Article
Eduard Lebedyuk · Jul 5, 2016 3m read

Useful auto-generated methods

For each defined property, query or an index, several corresponding methods would be automatically generated on a class compilation. These methods can be very useful. In this article, I would describe some of them.

Properties

Let ' s say you defined a property named " Property " . The following methods would be automatically available (bold property is a variable part, equal to property name):

```
ClassMethod PropertyGetStored(id)
```

For datatype properties this method returns their logical value, for object properties, it returns the id. It ' s a wrapped global reference to the class data global and the fastest way to retrieve the singular property value. This method is only available for stored properties.

```
Method PropertyGet()
```

Is a property getter. Can be redefined.

```
Method PropertySet(val) As %Status
```

Is a property setter. Can be redefined.

Object properties

If it ' s an object property, some additional methods, related to ID and OID access become available:

```
Method PropertySetObjectId(id)
```

This method sets property value by ID, so there is no need to open an object to set it as a property value.

```
Method PropertyGetObjectId()
```

This method returns property value ID.

```
Method PropertySetObject(oid)
```

This method sets property value by OID.

```
Method PropertyGetObject()
```

This method returns property value OID.

Datatype properties

For a datatype property several other methods for conversion between different formats become available:

```
ClassMethod PropertyDisplayToLogical(val)
ClassMethod PropertyLogicalToDisplay(val)
ClassMethod PropertyOdbcToLogical(val)
ClassMethod PropertyLogicalToOdbc(val)
ClassMethod PropertyXSDToLogical(val)
ClassMethod PropertyLogicalToXSD(val)
```

```
ClassMethod PropertyIsValid(val) As %Status
```

Checks if val is a valid property value

```
ClassMethod PropertyNormalize(val)
```

Returns normalized logical value

Notes

- Relationships are properties and can be get/set with these methods
- Input val is always a logical value, except for format conversion methods.

Indexes

For an index named " Index " , the following methods would be automatically available

```
ClassMethod IndexExists(val) As %Boolean
```

Returns 1 or 0 depending on whatever object with this val exists, where val is a logical value of the indexed property.

Unique Indexes

For unique indexes, additional methods become available:

```
ClassMethod IndexExists(val, Output id) As %Boolean
```

Returns 1 or 0 depending on whatever object with this val exists, where val is a logical value of the indexed property. Also returns object id (if found) as a second argument.

```
ClassMethod IndexDelete(val, concurrency = -1) As %Status
```

Deletes entry with index value equal to val.

```
ClassMethod IndexOpen(val, concurrency, sc As %Status)
```

Returns existing object with index value equal to val.

Notes:

a) As an index may be based upon several properties, the method signature would change to have several values as an input, for example, consider this index:

```
Index MyIndex On (Prop1, Prop2);
```

Then IndexExists method would have the following signature:

```
ClassMethod IndexExists(val1, val2) As %Boolean
```

Where val1 corresponds to Prop1 value and val2 corresponds to Prop2 value. Other methods follow the same logic.

b) Caché generates an IDKEY index that indexes the ID field (RowID). It can be redefined by the user and can also contain several properties. For example, to check if class has some property defined execute:

```
Write ##class(%Dictionary.PropertyDefinition).IDKEYExists(class, property)
```

c) All index methods check for a logical value

d) Documentation

Queries

For a query (which can be a simple SQL query or a custom class query, here ' s my post about them) named " Query " Func method gets generated:

```
ClassMethod QueryFunc(Arg1, Arg2) As %SQL.StatementResult
```

which returns a %SQL.StatementResult used to iterate over the query. For example Sample.Person class in Samples namespace has a ByName query accepting one parameter. It can be called from object context with this code:

```
Set ResultSet=##class(Sample.Person).ByNameFunc("A")
While ResultSet.%Next() { Write ResultSet.Name,! }
```

Additionally, there is a demo class on GitHub demonstrating these methods.

#Best Practices #Code Snippet #Object Data Model #Caché #InterSystems IRIS