Variables – Process and Service

## 变量 – 流程和服务

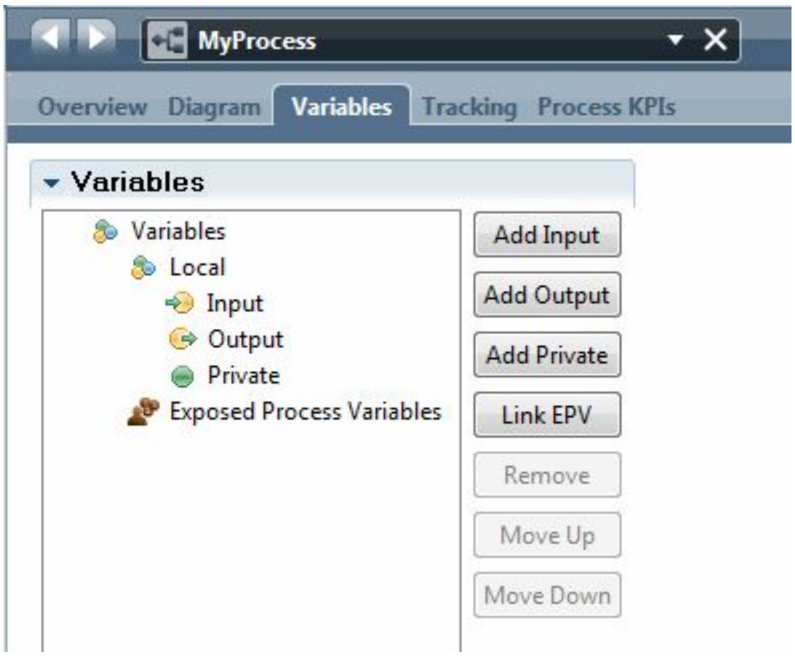
As an instance of a process executes, that process is going to have data maintained as part of its state during its complete life. The data may be input into the process when it starts or may be calculated or entered as the process progresses through its life.
当一个流程实例在执行的时候,在其完整的生命周期中的不同的阶段,它将有一部分数据维护性的操作.这些数据可能是流程开始时作为输入参数传递到流程中,也可能是流程处理中进行计算的结果或者是用户输入的.

Variables are used to hold the state data associated with an instance of a process. Variables are "named" entities which have associated data types. Variables occur in two distinct categories of places in IBPM. The first is the set of variables which can be defined in a BPD. These variables have a scope of the entire instance of the process. Variables are also constrained by the process in which they live. Variables in one process instance can't be accessed in another process (without an explicit passing of their values). A change to a variable in one activity will be seen in subsequent activities.
变量是用来存储流程实例的数据状态.变量是一个可命名的实体并且有数据类型.在IBPM中,变量可以在两个不同的分类中进行定义.第一种是在BPD中的定义的变量集.这些变量的可用范围是在流程实例的整个周期中.同时,它也仅在该流程存活时进行使用.在不同的流程中变量是不能相互访问(除非是进行显示传值).在一个活动中变量的改变将会传递到它的后来的活动中。

The other place variables can be defined is within a Service implementation. These variables exist for the life of the Service and are no longer available after the service completes.
另外一个可以定义变量的地方是在服务的实现中.这些变量只存在于服务运行周期中,当服务运行结束后就失效.

Variables can be defined with a qualifier of input, output or private. Input variables have their values passed into a process or service when that process or service is initiated. Output variables have their values returned from a process or service when it completes and private variables only have scope within a process or service and are deleted when they end.
变量定义可以用input,output或private来进行修饰.输入变量在流程或服务初始化时,将其值传入到流程或者服务中.输出变量在流程或服务完成时获得其返回值.私有变量只在当前的流程或服务中起作用,当流程或服务结束时

将被删除回收.



When a variable is created, it can be addressed from within JavaScript with the prefix:

tw.local.<variable name>

(Note: This prefix tw.local is used so frequently that the JavaScript editor has a special shortcut for that called "twl". If you enter twl and then CTRL+Space for content assist, the twl entered characters are changed to tw.local.)

当一个变量创建完后,它可以通过 JavaScript 用以下前缀的方式进行访问:

    tw.local.<variable name>

(注意: 由于前缀 tw.local 使用比较频繁,JavaScript 编译器为其提供了一个专用的缩写"twl".当你输入 twl 并且按下 Ctrl+Space 时,输入的 twl 将自动转成 tw.local.)

When a variable is defined, it is given a name and other attributes including:

• Data type - The type of data represented by the variable

• Default value - An optional default value

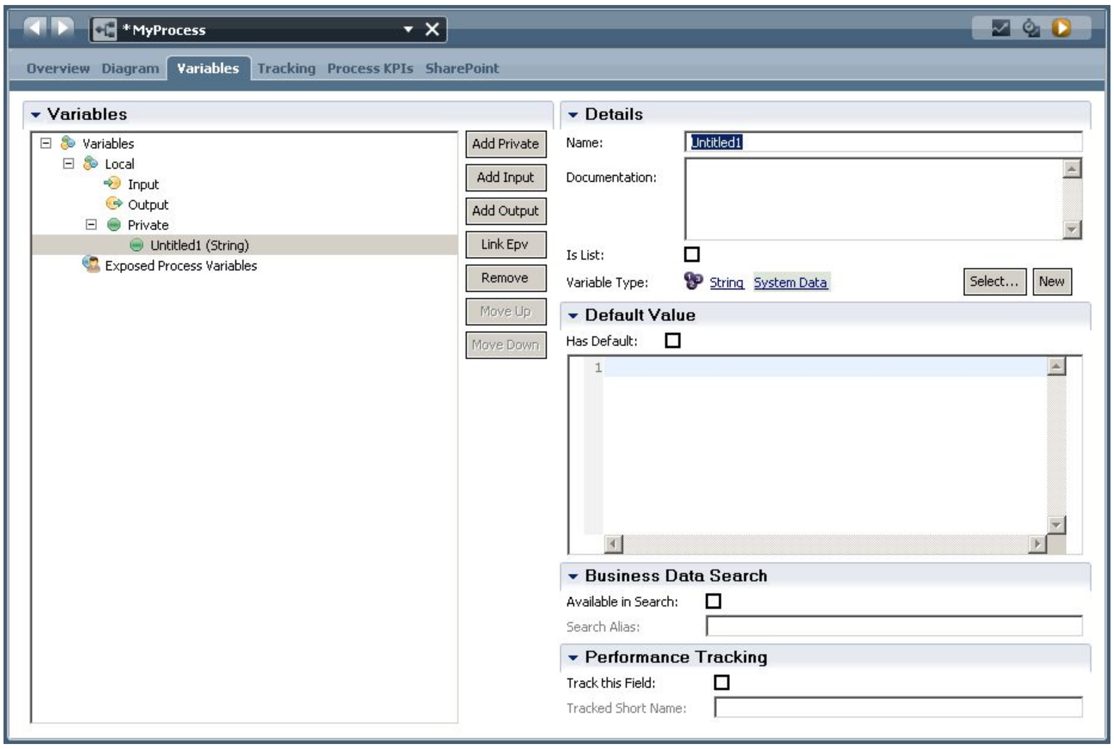
• Is List - A flag indicating that the variable represents a list/array of instances of the data type

• Documentation - Human readable documentation on the purpose of the variable

当一个变量定义完后,它将拥有一个名称,同时含有其它其他属性,包括:

- 变量类型 - 该变量所属数据类型
- 缺省值 - 一个可选的缺省值

- 是否列表 － 标记该变量是否是一个数据类型的列表/数组
- 文档 － 关于该变量说明的可读性文档



Name: Untitled1

Documentation:

Is List: ☐

Variable Type: String System Data

The data types initially supplied available include:

• ANY

• Boolean

• Date

• Decimal

• Integer

• Map

• NameValuePair

• String

These data types come from the System Data toolkit which is always present within the environment.

变量类型默认有以下几种类型:

- ANY
- Boolean
- Date
- Decimal
- Integer
- Map

- NameValuePair
- String

这些属性类型是在系统数据 Toolkit 中定义的,该 Toolkit 在不同环境中都必然存在.

When a new variable is defined, it can be declared as having a default value. What this means is that an instance will be created and will already have a value without having to explicitly perform a subsequent assignment to it.
当一个新的变量定义时,它可以申明有一个缺省值.这表明当一个实例创建时,它总是有一个值,不需要对其显示赋值.

System of Record data vs Business Intelligence data

**系统记录数据 vs 业务消息数据**

Let us take a few minutes to think about the data used and produced by a process instance. In my opinion, this data can be split into two logically distinct categories. These are "System of Record data" (SOR) and "Business Intelligence data" (BI).
让我们用几分钟来思考在流程实例中的数据使用和产生.在我看来,这些数据可以分为两种不同逻辑的类型.它们是系统记录数据(SOR)和业务消息数据(BI).

SOR data is information that can not afford to be lost once the process has ended. It is typically used by other applications in the future. As an example, imagine a process which books a seat on an airplane for a future trip I plan to make. It would be very wrong if I turned up at the airport three weeks from now and found that there was no record of my booking. BPM is not a repository for SOR data. Instead, BPM could be responsible for causing new SOR data to be created or existing SOR data to be updated. The SOR data is maintained by systems outside BPM such as a database or another application.
SOR 数据指的是流程结束后,那些还不能丢失的数据.它们将会被其他应用使用.比如我通过一个流程,在往后的航班系统中订了一个座位.然而当我三周后出现在机场时,发现没有我的订位记录,这显然是一个错误的信息.BPM 并不存储这些 SOR 数据.相反,BPM 将会触发创建新的 SOR 数据或者对现有的 SOR 数据进行更新.这些 SOR 数据是在 BPM 之外的系统中维护的,比如数据库或者其他应用程序.

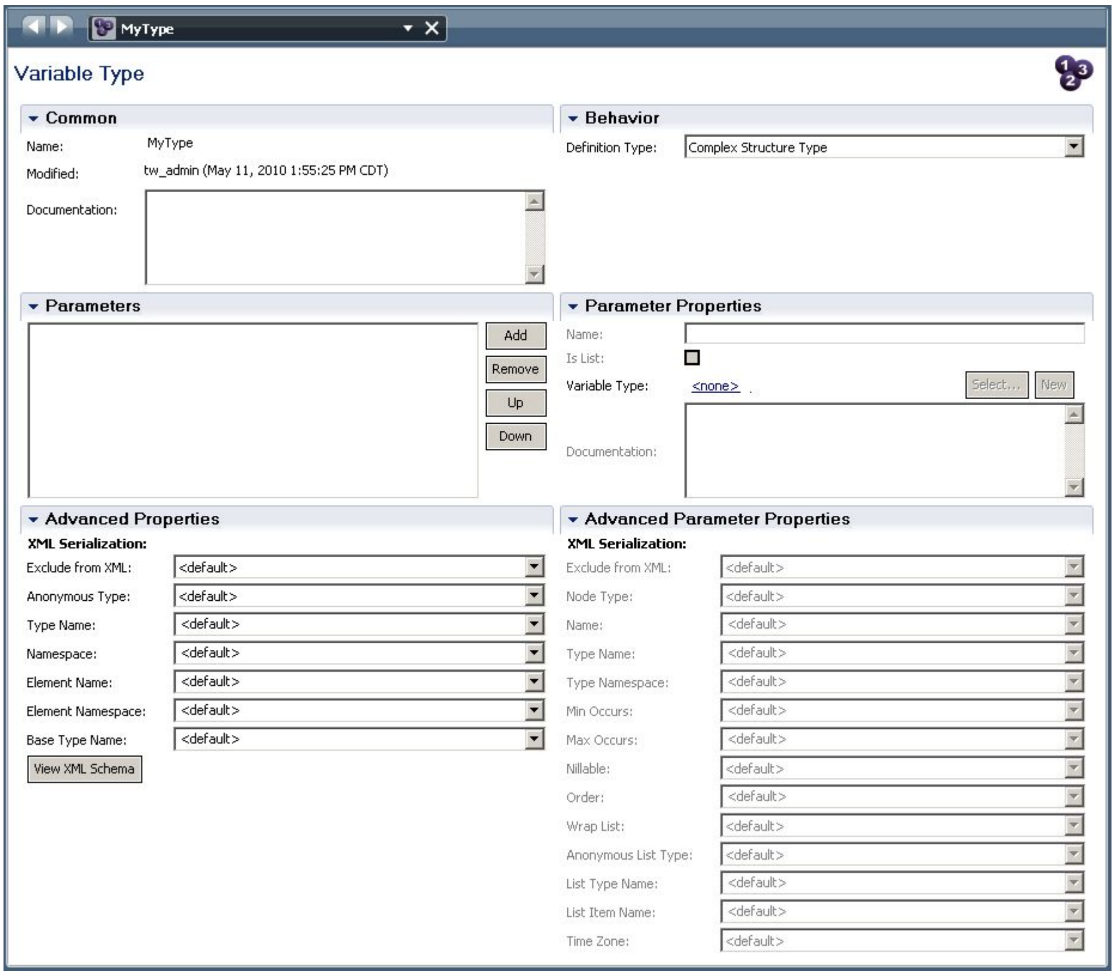
BI data is information that could, in principle, be lost without damaging the operation of your business.

This data is used to examine how your processes performed in the past. It includes times, paths, people, decisions and data relevant to those decisions. Its primary purpose is to provide the capability to perform business analytics. If it were lost, no customers would be upset, no monies would be lost, no corrective actions need be performed … the worst that would happen is that you would be unable to ask questions on how your processes performed in the past. IBM BPM can record BI data through the Performance Data Warehouse (PDW) architecture.
BI 数据指的是那些原则上可以丢失,并且不影响你的业务操作的数据.这些数据是用来检验过去你的流程性能的.它包含有次数,路径,人员,决策以及决策相关的数据.它主要是用来提供一些业务分析的指标.如果它们丢失了,没有客户会受到打击,也没有财产损失,不需要进行一些矫正的操作.最坏的情况是,你无法获知你的流程在过去运行的如何. IBM BPM 可以通过性能数据仓库(PDW)的结构记录这些数据.

Creating new Data structures

**创建新的数据结构**

The data types supplied with IBPM are important and useful but as we start to model our business processes we will commonly find that we want to create new data structures. New data structures can also be created which can be composed of fields (also called parameters). These new data structures will be used to represent different types of logical data such as "A Customer" or "A Loan".
IBPM 所提供的数据类型是重要并且有用的,但是我们通常要创建新的数据结构来对我们的业务流程进行模块化.新的数据结构可以通过组合一些领域(也称为参数)来创建.这些新的数据结构将用来代表不同类型的逻辑数据,比如’一个消费者’或者’一个贷款’.


MyType

Variable Type

Common

Name: MyType

Modified: tw_admin (May 11, 2010 1:55:25 PM CDT)

Documentation:

Behavior

Definition Type: Complex Structure Type

Parameters

Add | Remove | Up | Down

Parameter Properties

Name:

Is List: ☐

Variable Type: <none>

Select... | New

Documentation:

Advanced Properties

XML Serialization:

| | |
|---|---|
| Exclude from XML: | <default> |
| Anonymous Type: | <default> |
| Type Name: | <default> |
| Namespace: | <default> |
| Element Name: | <default> |
| Element Namespace: | <default> |
| Base Type Name: | <default> |

View XML Schema

Advanced Parameter Properties

XML Serialization:

| | |
|---|---|
| Exclude from XML: | <default> |
| Node Type: | <default> |
| Name: | <default> |
| Type Name: | <default> |
| Type Namespace: | <default> |
| Min Occurs: | <default> |
| Max Occurs: | <default> |
| Nillable: | <default> |
| Order: | <default> |
| Wrap List: | <default> |
| Anonymous List Type: | <default> |
| List Type Name: | <default> |
| List Item Name: | <default> |
| Time Zone: | <default> |

In the Behavior section of the data type editor, there are two choices:

• Complex Structure Type

• Simple Type

When a complex type variable is created, it must be initialized before it can be used. This can be done through JavaScript with the code:

tw.local.myVariable = new tw.object.*ComplexType*();

A similar story is also true for data of type list:

```
tw.local.myListOfStrings = new tw.object.listOf.String();
```

在数据类型编辑器的行为章节里,有两种选项:

- 复合结构类型
- 简单类型

当一个复合结构类型的变量创建后,它必须要进行初始化才能使用.这可以通过 JavaScript 来实现,具体如下:

```
tw.local.myVariable = new tw.object.ComplexType();
```

同样适用于列表类型:

```
tw.local.myListOfStrings = new tw.object.listOf.String();
```

Simple Types

The Simple Type allows us to place constraints on certain simple data types.

**简单类型**

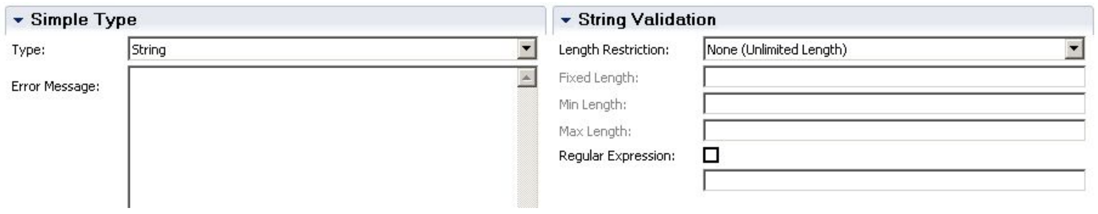
简单类型提供对确定的简单数据类型添加约束.

Simple type - String

Here we can place length restrictions on the string. This includes either a fixed length or a minimum/maximum length.

**简单类型 - String**

我们可以对 String 类型添加长度限制.这些可以是一个固定的长度,或者是一个最小/最大长度.



Simple Type

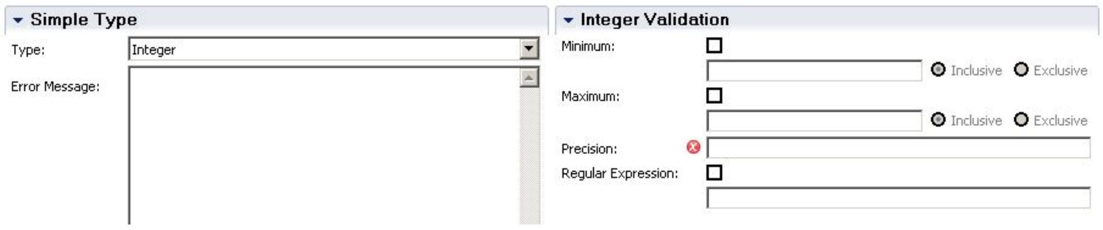
| | |
|---|---|
| Type: | String |
| Error Message: | |

String Validation

| | |
|---|---|
| Length Restriction: | None (Unlimited Length) |
| Fixed Length: | |
| Min Length: | |
| Max Length: | |
| Regular Expression: | ☐ |

Simple type - Integer

**简单类型 - Integer**



Simple Type

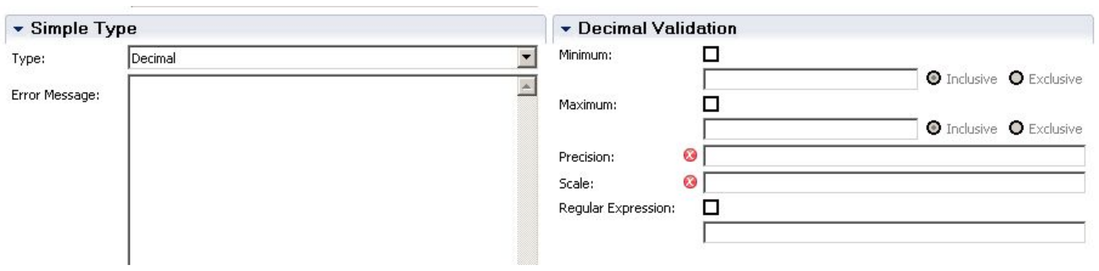
| | |
|---|---|
| Type: | Integer |
| Error Message: | |

Integer Validation

| | |
|---|---|
| Minimum: | ☐ Inclusive Exclusive |
| Maximum: | ☐ Inclusive Exclusive |
| Precision: | |
| Regular Expression: | ☐ |

Simple type - Decimal

## 简单类型 - Decimal


**Simple Type**

Type: Decimal

Error Message:

**Decimal Validation**

Minimum: ☐ [✓ Inclusive] Exclusive

Maximum: ☐ [✓ Inclusive] Exclusive

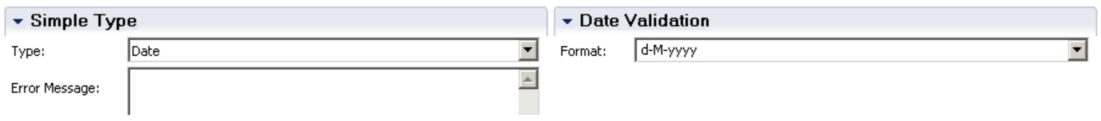
Precision:

Scale:

Regular Expression: ☐

Simple type - Date

## 简单类型 - Date


**Simple Type**

Type: Date

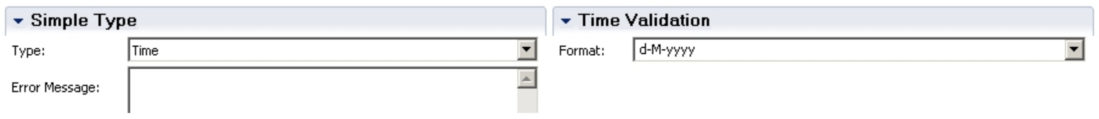
Error Message:

**Date Validation**

Format: d-M-yyyy

Simple type - Time

## 简单类型 - Time


**Simple Type**

Type: Time

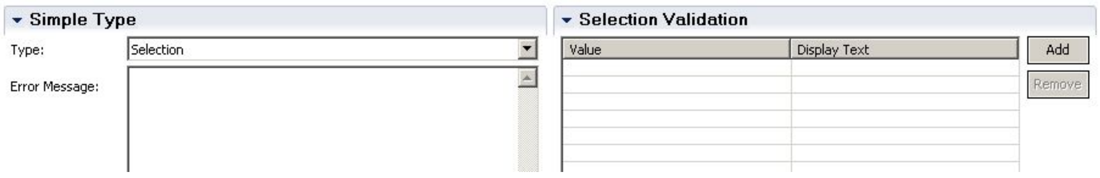
Error Message:

**Time Validation**

Format: d-M-yyyy

Simple type - Selection

## 简单类型 - Selection


**Simple Type**

Type: Selection

Error Message:

**Selection Validation**

| Value | Display Text |
|---|---|

Business Object Types

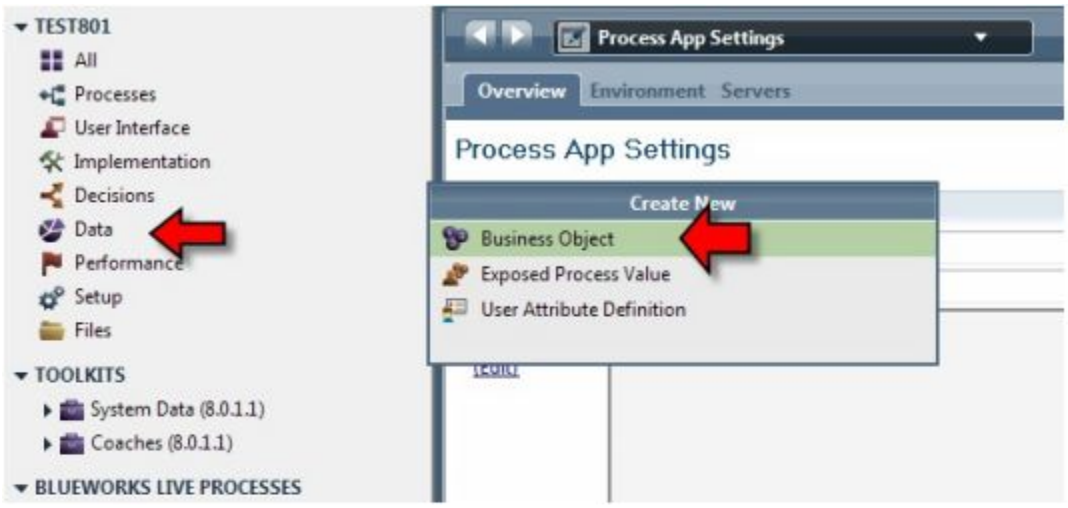
**业务对象类型**

A Business Object is commonly used to describe a series of attributes that apply to a concept being worked upon in a process. Consider the concept of a "Customer". A Customer has many attributes including name, address, payment info, purchase historyand perhaps much more. We can define the data type known as Customer by creating a new data type within IBPM PD.

一个业务对象常常用于描述一系列作用于流程的属性，假设 一个 "customer"的内容， 一个顾客有多个属性，包含名字，地址，付款信息，购买记录 或者更多。我们能够通过IBPM  PD 创建一个新的数据类型定义这些数据为顾客类型。

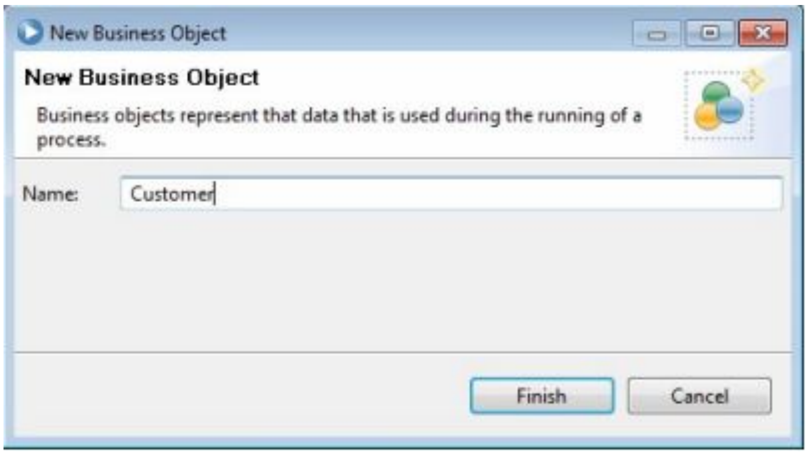
A new data type can be created from the Data entry in the list and selecting Business Object

可以通过Data entry 里的列表中选择Business Object 来创建新的数据类型.



Once selected, a dialog window appears asking us to name the new data type we are creating. In this example, we have called our new data type "Customer":

选择后，一个对话框要求我们输入新创建数据类型的名字，在这个例子里，我们命名新的数据类型为 "Customer"


New Business Object

**New Business Object**

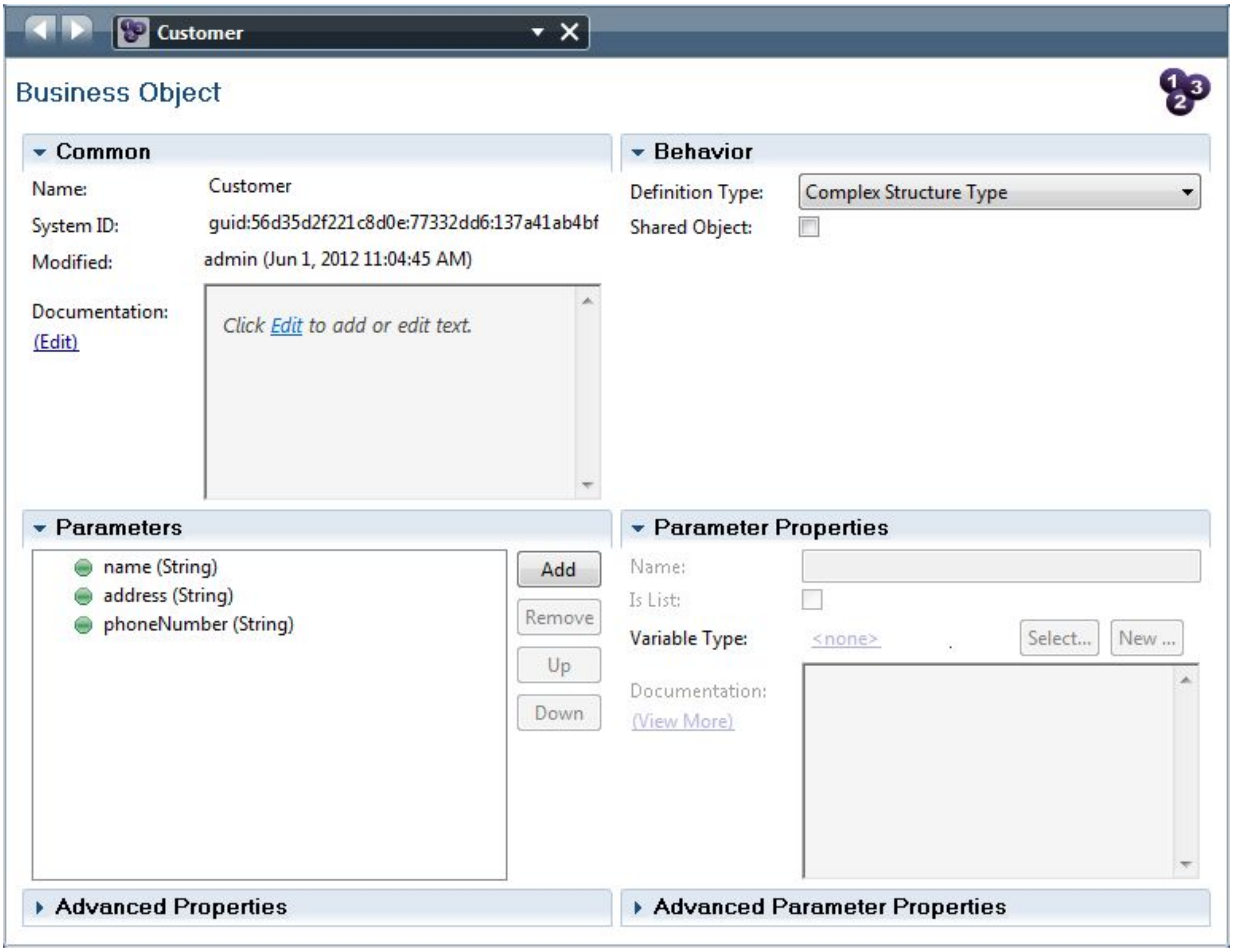
Business objects represent that data that is used during the running of a process.

Name: Customer

Once the data type has been created, the fields of the new structure may be defined. IBPM calls what I call field"Parameters". Some products call such data entries "fields", some call them "attributes", others call them "properties" … so why should we not allow IBPM to call them "Parameters".
一旦数据类型被创建后,新结构的属性字段能够去定义, 比较有趣的是IBPM 称这些为 “参数”,一些其他厂商的产品称这些数据实体为“fields", 一些称为 "attributes", 其他的称为"properties”... 所以,为什么IBPM不能称他们为 “参数".

An arbitrary number of parameters can be added to the structure by pressing the "Add" button.
点击 "Add" 按钮能够添加任意数量的参数到结构里。

When a parameter is added, its name and data type can be defined. The data type of a parameter can itself be a complex type allowing for a tree like structure to be created. The parameter can also be flagged as a "List" allowing the one parameter to hold zero or more values.
添加一个参数的时候，能够去定义它的名字和数据类型， 参数的数据类型能够是一个组合类型，能够创建一个树形结构，参数同样能够标示为能够持有0个或多个数据的列表。


Customer

Business Object

Common

| | |
|---|---|
| Name: | Customer |
| System ID: | guid:56d35d2f221c8d0e:77332dd6:137a41ab4bf |
| Modified: | admin (Jun 1, 2012 11:04:45 AM) |
| Documentation: (Edit) | Click Edit to add or edit text. |

Behavior

| | |
|---|---|
| Definition Type: | Complex Structure Type |
| Shared Object: | ☐ |

Parameters

- name (String)
- address (String)
- phoneNumber (String)

Parameter Properties

| | |
|---|---|
| Name: | |
| Is List: | ☐ |
| Variable Type: | <none> |
| Documentation: (View More) | |

When a variable type is created, that type is available to all artifacts within the defined Process Application. The type can alternatively be defined within a Toolkit and that toolkit re-used across multiple Process Applications. Creating a Toolkit of data types used across your multiple projects is a good idea.

当一个变量类型被创建后，该类型可用于已定义的 PA 里的任何工件。这样的类型当然也可以被定义在被多个 PA 所引用的 Tookit 里面，创建一个可以被多个项目所引用的 Toolkit 里面的数据类型会是一个很好的主意。

List Variables

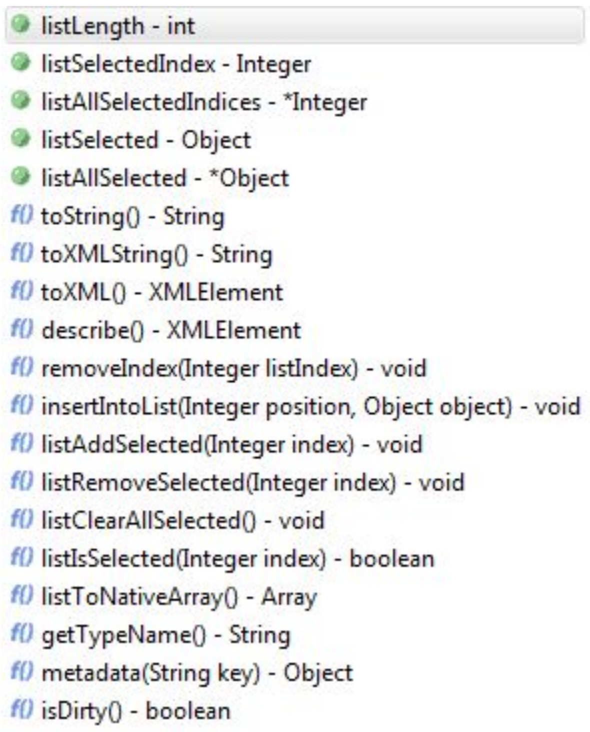
**列表型变量**

A variable can be defined as a list of some data type. Take care to realize that an IBM BPM List is not the same as a JavaScript array. The IBM BPM List data type is mapped to a much richer/different set of semantics. The IBM BPM List data type has the notion of a set of selected items from within the list. This means that given a List, we can also ask that list "Which (if any) items in that list has the property called 'selected'?". This becomes very useful when considering UI based functions.

一个变量可以被定义成一些数据类型的列表。要注意一个 IBM BPM 列表是不同于 Javascript 数组的。IBM BPM 列表数据类型是被映射到一个非常丰富而又不同的系列语法里。IBM BPM 列表数据有个‘选中的系列项’的概念。也就是说，给定一个列表，我们就能知道”在列表里那个（那些）项会有‘selected’这个属性”，这就在考虑基于 UI 的一些功能时变得非常有用了。

The List variable has the following operations and properties defined upon it.

列表变量有以下操作方法和属性的定义：



● listLength - int
● listSelectedIndex - Integer
● listAllSelectedIndices - *Integer
● listSelected - Object
● listAllSelected - *Object
f() toString() - String
f() toXMLString() - String
f() toXML() - XMLElement
f() describe() - XMLElement
f() removeIndex(Integer listIndex) - void
f() insertIntoList(Integer position, Object object) - void
f() listAddSelected(Integer index) - void
f() listRemoveSelected(Integer index) - void
f() listClearAllSelected() - void
f() listIsSelected(Integer index) - boolean
f() listToNativeArray() - Array
f() getTypeName() - String
f() metadata(String key) - Object
f() isDirty() - boolean

A common mistake that is made by BPM programmers using lists within JavaScript is to create the list variable but neglect to create the entries within the list.

For example:

Wrong

```
tw.local.myVar = new tw.object.listOf.MyBO();
tw.local.myVar[0].fieldA = "Hello";
```

Right

```
tw.local.myVar = new tw.object.listOf.MyBO();
tw.local.myVar[0] = new tw.object.MyBO();
tw.local.myVar[0].fieldA = "Hello";
```

A list variable tracks which values are "selected". An algorithm to remove selected items, the following snippet may be used:

```
for (var i = 0; i < tw.local.myList.listLength; i++) {
  if (tw.local.myList.listIsSelected(i)) {
    tw.local.myList.removeIndex(i);
    i--;
  }
}
```

一个BPM程序员常见的错误是，在JavaScript脚本里创建了列表变量却遗漏了创建列表里的实体。

例如:

错误的写法

```
tw.local.myVar = new tw.object.listOf.MyBO();
tw.local.myVar[0].fieldA = "Hello";
```

正确的写法

```
tw.local.myVar = new tw.object.listOf.MyBO();
tw.local.myVar[0] = new tw.object.MyBO();
tw.local.myVar[0].fieldA = "Hello";
```

列表中的跟踪变量‘selected’， 可以利用以下算法去移除选中项

```
for (var i = 0; i < tw.local.myList.listLength; i++) {
  if (tw.local.myList.listIsSelected(i)) {
    tw.local.myList.removeIndex(i);
    i--;
  }
}
```

To add an item to the list, the "insertIntoList()" method can be used. It is not a "replace" item function but instead will move other entries up as needed.

为了加入列表项，‘insertIntoList()’方法可以达到目的。它不是要‘替代’某项值而是根据需要去移动其他项的值。

Q: If I remove an element from the list using removeIndex, is the selected set "adjusted"?

问题：如果我使用‘removeIndex’从列表中移除一个元素，是不是选中项会‘调整’呢？

Setting defaults on variables

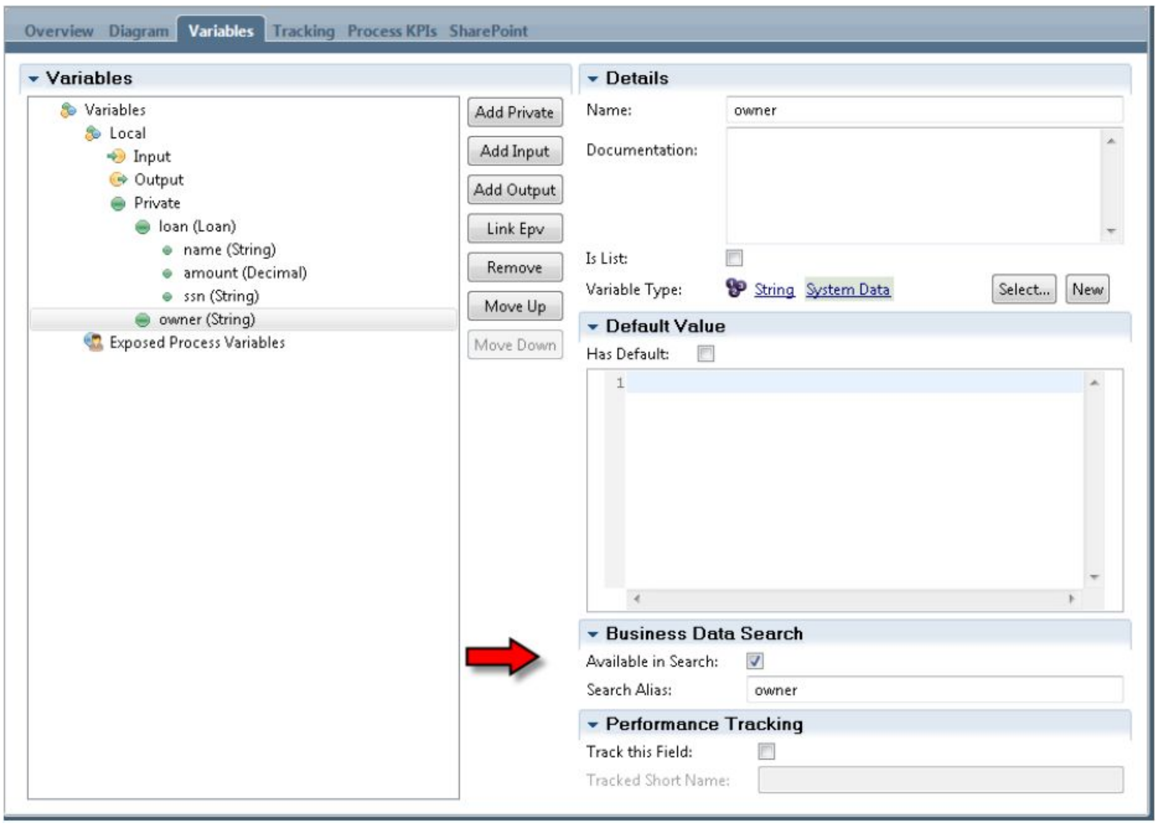
**变量默认值设置**

When a variable is defined in a service or BPD as either input or private, a default value can be provided. The default provides an initial value for private variables and a default value for input variables that are not supplied. The default value is set in the variables tab if the "Has Default" check box is selected. If selected, the text area provides a place in which the default values can be placed. The appearance of the entry of the default values changes depending on whether or not the PD is in advanced mode. If it is in advanced mode, JavaScript can be entered as long as it results in a value for the variable.
**当在一个服务或流程定义中定义一个输入或私有变量时，也可以给该变量指定默认值。默认情况下，输入或私有变量都是没有提供默认值的。在变量标签页上的‘Has Default’复选框选中时才会赋予默认值。当选中后，文本区域会提供输入默认值的位置，能不能显示默认值条目取决于 PD 是不是在高级模式。如果是出于高级模式，能够输入 JavaScript 脚本作为该变量的值。**

Making variables search-able

**创建可搜索的变量**

Process Portal can be used to find instances of processes based upon the values contained in variables within the process. Because IBPM has to do more work to index the values of the variables that are to be looked up, this feature has to be enabled on a variable by variable basis. In the declaration of variables, there is an option to enable searching for Business Data.
流程门户可以用作找到基于值在变量流程里的流程实例。因为 IBPM 要做更多的工作去索引出需要查找的变量值，这个特征需要通过变量的特征来启用。在变量的声明时候， 有一个选项来用来查询业务数据。



Variables

- Variables
  - Local
    - Input
    - Output
    - Private
      - loan (Loan)
        - name (String)
        - amount (Decimal)
        - ssn (String)
      - owner (String)
  - Exposed Process Variables

Details

Name: owner

Documentation:

Is List: ☐

Variable Type: String System Data

Default Value

Has Default: ☐

```
1
```

Business Data Search

Available in Search: ✓

Search Alias: owner

Performance Tracking

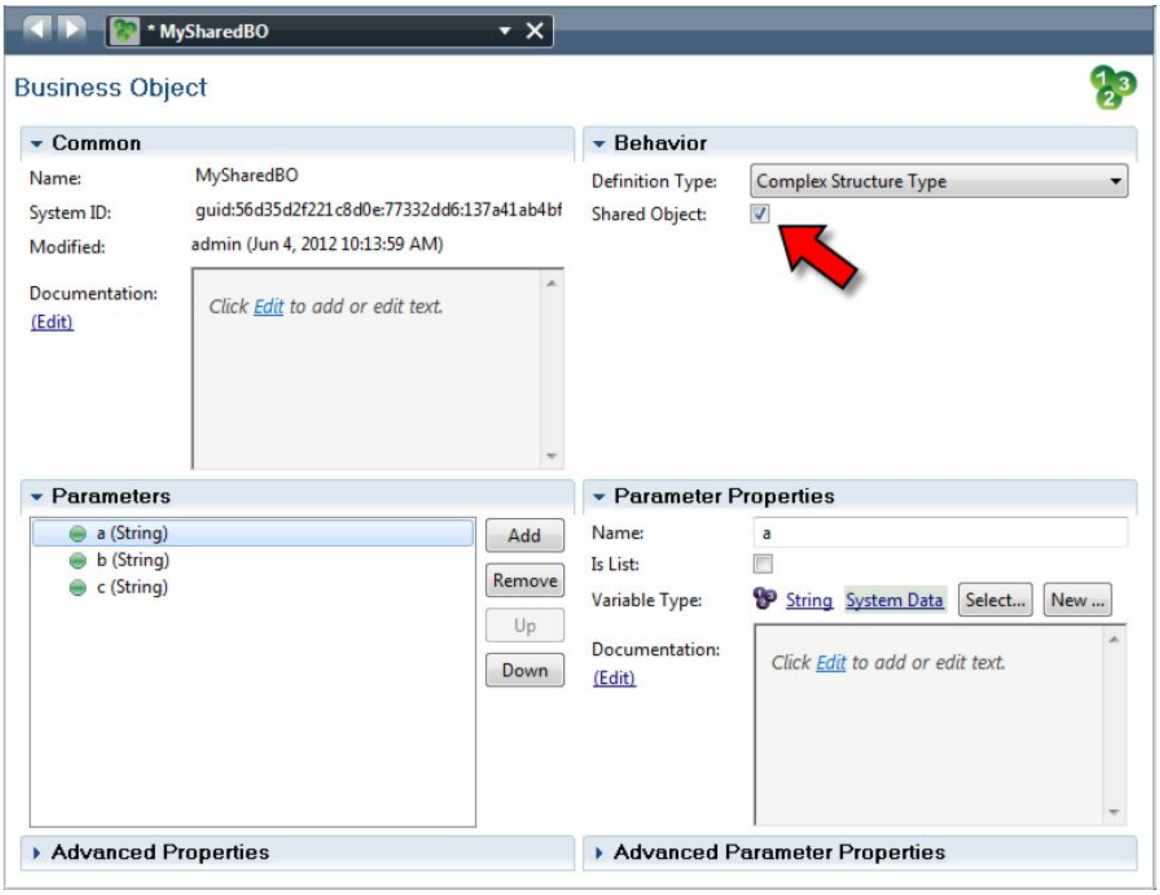
Track this Field: ☐

Tracked Short Name:

Defining and using shared objects

**定义和使用共享对象**

From IBPM v8 onwards, a capability called Shared Objects was added. Shared Objects have the ability to mark a Business Object definition as being a Shared Object. See the following image:
从 IBPM v8 之后， 引入了一个被称为共享对象的功能， 共享对象可以把一个业务对象标记为一个共享对象。看接下来的图：


* MySharedBO

Business Object

Common

Name: MySharedBO

System ID: guid:56d35d2f221c8d0e:77332dd6:137a41ab4bf

Modified: admin (Jun 4, 2012 10:13:59 AM)

Documentation: (Edit) Click Edit to add or edit text.

Behavior

Definition Type: Complex Structure Type

Shared Object: ✓

Parameters

a (String)
b (String)
c (String)

Add Remove Up Down

Parameter Properties

Name: a

Is List: ☐

Variable Type: String System Data Select... New ...

Documentation: (Edit) Click Edit to add or edit text.

Advanced Properties

Advanced Parameter Properties

Note that Business Objects flagged as shared have a "green" icon as opposed to the "purple" icon of non-shared business objects:

注意，业务对象是通过‘绿色’的图标来标记共享，相对来说，非共享业务对象会使用‘紫色’图标。

| | |
|---|---|
| ● | 非共享业务对象 |
| 🟢 | 共享业务对象 |

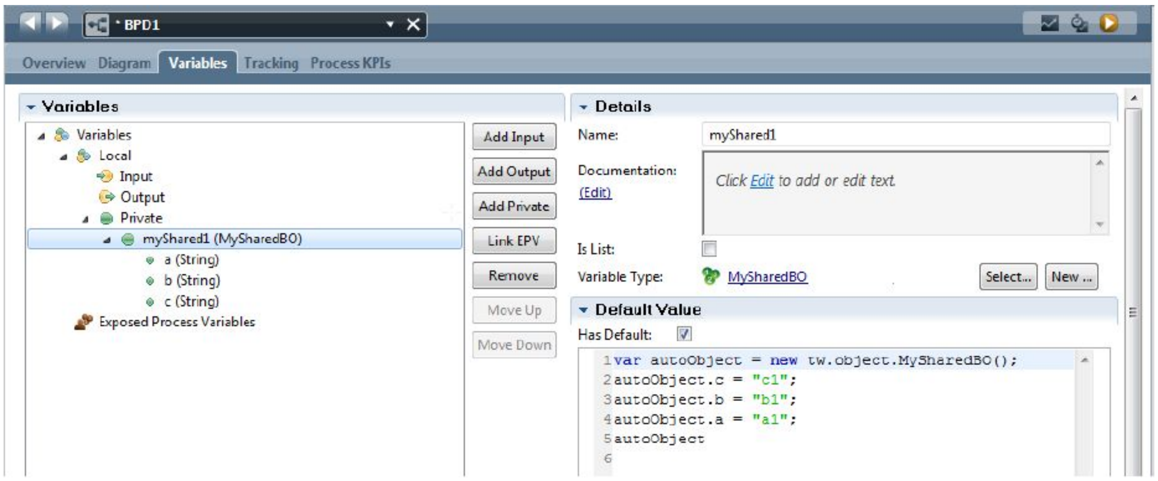
Once flagged, variables created of this BO type are now passed by reference as opposed to passed by value around the environment. What this means is that multiple processes or parallel steps within processes can "see" and "update" the variable simultaneously. If we wish to have multiple processes work with the shared object simultaneously, then we need to pass a "key" that can be used to access the variable in a different process.

一旦被标记为共享， 被创建的变量会通过值传递相反的方式在环境中进行引用传递。这就意味着多个流程或者平行的步骤在流程里能同时地‘查看’和‘更新’变量。 如果我们希望有多个流程同时地工作在一个共享对象上，

那么我们需要传递一个能够用作访问在不同流程中的变量的‘键’。

This is starting to get subtle so let us try and break this down. Consider the previous screen shot which shows the creation of a new Business Object type called "MySharedBO". Now consider an instance of a process which declares a variable of type "MySharedBO".

这个说法有些微妙，让我们来分解说明一下。 考虑到前面的截图已经创建了一个叫‘MySharedBO’的新业务对象，现在可以假定有一个申明了‘MySharedBO’类型变量的流程实例。



Variables
- Variables
  - Local
    - Input
    - Output
    - Private
      - myShared1 (MySharedBO)
        - a (String)
        - b (String)
        - c (String)
  - Exposed Process Variables

Details

Name: myShared1

Documentation: (Edit) Click Edit to add or edit text

Is List: ☐

Variable Type: MySharedBO

Default Value

Has Default: ☑

```
var autoObject = new tw.object.MySharedBO();
autoObject.c = "c1";
autoObject.b = "b1";
autoObject.a = "a1";
autoObject
```

We should be able to easily imagine an instance of the MySharedBO Business Object being "held" somewhere for future reference. Imagine that we wished to have a second process leverage the same shared object. The second process can declare a variable of type MySharedBO but … to get a reference to the object, will have to execute:

  tw.local.myShared2 = new tw.object.MySharedBO(keyValue);

but wait … where did the "keyValue" parameter come from?

On the original reference to the Business Object, we can access its "key" value with:

  var keyValue = tw.local.myShared1.metadata("key");

The 'metadata("key")' method returns a string representation of the key to that shared object. Note that "key" in this call is actually an explicit keyword and not a variable name.

我们应该很容易地想到， MySharedBO 业务对象的实例会被某个地方‘持有’以备将来的使用。 假设我们希望有第二个流程利用同一个共享对象。 第二个流程能定义一个类型为 MySharedBO 的变量，但…获取这个对象的引用，将要执行：

  tw.local.myShared2 = new tw.object.MySharedBO(keyValue);

但是等等…‘ keyValue’参数是从哪来的？

在对业务对象的原始引用上，我们能访问它的‘Key’值用：

```
  var keyValue = tw.local.myShared1.metadata("key");
```

‘metadata("key")’方法会返回一个代表 key 值的字符串给那个共享对象。注意， “key”实际上是一个显式的关键字而不是一个变量名。

When a change is made to the data contained within the Shared Object that data is written to a database behind the scenes. Only when the data is written, will the data be able to be seen by a partner. The data is written when a service that modifies it completes or when an explicit call to the save() method (found on the business object) is called.

共享对象中包含的数据的变化在幕后被写入数据库。只有当数据被写入时, 共同使用者将可以看到这些数据。当一个服务修改完成或者当一个显式 save()方法(业务对象)的调用时，该数据被写入。

The data associated with the Shared Object is associated with a process instance that created it. When the process instance data is cleared, so is the Shared Object.

数据相关的共享对象与一个创建的流程实例相关联， 当一个流程实例数据被清除，共享对象也被清除。

One important area that requires further consideration is **when** a process sees the changes that another process may have made to the data. The answer is **not** quite what one may trivially expect. Imagine two process instances, P1 and P2. P1 creates an instance of a Shared Object and sets its values. P2 then also creates an instance referencing the key from P1. Now P2 changes a value in the data and executes a save(). What does P1 now see?

要进一步的考虑的一个重要问题是，什么时候一个流程看见另外一个流程做的数据更改， 答案是不能完全确定。假设两个流程实例，P1 和 P2，P1 创建一个共享对象的实例并且设置它的值，然后 P2 也创建一个引用来自 P1 Key 的实例。 现在 P2 更改一个数据的值并且执行一个 save（）方法。现在 P1 能看到什么呢？

The choices would appear to be:

1. The original value of the field as originally set by P1
2. The new value of the field as set by P2

The answer is "it depends".

结果也许会是:

1. 字段的原始值和原始 P1 设置的值一致
2. 字段的新值和 P2 设置的一致

答案是‘看情况’。

This is the concept of the "latest version of the data in the Shared Object". This is what will be used when either a new instance of the Shared Object with the given key is created or an existing instance is re-loaded with an explicit load() call. So if P1 executes a load(), it will always get the latest data.
这就是‘共享对象数据的最新版本’概念， 当任何一个新的有给定的 Key 的共享对象实例被创建的时候，或者一个存在的实例被一个显式的 load（）方法重新加载的时候， 那个概念就会被用到。因此如果 P1 执行一个 load（）方法，它会永远获得最新的数据。

A reload of the data within P1 happens automatically when a step in a process is transitioned or when a task is woken up. However, within the context of a single straight through service, the values of the variable will not change unless it is explicitly re-loaded. The transition from step to step in a process is an implicit re-load as is the awakening of the task.
当流程里的某一步骤发生转换或者某一任务被唤醒时，P1 里就会自动地重载数据。然而，在直接通过一个服务的上下文里面，变量的值不会改变，除非它被显式的重新加载。流程里一步一步的转变是一个唤醒任务的隐式重载。

## BPD Variables and Service Variables - Mapping

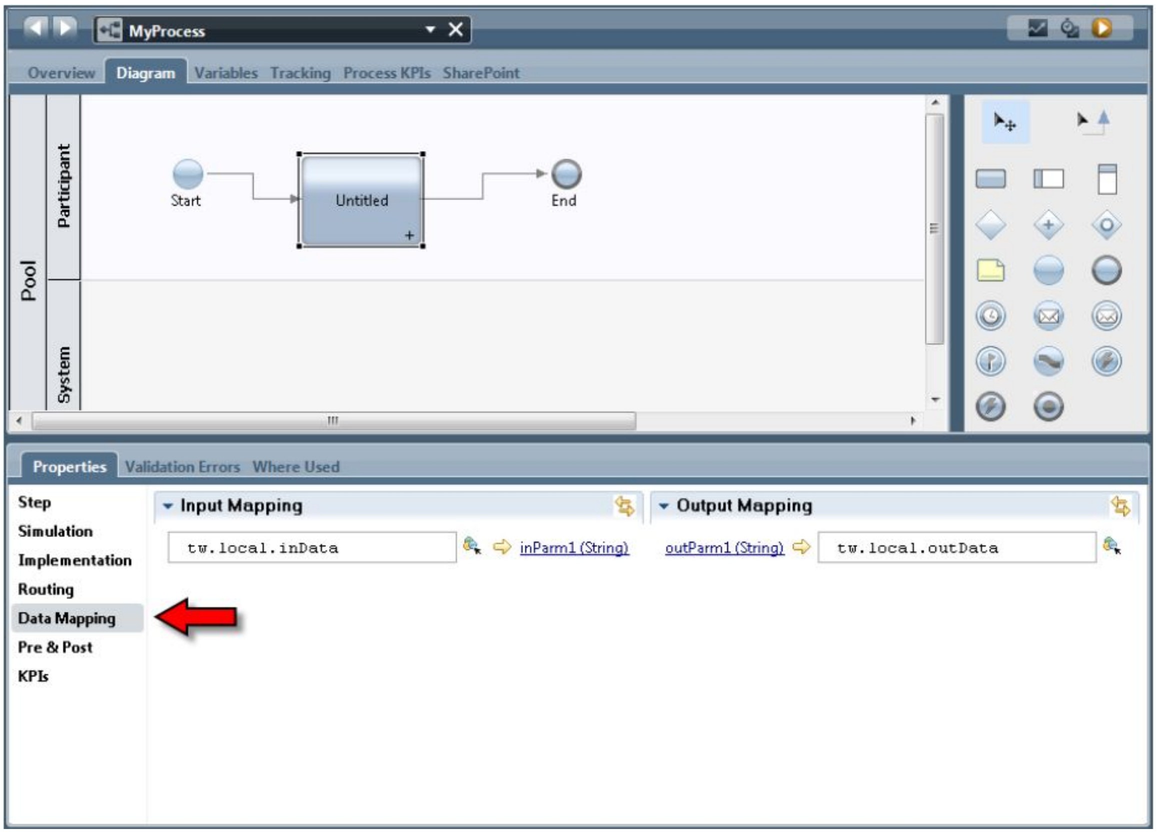
## **BPD 变量和 Service 变量 - 映射**

A Business Process Definition has variables associated with it. Some variables might be input to the process, some may be output from the completed process and some may be local variables only visible to the process.
一个业务流程定义有相关的变量。有些变量可能是流程输入，有些可能是从结束的流程输出和有些可能是只对流程可见的本地变量。

In addition to the a BPD having variables, so too can a IBPM Service. Just like a BPD, a IBPM Service has input, output and local variables. These are defined in the definition of the IBPM Service.
除了 BPD 可以有相应的变量，一个 IBPM Service 也可以有。 像 BPD 一样，IBPM Service 有输入，输出和本地变量。

这些被定义在 IBPM Service 的定义里。

A IBPM Service is usually invoked from within the context of a BPD through an Activity node. When the node is defined, it is associated with a IBPM Service definition. If we think about this for a moment, we see that the IBPM Service has some input variables that it is expecting to contain data and will return some output variables that make data available to the process as a whole. These variables in the IBM service need to be "mapped" to variables that are in scope within the BPD that is calling the service.
一个 IBPM Service 通常被一个 BPD 经过一个活动节点的上下文里面调用。节点的定义会和一个 IBPM Service 的定义相关联。如果我们思考一下，我们看见这个 IBPM Service 有一些待输入变量和返回某些输出变量， 使数据对于流程可用的过程作为一个整体。 这些在 IBM Service 变量需要被‘映射’到正在调用 Service 的 BPD 的变量范围里。



tw.local.inData → inParm1 (String)
outParm1 (String) → tw.local.outData

When an Activity is selected in the BPD editor, the properties section shows a Data Mapping tab. Selecting this shows the expected input variables to the associated IBPM Service as well as the generated output variables. These can then be mapped to variables within the current BPD. The

icon beside the "boxes" ( )provides for a smart search of the defined variables so we do not need to remember the JavaScript variable names.
当一个活动在 BPD 编辑器里面被选择， 属性部分显示一个数据映射标签。选择这个待输入变量对应相关的 IBPM Service 和生成的输出变量。然后这些能被映射到在当前里的变量。‘盒子’旁边的图标（ ）提供一个已定义变量的智能查询，因此我们不需要记住 Javascript 变量名。

If a variable is defined as a complex type, it will need to be initialized before values can be assigned to it. Its initial value is "null" which has no fields. A good way to achieve initialization is to check the has default check box in the data type definition.
如果一个变量被定义为一个复合类型， 它将需要在被赋值前被初始化。它的初始化值是‘null’，它没有任何的字段。 勾选数据类型定义的‘has default’复选框是一个实现初始化的好办法。

If a variable is defined as a complex type, it will need to be initialized before values can be assigned to it. Its initial value is "null" which has no fields. A good way to achieve initialization is to check the has default check box in the data type definition.
如果一个变量被定义成复合类型，它在初始化之前就可以指定值给它。如果他的值是空，没有字段，有一个好的方式去达成初始化，就是在数据类型定义处勾选‘has default’复选框。

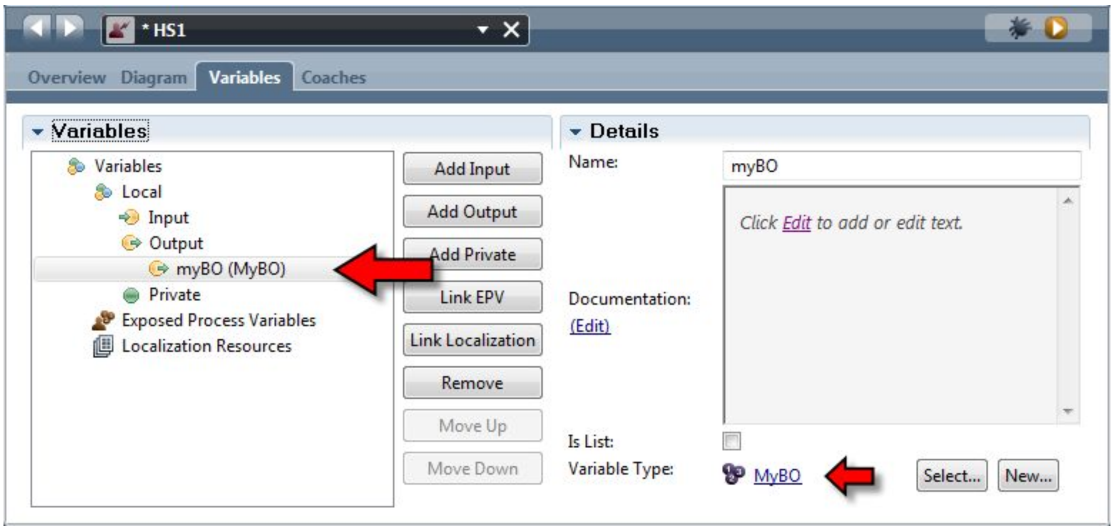
Variable identity and UUID

**变量特征和 UUID**

Imagine you create a new Business Object type called "MyBO". What is the identity of that Business Object type? As you might imagine, there is more to this simple question than meets the eye.The normal way of thinking is that if I create a data type called "X" then it has an identity of "X". Unfortunately, IBM BPM doesn't think that way. Internally within the heart of the product, all artifacts (including all data types) are given a generated UUID value. It is this opaque UUID that the product uses to track references to data types.
假设你创建一个新的业务对象的类型为”MyBO” 。那么什么才是这个业务对象的类型的标识？ 如你所想，这个看似简单的问题，其实很繁琐. 按照通常的思维，如果我创建一个数据类型为“X”，那么我们会认为它的标识就是“X”。很不幸的是，IBM BPM并没有如我们想象的一样，在产品的内部核心中所有的部件（包括所有数据类型）都会生成独一无二的UUID值。正是这种不可见的UUID值，产品使用它来进行跟踪引用数据类型。

To illustrate, consider the following Human Service definition which has an output parameter called "myBO" of data type "MyBO":

为了说明这一点，给下面的人工服务定义一个名为“myBO”的输出参数，它的类型为“MyBO”



Name: myBO

Documentation: (Edit) Click Edit to add or edit text.

Is List: ☐

Variable Type: MyBO

The Human Service definition (called HS1) can be read as:

"This service returns an output called 'myBO' of type 'MyBO'".

However, this is only an illusion. The way the product really thinks of this is:

"This service returns an output called 'myBO' of type 'guid:7b9691c16dd21a42:1e6925ca:142dd70d25a:-7ffa'"

Hmm. That is quite a difference. If we switch on advanced mode in PD and then look at the definition of the business object, we can see that UUID there:

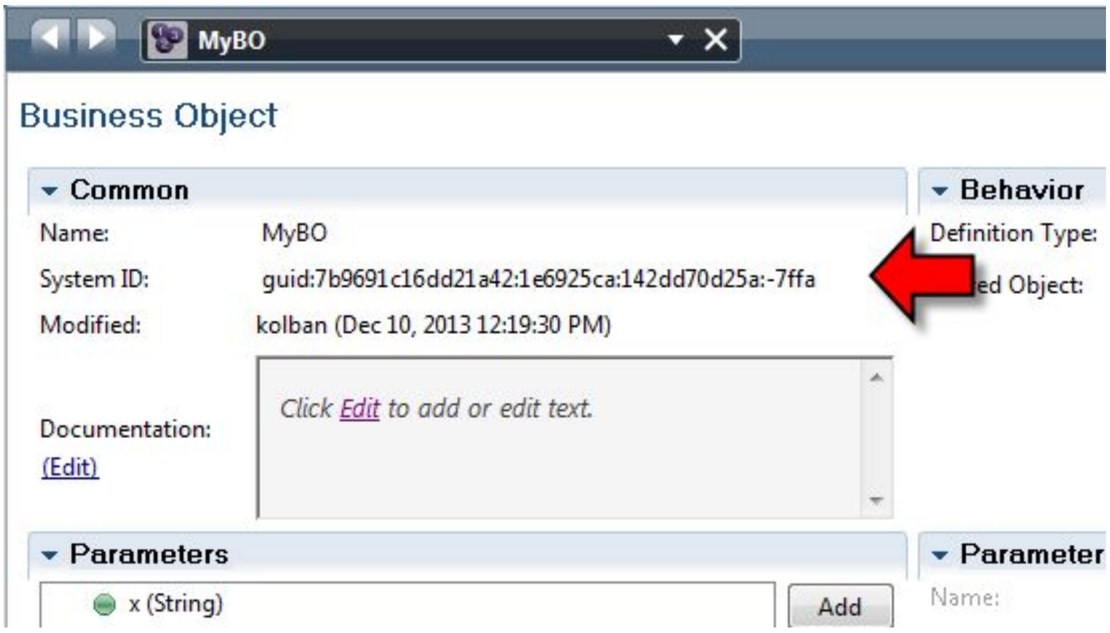
这个人工服务（名为：HS1）可以理解为：

“这个人工服务返回一个名为‘myBO’类型为‘MyBO’的输出参数”

然而这是一个错觉，产品真正的思维方式是：

“这个人工服务返回一个名为'myBO'类型为'guid:7b9691c16dd21a42:1e6925ca:142dd70d25a:-7ffa'的输出参数”

恩，这是相当不同的，如果我们在PD里打开高级模式，然后看看业务对象的定义，我们能看到它的UUID在那里。



Name: MyBO
System ID: guid:7b9691c16dd21a42:1e6925ca:142dd70d25a:-7ffa
Modified: kolban (Dec 10, 2013 12:19:30 PM)
Definition Type:
ed Object:

Normally, I wouldn't try and uncover internals knowledge such as this as it is usually irrelevant to the model and used of the BPM product itself however, there are some ugly ramifications associated with this implementation story.

正常情况下，我不会尝试去发现内部结构与知识等，这通常是无关紧要的模型和BPM产品本身的东西，不过还是会有一些理解上的分歧在运用过程中。

As a thought process, consider the following:

1. Create a new Business Object type called MyBO
2. Create a Human Service that returns a variable of type MyBO
3. Delete the Business Object type called MyBO
4. Create a new Business Object type called MyBO identical in structure to the first

模拟一个思考过程，考虑一下几点:

1.创建一个新的业务对象类型名为MyBO。

2.创建一个新的人工服务返回一个类型为MyBO的变量。

3.删除MyBO这个业务对象类型。

4.创建一个新的业务逻辑类型名为MyBO与第一次的结构相同。

Are we now consistent? It would seem so, we have a business object called MyBO which contains all the fields we want. Unfortunately, we are broken.

当前的MyBO与之前的MyBO是一致的么？似乎是这样的，我们有一个业务逻辑对象MyBO其中包含所有想要的字段，不

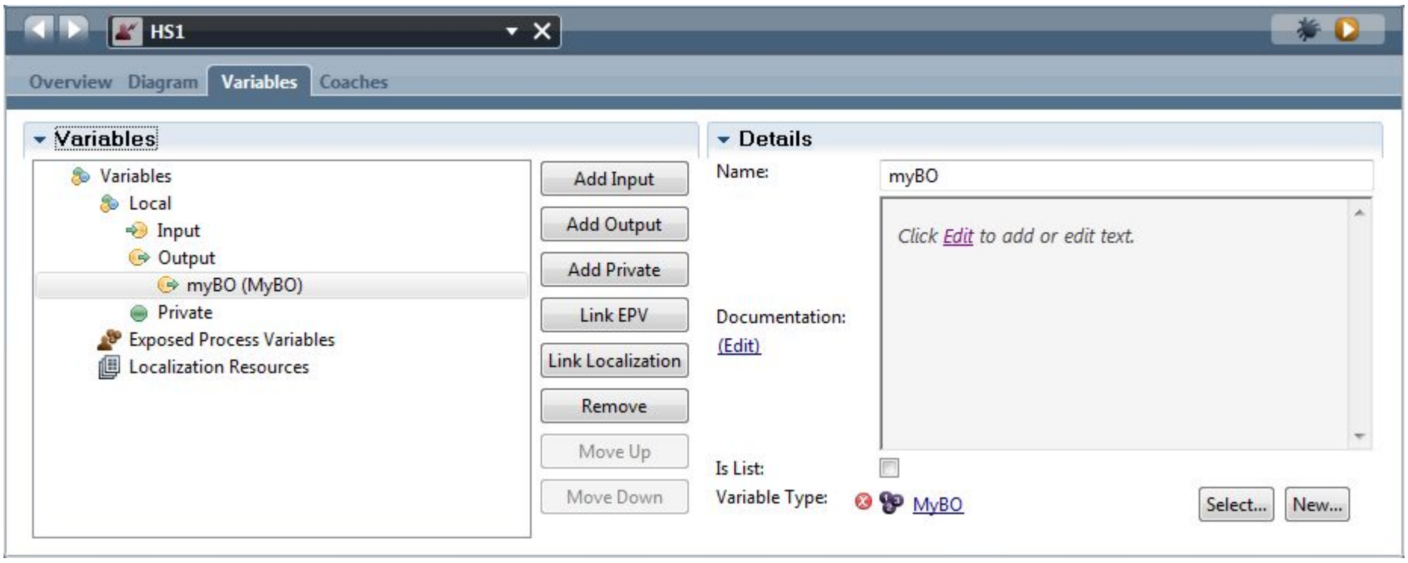
幸的是，我们都想错了。

In step 2, when we create a Human Service that returned a variable of type MyBO, the Human Service actually defined that it returned a data type of a UUID. That UUID was associated with the data type created in step 1. When we performed step 4, a brand new UUID data type was created that is not the same as that of step 1.

在第二步的时候，我们创建一个人工服务，它返回了一个类型为MyBO的变量，实际上人工服务定义的返回数据类型是一个UUID .那个UUID相关的数据类型是在第一步中创建的。然后我们执行第4步 ， 一个全新的UUID数据类型的创建与第一步是不一样的。

We end up with the Human Service error showing:

最终我们的人工服务上会显示错误:



Name: myBO

Click Edit to add or edit text.

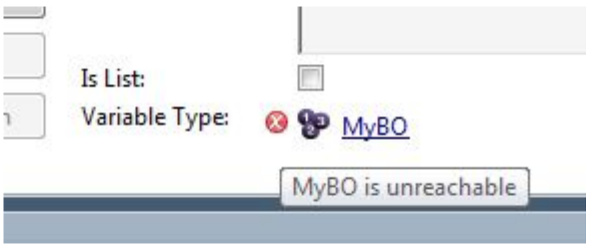
Documentation: (Edit)

Is List:

Variable Type: MyBO

If we hovered over the error we would see:

如果我们查看这个错误我们会看到:


Is List:

Variable Type: MyBO

MyBO is unreachable

But yet if we examined the list of data types, we would see that MyBO is a defined data type.

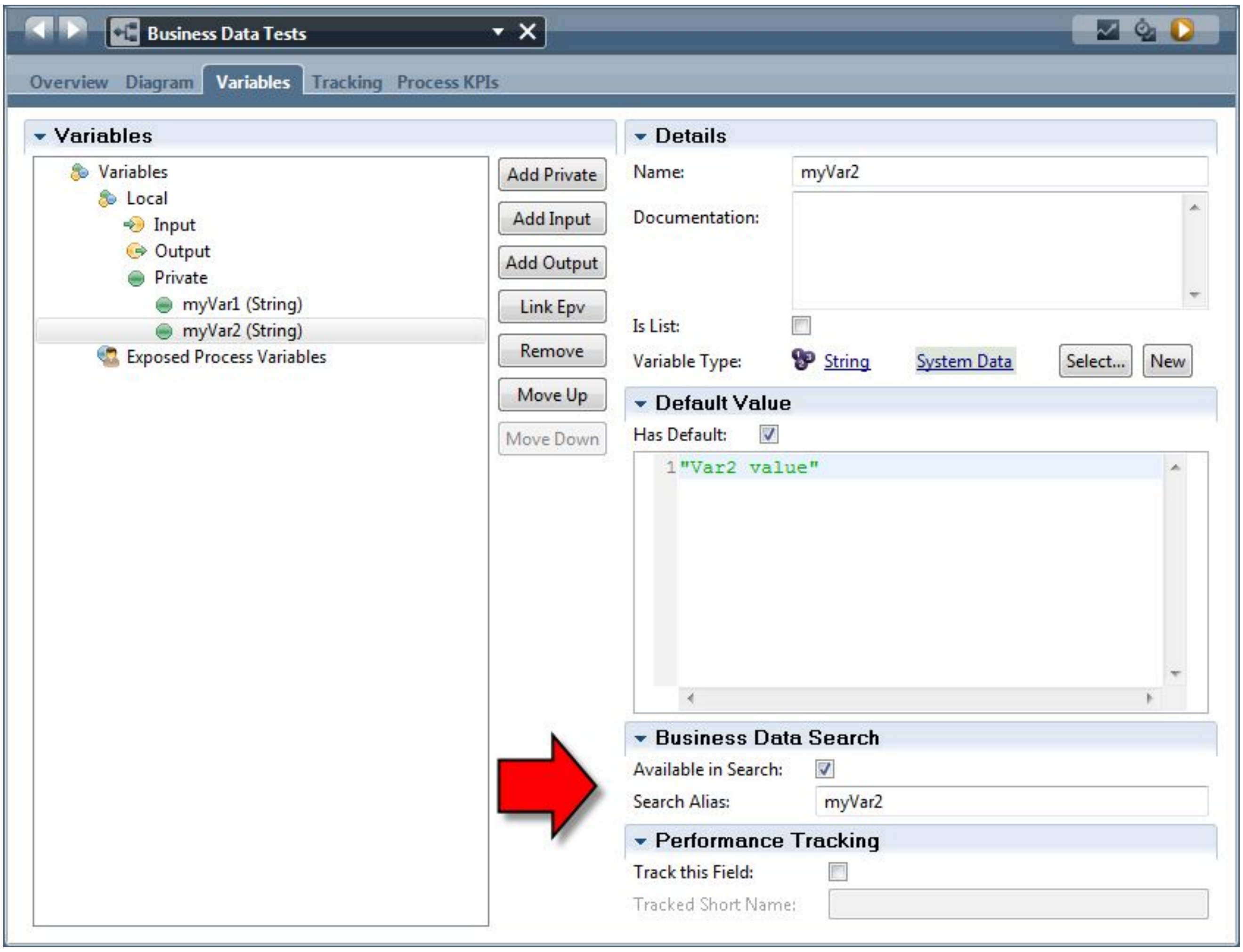
The fix would be to remap the variable type to the new MyBO definition. To most folks (including myself) this is an odd situation to be in.
The moral of the story is to be very careful with the concept of deleting and recreating data types. Deleting something and then recreating it is **not** the same as modifying it. This story also comes into play when you think that you can move data types into toolkits.
但是如果我们检查数据类型的列表，我们将看到这个那个MyBO 是一个已经定义的数据类型。解决办法是重新映射这个变量类型为新定义的MyBO类型。对于大多数人来说（包括我自己）这是个奇怪的情况。
这则故事的寓意是要非常小心删除和重新创建数据类型的概念。删除数据类型然后重新创建它并不等同于修改它，当你认为你可以移动数据类型到toolkits的时候也可以参考这个故事。

Exposing Business Data for Searches

**暴露业务数据到Search功能**

Imagine that we have hundreds (perhaps thousands) of process instances running handling orders for customers. If a customer calls in and asks "What is the status of my order #1234?" how can we find the process associated with this?
假设我们有为数百（也许成千上万）的流程实例运行为客户处理订单,如果客户打电话过来问“我的订单#1234的状态是什么？”我们怎么样才能找到与这相关的流程实例？

When process variables are defined in IBPM, we have an opportunity to flag them as being available in Business Data Searches. This means that we can execute searches using the exposed names and values of variables in a process. For example, if a process has a variable called "orderNumber" and we exposed that for searching, we can now execute a search for process instances where "orderNumber = 1234".
当在IBPM中定义流程变量的时候，我们有机会标记它们,使得它们可以在业务数据中被搜索到。这意味着我们可以使用执行搜索找到过程中暴露出的变量名称和值.例如,如果一个流程有一个名为“orderNumber”的变量并且我们将它暴露到search功能 ,那么我们现在可以执行一个流程实例的search功能，条件为"orderNumber = 1234"。



Name: myVar2

Documentation:

Is List: ☐

Variable Type: String System Data

Has Default: ✓

```
"Var2 value"
```

Available in Search: ✓

Search Alias: myVar2

Track this Field: ☐

Tracked Short Name:

Accessing variables from JavaScript

## 访问来自 JavaScript 的变量

It is common to want to read or write the values of variables from within a JavaScript environment. When a variable is defined at the BPD or service level, it is available within that BPD or service within the JavaScript scope of:

tw.local.*

通常想要读取或写入在一个JavaScript环境中的变量的值的时候。当一个变量定义在BPD或服务级别，它在BPD或服务里面的JavaScript范围里是可见的：

通过tw.local.*取得可见变量。

The IBPM PD commonly provides entry assist to show you the names of the in-scope variables that can be used in a JavaScript fragment.

Note that the entry assistance accessed by CTRL+Space in the editor can be used to substitute the characters

″twl″ for the string ″tw.local.″.

IBPM PD 通常提供入口帮助向您展示JavaScript环境里的作用范围内变量的名称 。

注意，在编辑器中可以使用 替代字符串’twl ‘得到“tw.local”输入联想通过CTRL+Space访问 .

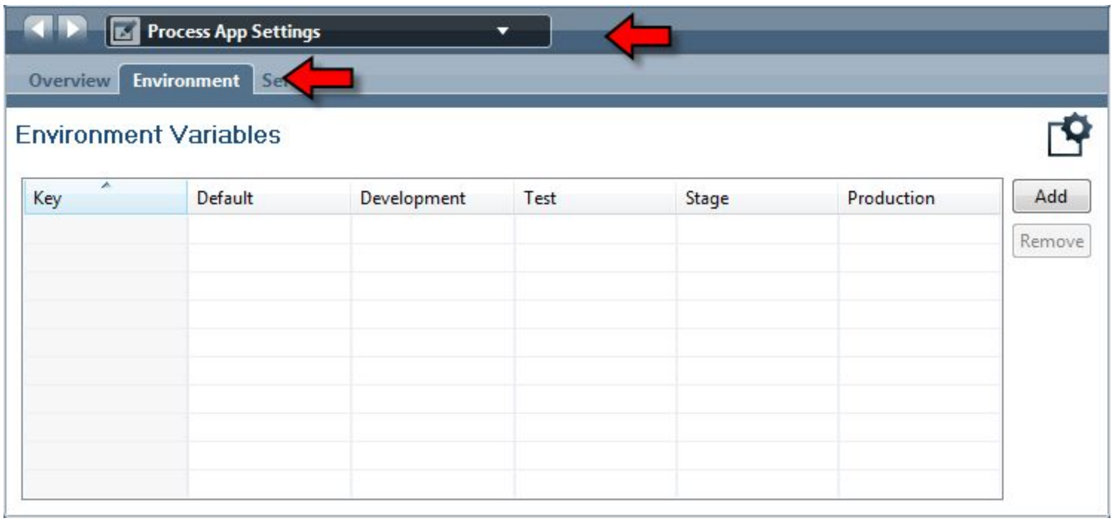
List variables can be accessed using square bracket notation with an index value starting at 0. When assigning to a list, if an index value is used that is greater than the size of the list, this is acceptable. To make this clear, imagine a list with two existing entries which would be [0] and [1]. If we assign to the list with an index of [3], this would produce an entry for [2] of data type ANY and no value.

如果变量是一个集合，那么我们可以通过方括号访问索引值从0开始。当我们访问一个集合的时候，如果它的索引值大于这个集合的大小，这是可以接受的。详细的说，想象一下一个集合里有下标为[0]和[1]两个元素,如果我们指定一个索引为[3]的话，那么它将产生一个索引为[2]的元素，这个元素的数据类型为ANY并且它是无任何值的。

**环境变量**

Instead of hard-coding values in a process, we may wish to externalize these values and re-use them across our solution. IBPM PD allows us to define environment variables in either Process Applications or in Toolkits. Once defined in there, they can then be referenced in the solution. The environment variables section can be accessed from the Process App Settings within the Environment tab:

我们希望在我们的解决方案里能够外部化某些值然后重用它们，而不是在流程中直接使用硬编码。IBMP PD 允许我们在任何应用程序和 Toolkits 中定义环境变量。一旦定义，他们就能在我们的解决方案中被引用。环境变量可以通过 Environment tab 里的应用程序设置来进行访问。



| Key | Default | Development | Test | Stage | Production |
|---|---|---|---|---|---|

Variables defined in the Process Application can be accessed as:

tw.env.<Env Variable Name>

Variables defined in a Toolkit can be accessed as:

tw.env.toolkit.<Toolkit Name>.<Env Variable Name>

**Note**: When setting initial values of environment variables within Process Designer, make sure that you do **not** surround the values with quotes. There is no need. The values entered are already strings. For some reason, a common error is that folks are adding quotes which adds the quotes themselves into the value of the string:

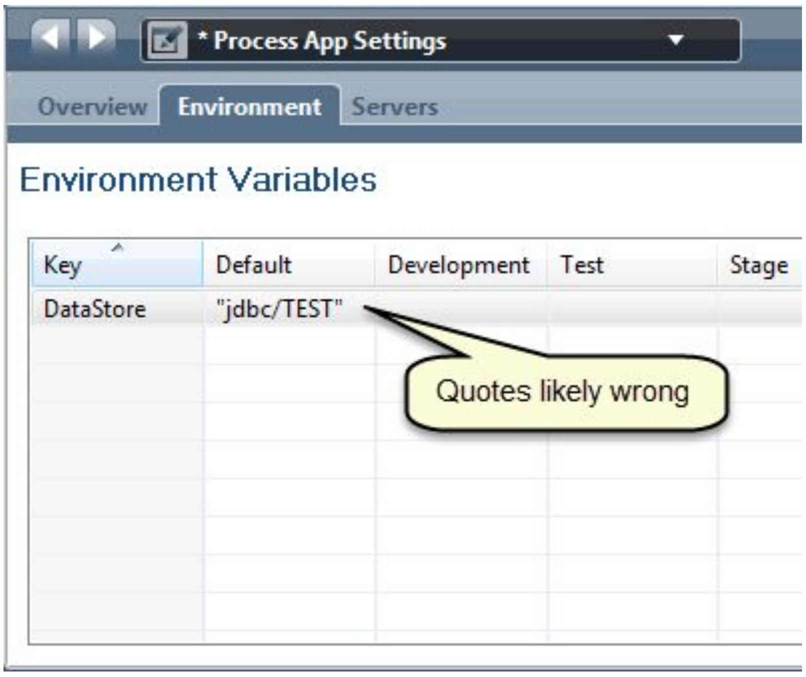
定义在应用程序里的变量可以通过以下方式访问：

tw.env.<Env Variable Name>

定义在 Tookit 的值的访问：

tw.env.toolkit.<Toolkit Name>.<Env Variable Name>

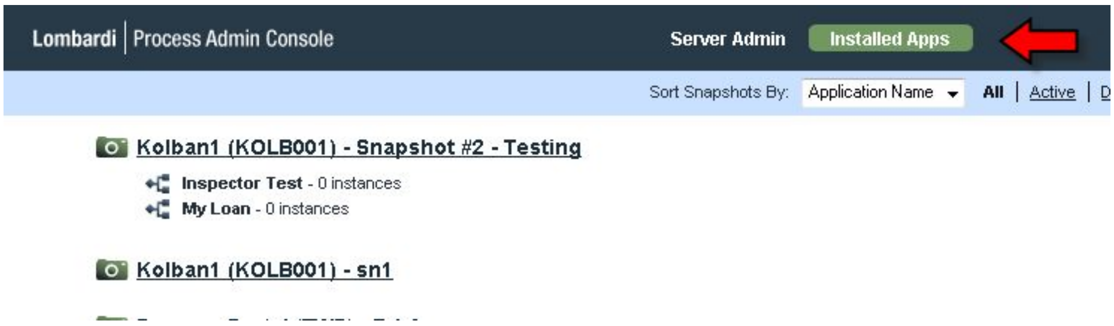
注意：当在流程设计器里给环境变量赋初始值的时候，值不能加引号。因为这个值已经是 String 值。一个常见的错误是人们总是给他们的 String 值加引号。



| Key | Default | Development | Test | Stage |
|---|---|---|---|---|
| DataStore | "jdbc/TEST" | | | |

The values of the environment variables can be changed through the Process Admin Console. After starting it up, select the Installed Apps button:

环境变量的值可以通过流程管理控制台来改变。在程序启动之后，选择’已安装的应用’按钮:


Lombardi | Process Admin Console
Server Admin
Installed Apps
Sort Snapshots By: Application Name
All | Active | D

Kolban1 (KOLB001) - Snapshot #2 - Testing

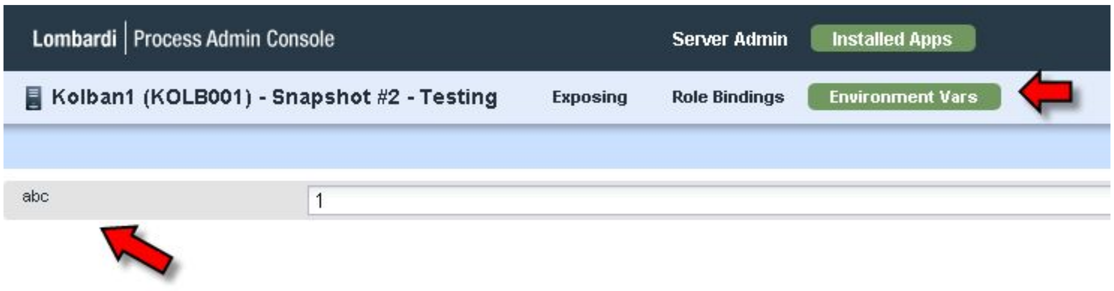
Inspector Test - 0 instances

My Loan - 0 instances

Kolban1 (KOLB001) - sn1

Next, select the application who's environment variables are to be changed. A list of the environment variables and their current values can then be seen:

下一步：选中要修改环境变量的选项。将显示一个环境变量列表并且能看到它们的当前值。


Lombardi | Process Admin Console
Server Admin
Installed Apps
Kolban1 (KOLB001) - Snapshot #2 - Testing
Exposing
Role Bindings
Environment Vars

abc 1

The environment variables default values can be based on the setting of the environment.type attribute set in the install.properties used when IBPM is installed.

To make changes to environment variables, one must be a member of the administrators group.

环境变量的默认值是基于安装 IBPM 时，设置 install.properties 里的 environment.type 属性。改变环境变量的值，必须有管理组成员的权限。

Exposed Process Values (EPVs)

**暴露流程变量（EPVs）**

Consider a business process that utilizes some business data value such as the current tax rate or the interest rate on loans. This value is obviously not a constant as it may change over time. What we want is a way to supply such values to processes while at the same time making them easy to modify as needed. The concept of Exposed Processes Values (EPVs) is the IBPM solution.

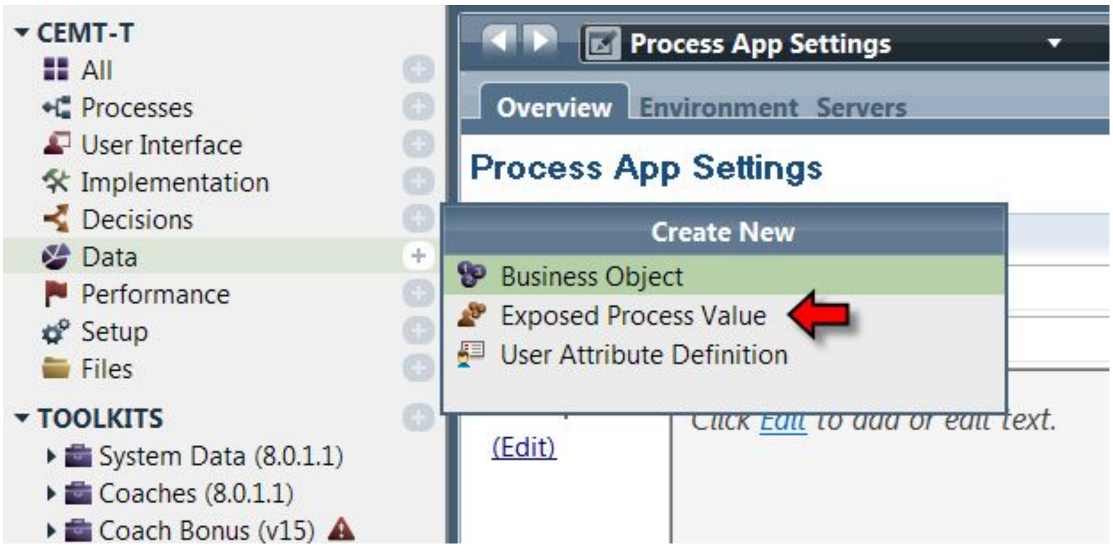
考虑到业务流程会用到一些流程数据例如当前出借的利率或税率。这个值显然不是一个常量，但会随着时间而改变。我们需要一种方式提供这样的流程值同时让它们能轻易的根据需要而改变。流程变量(EPVs)的概念是 IBPM 的解决方案。

An EPV is a named container that holds one or more variables where the values of those variables can be defined through the Process Administration console.

EPV 是一个存储一个或者多个变量的容器,这些变量的值是可以通过流程管理控制台页面来定义的.

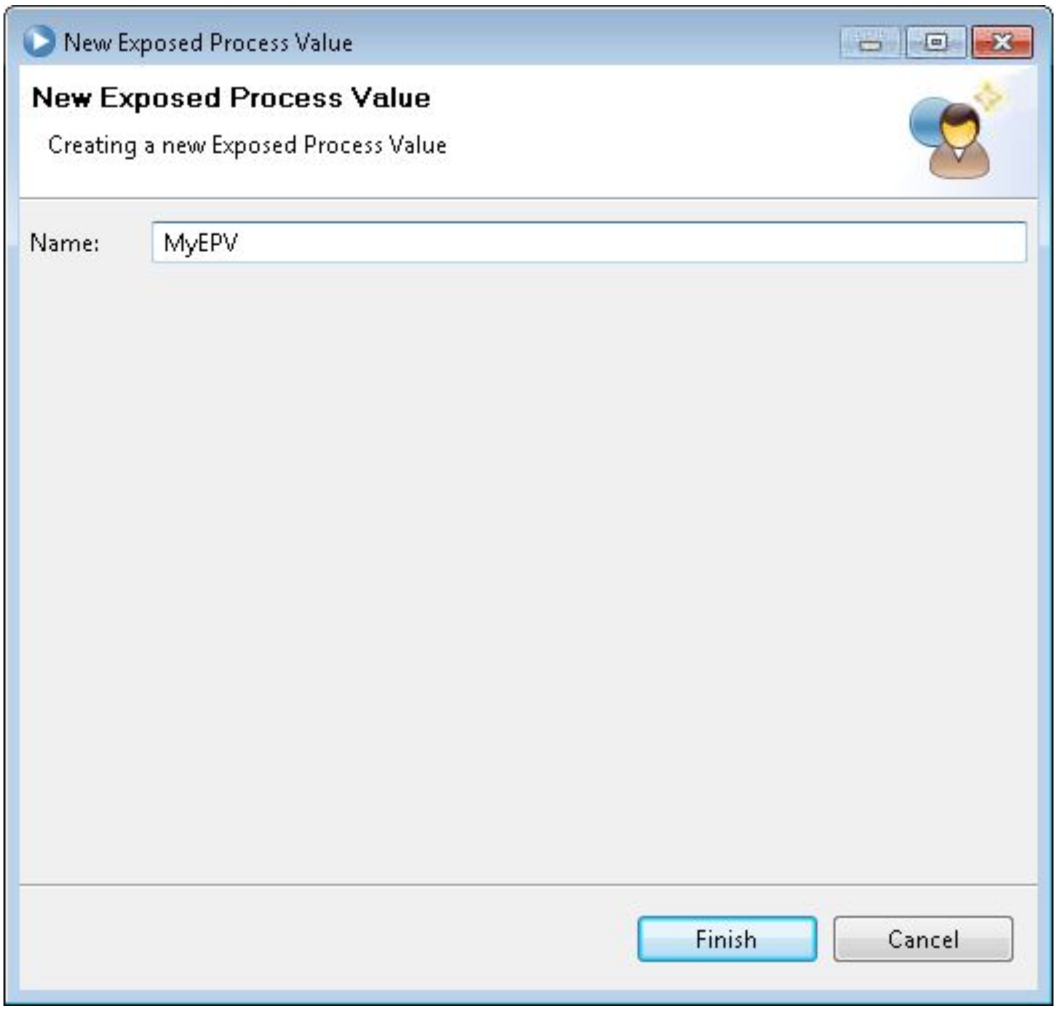
An EPV is defined within a Process Application or Toolkit from the Data catalog in the library.

EPV 是定义在 PA 或 Toolikit 里面的公用库的数据类别



When the wizard appears to create a new EPV, a name can be supplied for it.

使用向导创建新的 EPV 是,需要输入新创建 EPV 的名称



New Exposed Process Value

Creating a new Exposed Process Value

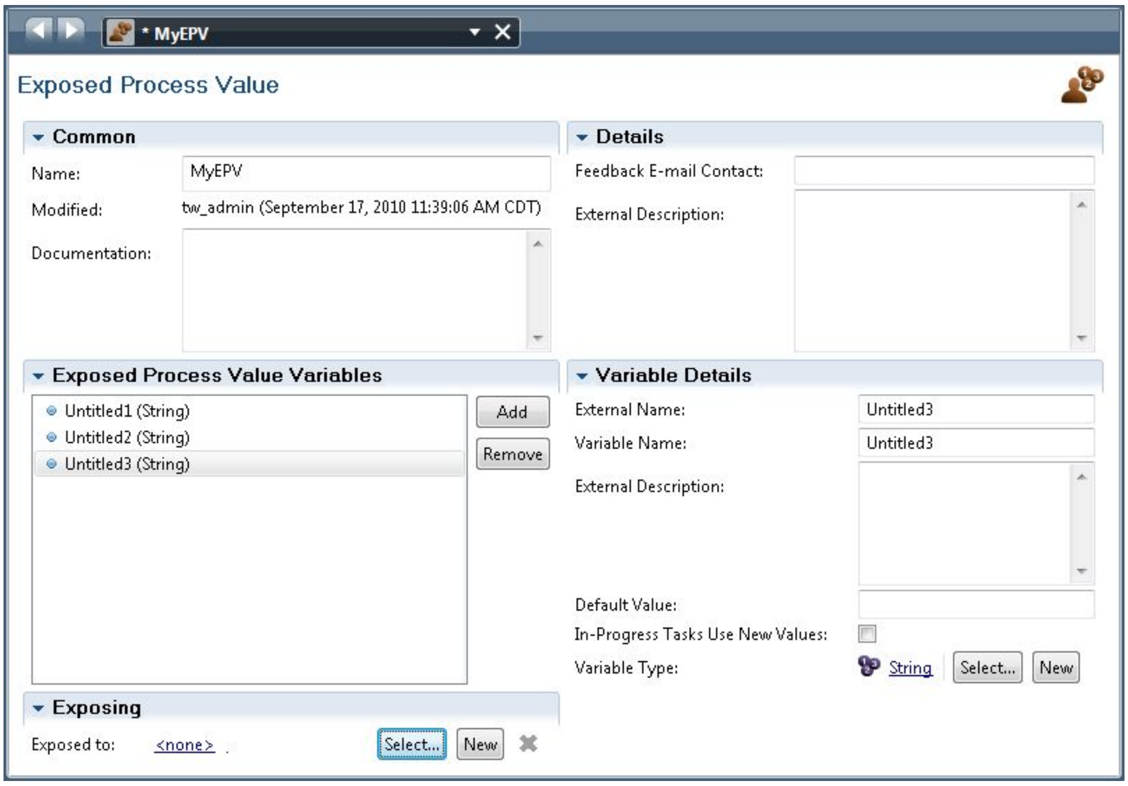
Name: MyEPV

Once created, the editor provides a rich set of settings for it. At its basic level, we define a set of one or more variables and give each variable a name, data type and default value.

创建成功后，在编辑器里添加它的各种设置。定义一个或多个变量，给每个变量设置其名字，数据类型，默认值。

Updates and access to the EPV is performed through the Process Admin Console. Only those users to whom the EPV is exposed can modify its value. The Exposed to section names a group of users who are allowed to manage the EPV.

通过流程管理控制台更新和访问 EPV。EPV 的值暴露给一些用户，这些用户可以修改 EPV 的值。允许暴露的用户组管理 EPV.



Name: MyEPV
Modified: tw_admin (September 17, 2010 11:39:06 AM CDT)
Documentation:
Feedback E-mail Contact:
External Description:

Untitled1 (String)
Untitled2 (String)
Untitled3 (String)

External Name: Untitled3
Variable Name: Untitled3
External Description:
Default Value:
In-Progress Tasks Use New Values:
Variable Type: String

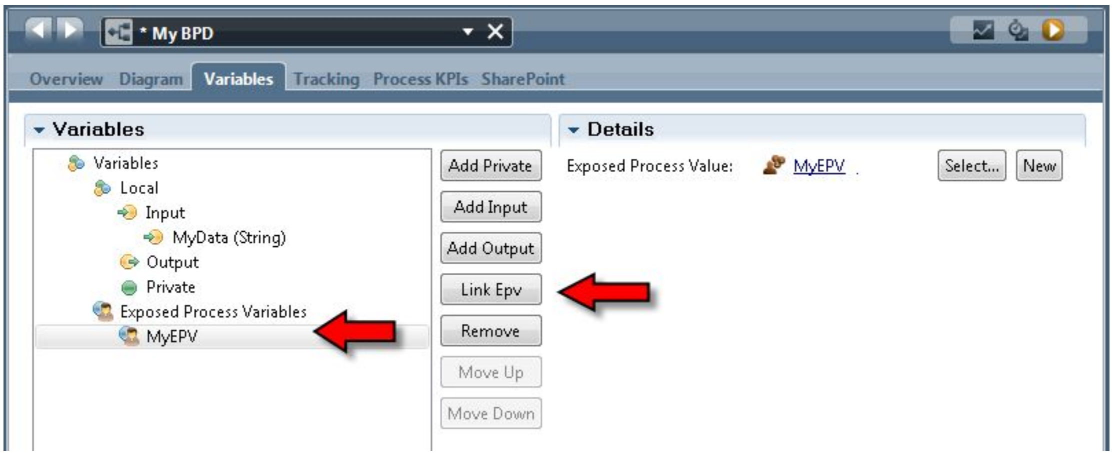
Exposed to: <none>

The Feedback E-mail contact provides the email address of the "owner" of this EPV. This allows users to contact the owner for questions and requests.

反馈邮件联系章节提供了 EPV 持有者的的邮件地址.这个允许用户联系持有者来解答相应的问题和需求。

In a BPD or service definition, within the variables section, an EPV can be linked or associated with the BPD or service:

在一个定义 BPD 或 service 中，在内部变量部分，可以使一个 EPV 和 BPD 连接或使一个 EPV 和 service 关联。BPD 或 service:



Variables
Local
Input
MyData (String)
Output
Private
Exposed Process Variables
MyEPV

Exposed Process Value: MyEPV

Once the link has been made, the values of the EPV can be used within the solution. The JavaScript expression:

tw.epv.[EPV Name].[EPV Variable Name]

can be used to refer to those values.

See also:

• Admin Tools > Manage EPVs

一旦关联 EPV, EPV 的值就可以被解决方案使用。Javascript 表达式如下：

tw.epv.[EPV Name].[EPV Variable Name]

以上可以用来引用那些 EPV 的值。

另见:

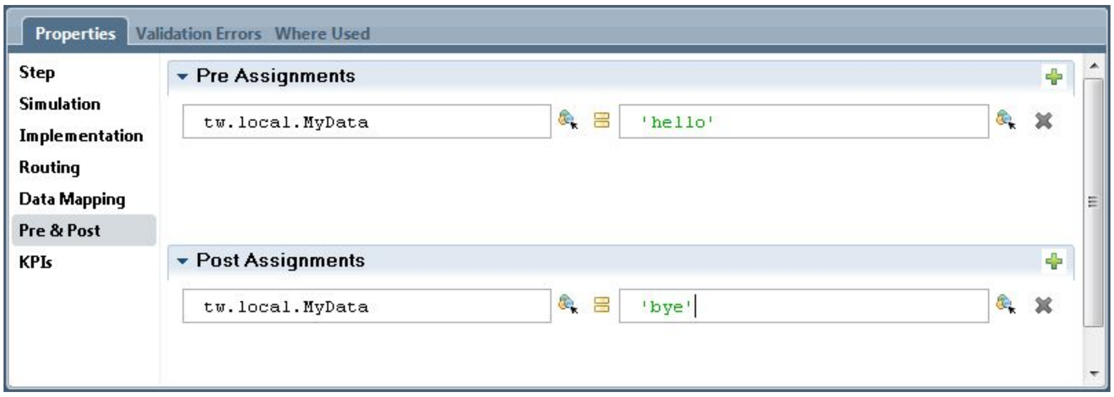
• 管理工具>管理 EPVs

Pre and Post Assignments

**前置和后置的赋值**

Associated with an activity element is the concept of pre and post assignments. These sections of the activity definition allow for updates of variables prior to executing the core of the activity and also to allow for updates immediately following the core of the activity.

与一个活动元素相关联的概念是前置和后置赋值。这部分的活动定义会允许在执行核心活动之前进行变量的更新和也允许在核心活动之后的立即更新操作。



tw.local.MyData = 'hello'

tw.local.MyData = 'bye'

The number of pre and post assignments (if any) on an individual activity is configurable. There have been a number of discussions on the value and appropriateness of pre and post assignments. The questions arise when thinking about readability and maintainability of a solution that uses this technique. The markers added to an activity to indicate that it has pre or post assignments are small and easily missed. There is a school of thought which says that this capability should not be used often and instead, explicit steps be added to the process diagram to achieve the same effect.

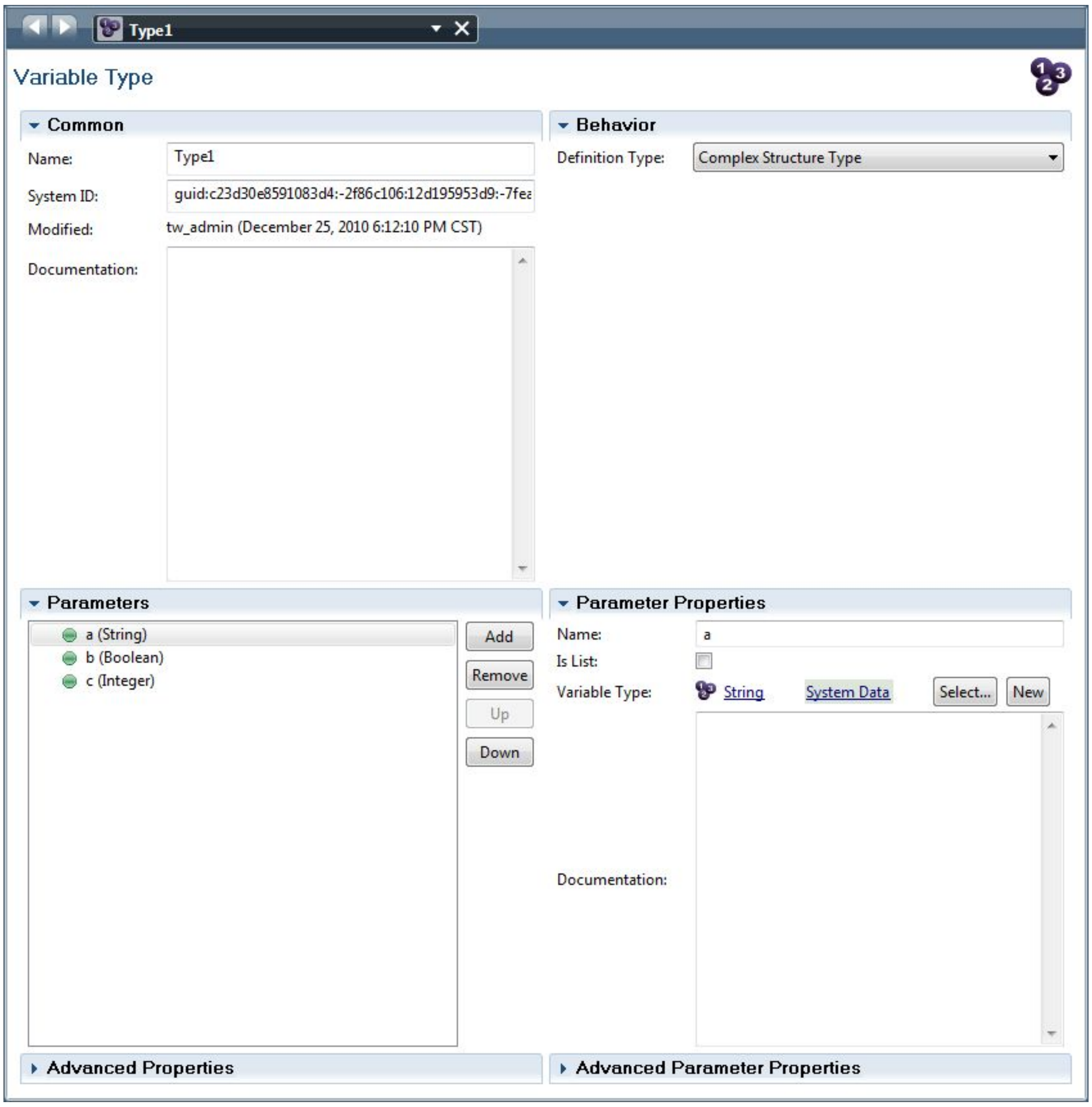
前置和后置赋值的个数(如果有的话)对单个活动是可配置的。已经有很多关于前后赋值的价值和合理性的讨论。在思考使用这种技术方案的可读性和可维护性时会出现问题，前置或后置赋值的标记添加到一个活动上是很小的且很容易被忽略。有一种思想学派认为,这种方式不应被频繁使用, 相反地, 显式添加步骤到流程图能达到同样的效果。

Variables and XML

## 变量和 XML

Each complex variable has an XML representation. To examine this, let us initially look at a complex data type called Type1 that looks as follows:

每个复杂的变量都可以有一个 XML 表现形式。为了验证这个问题,让我们开始看
一个被称作 *Type1* 的复杂数据类型, 如下:



Name: Type1
System ID: guid:c23d30e8591083d4:-2f86c106:12d195953d9:-7fea
Modified: tw_admin (December 25, 2010 6:12:10 PM CST)
Documentation:
Definition Type: Complex Structure Type

a (String)
b (Boolean)
c (Integer)

Name: a
Is List: ☐
Variable Type: String System Data
Documentation:

This structure contains three fields called ″a″, ″b″ and ″c″. If we create an instance of this variable and execute tw.system.serializer.toXml method, we get the following:

```
<variable type="Type1">
<a type="String"><![CDATA[value for a]]></a>
<b type="Boolean"><![CDATA[false]]></b>
<c type="Integer"><![CDATA[123]]></c>
</variable>
```

结构中包含三个名为“a”“b”和“c”的字段，如果我们创建一个这个变量的实例并执行 tw.system.serializer.toXml 方法，我们可以得到：

```
<variable type="Type1">
    <a type="String"><![CDATA[value for a]]></a>
    <b type="Boolean"><![CDATA[false]]></b>
    <c type="Integer"><![CDATA[123]]></c>
</variable>
```

Breaking this apart, we see an element called <variable> as the root with type attribute that names the data type of the variable as a whole. The nested elements are named after the fields within the structure again using the type attribute to define the data type of the fields.
打开这部分来看，我们看到一个名为<variable>的带有 type 属性的根元素来命名整个数据类型变量，嵌套的元素所命名的字段结构内又再次使用 type 属性来定义字段的数据类型。

Given an XML representation of the data, we can convert back to a variable instance with the method called:
tw.system.serializer.fromXml()
鉴于数据的 XML 表示法,我们可以调用一个方法来转换回一个变量的实例:
tw.system.serializer.fromXml()

This takes either a String or XMLElement object as a parameter and returns a new Object instance. For variables defined as lists, the structure is a little different.

```
<variable type="Item[]">
<item type="Item">
<name type="String"><![CDATA[Item1]]></name>
<quantity type="Integer"><![CDATA[10]]></quantity>
</item>
<item type="Item">
<name type="String"><![CDATA[Item2]]></name>
<quantity type="Integer"><![CDATA[20]]></quantity>
</item>
</variable>
```

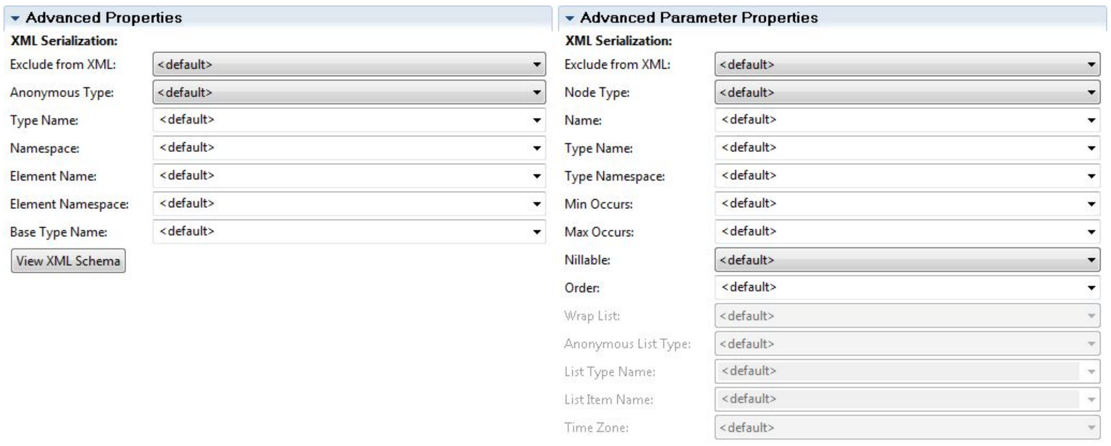
这需要一个 String 或 XMLElement 对象作为参数,并返回一个新对象实例。
对于变量列表，结构又有些不一样：

```
<variable type="Item[]">
  <item type="Item">
```

```
    <name type="String"><![CDATA[Item1]]></name>
    <quantity type="Integer"><![CDATA[10]]></quantity>
  </item>
  <item type="Item">
    <name type="String"><![CDATA[Item2]]></name>
    <quantity type="Integer"><![CDATA[20]]></quantity>
  </item>
</variable>
```

In the Data Type definition, there are properties for the XML serialization:

在数据类型定义里，有些属性专用于 XML 的序列化：



Advanced Properties

XML Serialization:

| | |
|---|---|
| Exclude from XML: | <default> |
| Anonymous Type: | <default> |
| Type Name: | <default> |
| Namespace: | <default> |
| Element Name: | <default> |
| Element Namespace: | <default> |
| Base Type Name: | <default> |

View XML Schema

Advanced Parameter Properties

XML Serialization:

| | |
|---|---|
| Exclude from XML: | <default> |
| Node Type: | <default> |
| Name: | <default> |
| Type Name: | <default> |
| Type Namespace: | <default> |
| Min Occurs: | <default> |
| Max Occurs: | <default> |
| Nillable: | <default> |
| Order: | <default> |
| Wrap List: | <default> |
| Anonymous List Type: | <default> |
| List Type Name: | <default> |
| List Item Name: | <default> |
| Time Zone: | <default> |

Notice the View XML Schema button. When clicked, an XML Schema representing this data type is shown. Notice also that the name space for this schema looks *odd*. It is created from the hostname and port number of the machine running the authoring environment. This can cause problems as this schema value will change depending on where the schema is displayed. If your IBPM solution is going to utilize XSDs it is strongly recommended to explicitly name a Namespace value. This can be any constant of the form:

http://<value here>

注意 View XML Schema 按钮。当点击时,XML Schema 表示这个数据类型会被显示出来。还需要注意的是,这个模式的名称空间看起来很奇怪。它是依据机器运行的编辑环境的主机名和端口号而来，这可能会导致一个问题，就是这个 schema 的值将会依据在那里显示这个 schema 而改变。如果你的 IBPM 解决方案将利用 XSDs, 强烈建议显示地给一个命名空间的值。它可以是任何形式的常数:

http://<value here>

Using Variables

Since variables in IBPM are available within JavaScript then any JavaScript functions that execute on such variables can be used.

If a variable is used to represent a business object, remember that the variable is a reference to that data and **not** its value. For example, if we have code that looks as follows:

```
var x = new tw.object.MyDataType();
var y = x;
```

## 变量的使用

既然 IBPM 的变量在 JavaScript 中也是可用的，任何 JavaScript 函数的执行也可以使用这样的变量。如果一个变量是用来表示一个业务对象，记住，变量只是一个数据的引用,而不是它的值。例如，如果我们有如下代码:

```
var x = new tw.object.MyDataType();
var y = x;
```

Then the variable y is actually a reference to the same object as the variable called x and importantly not a copy. Changing a field in x will cause the exact same field in y to be also changed. If the desire is to explicitly make a copy of a variable, consider using the IBPM provided XML serializer and de-serializer. For example:

```
var x = new tw.object.MyDataType();
var temp = tw.system.serializer.toXml(x);
var y = tw.system.fromXml(temp);
```

那么变量 y 实际上是对相同的对象变量 x 的引用，重要的是它并不是一个副本。改变 x 的一个字段将导致 y 里完全相同的字段也改变了。如果想显式地复制一个变量,考虑使用 IBPM 提供的 XML 序列化器和反序列化器，例如:

```
var x = new tw.object.MyDataType();
var temp = tw.system.serializer.toXml(x);
var y = tw.system.fromXml(temp);
```

On a broader scale, there is a question on what happens when a business object is passed from one entity to another (for example, a process to a sub-process)? In computer science there is the concept of pass-by-value vs pass-by-reference. With pass-by-value, a copy of the variable is passed and hence chances to that copy do not affect the value of the original variable. With pass-by-reference, a reference (pointer) to the original variable is passed. This means a change to the content of one variable immediately affects the value of the other variable. In IBPM both pass-by-value and passby-reference are used in different circumstances.

更广泛来说，有一个问题， 所发生的业务对象是通过从一个实体到另一个(例如,流程的子流程)， 在计算机科学中有这个概念， 按值传递和按引用传递。按值传递，传递的复制的变量值，因此，复制的值的改变不影响原始变量的值。按引用传递，原始变量的引用(指针)传递，这意味着更改的内容立即会影响到其他变量的值。在 IBPM 中，按值传递和按引用传递被用在不同的环境里：

| 场景 | 传递机制 |
|---|---|
| 流程调用子流程 | 按引用传递 |
| 流程调用服务 | 按值传递 |
| 服务调用内嵌服务 | 按引用传递 |

When assigning a value to a Business Object, one can use the JavaScript style of definition. For example:

```
tw.local.myData = {
a: "My A Value",
b: 123,
c: 3.141,
d: {
x: "X Value",
y: "Y Value"
}
};
```

would work for an appropriate definition of the data type of myData. The use of this technique can dramatically simplify data assignment.

当给一个业务对象赋值，可以使用 Javascript 风格的定义。比如：

```
tw.local.myData = {
  a: "My A Value",
  b: 123,
  c: 3.141,
  d: {
    x: "X Value",
    y: "Y Value"
  }
};
```

上述给予数据类型 myData 的定义是可以正常工作的，这种技术的使用可以极大简化数据的赋值操作。

Determining the type of a variable

On occasion, you may find that you have a local JavaScript variable and not know what type it is. You can dynamically determine its type. See the following example:

```
var myDate = new tw.object.Date();
log.info("The data type of the object is: " + Object.prototype.toString.call(myDate).slice(8, -1));
```

**确定一个变量的类型**

有时，你会发现你有一个本地 JavaScript 变量,不知道它是什么类型。 您可以动态地确定它的类型， 看下面的例子:

```
var myDate = new tw.object.Date();
log.info("The data type of the object is: " + Object.prototype.toString.call(myDate).slice(8, -1));
```

The case of the mysteriously changing variables

Consider the following story:

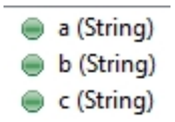
First, imagine a data type called "MyBO" that has three fields:

Now imagine a process called "LinkedProcess1" which looks as follows:

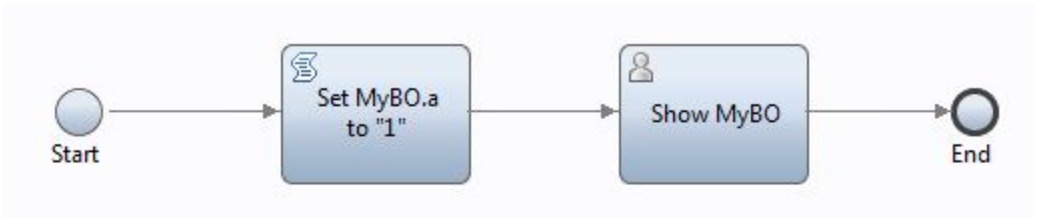
**神秘的情况下改变变量值**

考虑下面的状况:

首先,想象一个数据类型称为“MyBO”， 它有三个字段:

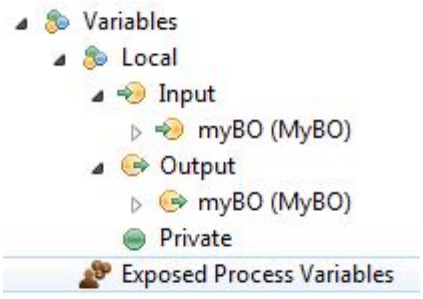

a (String)
b (String)
c (String)

现在想象一下有一个被称为“LinkedProcess1”的流程，如下:



The parameters to this process are:

这个流程里有如下变量：

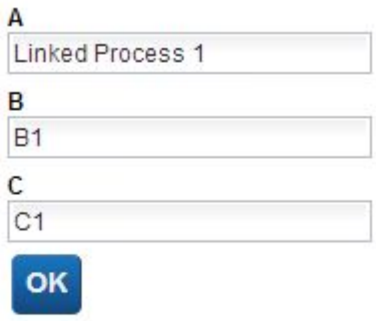


What this process does is take an instance of "MyBO" as input, set the value of property "a" to say that it is "Linked Process 1" that set it and finally create a task that shows the content of the MyBO data type.

流程需要做的是用 MyBO 的实例作输入参数，把属性 a 值设为“Linked Process 1”，最后创建一个任务显示 MyBO 数据类型的内容。

If we ran it by itself, an example of what the task would show is:

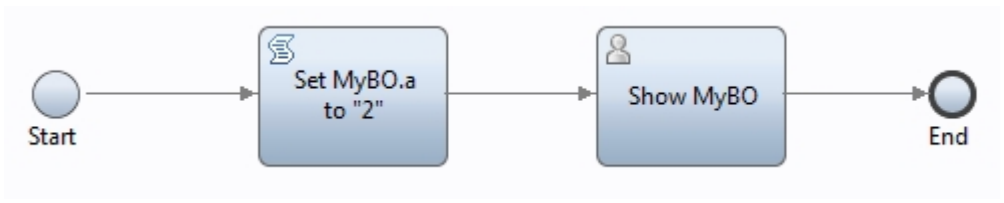
如果运行这个流程，任务显示如下：



A
Linked Process 1
B
B1
C
C1

Now imagine a second process called "LinkedProcess2". It looks like:

现在设想第二个流程"LinkedProcess2"。如下：



It is identical to "LinkedProcess1" with the exception that it sets the "a" property to say that we are in "Linked Process 2".

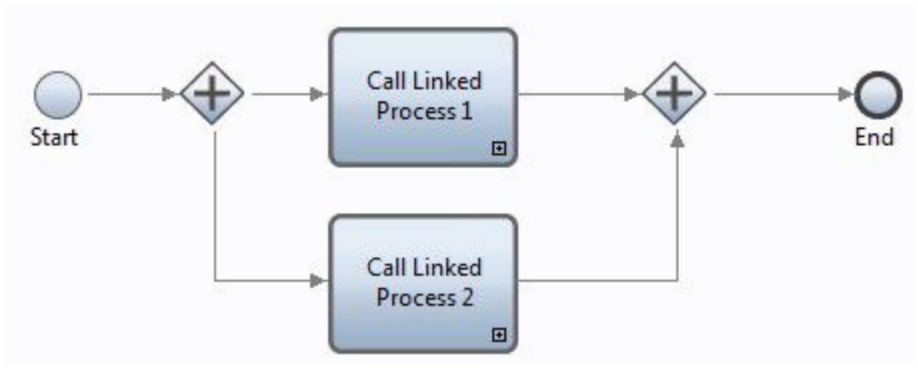
"Linked Process 2"流程，除了属性 a 的值为"Linked Process 2"，其他部分都一样的。

Now ... here comes the next part ...

Imagine one last process called "MyProcess" which looks like:

现在下一步

设想最后一个流程叫"MyProcess"，如下：

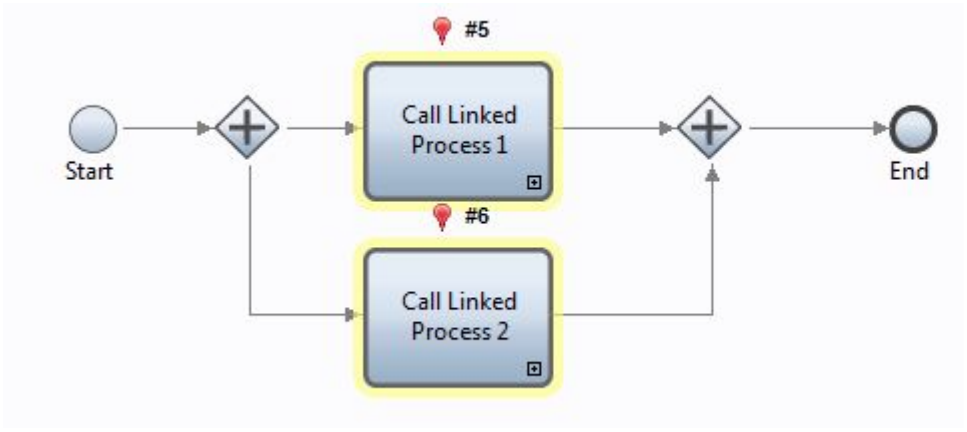


This process has a single "MyBO" variable which is passed as input to two parallel linked processes. These are mapped to "LinkProcess1" and "LinkProcess2".

这个流程有唯一一个"MyBO"类型的输入变量传向两个平行的流程，它被映射给"LinkProcess1" 和 "LinkProcess2" 流程.

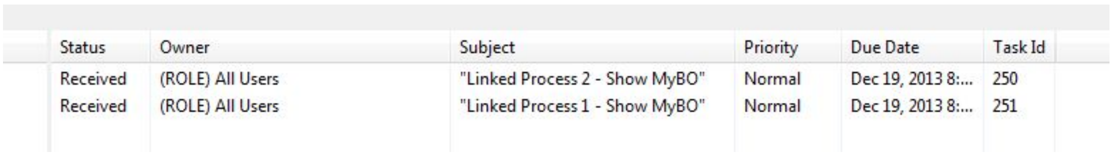
When I run this process, as expected, I see it block waiting on two tasks:

当运行这个过程，不出所料，两个任务将处于等待状态：



And I see two tasks available to be worked upon:

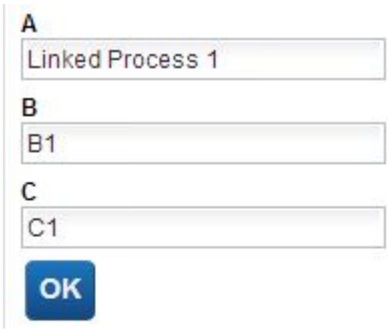
正在执行的两个任务如下：



| Status | Owner | Subject | Priority | Due Date | Task Id |
|---|---|---|---|---|---|
| Received | (ROLE) All Users | "Linked Process 2 - Show MyBO" | Normal | Dec 19, 2013 8:... | 250 |
| Received | (ROLE) All Users | "Linked Process 1 - Show MyBO" | Normal | Dec 19, 2013 8:... | 251 |

Now I open the task for "Linked Process 1" ... and here is what I see:

现在打开"Linked Process 1"任务 ，如下：



A

Linked Process 1

B

B1

C

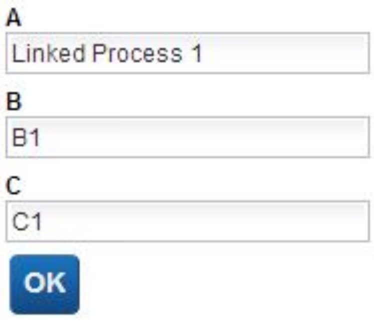
C1

So far, all as expected.

Now I open up the task for "Linked Process 2" ... and here is what I see:

到目前为止，一切如我们所想。

现在打开"Linked Process 1"任务 ，如下：



A

Linked Process 1

B

B1

C

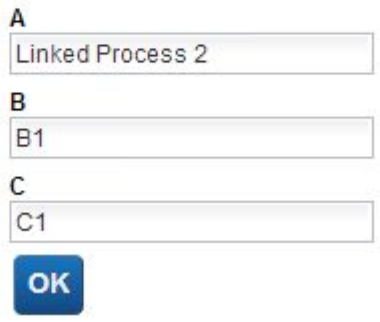
C1

Do you see it? :-)

你看到了吗？:-)

The "a" field has the **wrong** value!!! It should say:

这个“a”字段有了一个错误的值!!! 它本该是：



A

Linked Process 2

B

B1

C

C1

So .... what happened?

那么……到底发生了什么？

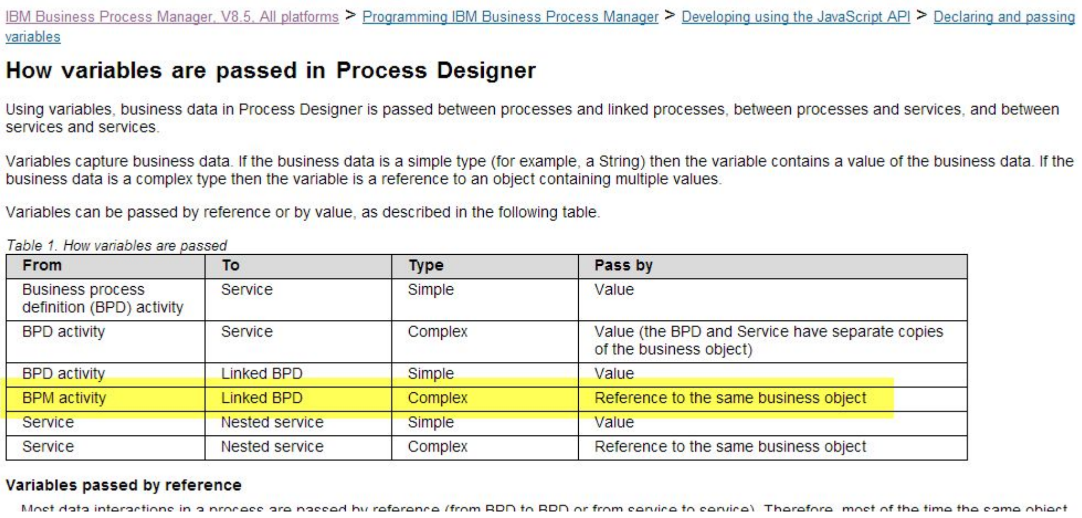
The answer can be found in the IBM BPM documentation found here:

我们可以从以下链接，来自IBM BPM文档寻求答案：

http://pic.dhe.ibm.com/infocenter/dmndhelp/v8r5m0/topic/com.ibm.wbpm.bpc.doc/modeling/topic/declaring_variables_A.html

Here is what caught my eye:

我们可以看到：


IBM Business Process Manager, V8.5, All platforms > Programming IBM Business Process Manager > Developing using the JavaScript API > Declaring and passing variables

**How variables are passed in Process Designer**

Using variables, business data in Process Designer is passed between processes and linked processes, between processes and services, and between services and services.

Variables capture business data. If the business data is a simple type (for example, a String) then the variable contains a value of the business data. If the business data is a complex type then the variable is a reference to an object containing multiple values.

Variables can be passed by reference or by value, as described in the following table.

*Table 1. How variables are passed*

| From | To | Type | Pass by |
|---|---|---|---|
| Business process definition (BPD) activity | Service | Simple | Value |
| BPD activity | Service | Complex | Value (the BPD and Service have separate copies of the business object) |
| BPD activity | Linked BPD | Simple | Value |
| BPM activity | Linked BPD | Complex | Reference to the same business object |
| Service | Nested service | Simple | Value |
| Service | Nested service | Complex | Reference to the same business object |

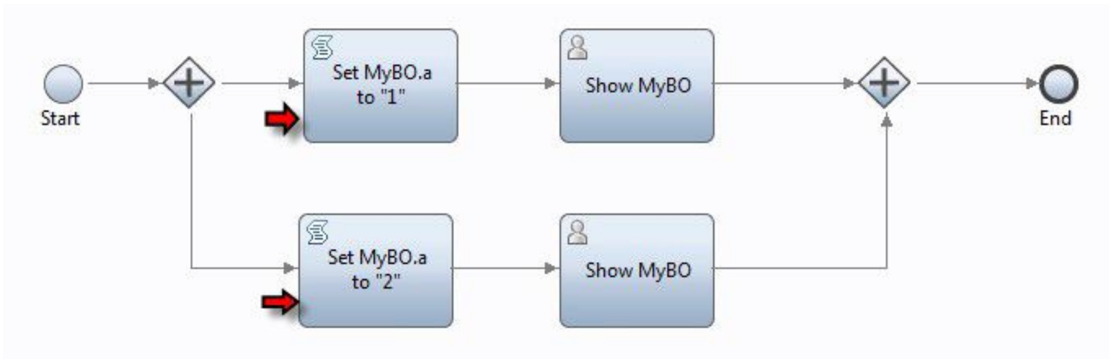
**Variables passed by reference**

When we call from one process to another as a link, the variable is passed by reference and not by value (a copy).

当我们调用一个流程到另一个流程，就像一个链接，其变量的传递是通过引用而非值（副本）。

What this means ... is that our BPM process called "MyProcess" (see above) ... is logically the same as:

意思就是……BPM流程调用“MyProcess”(参照上面)……在逻辑上如同:



And if we look closely at this diagram, we see that the two marked activities BOTH set the same business object instance field to two different values ... in parallel and since one of them will run before the other in reality, the last one executed will set the single value ... and it is that single value that is shown in the Coaches.

如果我们仔细看上图，可以看到，这两个标记活动都设置了相同的业务对象变量，和有不同的值……并行的，实际上由于其中一个比另外一个先运行，后运行的将会设置为同一个值，而且这就是显示在页面上的值。

With this understanding in mind, we can now see how to avoid the problem ... have two business objects ... one for "Linked Process 1" and one for "Linked Process 2" ... and don't share the same business object between both.

在此理解的基础之上，我们现在可以避免这个问题……有两个业务对象，一个是“Linked Process 1”，另一个是“Linked Process 2”……在他们之间不要使用相同的业务对象。