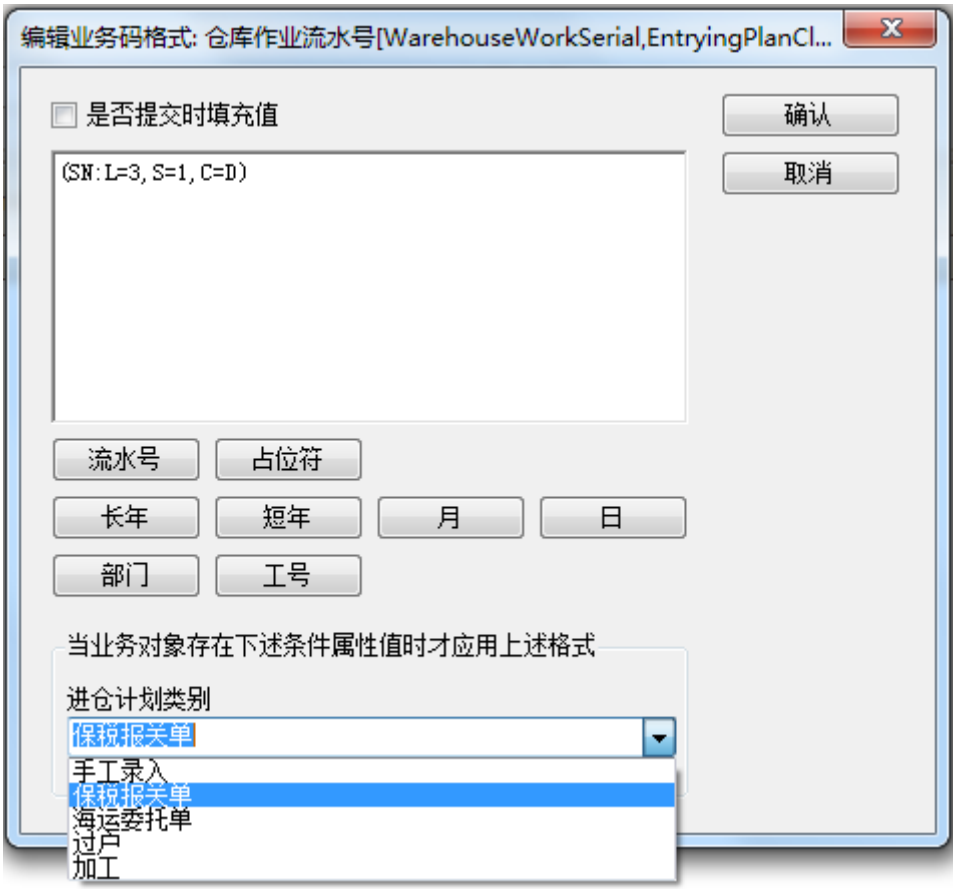
```
/// <summary>
/// 获取业务码(忽略属性名标记的占位符)
/// </summary>
public string GetValue()

/// <summary>
/// 获取业务码(忽略属性名标记的占位符)
/// </summary>
public string GetValue(DbConnection connection)

/// <summary>
/// 获取业务码(忽略属性名标记的占位符)
/// </summary>
public string[] GetValues(int count)

/// <summary>
/// 获取业务码(忽略属性名标记的占位符)
/// </summary>
public string[] GetValues(DbConnection connection, int count)
```

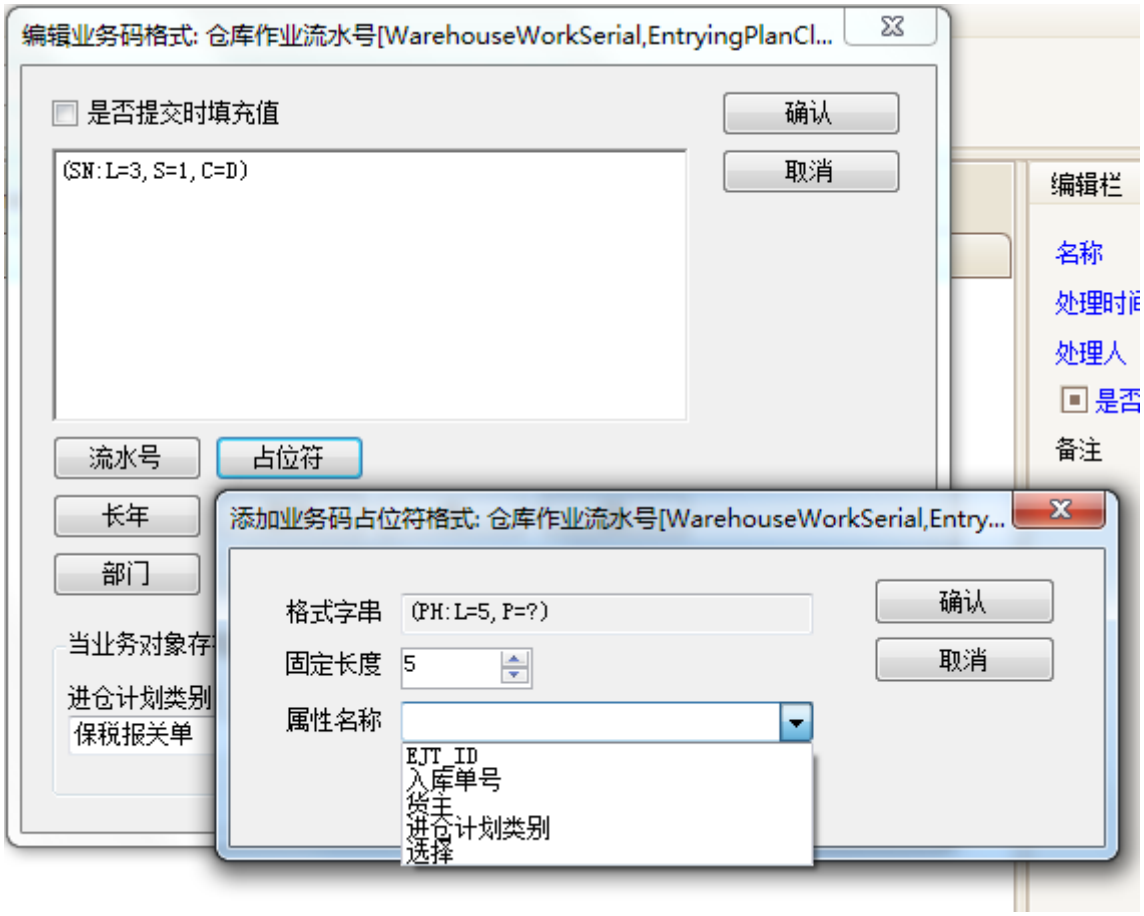
但是，只要业务码格式不属于上述范畴，通过这些函数获得的半成品业务码，其实与自动填充的业务对象的业务码属性值是没有任何区别的。

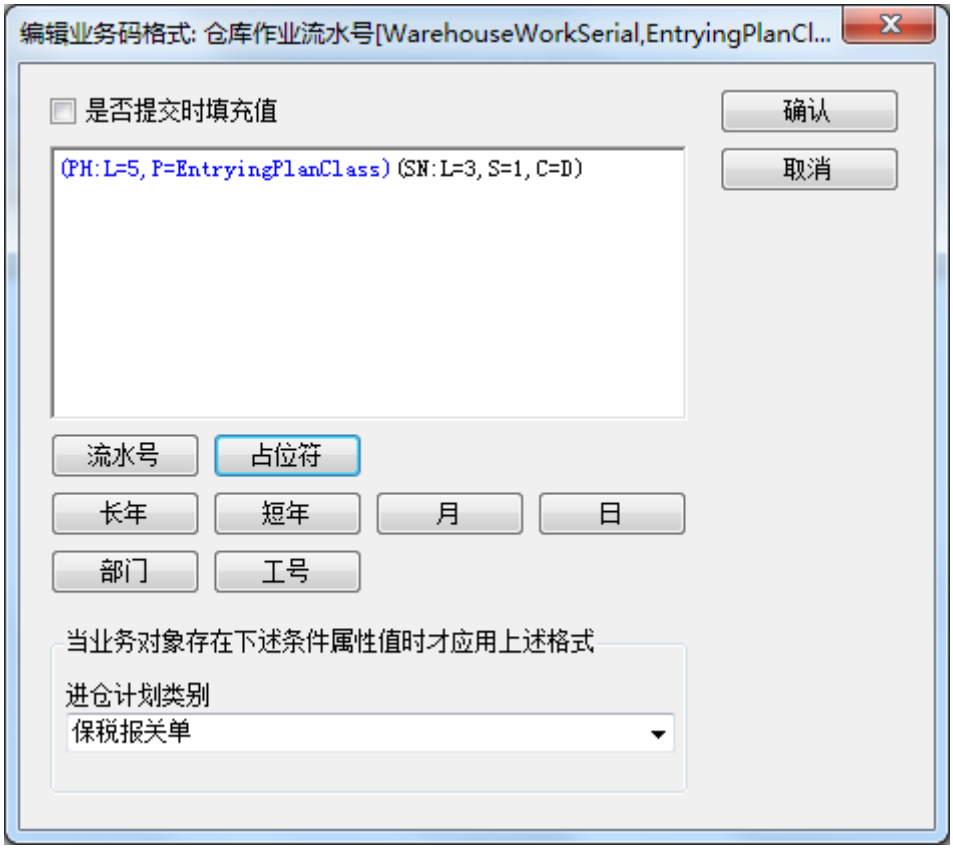
15.6.6 案例

15.6.6.1 演示动态业务码格式的注册和编辑过程

```
        //注册编辑业务码格式
        BusinessCodeFormat businessCodeFormat = new
BusinessCodeFormat("WarehouseWorkSerial,EntryingPlanClass=1", "仓库作业流水号");
        bool editSucceed =
businessCodeFormat.Edit(typeof(SHB.Component.Warehouse.EntryingWork.Business.EntryingJobTicket),
Phenix.Core.Rule.EnumKeyCaptionCollection.Fetch<SHB.Component.Warehouse.EntryingPlan.Rule.EntryingPla
nClass>());
```








```
//保存业务码格式
if (editSucceed)
```

```
businessCodeFormat.Save();
```

15.6.6.2 演示普通业务码格式的注册、编辑和获取业务码过程

```
//注册编辑业务码格式
BusinessCodeFormat businessCodeFormat = new BusinessCodeFormat("WarehouseWorkSerial", "仓库作业流水号");
businessCodeFormat.Edit();
//保存业务码格式
businessCodeFormat.Save();

//构建业务码格式
businessCodeFormat = BusinessCodeFormat.Fetch("WarehouseWorkSerial");
//获取业务码
string businessCode = businessCodeFormat.GetValue();
MessageBox.Show(businessCode, businessCodeFormat.Caption, MessageBoxButtons.OK, MessageBoxIcon.Information);
```