Article
Ed de Moel · Aug 29, 2019 7m read

DeSerializing JSON stream into well-defined object

Some time ago, InterSystems introduced the concept of %DynamicObjects.
This feature is a powerful tool that makes it very easy to convert any string of JSON text to objects and vice versa.
However, in the work that J2 Interactive is doing for our customers, there are a couple of things that "need some tweaking".

Very often, we create REST services for our customers. Those services have a payload (in JSON format) that contains information that our customer expects in the form of an object that is proprietary to them.
We can easily convert that JSON stream to a %DynamicObject, but, if we try to pass such an object to the (legacy) software of our customer, there are some issues:

- If the expected object contains a "collection", their well-defined object would use methods like **Count()** and **GetAt()** to get at the individual elements of their object, whereas the %DynamicObject does not recognize those methods and would require the methods **%Size()** and **%Iterate**.
- If the customer's software references a property that was omitted in the JSON stream, then we'll get a <Property Does Not Exist> type of error

So, in order to pass an appropriate object to the customer's software, we'll have to create a new instance of the intended object, and copy all relevant properties from the %DynamicObject into the well-defined object.
When the object in question only has a handful of properties, that would require only a handful of Set commands.
However, reality is that many of those objects have dozens of properties, and some of those are arrays or lists of other (non-trivial) objects.

So, we developed a method does the tedious work for us. We called it DeSerialize. It takes a JSON-string and the name of a target object, and it creates a new instance of that object, populating those properties (recursively for properties that are objects, and iteratively for properties that are arrays or lists) from the information in the %DynamicObject that can be created from the JSON string.

Here's the code:

```
/// This method is intended to replace the "copy payload to Customer's Object" sequences
/// in the various REST calls.
///
/// The idea here is as follows:
/// Our input stream has a JSON object, we typically store it in local variable tPayLoad
/// At some point, we have to pass an object to the Customer's software, but if we pass
/// the tPayload object, and the Customer's software references a property that isn't
/// specified in the input stream, we get a <no such property> kind of error.
/// We also cannot just copy over any property that is in pPayLoad into the target
/// object, because, if there happens to be a typo in the name of a property, we
/// will also get a <no such property> type of error.
/// So, this method is intended to take the tPayLoad object and copy it into
/// a Customer's object.
/// It does so by looping through the properties in the definition of the Conduent object
/// and, if these exist in tPayLoad, copy them over to the target object.
/// If a property is an object, it does so recursively.
/// If a property is an array (or list), it does so repeatedly.
ClassMethod DeSerialize(pJson As %String, pObjectName As %String)
```

```
{
  Kill %Array,%Object,%Value,%Count
  Set tPayLoad=##class(%DynamicAbstractObject).%FromJSON(pJson)
  Do ..SimpleTypes(.types)
  Set pObject=$Xecute("() Quit ##class("_pObjectName_").%New()")
  Do ..CopyObject("pPayLoad","pObject",1,pObjectName,.types,.pObject,.tPayLoad)
  Quit pObject
}

ClassMethod CopyObject(pName1 As %String, pName2 As %String, pLevel As %Integer,
pObjectName As %String, pTypes As %String, pObject As %RegisteredObject, pPayLoad As
%DynamicAbstractObject)
{
  Set tProp="" For
  {
  Set tProp = $Order(^oddCOM(pObjectName,"a",tProp)) Quit:tProp=""
  Continue:tProp="%%OID"
  Continue:'$Xecute("(pPayLoad) Quit "_pName1_".%IsDefined("""_tProp_""")",pPayLoad)
  Set tParent = $Get(^oddCOM(pObjectName,"a",tProp,2))
  Set:tParent="" tParent=pObjectName
  Set tType=$Get(^oddDEF(tParent,"a",tProp,5))
  Set tStream = tType["Stream"
  Set:tStream pTypes(tType)="string"
  Set array=$Get(^oddCOM(pObjectName,"a",tProp,40))
  Set isarray=0
  Set:array="array" isarray=1
  Set:array="list" isarray=1
  Set:array="array" isarray=1
  Set:array="list" isarray=1
  Set:array["Collection" isarray=1
  Set:$Get(^oddCOM(pObjectName,"a",tProp,"m","BuildValueArray",2))["Collection" isarray=1
  Set acttype=$Get(pTypes(tType))
  If isarray
  {
  If $Xecute("(pPayLoad) Quit "_pName1_".%IsDefined("""_tProp_""")",pPayLoad)
  {
  If $Xecute("(pPayLoad) Quit "_pName1_"."_tProp_".%Size()",pPayLoad)
  {
  Set %Object(pLevel)=$Xecute("() Quit ##class("_tType_").%New()")
  Set tIterator = $Xecute("(pPayLoad) Quit "_pName1_"."_tProp_".%GetIterator()",pPayLoad)
  Set %Count(pLevel)=0
  While tIterator.%GetNext(.tKey,.tValue)
  {
  Set %Value(pLevel)=tValue,%Count(pLevel)=%Count(pLevel)+1
  If acttype=""
  {
  Set add1="%Value("_pLevel_")"
  Set add2="%Array("_pLevel_")"
  Set %Array(pLevel)=$Xecute("() Quit ##class("_tType_").%New()")
  Do ..CopyObject(add1,add2,pLevel+1,tType,.pTypes,.pObject,%Value(pLevel))
  Set void=$Xecute("(pObject) Do "_pName2_"."_tProp_".Insert(%Array("_pLevel_")) Quit 0",pObject)
  }
  Else
  {
  Set void=$Xecute("(pValue) Do "_pName2_".Insert(pValue) Quit 0",tValue)
  }
  }
  }
  }
```

```
  }
  Else
  {
  If acttype=""
  {
  If $Xecute("(pPayLoad) Quit "_pName1_".%IsDefined("""_tProp_""")",pPayLoad)
  {
  Set %Object(pLevel)=$Xecute("() Quit "_"##class("_tType_").%New()")
  Do ..CopyObject(pName1_"."_tProp,"%Object("_pLevel_")",pLevel+1,tType,.pTypes,pObject,pPayLoad)
  Set void=$Xecute("(pObject) Set "_pName2_"."_tProp_" = %Object("_pLevel_") Quit 0",pObject)
  }
  }
  Else
  {
  If tStream
  {
  Set void=$Xecute("(pObject,pPayLoad) Do:"_pName1_".%IsDefined("""_tProp_""")
"_pName2_"."_tProp_".Write("_pName1_"."_tProp_") Quit 0",pObject,pPayLoad)
  }
  Else
  {
  Set void=$Xecute("(pObject,pPayLoad) Set:"_pName1_".%IsDefined("""_tProp_""") "_pName2_"."_tProp_" =
"_pName1_"."_tProp_" Quit 0",pObject,pPayLoad)
  }
  Continue
  }
  }
  }
}

ClassMethod SimpleTypes(pTypes As %String)
{
  Kill pTypes
  Set pTypes("%Integer")="integer"
  Set pTypes("%String")="string"
  Set pTypes("%Boolean")="boolean"
  Set pTypes("%Date")="string"
  Set pTypes("%Time")="string"
  Set pTypes("%DateTime")="string"
  Set pTypes("%GlobalBinaryStream")="string"
  Set pTypes("Datatype.Boolean")="boolean"
  Set pTypes("Datatype.Boolean01")="boolean"
  Set pTypes("Datatype.Currency")="string"
  Set pTypes("Datatype.Date")="string"
  Set pTypes("Datatype.Time")="string"
  Set pTypes("Datatype.Float")="number"
  Set pTypes("Datatype.Integer")="integer"
  Set pTypes("Datatype.String")="string"
  Set pTypes("Enum.SmarTrackWorklistColumnType")="string"
  Set pTypes("Enum.FunctionNavigateType")="string"
  Set pTypes("!")="string"
}
```

#JSON #Object Data Model #REST API #InterSystems IRIS