```
/// A message class generator for JDBC stored procedure calls
Class Common.JDBC.MakeRequestClass Extends (%RegisteredObject, %XML.Adaptor)
{

/// implementation requirements, leave empty here
Parameter LINK;

Parameter MAKE;

Parameter GROUP;

/// create a new message from LINK's stored procedure spec
ClassMethod GetParamArray(ByRef pParamArray As %String) [ CodeMode = objectgenerator ]
{
	// build an array of the stored procedure's parameters
	set dict = ##class(%Dictionary.ClassDefinition).%OpenId(%parameter("LINK"))
	if '$IsObject(dict) quit $$$OK

	// the follow method-generated code is for post-compile debugging purposes only
	// there can be only one
	if dict.Methods.GetAt(1).ClassMethod {
		do %code.WriteLine(" set dict = ##class(%Dictionary.ClassDefinition).%OpenId(..#LINK)")
		do %code.WriteLine(" set proc = dict.Methods.GetAt(1).Name")
		do %code.WriteLine(" set spec = dict.Methods.GetAt(1).FormalSpec")
		do %code.WriteLine(" for i = 1:1:$l(spec,"","") {")
		// render the parameter list
		set proc = dict.Methods.GetAt(1).Name
		set spec = dict.Methods.GetAt(1).FormalSpec
		do %code.WriteLine(" set pParamArray(0)=""_%class.Name_""""")
		for i = 1:1:$l(spec,",") {
			do %code.WriteLine(" set pParamArray(i,""Name"")=$p($p(spec,"","",i),"":"")")
			do %code.WriteLine(" set pParamArray(i,""Type"")=$p($p($p(spec,"","",i),"":"",2),""("")")
		}
		do %code.WriteLine(" }")
	}
	do %code.WriteLine(" quit")

	// the real work begins here
	// re-write this message class's properties
	do ##class(%Dictionary.ClassDefinition).%DeleteId(%parameter("MAKE"))
```

```
	set mesg = ##class(%Dictionary.ClassDefinition).%New(%parameter("MAKE"))
	if '$IsObject(mesg) quit '$$$OK
	set mesg.Description = "DO NOT TOUCH!"_$c(13,10)_"auto-generated by Common.JDBC.MakeRequestClass via
"_%class.Name_$C(13,10)_"Authored by Robert Hurst"
	// set mesg.ProcedureBlock = 1
	set mesg.Super = "Common.JDBC.StoredProcedureRequest"
	set parm = ##class(%Dictionary.ParameterDefinition).%New()
	set parm.Description = "the name of the JDBC proxy class created by SQL Link Procedure Wizard"
	set parm.Name = "LINK"
	set parm.Default = %parameter("LINK")
	do mesg.Parameters.Insert(parm)
	; re-write GROUP parameter as false (0), because this is the base message
	set parm = ##class(%Dictionary.ParameterDefinition).%New()
	set parm.Description = "supply whether message class is a group of messages, or not"
	set parm.Name = "GROUP"
	set parm.Type = "INTEGER"
	set parm.Default = 0
	do mesg.Parameters.Insert(parm)
	set n = $l(spec,",")
	set pad = $s(n<10:1, n<100:2, 1:3)
	for i = 1:1:n {
		set prop = ##class(%Dictionary.PropertyDefinition).%New()
		set name = $p($p(spec,",",i),":")
		set type = $p($p(spec,",",i),":",2)
		set prop.Name = "x"_$tr($j(i,pad)," ","0")_name
		set prop.SequenceNumber = i + 1
		set prop.Type = $p(type,"(")
		do mesg.Properties.Insert(prop)
		if prop.Type="%String",type'["(" set type=type_"(MAXLEN=512)"
		if type["(" {
			set parm = $p($p(type,"(",2),")")
			set element = $p(parm,"=",2)
			set key = $p(parm,"=")
			for p = 1:1:$l(parm,",") do prop.Parameters.SetAt(element,key)
		}
	}
	do mesg.%Save()

	set status = $SYSTEM.OBJ.Compile(mesg.Name)
	if status '= $$$OK quit status
```

```
	// generate a super-class to the MAKE target for an array of messages
	if %parameter("GROUP") {
		set reqArray = %parameter("MAKE") _ "Group"
		do ##class(%Dictionary.ClassDefinition).%DeleteId(reqArray)
		set mesg = ##class(%Dictionary.ClassDefinition).%New(reqArray)
		if '$IsObject(mesg) quit '$$$OK
		set mesg.Description = "DO NOT TOUCH!"_$c(13,10)_"auto-generated by Common.JDBC.MakeRequestClass via
"_%class.Name_$C(13,10)_"Authored by Robert Hurst"
		// set mesg.ProcedureBlock = 1
		set mesg.Super = "Common.JDBC.StoredProcedureRequest"
		set parm = ##class(%Dictionary.ParameterDefinition).%New()
		set parm.Description = "the name of the JDBC proxy class created by SQL Link Procedure Wizard"
		set parm.Name = "LINK"
		set parm.Default = %parameter("LINK")
		do mesg.Parameters.Insert(parm)
		set parm = ##class(%Dictionary.ParameterDefinition).%New()
		set parm.Description = "supply whether message class is a group of messages, or not"
		set parm.Name = "GROUP"
		set parm.Type = "INTEGER"
		set parm.Default = 1
		do mesg.Parameters.Insert(parm)
		set prop = ##class(%Dictionary.PropertyDefinition).%New()
		set prop.Name = "group"
		set prop.SequenceNumber = 1
		set prop.Collection = "array"
		set prop.Type = %parameter("MAKE")
		do mesg.Properties.Insert(prop)
		do mesg.%Save()
		set status = $SYSTEM.OBJ.Compile(mesg.Name)
	}

	quit status
}

}
```

```
/// Adapter that handles internal service requests by acting as a JDBC client to an external SQL stored procedure.
Class Common.JDBC.OutboundAdapter Extends Ens.OutboundAdapter [ ProcedureBlock ]
{

/// Name of the auto-generated class from JDBC.MakeRequestClass
Property RequestClassname As %String;

Parameter SETTINGS = "RequestClassname";

}
```

```
/// Takes parameters from Cach&eacute; SQL <a href="/csp/sys/exp/UtilSqlHome.csp" target="_blank">Link Procedure Wizard</a> (proxy
class)
/// and invokes it.
Class Common.JDBC.StoredProcedureOperation Extends Ens.BusinessOperation [ ProcedureBlock ]
{

Parameter ADAPTER = "Common.JDBC.OutboundAdapter";

Parameter INVOCATION = "Queue";

/// Name of the Cach&eacute; SQL <a href="/csp/sys/exp/UtilSqlHome.csp" target="_blank">Link Procedure Wizard</a>'s proxy class.
Property TargetConfigName As %String(MAXLEN = 100);

Parameter SETTINGS = "TargetConfigName";

XData MessageMap
{
<MapItems>
  <MapItem MessageType="Ens.Request">
    <Method>CallJDBC</Method>
  </MapItem>
</MapItems>
}

Method OnInit() As %Status
{
	$$$TRACE("OnInit()")
	quit $$$OK
}

Method CallJDBC(pRequest As Ens.Request, Output pResponse As Ens.Response) As %Status
{
	//do ..Adapter.CallProc(..TargetConfigName, ..Adapter.RequestClassname, pRequest.OutputLine())
	set i = pRequest.CallProc()
	if i = 0 {
		quit $$$OK
	}
	else {
		quit $$$ERROR(i)
	}
```

```
}

Method OnTearDown() As %Status
{
	$$$TRACE("OnTearDown()")
	quit $$$OK
}

}
```

```
Include %occErrors

/// message operation for invoking a stored procedure call in another database
/// via a JDBC gateway connection.
Class Common.JDBC.StoredProcedureRequest Extends (Ens.Request, %XML.Adaptor) [ Inheritance = right ]
{

/// implementation requirements, leave empty here
Parameter LINK;

Parameter GROUP;

Method CallProc() As %Integer [ CodeMode = objectgenerator ]
{
	do %code.WriteLine(" // auto-generated by "_%class.Name)
	set dict = ##class(%Dictionary.ClassDefinition).%OpenId(%parameter("LINK"))
	if '$IsObject(dict) quit $$$OK

	// there can be only one
	if dict.Methods.GetAt(1).ClassMethod {
		set proc = dict.Methods.GetAt(1).Name
		set spec = dict.Methods.GetAt(1).FormalSpec
		set n = $l(spec,",")
		set pad = $s(n<10:1, n<100:2, 1:3)
		set JDBC = "##class("_dict.Name_").""_proc_""("

		if %parameter("GROUP") {
			do %code.WriteLine(" set key = """ for { ")
			do %code.WriteLine(" set key = ..group.Next(key) quit:key=""""")
		}

		// render the parameter list
		for i = 1:1:$l(spec,",") {
			if i>1 set JDBC = JDBC_","
			if %parameter("GROUP") {
				set JDBC = JDBC_"..group.GetAt(key).x"_$tr($j(i,pad)," ","0")_$p($p(spec,",",i),":")
			}
			else {
				set JDBC = JDBC_"..x"_$tr($j(i,pad)," ","0")_$p($p(spec,",",i),":")
			}
```

```
		}
		set JDBC = JDBC_")"
		do %code.WriteLine(" set result = " _ JDBC)

		if %parameter("GROUP") {
			do %code.WriteLine(" if result '= 0 quit")
			do %code.WriteLine(" }")
		}
	}

	do %code.WriteLine(" quit $s($d(result):result, 1:$lb(0, ""invalid SP method "_ %parameter("LINK") _""""))")
	quit $$$OK
}

}
```