Question
Yone Moreno · Jan 12, 2021

How could we get the name, port and type of all services in a namespace

Hello,

First of all thanks for your help,

We would need **to get the list of all names, ports and types from the services listed in a namespace**

We have read the following documentation:

https://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?...

https://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?...

In addition, we have read the following topics:

https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls...

https://community.intersystems.com/post/how-do-i-retrieve-values-specifi...

We have found that we could get the list of all components inside an instance as follows:

```
SELECT
ID, AlertGroups, Category, ClassName, Comment, DisableErrorTraps, Enabled, Foreground
, LogTraceEvents, Name, PoolSize, Production, Schedule, Settings
FROM Ens_Config.Item
```

Also, we have read Marc Mundt's last comment in this topic:

https://community.intersystems.com/post/how-do-i-retrieve-values-specifi...

We have imported the class

And we have tried to use it, however we are not using it properly:

```
ESBDESARROLLO:ESBSSCC>##class(Sample.Util).SettingsByNameExecute("Port")

##class(Sample.Util).SettingsByNameExecute("Port")
^
<SYNTAX>
ESBDESARROLLO:ESBSSCC>do ##class(Sample.Util).SettingsByNameExecute("Port")
```

Besides, we have seen that the following classes and methods could be helpful, but we do not know how to use them:

Ens.Director getProductionItems

```
ClassMethod getProductionItems(tProduction As Ens.Config.Production, pDefinition As %
String, pAutoAdd As %String) As %Status
{
Set $ZT="Trap",tSC=$$$OK
Do {
#; Set definitions for all ConfigItems in the configured Production
For i=1:1:tProduction.Items.Count() {
Set tItem = tProduction.Items.GetAt(i)
Set tConfigName=tItem.Name, tBusinessType=tItem.BusinessType()
#; To understand the following logic, do it the "complement" way, think "set pDefinit
ion" as the reverse of the "continue" command,
#; then the logic becomes:
#; if tItem.Enabled, then set pDefinition
#; if pDefinition is not defined, then set pDefinition
#; if it is auto-add, then set pDefinition.
#; In the end:
#; 1) the last Enabled version with the same ConfigName is the one that gets defined
at runtime
#; 2) any defined item, enabled or disabled, will override the auto-add.
#; this means, if you have a disabled Ens.Alarm, it will override the system default
one and you will end up with a disabled Ens.Alarm.
Continue:'tItem.Enabled&&$data(pDefinition(tConfigName))&&'$G(pAutoAdd(tConfigName))
#; overwrite registration of AutoAdd items with user settings if any
Set pAutoAdd(tConfigName)=0 ; clear this so behavior will be the same for multiply-
defined normal and AutoAdd items Set tSC = tItem.PopulateModifiedSettings()
Kill pDefinition(tConfigName)
Set pDefinition(tConfigName,"IsEnabled")=tItem.Enabled
Set pDefinition(tConfigName,"ClassName")=tItem.ClassName
Set pDefinition(tConfigName,"OnTaskClassName")=tItem.ClassName
Set pDefinition(tConfigName,"QueueName")=tItem.QueueName()
Set pDefinition(tConfigName,"PoolSize")=tItem.PoolSize
Set pDefinition(tConfigName,"Foreground")=tItem.Foreground
Set pDefinition(tConfigName,"DisableErrorTraps")=tItem.DisableErrorTraps
Set pDefinition(tConfigName,"BusinessType")=tBusinessType
Set pDefinition(tConfigName,"InactivityTimeout")=tItem.InactivityTimeout
Set pDefinition(tConfigName,"Checksum")=tItem.Checksum() ; depends on PopulateModifie
dSettings()
Set pDefinition(tConfigName,"%Id")=tItem.%Id()
Set pDefinition(tConfigName,"Schedule")=tItem.Schedule
Set pDefinition(tConfigName,"Register")=1
Set pDefinition(tConfigName,"RunAsJob")=1
Set pDefinition(tConfigName,"Trace")=tItem.LogTraceEvents Set tIndex="" For { Set tIn
dex = tItem.ModifiedSettings.Next(tIndex) Quit:tIndex=""
Set tSetting = tItem.ModifiedSettings.GetAt(tIndex)
Set pDefinition(tConfigName,"Setting",tSetting.Target,tSetting.Name)=tSetting.Value
} If ('tItem.Enabled)||($classmethod(tItem.ClassName,"%GetParameter","INVOCATION")'="
Queue") {
Set pDefinition(tConfigName,"RunAsJob")=0
} Elseif tItem.Schedule'="" {
Set tSC=##class(Ens.ScheduleHandler).ParseScheduleSpec(tItem.Schedule,.tCurrentState)
If $$$ISERR(tSC)||(tCurrentState="DISABLED") {
$$$LOGWARNING("ConfigItem '"_tItem.Name_"' is disabled because its schedule string is
 invalid.")
```

```
Set pDefinition(tConfigName,"IsEnabled")=0
Set pDefinition(tConfigName,"RunAsJob")=0
} Elseif (tCurrentState="STOP") {
Set pDefinition(tConfigName,"RunAsJob")=0
}
}
If tBusinessType=$$$eHostTypeProcess {
Set pDefinition(tConfigName,"OnTaskClassName")="Ens.Actor"
If tItem.PoolSize=0 Set pDefinition(tConfigName,"QueueName")="Ens.Actor"
}
#; No RunAsJob for Services with no Adapter
#; No RunAsJob for items with PoolSize=0
#; RunAsJob=-1 if Adapter.#SINGLEPOOLJOB to run only 1 job regardless of PoolSize, e.
g. for TCP.InboundAdapter
#; Otherwise no change
Set tRunAsJob=pDefinition(tConfigName,"RunAsJob"), pDefinition(tConfigName,"RunAsJob"
)=$S(
  (tBusinessType=$$$eHostTypeService)&&(""=tItem.AdapterClassName()):0
, pDefinition(tConfigName,"PoolSize")=0:0
, tRunAsJob&&(""'=tItem.AdapterClassName())&&$parameter(tItem.AdapterClassName(),"SIN
GLEPOOLJOB"):-1
, 1:tRunAsJob)
}
#; recursively get all the items in subproductions
For i=1:1:tProduction.SubProductions.Count() {
Set tSubProduction = tProduction.SubProductions.GetAt(i)
Set tSC=..getProductionItems(tSubProduction,.pDefinition,.pAutoAdd)
Quit:$$$ISERR(tSC)
}
} While 0
Exit
Quit tSC
Trap
Set $ZT="",tSC=$$$EnsSystemError
Goto Exit
}
```

Ens.Config.Item

```
/// Get the config value of the named setting, return 0 if not defined, 1 if defined.
Method GetSetting(pSettingName As %String, ByRef pValue As %String) As %Boolean
{
Kill pValue
Set tKey="" For { Set tSetting=..Settings.GetNext(.tKey) Quit:""=tKey
If tSetting.Name = pSettingName {
Set pValue = tSetting.Value
Quit
}
}
Quit ''$D(pValue)
}
```

Ens.Production

```
ClassMethod GetSettingValue(pName As %String, Output pStatus As %Status) As %String [
 CodeMode = expression ]
{
##class(Ens.Director).GetProductionSettingValue("",.pName,.pStatus)
} ClassMethod GetSettingsArray(Output pSettings) As %Status [ CodeMode = expression ]
{
##class(Ens.Director).GetProductionSettings("",.pSettings)
}
```

Could you help us, please?

→ We would need documentation or examples

Thanks for your help

#Caché #Ensemble

Source URL:https://community.intersystems.com/post/how-could-we-get-name-port-and-type-all-services-namespace%E2%80%BD