Article
Yuri Marx · Nov 25, 2021 3m read

Open Exchange

The power of XDATA applied to the API Security

The XData (https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GOBJXDATA) is a powerful feature to set documentation and metadata information for classes and methods. The %CSP.REST class uses XDATA to mapping REST calls (https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GRESTcsprest), so in this article you will see how to use XData into your apps as code, not only as documentation.

When you write XData comments/definitions, the IRIS store it into %Dictionary.ClassDefinition (for classes) %Dictionary.MethodDefinition (for methods). If you query these tables, you will be able get metadata information and write code to this metadata configuration. %CSP.REST do this when you write your REST mappings for your REST Services using ObjectScript.

I wrote an application that is using XDATA to enforce authorization rules to the class method endpoints, see:

```
/// Retreive all the records of dc.Sample.Person
/// @security.and: roles: { PersonAdmin }
ClassMethod GetAllPersons() As %Status
{

  #dim tSC As %Status = $$$OK
....
}
```

The @security.and does not exists into IRIS. So I need to read this configuration and write code to enforce access to the users with PersonAdmin role only.

To get this @security.and, you need to read this XData. See:

```
ClassMethod GetXDataContent(className, methodName) As %String
{

  Set qryXdata = "SELECT parent, Name, Description FROM
%Dictionary.MethodDefinition WHERE parent = ? and Name = ?"
  Set stmXdata = ##class(%SQL.Statement).%New()
  Set qStatus = stmXdata.%Prepare(qryXdata)
  If qStatus'=1 {Write "%Prepare
failed:" Do $System.Status.DisplayError(qStatus) Quit}
  Set rsetXdata = stmXdata.%Execute(className, methodName)
    While rsetXdata.%Next() {
  // Return rsetXdata.Name
  Return rsetXdata.Description
```

```
  }
}
```

With this method you be able to get any xdata content for methods.

Now, to restrict access only to the users with the PersonAdmin role is simple. You need to override AccessCheck ClassMethod from %CSP.REST class. See:

```
ClassMethod AccessCheck(Output pAuthorized As %Boolean = 0) As %Status
{

 Do ##super()

 Set message = {}

 Set tSC = $$$OK

 Set message.verb = %request.Method
   Set message.url = %request.URL
   Set message.url = "/"_$REPLACE(message.url, %request.Application, "")
   Set message.application = %request.Application

 Set methodName = ""
 Do ..GetClassMethodName(message.url, %request.Method, .methodName)

 Set message.method = methodName
   Set xdata = ##class(dc.SecurityMediator.XDataUtil).
GetXDataContent($CLASSNAME(), methodName)
   Do ..GetSecurityRules(xdata, .rules, .roles, .header, .operator)
   Set UserRoles = $LISTFROMSTRING($ROLES,",")
 Set RolesAllowed = UserRoles

 If $FIND(xdata, "@security") > 0 {
  Set RolesAllowed = $LISTFROMSTRING(roles,",")
 }
   Set HasRole = 0

 For RoleIdx=1:1:$LISTLENGTH(UserRoles) {
  If $LISTFIND(RolesAllowed, $LIST(UserRoles, RoleIdx)) {
  Set HasRole = 1
  Quit
  }
```

```
  }
    If HasRole {
   Set pAuthorized = 1
  } Else {
   Set pAuthorized = 0
   Set message.error = $USERNAME_" is not authorized for this request. User
Roles Allowed is not in User Roles"
   Write message.%ToJSON()
  }
    Return tSC
}
```

With the rule match, set pAuthorized = 1, otherwise, set 0.

Now the roles allowed is based into XData configuration to your REST Class. Great!

If you want to see this in action, get my new app: https://openexchange.intersystems.com/package/API-Security-Mediator.

#ObjectScript #REST API #Security #InterSystems IRIS
Check the related application on InterSystems Open Exchange