Portrait
Foundation

Scripting Reference

Edition 1.2

11 January 2013

Pitney Bowes
Software

Portrait Foundation
Scripting Reference

Acknowledgement of trademarks

About Portrait Software

Portrait Software is now part of Pitney Bowes Software Inc.

Portrait Software enables organizations to engage with each of their customers as individuals, resulting in improved customer profitability, increased retention, reduced risk, and outstanding customer experiences. This is achieved through a suite of innovative, insight-driven applications which empower organizations to create enduring one-to-one relationships with their customers.

Portrait Software was acquired in July 2010 by Pitney Bowes to build on the broad range of capabilities at Pitney Bowes Software for helping organizations acquire, serve and grow their customer relationships more effectively. The Portrait Customer Interaction Suite combines world leading customer analytics, powerful inbound and outbound campaign management, and best-in-class business process integration to deliver real-time customer interactions that communicate precisely the right message through the right channel, at the right time.

Our 300 + customers include industry-leading organizations in customer-intensive sectors. They include 3, AAA, Bank of Tokyo Mitsubishi, Dell, Fiserv Bank Solutions, Lloyds Banking Group, Merrill Lynch, Nationwide Building Society, RACQ, RAC WA, Telenor, Tesco Bank, T-Mobile, Tryg and US Bank.

Pitney Bowes Softweare Inc. is a division of Pitney Bowes Inc. (NYSE: PBI).

For more information please visit: http://www.pitneybowes.co.uk/software/

UK

Portrait Software
The Smith Centre
The Fairmile
Henley-on-Thames
Oxfordshire, RG9 6AB, UK

Email: support@portraitsoftware.com
Tel: +44 (0)1491 416778
Fax: +44 (0)1491 416601

America

Portrait Software
125 Summer Street
16th Floor
Boston, MA 02110
USA

Email: support@portraitsoftware.com
Tel: +1 617 457 5200
Fax: +1 617 457 5299

Norway

Portrait Software
Portrait Million Handshakes AS
Maridalsveien. 87
0461 Oslo
Norway

Email: support@portraitsoftware.com
Tel: +47 22 38 91 00
Fax: +47 23 40 94 99

About this document

Purpose of document

The purpose of this document is to provide configurers with a reference to methods and properties of Portrait Foundation data objects that are available to be used within the Configuration Suite script editor.

The document contains examples which can be copied and pasted for use on projects and many of the examples are from existing Foundation configuration. i.e. they are tried and tested.

Intended audience

Portrait Foundation configurers.

Related documents

None

Software release

Portrait Foundation 4.2 or later.

Contents

1 Data object properties

1.1 Count

Holds the number of configured properties in a data object. It is important to note that you should avoid configuring a property called **Count** in a data object. If you attempt to access the property in a Portrait script you will always get the number of properties in the data object rather than the value of the **Count** property that you have created. Access to the property in other contexts is likely to have the same effect.

Example

The following example merges one data object (**DOToMerge**) into another (**TargetDO**).

```
DOCount = DOToMerge.Count
DOIndex = 0
For DOIndex = 0 To DOCount - 1
   Value = EmptyValue
   Name = DOToMerge.Index(DOIndex, Value)
   Dirty = DOToMerge.IsPropertyDirty(Name)
   ' If the property is dirty on the merge data object or all the
   ' properties are to be merged.
   If Dirty <> False OR MergeOnlyModified = False Then
      If TargetDO.HasProperty(Name) <> False Then ' If property exists in target.
         SrcType = DOToMerge.GetPropertyType(Name)
         DestType = TargetDO.GetPropertyType(Name)
         If MatchDataTypes(SrcType, DestType) Then ' Check properties are same type
            ' Copy the value from merge data object to the target data object.
            Call TargetDO.SetValue(Name, DOToMerge.GetValue(Name))
            Call LogMessage("Merged property" + Name + ". Value is " +
                            CStr(DOToMerge.GetValue(Name)))
         else
            Call LogMessage("Failed to merge property" + Name +
                           ". Source data type is " + CStr(SrcType) +
                           ". Destination data type is " + CStr(DestType))
         End If
      else
         Call LogMessage("Failed to merge property" + Name +
                         ". No corresponding destination property")
      End If
   else
      Call LogMessage("Failed to merge property" + Name +
                      ". Merge only modified is " + CStr(MergeOnlyModified) +
                      ". Property dirty flag is " + CStr(Dirty))
   End If
Next
```

2 Data object Methods

2.1 AddPropertyDefinition(PropertyName, Type)

Adds a property to a data object without giving the property a value.

See also: `AddRefPropertyDefinition (PropertyName, Type)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the new property
Type	Integer	The portrait data type of the property. This can be one of: • Integer: **vbLong** • Float: **vbDouble** • DateTime: **vbDate** • String: **vbString** • Boolean: **vbBoolean** • Data object: **12** • Data object collection: **13** • Reference: **16**

Return

None

Example

The following example creates a **ViewAttribute** data object that is used to display the value of a property in another data object (not shown). The code fragment sets up standard properties on the **ViewAttribute** data object and then adds a final property (**Value**) which may or may not be a reference property. (See also example in `GetPropertyDisplayName (PropertyName)`.)

```
Set ViewAttribute = CreateDataObject("ViewAttributes", "ViewAttribute")
ViewAttribute.DisplayTypeIndicator = "PlainTextViewField"
ViewAttribute.ItemOrder = ItemOrder
ViewAttribute.Name = Prefix + DisplayName
ViewAttribute.SystemName = SystemName
ViewAttribute.Type = TypeToViewTypeString(SrcType)

If SrcType = ReferenceType Then
   ' The property has to be added dynamically as the type is changeable.
   RDGroup = DOToSerialise.GetRefPropertyDefinition(SystemName)
   Call ViewAttribute.AddRefPropertyDefinition("Value", RDGroup)
Else
   ' The property has to be added dynamically as the type is changeable.
   Call ViewAttribute.AddPropertyDefinition("Value", SrcType)
End If
```

2.2 AddRefPropertyDefinition (PropertyName, Type)

Adds a reference property to a data object without giving the property a value. A reference property is a property that refers to a Reference Data Group. The property can only contain a value that is the system name of one of the items in the group.

See also: `AddPropertyDefinition(PropertyName, Type)`
`GetRefPropertyDefinition (PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the new property
RDG name	String	The system name of the reference data group that contains the valid items for this property.

Return

None

Example

See the fuller example in `AddPropertyDefinition(PropertyName, Type)` and example in `GetPropertyDisplayName (PropertyName)`.

```
…
If SrcType = ReferenceType Then
   ' The property has to be added dynamically as the type is
changeable.
   RDGroup = DOToSerialise.GetRefPropertyDefinition(SystemName)
   Call ViewAttribute.AddRefPropertyDefinition("Value", RDGroup)
…
```

2.3 CreateCopy()

Creates a copy of a data object. This method is useful if you need to have a copy of a data object to, for example, keep the original state of an object.

It is worth noting that you cannot achieve the same effect by mapping a data object out of a data manipulation node twice. This merely creates two context references to the same data object. Changes to one context entry will be seen in the other.

Inputs

None

Return

Data object: A copy of the data object.

Example

Also see example `CreateCopy()` in Data object collection methods.

```
Set CopiedDataObject = DataObjectToCopy.CreateCopy()
```

2.4 GetBooleanValue(PropertyName)

Returns the value of a boolean property. Use the `GetValue(PropertyName)` method instead. The behavior will be the same.

See also: `GetValue(PropertyName).`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Boolean: The value of the named property

Example

See `GetValue(PropertyName).`

2.5 GetCategory()

Returns the name of the category of the data object. This is the name that would be used in a call to `CreateDataObject().`

See also: `GetObjectType()`
`CreateDataObject(Category, Type)`

Inputs

None

Return

String: The name of the data object category.

Example

The following example creates a data object with a given category and type. Then it compares this with the category and type of another data object to determine whether the two objects should be merged.

```
If IsEmpty(ClassifiedDataObject) <> False or IsNull(ClassifiedDataObject) <> False then
    'Create the new object
    Set ClassifiedDataObject = CreateDataObject(DOCategory, DOType)
elseIf ClassifiedDataObject.GetCategory() <> DOCategory or
                                ClassifiedDataObject.GetObjectType() <> DOType then
    'Create a new object and merge the existing one into it
    Set NewDataObject = CreateDataObject(DOCategory, DOType)
    Set ClassifiedDataObject = MergeDataObjects(ClassifiedDataObject, NewDataObject)
end if
```

2.6 GetDataObjectValue(PropertyName)

Returns the data object (value) in the named data object property. Use this method instead of the **GetValue** method when accessing a data object. Bear in mind that the data object returned is not a copy. Any changes that you make in the data object returned are reflected in the originating property.

See also: GetDOCollectionValue(PropertyName)

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Data object: The data object in the named property.

Example

This example returns a value from a named data object property:

```
If DataObject.HasProperty(PropertyName) then
   SrcType = DataObject.GetPropertyType(PropertyName)
   Select case SrcType
      Case DataObjectType
         Set DataObjectProperty = DataObject.GetDataObjectValue(PropertyName)
      Case DOCollectionType
         Set DOCProperty = DataObject.GetDOCollectionValue(PropertyName)
      Case IntegerType
         IntegerProperty = DataObject.GetValue(PropertyName)
      Case FloatType
         FloatProperty = DataObject.GetValue(PropertyName)
      Case DateTimeType
         DateTimeProperty = DataObject.GetValue(PropertyName)
      Case StringType
         StringProperty = DataObject.GetValue(PropertyName)
      Case BooleanType
         BooleanProperty = DataObject.GetValue(PropertyName)
      Case ReferenceType
         StringProperty = DataObject.GetValue(PropertyName)
   end select
else
   Call Err.Raise(6, , "Could not find property with name '" + PropertyName + "'.")
end if
```

2.7 GetDateTimeValue(PropertyName)

Returns the value of a date/time property. Use the `GetValue(PropertyName)` method instead. The behavior will be the same.

See also: `GetValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

DateTime: The value of the named property

Example

See `GetValue(PropertyName).`

2.8 GetDOCollectionValue(PropertyName)

Returns the data object collection (value) in the named collection property. Use this method instead of the **GetValue** method when accessing a data object collection. Bear in mind that the collection returned is not a copy. Any changes that you make in the collection returned are reflected in the originating property.

See also: `GetDataObjectValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Data object collection: The collection in the named property

Example

See fuller example in `GetDataObjectValue(PropertyName)`:

```
Set DOCProperty = DataObject.GetDOCollectionValue(PropertyName)
```

2.9 GetDoubleValue(PropertyName)

Returns the value of a double (floating-point number) property. Use the `GetValue(PropertyName)` method instead. The behavior will be the same.

See also: `GetValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Double: The value of the named property

Example

See `GetValue(PropertyName).`

2.10 GetLongValue(PropertyName)

Returns the value of a long (integer) property. Use the `GetValue(PropertyName)` method instead. The behavior will be the same.

See also: `GetValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Long: The value of the named property

Example

See `GetValue(PropertyName)`.

2.11 GetObjectType()

Returns the name of the type of the data object. This is the name that would be used in a call to `CreateDataObject()`.

See also: `GetCategory()`
`CreateDataObject(Category, Type)`

Inputs

None

Return

String: The name of the data object type.

Example

The following example creates a data object with a given category and type. Then it compares this with the category and type of another data object to determine whether the two objects should be merged.

```
If IsEmpty(ClassifiedDataObject) <> False or IsNull(ClassifiedDataObject) <> False then
    'Create the new object
    Set ClassifiedDataObject = CreateDataObject(DOCategory, DOType)
elseIf ClassifiedDataObject.GetCategory() <> DOCategory or ~
                                  ClassifiedDataObject.GetObjectType() <> DOType then
    'Create a new object and merge the existing one into it
    Set NewDataObject = CreateDataObject(DOCategory, DOType)
    Set ClassifiedDataObject = MergeDataObjects(ClassifiedDataObject, NewDataObject)
end if
```

2.12 GetPropertyDisplayName (PropertyName)

Returns the display name of the given property of the data object. The display name of a property is the name seen in the configuration suite and the name used as the label of any property of a CBE (that is, an attribute) that is displayed in a generated view.

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

String: The display name of the data object property.

Example

This example converts a data object into a collection of view attributes. The **GetPropertyDisplayName** method creates the label value for the view field.

```
PropertyCount = DOToSerialise.Count
DOIndex = 0
ParseDO = ""
For DOIndex = 0 To PropertyCount - 1
   Value = EmptyValue
   RDGroup = EmptyValue
   SystemName = DOToSerialise.Index(DOIndex, Value)
   DisplayName = DOToSerialise.GetPropertyDisplayName(SystemName)
   SrcType = DOToSerialise.GetPropertyType(SystemName)
   Select Case SrcType
      Case DataObjectType
         If IsObject(Value) Then
            ItemOrder = ParseDO(Value, ViewAttributes, Prefix + DisplayName + " - ",
                                ItemOrder)
         End If
      Case DOCollectionType
         If IsObject(Value) Then
            ItemOrder = ParseDOC(Value, ViewAttributes, Prefix + DisplayName + " - ",
                                 ItemOrder)
         End If
      Case Else ' This is a simple property of the data object.
         If (UsePropertyPrefixLen < 1) Or (StrComp(LCase(UsePropertyPrefix),
            LCase(Left(SystemName, UsePropertyPrefixLen))) = 0) Then
            LogMessage "About to add the view attribute DO '" & SystemName &
                    "' with the value = '" & Value & "'."
               Set ViewAttribute = CreateDataObject("ViewAttributes", "ViewAttribute")
            ViewAttribute.DisplayTypeIndicator = "PlainTextViewField"
            ViewAttribute.ItemOrder = ItemOrder
            ViewAttribute.Name = Prefix + DisplayName
            ViewAttribute.SystemName = SystemName
            ViewAttribute.Type = TypeToViewTypeString(SrcType)
            If SrcType = ReferenceType Then
               ' The property has to be added dynamically as the type is changeable.
               RDGroup = DOToSerialise.GetRefPropertyDefinition(SystemName)
               Call ViewAttribute.AddRefPropertyDefinition("Value", RDGroup)
            Else
               ' The property has to be added dynamically as the type is changeable.
```

```
            Call ViewAttribute.AddPropertyDefinition("Value", SrcType)
         End If
         If (IsEmpty(Value) = False) And (IsNull(Value) = False) Then
            ViewAttribute.Value = Value
         Else
            ViewAttribute.Value = EmptyValue
         End If
         Call ViewAttributes.Add(ViewAttribute)
         ItemOrder = ItemOrder + 1
      End If
   End Select
Next
```

2.13 GetPropertyType(PropertyName)

This method returns the data type of the named property.

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Integer: The portrait data type of the property expressed as an integer. This will be one of:

- Integer: **vbLong**
- Float: **vbDouble**
- DateTime: **vbDate**
- String: **vbString**
- Boolean: **vbBoolean**
- Data object: **12**
- Data object collection: **13**
- Reference: **16**

Example

This example shows the property types of properties in two data objects being compared. It is important to note that a string property (**vbString**) and a reference property (16) are compatible (that is, the value of one can be assigned to the other). (See the fuller example in `Count`).

```
...
   ' Check the two properties are of the same type.
   SrcType = DOToMerge.GetPropertyType(Name)
   DestType = TargetDO.GetPropertyType(Name)
   If MatchDataTypes(SrcType, DestType) Then
      ...
```

2.14 GetReferenceDisplayValue (PropertyName, Language)

Returns the display name for the value of a reference property in the data object. This method performs the same function as the scripting helper function of the same name. In the case of the method, the reference data group is determined from the property definition and the reference data item is the value of the named property.

See also: `GetReferenceDisplayValue(RDGSystemName, RDISystemName, Language)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the reference property. If a none reference property name is provided, the method will fail.
Language	String	The language in which the display name should be returned. The value supplied must be the system name of one of the values in the Implementation languages reference data group. “Default” is the value normally supplied unless your implementation requires multi-language support.

Return

String: The display name for the reference data item associated with the value in the named data object property.

Example

```
DisplayText = Task.GetReferenceDisplayValue("PartyType", "Default")
```

2.15 GetRefPropertyDefinition (PropertyName)

Returns the system name for the reference data group of a reference property in the data object.

See also: `AddRefPropertyDefinition (PropertyName, Type)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the reference property. If a none reference property name is provided, the method will fail.

Return

String: The system name of the reference data group associated with the named data object property.

Example

```
RDGName = Task.GetReferenceDisplayValue("PartyType")
```

2.16 GetStringValue(PropertyName)

Returns the value of a string property. Use the `GetValue(PropertyName)` method instead. The behavior will be the same.

See also: `GetValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

String: The value of the named property

Example

See `GetValue(PropertyName)`.

2.17 GetValue(PropertyName)

Returns the value of a simple property. The data type of the returned value is determined by the data type of the requested property. If you want to return the value of a data object or data object collection, use the specific **Get*Value** method.

See also: `GetBooleanValue(PropertyName)`
`GetDateTimeValue(PropertyName)`
`GetDoubleValue(PropertyName)`
`GetLongValue(PropertyName)`
`GetStringValue(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.

Return

Variant: The value of the named property

Example

See fuller example in `GetDataObjectValue(PropertyName)`.

```
PropertyValue = DataObject.GetValue(PropertyName)
```

2.18 HasProperty(PropertyName)

This method determines whether a data object has a property with a given name. This method should be used prior to using any other methods that use the property name.

Inputs

Name	Data type	Description
PropertyName	String	The name of the property to look for.

Return

Boolean: True if a property named **PropertyName** exists in the object. False if not.

Example

This example tests for the existence of a property before performing a merge operation (see fuller example in `Count`).

```
...
   ' If the property exists in the target data object.
   If TargetDO.HasProperty(Name) <> False Then
      Dim SrcType
      Dim DestType
      ...
```

2.19 Index(Index, Value)

This method can be used to extract the name and value of a property based on its index (position) in a data object. The name of the property is the return value of the method, whereas the value of the property is returned into a variable that is supplied as the second input parameter.

Inputs

Name	Data type	Description
Index	Integer	The index of the property. The index of the first property is 0 (zero), the index of the second property is 1 and so on.
Value	String	This is the return variable into which the value of the property will be placed. You must declare the variable and re-initialize it before every use. See the example below. Because this parameter is in fact a pointer, you cannot enter a string constant.

Return

String: The name of the property identified by the index.

Example

The following example is a fragment of the fuller example in Count above. Note how to variables (**Value** and **EmptyValue**) are declared. **EmptyValue** is never assigned a value of its own. It remains empty and is only used to effectively re-initialize Value in every iteration of the loop. **Value** must be reset in this way in order for the Index method to work.

```
Dim DOCount
Dim DOIndex
Dim Name
Dim Value
Dim EmptyValue
Dim Dirty

DOCount = DOToMerge.Count
DOIndex = 0

For DOIndex = 0 To DOCount - 1
       Value = EmptyValue
       Name = DOToMerge.Index(DOIndex, Value)
...
```

2.20 Initialise(Category, Type)

This method sets the Category and Type of a data object. Although the data object is then fully classified, the method does NOT create the data object's properties. Instead of this method, use the **CreateDataObject** scripting helper function.

See also: `GetDataObjectValue(PropertyName)`
`SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
Category	String	The system name of the data object category.
Type	String	The system name of the data object type.

Return

None

Example

The following example:

```
Set DynamicDO = CreateObject("AIT.AMC.DataObjects.DataObject")
Call DynamicDO.Initialise("General","DynamicDO")
```

is equivalent to:

```
Set DynamicDO = CreateDataObject("General","DynamicDO")
```

excepting that, in the second example, the data object is created complete with the full set of properties.

2.21 IsDirty()

Indicates whether the value of any property on the data object has changed since the data object was cleaned or created. Whenever a property on the data object has its value set, the dirty flag is set on the property and on the data object. This flag can be used to determine whether the data object should be considered in subsequent operations (for example, the flag is used by some DATs for CBE data objects when determining which properties should be written to the database).

See also: `IsPropertyDirty(PropertyName)`
`SetClean()`

Inputs

None

Return

Boolean: True if any property on the object is dirty, False if not.

Example

The following example tests to see if any property on the data object is dirty before entering the loop to determine which properties are dirty.

```
If DOToCheck.IsDirty() then
   DOCount = DOToCheck.Count
   DOIndex = 0
   For DOIndex = 0 To DOCount - 1
      Value = EmptyValue
      Name = DOToCheck.Index(DOIndex, Value)
      If DOToCheck.IsPropertyDirty(Name) then
         ...
```

2.22 IsPropertyDirty(PropertyName)

Indicates whether the property value has been changed since the data object was cleaned. Whenever a property on the data object has its value set, the dirty flag is set. This can be used to determine which properties to consider in subsequent operations.

See also: `IsDirty()`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property to check.

Return

Boolean: True if the property value has been changed. False if it has not.

Example

In this example, the returned flag is tested in order to decide whether a property should be merged. See also the fuller example in Count above.

```
...
Dirty = DOToMerge.IsPropertyDirty(Name)
' If the property is dirty on the merge data object
' or all the properties are to be merged.
If Dirty <> False OR MergeOnlyModified = False Then
   ...
```

2.23 IsPropertyEmpty(PropertyName)

Indicates whether the property value is empty (has no value – this is not the same as a string of zero length). If you use the Index method to get the value of a data object property, you will not be able to test for empty using the **IsEmpty** VBScript function, since the function will always return false. Instead you should use this method to test

In general, you should use this method to test the properties of Portrait data objects.

See also: `Index(Index, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property to check.

Return

Boolean: True if the named property is empty, False if it is not.

Example

```
Name = DOToMerge.Index(DOIndex, Value)
If DOToMerge.IsPropertyEmpty(Name) = False then
   OutputValue = Value
Else
   OutputValue = "(Empty)"
End if
```

2.24 SetBooleanValue(PropertyName, Value)

Method used to set the value of a Boolean property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type of the property will be set to Boolean. You can use the `SetValue(PropertyName, Value)` method instead since it will have the same effect.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	Boolean	Variable or constant containing the boolean value to which the property will be set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.25 SetClean()

Whenever a property on the data object has its value set, the dirty flag is set on the property and on the data object to indicate that the object has been changed. This method can be used clear the dirty flag on all the properties of a data object as well as the data object itself.

See also: `IsDirty()`
`IsPropertyDirty(PropertyName)`
`SetPropertyDirty(PropertyName, DirtyState)`

Inputs

None

Return

None

Example

```
Call DOToClean.SetClean()
```

2.26 SetDataObjectValue(PropertyName, Value)

Method used to set the value of a data object property. If the **PropertyName** property does not exist, then the method will create a data object property with that name. Bear in mind that the method does not copy the data object. It merely set the property to point at the same data object.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	DateObject	Variable or constant containing the data object value to which the property is set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.27 SetDateTimeValue(PropertyName, Value)

Used to set the value of a date/time property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type will be set to date/time. You can use the `SetValue(PropertyName, Value)` method instead as it will have the same effect.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	DateTime	Variable or constant containing the date time value to which the property is set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.28 SetDOCollectionValue(PropertyName, Collection)

Used to set the value of a collection property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type will be set to Data Object Collection.

Inputs

Name	Data type	Description
PropertyName	String	The name of the collection property, the value of which is to be set.
Collection	Data object collection	The collection to which the named property refers. Note, the collection is not copied, the property holds a reference to it. If the collection content changes in another context, then the content of the data object reflects that change.

Return

None

Example

The following example sets up a multiple search criterion object as used in the Search for task 3 DAT.

```
Dim TaskCriterionDO
Dim PartyCriterionDO
Dim MultiCriterionDOC
Set TaskCriterionDO = CreateDataObject("General", "SearchCriteria")
Call TaskCriterionDO.SetValue("SearchableColumnName", "TaskType")
Call TaskCriterionDO.SetValue("SearchableColumnValue", TaskType)
Call TaskCriterionDO.SetValue("StringComparisonType", "Equal")
Set PartyCriterionDO = CreateDataObject("General", "SearchCriteria")
Call PartyCriterionDO.SetValue("SearchableColumnName", "PartyID")
Call PartyCriterionDO.SetValue("SearchableColumnValue", PartyID)
Call PartyCriterionDO.SetValue("StringComparisonType", "Equal")
Set MultiCriterionDOC = CreateDOCollection()
Call MultiCriterionDOC.Add(TaskCriterionDO)
Call MultiCriterionDOC.Add(PartyCriterionDO)
Call MultiCriterionDO.SetDOCollectionValue("SearchableColumnValue",
MultiCriterionDOC)
Call MultiCriterionDO.SetValue("StringComparisonType", "AllOf")
```

2.29 SetDoubleValue(PropertyName, Value)

Method used to set the value of a Double (floating point number) property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type of the property will be set to Double. You can use the `SetValue(PropertyName, Value)` method instead since it will have the same effect.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	Double	Variable or constant containing the double value to which the property will be set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.30 SetLongValue(PropertyName, Value)

Method used to set the value of a Long (integer) property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type of the property will be set to Long. You can use the `SetValue(PropertyName, Value)` method instead since it will have the same effect.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	Long (Integer)	Variable or constant containing the long value to which the property will be set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.31 SetPropertyDisplayName (PropertyName, DisplayName)

This method sets the display name of the given property of the data object. The display name of a property is the name seen in the configuration suite and the name used as the label of any property of a CBE (that is, an attribute) is displayed in a generated view.

See also: `GetPropertyDisplayName (PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property.
DisplayName	String	The display name of the property

Return

String: The display name of the data object property.

Example

Also see the example in `GetPropertyDisplayName (PropertyName.`

```
Call DO.SetPropertyDisplayName(PropertyName, DisplayName)
```

2.32 SetPropertyEmpty(PropertyName)

Some nodes and DATs treat empty properties differently from properties where the value is an empty string (“”). This method sets the value of the named property to **<Empty>**.

See also: `IsPropertyEmpty(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property to empty.

Return

None

Example

The following example copies a data object and then empties selected properties in the new copy.

```
Set NewProdDef = OrigProdDef.CreateCopy()
Call NewProdDef.SetPropertyEmpty("PRODUCT_DEFINITION_ID")
Call NewProdDef.SetPropertyEmpty("EFFECTIVE_FROM")
Call NewProdDef.SetPropertyEmpty("EFFECTIVE_TO")
```

2.33 SetPropertyDirty(PropertyName, DirtyState)

Sets or clears the dirty flag on a property. The dirty flag indicates whether a property value has been changed since the object was last cleaned. The flag can be used to determine which properties should be considered in any processing of the data object (for example, saving its contents to the database).

See also: `IsPropertyEmpty(PropertyName)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property for which the dirty flag should be set or cleared.
DirtyState	Boolean	The new state of the Dirty flag. True if the property should be regarded as changed, False if not.

Return

None

Example

This example sets the dirty flag on all the properties of a given data object. This can be useful in the case of nodes that do not set the dirty flag on properties that are empty (for examople, GI node). In these cases, the empty properties will not be processed by DATs and so the corresponding columns in the database will not be cleared.

```
PropertyCount = DOToCheck.Count
DOIndex = 0
For DOIndex = 0 To PropertyCount - 1
   Value = EmptyValue
   Name = DOToCheck.Index(DOIndex, Value)
   SrcType = DOToCheck.GetPropertyType(Name)
   Select case SrcType
      Case 12 ' This is a data object
         Set DOValue = DOToCheck.GetDataObjectValue(Name)
         Call ParseDO(  DOValue, FlagValue, SetFlag)
      Case 13 ' This is a collection
         Set DOCValue = DOToCheck.GetDOCollectionValue(Name)
         Call ParseDOC(  DOCValue, FlagValue, SetFlag)
      Case else ' This is a simple property of the object
         Call DOToCheck.SetPropertyDirty(Name, FlagValue)
   end select
Next
```

2.34 SetStringValue(PropertyName, Value)

Method used to set the value of a String property. If the **PropertyName** property does not exist, then the method will create a property with that name and the data type of the property will be set to String. You can use the `SetValue(PropertyName, Value)` method instead since it will have the same effect.

See also: `SetValue(PropertyName, Value)`

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	String	Variable or constant containing the string value to which the property will be set.

Return

None

Example

See `SetValue(PropertyName, Value)`

2.35 SetValue(PropertyName, Value)

General purpose method to set the value of a property regardless of the property's data type. If the **PropertyName** property does not exist, then the method will create a property with that name (provided that the data object has not been locked). The data type of the property will be determined by the data type of Value supplied in the second parameter.

If you are intending to set the value of a reference property that does not yet exist in the data object, you must use the `AddRefPropertyDefinition (PropertyName, Type)` method in order to determine the reference data group for the property.

See also: SetBooleanValue(PropertyName, Value)
SetDateTimeValue(PropertyName, Value)
SetDoubleValue(PropertyName, Value)
SetLongValue(PropertyName, Value)
SetStringValue(PropertyName, Value)

Inputs

Name	Data type	Description
PropertyName	String	The name of the property, the value of which is to be set.
Value	Variant	Variable or constant containing the value to which the property will be set.

Return

None

Example

This example sets the value of the target property to be the same as the value of the source property. (See fuller example in `Count`.)

```
...
If MatchDataTypes(SrcType, DestType) Then
   ' Copy the value from merge data object to the target data object.
   Call TargetDO.SetValue(Name, DOToMerge.GetValue(Name))
   ...
```

3 Data object collection properties

3.1 Size

Contains the number of data objects in the collection.

Example

The following example removes all the data objects from a collection.

```
While DOCollection.Size > 0
       Call DOCollection.RemoveElementAt(0)
WEnd
```

4 Data object collection methods

4.1 Add(DataObject)

Adds a data object to a collection. The object is added to the end of the collection. Data objects in a collection do not have to be homogeneous (i.e. do not have to be of the same category and type).

Inputs

Name	Data type	Description
DataObject	Data object	The data object to be added to the collection.

Return

None

Example

The following example converts a data object collection with object of an unspecified category and type into a collection of List item data objects for use in generated interaction combo boxes.

```
For Each Item In InputCollection
   Name = ""
   SysName = ""
   Master = ""
   If Item.HasProperty(NamePropertyName) then
      'Test for Reference Data Attribute
      If Item.GetPropertyType(NamePropertyName) = 16 then
         Name = Item.GetReferenceDisplayValue(NamePropertyName,
                                              "Default" )
      else
         Name = Item.GetValue(NamePropertyName)
      end If
   else
      Call LogMessage("Create GI Collection: Error - The Name property ("
           + NamePropertyName +
           ") does not exist in an object in the input collection")
   end if

   If Item.HasProperty(SystemNamePropertyName) then
      SysName = Item.GetValue(SystemNamePropertyName)
   else
      Call LogMessage("Create GI Collection: Error - The System Name
           property (" + SystemNamePropertyName +
           ") does not exist in an object in the input collection")
   end if
   If Len(MasterPropertyName) > 0 then
      If Item.HasProperty(MasterPropertyName) then
         Master = Item.GetValue(MasterPropertyName)
      else
         Call LogMessage("Create GI Collection: Error - The Master
            property (" + MasterPropertyName +
            ") does not exist in an object in the input collection")
      end if
   elseif Len(MasterValue) > 0 then
      Master = MasterValue
```

```
   end if
   ' If name or system name are not supplied then don't
   ' add anything into collection
   If Name <> "" And SysName <> "" Then
      Set ListItem = CreateDataObject("DQEListModel", "ListItem")
      ' Get rid of any new lines in the display name
      ' and replace with commas
      Name = Replace(Name, vbNewLine, ", ")
      Call ListItem.SetValue("Name", Name)
      Call ListItem.SetValue("SystemName", SysName)
      If Master <> "" Then
         Call ListItem.SetValue("Master", Master)
      end if
      Call GIListCollection.Add(ListItem)
   else
      Call LogMessage("Create GI Collection: Error -
                       Both the Name and System Name properties must be
                       present.  The list item has not been added.")
   end if
next
```

Clear `HRESULT Clear();`

4.2 CreateCopy()

Creates a copy of a data object collection. This method is useful if you need to keep an original copy of a collection for comparison purposes.

It is worth noting that you cannot achieve the same effect by mapping a collection out of a data manipulation node twice. This merely creates two context references to the same collection. Changes to one context entry will be seen in the other.

Inputs

None

Return

Data object collection: A copy of the data object collection on which the method is invoked.

Example

```
Set CopiedDataObjectCollection = DataObjectCollectionToCopy.CreateCopy()
```

4.3 ElementAt (Index)

Returns a data object in a collection.

See also: `RemoveElementAt (Index)`
`Item (Offset)`

Inputs

Name	Data type	Description
Index	Integer	The index of the data object to be returned. The index of the first data object is 0 (zero), the index of the second is 1 and so on.

Return

Data object: The data object at the position identified by Index.

Example

The following example is an alternative way of iterating through a data object collection.

```
Index = 0
While Index < DOCollection.Size
   Set ExtractedDataObject = DOCollection.ElementAt(Index)
   Index = Index + 1
WEnd
```

4.4 IsDirty()

Indicates whether the value of any property on any data object in the collection has been changed since the data object collection was cleaned/created. Whenever a property on the data object has its value set, the dirty flag is set on the property and on the data object.

See also: `IsDirty()` (Data object method)
IsPropertyDirty(PropertyName) (Data object method)
SetClean() (Data object method)
SetClean() (Data object collection method)

Inputs

None

Return

Boolean: True if any property on the object is dirty, False if not.

Example

See the example for the Data object method `IsDirty()`.

4.5 IsEmpty()

Evaluates the number of data objects in the collection. Calling this method is exactly the same as examining the **Size** property.

See also: **Size**

Inputs

None

Return

Boolean: True if the collection contains 1 or more data objects. False if it contains none.

Example

The following example removes all the data objects from a collection.

```
While DOCollection.IsEmpty() > 0
       Call DOCollection.RemoveElementAt(0)
WEnd
```

4.6 Item (Offset)

Returns a data object in a collection. This method is the same as the `ElementAt (Index)` method except for the fact that the Offset input is '1' based (that is, starts at 1) whereas the Index input used on the other method is '0' based.

See also: `ElementAt (Index)`

Inputs

Name	Data type	Description
Offset	Integer	The offset of the data object to be returned. The offset of the first data object is 1, the offset of the second is 2 and so on.

Return

Data object: The data object at the position identified by Offset.

Example

The following example is an alternative way of iterating through a data object collection.

```
Offset = 0
While Offset <= DOCollection.Size
       Set ExtractedDataObject = DOCollection.Item(Offset)
       Offset = Offset + 1
WEnd
```

4.7 RemoveElementAt (Index)

Removes a data object from a collection. The data object itself may be referred to elsewhere, so the method does not delete the data object.

See also: `ElementAt (Index)`

Inputs

Name	Data type	Description
Index	Integer	The index of the data object to be removed. The index of the first data object is 0 (zero), the index of the second is 1 and so on.

Return

None

Example

The following example removes all the data objects from a collection.

```
While DOCollection.IsEmpty() > 0
       Call DOCollection.RemoveElementAt(0)
WEnd
```

4.8 SetClean()

Whenever a property on the data object has its value set, the dirty flag is set on the property and on the data object to indicate that the object has been changed. This method can be used clear the dirty flag on all the properties of all the data objects in the collection as well as each data object itself.

See also: `IsDirty()` (Data object method)
IsPropertyDirty(PropertyName) (Data object method)
SetPropertyDirty(PropertyName, DirtyState) (Data object method)

Inputs

None

Return

None

Example

```
Call DOCollectionToClean.SetClean()
```

5 Scripting helper functions

5.1 CreateDataObject(Category, Type)

Creates an initialized and classified Portrait data object. The data object will contain all the properties identified by the data object factory. Each of the properties will be empty.

Inputs

Name	Data type	Description
Category	String	The name of the data object category to use.
Type	String	The name of the data object type to use. The type must be declared within the category specified above.

Return

Data object: The classified data object.

Example

This example creates a data object and then adds the object to a collection.

```
Dim SingleStringItem
Set SingleStringItem = CreateDataObject("General", "SingleString")
Call SingleStringItem.SetValue("SingleValue", (SingleValue))
Call SingleStringCollection.Add(SingleStringItem)
```

5.2 CreateDOCollection()

Creates an empty data object collection.

Inputs

None

Return

Data object collection: An empty data object collection

Example

This example creates an empty data object collection (collection with no elements) if the variable identifying the collection is empty (an empty variable is not the same as an empty collection).

```
If IsEmpty(SingleStringCollection) then
   Set SingleStringCollection = CreateDOCollection()
end if
```

5.3 DecryptID(EncryptedID)

Decrypts a string value in accordance with the algorithm used to decrypt configurable business entity IDs that are passed around Portrait process models.

The IDs are encrypted by DATs when data is fetched from the database and decrypted when the data is returned to the database.

If IDs are stored in configurable columns, the DATs will not convert the IDs and the stored data will be encrypted. This can make it difficult to extract management information from the database because it is not possible for a query to use the encrypted ID to join to the relevant table in the database.

In these circumstances it is useful to decrypt IDs before storing them in configurable columns.

WARNING: The decrypted value will be visible in the model context for a brief period. This may be regarded as a security risk in some implementations.

See also: `EncryptID(DecryptedID)`

Inputs

Name	Data type	Description
EncryptedID	String	The ID to be decrypted

Return

Integer: The decrypted ID. This can be stored in configurable columns in Portrait configurable business entities to allow joins in MIS.

Example

```
DecryptedID = DecryptID(EncryptedID)
```

5.4 EncryptID(DecryptedID)

Encrypts a numeric value in accordance with the algorithm used to encrypt configurable business entity IDs that are passed around Portrait process models.

The IDs are encrypted by DATs when data is fetched from the database and decrypted when the data is returned to the database.

If IDs are stored in configurable columns, the DATs won't convert the IDs and the stored data will be encrypted. This can make it difficult to extract management information from the database because it is not possible for a query to use the encrypted ID to join to the relevant table in the database.

In these circumstances, IDs are sometimes decrypted and it is then necessary to encrypt them again for use in Portrait process models.

WARNING: The decrypted value will be visible in the model context for a brief period. This may be regarded as a security risk in some implementations.

See also: `DecryptID(EncryptedID)`

Inputs

Name	Data type	Description
DecryptedID	Integer	Although floating point numbers can be supplied these are rounded to the nearest whole number before being encrypted.

Return

String: The encrypted ID. This can be passed around a model context and then passed into DATs that expect an encrypted ID.

Example

```
EncryptedID = EncryptID(DecryptedID)
```

5.5 FormatString(FormatString, Array)

This function creates a string by merging items in the array into the format string as follows:

- The format string is a blend of text and zero or more tokens. (for example, "Your current role has changed from %1 to %2")
- A token is identified by the % (percent) symbol followed immediately by a whole number. (for example, "%1")
- Each array element replaces the corresponding token in the merge string. (that is, element 1 replaces token 1, element 2 replaces token 2 and so on.).
- The same token can appear any number of times in the format string
- The first token considered is token 1. So element 0 (zero) in the array will be ignored.
- The **%** symbol will be treated literally in the format string if it is preceded by another %
- The **%** symbol is ignored if it is not followed immediately by a whole number.
- Any array elements that have no corresponding token will be ignored
- Any token that has no corresponding array element will result in an error.

Inputs

Name	Data type	Description
FormatString	String	This is a string containing text and tokens. The string can contain zero or more tokens. The first token is token 1 (that is, %1)
Array	Array	A one-dimensional array. The first element of the array (element 0) is ignored. The data type of each element can vary - provided that it can be coerced into a string.

Return

String: The formatted string. All tokens will have been replaced by text from corresponding elements in the array

Example

The following example retrieves the format string from a reference data group and uses this in conjunction with 9 string inputs to create the formatted message.

```
Dim InputArray
Dim ReferenceDataItem
Dim MessageFormat

' Default the Message output to be the MessageFormatID, so that if no item is
found, the message is not empty and gives some idea of the problem
Message = MessageFormatID

' Create the array making sure that the first element is not used
InputArray = Array("", String1, String2, String3, String4, String5, String6,
                   String7, String8, String9)

Set ReferenceDataItem = GetReferenceDataItem("MessageFormats",
MessageFormatID, Language)
' If we get a valid object
```

```
If IsEmpty(ReferenceDataItem) = False then
   ' Make sure it has the right property
   If ReferenceDataItem.HasProperty("MessageFormat") <> False then
      ' Check that we get a message and that the array has at least
      ' one element in the range 1 to 9 (not 0)
      If Len(ReferenceDataItem.MessageFormat) > 0
              and UBound(InputArray) > 1 then
         Message = FormatString(ReferenceDataItem.MessageFormat, InputArray)
      end if
   end if
end if
```

5.6 GetReferenceDisplayValue(RDGSystemName, RDISystemName, Language)

Returns the display name associated with a given reference data item.

Inputs

Name	Data type	Description
RDGSystemName	String	The system name of the reference data group in which the item occurs.
RDISystemName	String	The system name of the reference data item for which the display name is required.
Language	String	The language in which the display name should be returned. The value supplied must be the system name of one of the values in the Implementation languages reference data group. “Default” is the value normally supplied unless your implementation requires multi-language support.

Return

String: The display name for the reference data item.

Example

```
DisplayText = GetReferenceDisplayValue(RDGSystemName, RDISystemName, "Default" )
```

5.7 GetReferenceDataItem (RDGSystemName, RDISystemName, Language)

Returns a reference data item data object containing the attribute values associated with a given reference data item's system name.

Inputs

Name	Data type	Description
RDGSystemName	String	The system name of the reference data group in which the item occurs.
RDISystemName	String	The system name of the reference data item for which the item's attributes are required.
Language	String	The language in which the display name and other configurable attributes should be returned. The value supplied must be the system name of one of the values in the Implementation languages reference data group. "Default" is the value normally supplied unless your implementation requires multi-language support.

Return

Data object: A data object representing the reference data item requested. The data object will be classified as follows:

Category: ReferenceDataItem

Type: RDGSystemName

The object will contain one property for each attribute of the reference data item excepting the Description attribute.

Example

The following example iterates over a collection produced by the Smart Lookup node. The collection contains the system names of reference data items. The script is used to filter a reference data group using a smart lookup.

```
Dim Item
Dim ReferenceDataItem

For each Item in SmartLookup.Results
   Set ReferenceDataItem =
         GetReferenceDataItem(RDGName, Item.Result, Language)
   Call ReferenceDataItems.Add(ReferenceDataItem)
Next
```

5.8 LogMessage(MessageText)

The function logs messages for use in the various Portrait log destinations (for example, MDT, LogViewer). Messages logged in this way appear if the log filter includes category 16. For example:

```
[(*):(*):(*):(5,6,16,17,18,19,20,21,22)][(2):!(1):(*):(*)])
```

This function will also display the **MessageText** in a popup window in the script test environment.

Logged messages appear in the form:

Method	AdditionalInfo	Category
CAmcRuntimeScriptingHelper::LogMessage	Found the tab 'Root tab' at the position '0'.	16 AMC_LOGCAT_SCRIPTPROCESSING

Inputs

Name	Data type	Description
MessageText	String	The text of the message to be logged.

Return

None

Example

This example logs a message when a boolean value is true.

```
If ParentTabFound Then
   Call LogMessage("Found the tab '"
                   + ParentTab.Label +
                   "' at the position '"
                   + CStr(ParentTabPosition) + "'.")
   Set FindTab = ParentTab
Else
   Set FindTab = Nothing
End If
```

6 Scripting helper variables

get_ActivityToken	BSTR* pbstrActivityToken;
put_ActivityToken	BSTR bstrActivityToken;
get_ScriptName	BSTR* pbstrScriptName;
put_ScriptName	BSTR bstrScriptName;