記事
Toshihiko Minamoto · 2021年11月9日 18m read

Open Exchange

別のIRISメッセージビューワを作成する

IRISインターオペラビリティのメッセージビューワで何かを変更できるとしたら、何を変更しますか？

「Dashboard IRIS History Monitor
」の記事を公開したところ、素晴らしいフィードバックやリクエストをいただきました。
中には、メッセージビューワの拡張に関するリクエストがありました。

まだプロジェクトを
確認していない方は、ぜひご覧ください。絶対に見る価値がありますし、2019年の最高のInterSystems Open Exchange開発者およびアプリケーションの1つとしてブロンズ賞を受賞しました。

「新しい」メッセージビューワに含めようと思う機能についてのアイデアを書き留め始めましたが、これらのリソースをどのようにすれば素早く簡単に見せることができるのでしょうか。

まずは、
一般的に、相互運用性の本番環境をセットアップし、ドキュメント
の指示のとおりに、ターゲットシステムにエクスポートしてデプロイすることから始めます。
これは私があまり好まないプロセスです。 特に何か悪いというわけではありませんが、
コードを使ってすべてを行う考えがあるためです。

誰かがこういったプロジェクトを実行するたびに、次のように開始することを期待しています。

```
$ docker-compose build

$ docker-compose up -d
```

いかがでしょうか!!!

たったこれだけのステップを思い浮かべながら、InterSystemsコミュニティを調べ始めると、いくつかのヒントが見つかりました。
ある投稿では、私が自問していた質問が挙げられ
ていました。「ルーチンを使って本番環境を作成するにはどうすればよいのか。」

その投稿の中で、@Eduard Lebedyuk
が、コードを使って本番環境を作成する方法を次のように回答しています。

「本番クラスを自動的に作成するには、次を行う必要があります。

1. テストプロダクション用の%Dictionary.ClassDefinitionオブジェクトを作成します。
2. Ens.Config.Productionオブジェクトを作成します。
3. %Dictionary.XDataDefinitionを作成します。
4. (2) を (3) にシリアル化します。
5. XData (3) を (1) に挿入します。
6. (1) を保存してコンパイルします。」

@Jenny Amesのコメントにも、次のように書かれていました。

「私たちがよくお勧めしているベストプラクティスは、逆方向に構築することです。 ビジネスオペレーションを先に構築してから、ビジネスプロセス、そしてビジネスサービスを構築していく方法です...」

というわけで、早速やってみましょう！

# #

リクエスト、ビジネスオペレーション、およびビジネスサービス

クラス
diashenrique.messageviewer.util.InstallerProduction.cls
は、名前から想像できるように、プロダクションのインストールを担当するクラスです。
インストーラのマニフェストは、そのクラスからClassMethod **Install**を呼び出します。

```
/// ????????????????????????????????

ClassMethod Install() As %Status

{

    Set sc = $$$OK

    Try {

        Set sc = $$$ADDSC(sc,..InstallProduction()) quit:$$$ISERR(sc)

        Set sc = $$$ADDSC(sc,..GenerateMessages()) quit:$$$ISERR(sc)

        Set sc = $$$ADDSC(sc,..GenerateUsingEnsDirector()) quit:$$$ISERR(sc)

    }

    Catch (err) {

        Set sc = $$$ADDSC(sc,err.AsStatus())

    }

    Return sc

}
```

クラスメソッド**InstallProduction**
は、次を作成することで、プロダクションを作成するためのメインの構造をまとめます。

- リクエスト
- ビジネスオペレーション
- ビジネスサービス
- 相互運用性プロダクション

コードを使用して相互運用性プロダクションを作成しようと考えているため、完全なコーディングモードに移行して、リクエスト、ビジネスオペレーション、およびビジネスサービスの全クラスを作成しましょう。
これを行うには、いくつかのInterSystemsライブラリパッケージを広範に使用します。

- %Dictionary.ClassDefinition
- %Dictionary.PropertyDefinition
- %Dictionary.XDataDefinition
- %Dictionary.MethodDefinition
- %Dictionary.ParameterDefinition

クラスメソッド**InstallProduction**は、次のコードを使用して、**Ens.Request**を継承した2つのクラスを作成します。

```
Set sc = $$$ADDSC(sc,..CreateRequest("diashenrique.messageviewer.Message.SimpleReques
t","Message")) quit:$$$ISERR(sc)

Set sc = $$$ADDSC(sc,..CreateRequest("diashenrique.messageviewer.Message.AnotherReque
st","Something")) quit:$$$ISERR(sc)

ClassMethod CreateRequest(classname As %String, prop As %String) As %Status [ Private
 ]

{

    New $Namespace

    Set $Namespace = ..#NAMESPACE

    Set sc = $$$OK

    Try {

        Set class = ##class(%Dictionary.ClassDefinition).%New(classname)

        Set class.GeneratedBy = $ClassName()

        Set class.Super = "Ens.Request"

        Set class.ProcedureBlock = 1

        Set class.Inheritance = "left"

        Set sc = $$$ADDSC(sc,class.%Save())

        #; create adapter

        Set property = ##class(%Dictionary.PropertyDefinition).%New(classname)

        Set property.Name = prop

        Set property.Type = "%String"

        Set sc = $$$ADDSC(sc,property.%Save())

        Set sc = $$$ADDSC(sc,$System.OBJ.Compile(classname,"fck-dv"))

    }

    Catch (err) {

        Set sc = $$$ADDSC(sc,err.AsStatus())

    }
```

```
    Return sc

}
```

では、**Ens.BusinessOperation**を継承したビジネスオペレーションのクラスを作成しましょう。

```
Set sc = $$$ADDSC(sc,..CreateOperation()) quit:$$$ISERR(sc)
```

このクラスを作成するほかに、MessageMapとメソッドConsumeを作成します。

```
ClassMethod CreateOperation() As %Status [ Private ]
{
    New $Namespace
    Set $Namespace = ..#NAMESPACE
    Set sc = $$$OK
    Try {
        Set classname = "diashenrique.messageviewer.Operation.Consumer"
        Set class = ##class(%Dictionary.ClassDefinition).%New(classname)
        Set class.GeneratedBy = $ClassName()
        Set class.Super = "Ens.BusinessOperation"
        Set class.ProcedureBlock = 1
        Set class.Inheritance = "left"

        Set xdata = ##class(%Dictionary.XDataDefinition).%New()
        Set xdata.Name = "MessageMap"
        Set xdata.XMLNamespace = "http://www.intersystems.com/urlmap"
        Do xdata.Data.WriteLine("<MapItems>")
        Do xdata.Data.WriteLine("<MapItem MessageType=""diashenrique.messageviewer.Me
ssage.SimpleRequest"">")
        Do xdata.Data.WriteLine("<Method>Consume</Method>")
        Do xdata.Data.WriteLine("</MapItem>")
        Do xdata.Data.WriteLine("<MapItem MessageType=""diashenrique.messageviewer.Me
ssage.AnotherRequest"">")
        Do xdata.Data.WriteLine("<Method>Consume</Method>")
        Do xdata.Data.WriteLine("</MapItem>")
        Do xdata.Data.WriteLine("</MapItems>")
        Do class.XDatas.Insert(xdata)
        Set sc = $$$ADDSC(sc,class.%Save())

        Set method = ##class(%Dictionary.MethodDefinition).%New(classname)
        Set method.Name = "Consume"
        Set method.ClassMethod = 0
        Set method.ReturnType = "%Status"
        Set method.FormalSpec = "input:diashenrique.messageviewer.Message.SimpleReque
st,&output:Ens.Response"
        Set stream = ##class(%Stream.TmpCharacter).%New()
        Do stream.WriteLine("   set sc = $$$OK")
        Do stream.WriteLine("   $$$TRACE(input.Message)")
        Do stream.WriteLine("   return sc")
        Set method.Implementation = stream
        Set sc = $$$ADDSC(sc,method.%Save())

        Set sc = $$$ADDSC(sc,$System.OBJ.Compile(classname,"fck-dv"))
    }
```

```
    Catch (err) {
        Set sc = $$$ADDSC(sc,err.AsStatus())
    }
    Return sc
}
```

相互運用性プロダクションを作成する直前のステップでは、ビジネスサービスクラスを作成しましょう。

```
Set sc = $$$ADDSC(sc,..CreateRESTService()) quit:$$$ISERR(sc)
```

このクラスにはHttpリクエストを受信するためのUrlMapとRoutesがあります。

```
ClassMethod CreateRESTService() As %Status [ Private ]
{
    New $Namespace
    Set $Namespace = ..#NAMESPACE
    Set sc = $$$OK
    Try {
        Set classname = "diashenrique.messageviewer.Service.REST"
        Set class = ##class(%Dictionary.ClassDefinition).%New(classname)
        Set class.GeneratedBy = $ClassName()
        Set class.Super = "EnsLib.REST.Service, Ens.BusinessService"
        Set class.ProcedureBlock = 1
        Set class.Inheritance = "left"

        Set xdata = ##class(%Dictionary.XDataDefinition).%New()
        Set xdata.Name = "UrlMap"
        Set xdata.XMLNamespace = "http://www.intersystems.com/urlmap"
        Do xdata.Data.WriteLine("<Routes>")
        Do xdata.Data.WriteLine("<Route Url=""/send/message"" Method=""POST"" Call=""
SendMessage""/>")
        Do xdata.Data.WriteLine("<Route Url=""/send/something"" Method=""POST"" Call=
""SendSomething""/>")
        Do xdata.Data.WriteLine("</Routes>")
        Do class.XDatas.Insert(xdata)
        Set sc = $$$ADDSC(sc,class.%Save())

        #; create adapter
        Set adapter = ##class(%Dictionary.ParameterDefinition).%New(classname)
        Set class.GeneratedBy = $ClassName()
        Set adapter.Name = "ADAPTER"
        Set adapter.SequenceNumber = 1
        Set adapter.Default = "EnsLib.HTTP.InboundAdapter"
        Set sc = $$$ADDSC(sc,adapter.%Save())

        #; add prefix
        Set prefix = ##class(%Dictionary.ParameterDefinition).%New(classname)
        Set prefix.Name = "EnsServicePrefix"
        Set prefix.SequenceNumber = 2
        Set prefix.Default = "|demoiris"
        Set sc = $$$ADDSC(sc,prefix.%Save())

        Set method = ##class(%Dictionary.MethodDefinition).%New(classname)
        Set method.Name = "SendMessage"
        Set method.ClassMethod = 0
```

```
        Set method.ReturnType = "%Status"
        Set method.FormalSpec = "input:%Library.AbstractStream,&output:%Stream.Object
"
        Set stream = ##class(%Stream.TmpCharacter).%New()
        Do stream.WriteLine("   set sc = $$$OK")
        Do stream.WriteLine("   set request = ##class(diashenrique.messageviewer.Mess
age.SimpleRequest).%New()")
        Do stream.WriteLine("   set data = {}.%FromJSON(input)")
        Do stream.WriteLine("   set request.Message = data.Message")
        Do stream.WriteLine("   set sc = $$$ADDSC(sc,..SendRequestSync(""diashenrique
.messageviewer.Operation.Consumer"",request,.response))")
        Do stream.WriteLine("   return sc")
        Set method.Implementation = stream
        Set sc = $$$ADDSC(sc,method.%Save())

        Set method = ##class(%Dictionary.MethodDefinition).%New(classname)
        Set method.Name = "SendSomething"
        Set method.ClassMethod = 0
        Set method.ReturnType = "%Status"
        Set method.FormalSpec = "input:%Library.AbstractStream,&output:%Stream.Object
"
        Set stream = ##class(%Stream.TmpCharacter).%New()
        Do stream.WriteLine("   set sc = $$$OK")
        Do stream.WriteLine("   set request = ##class(diashenrique.messageviewer.Mess
age.AnotherRequest).%New()")
        Do stream.WriteLine("   set data = {}.%FromJSON(input)")
        Do stream.WriteLine("   set request.Something = data.Something")
        Do stream.WriteLine("   set sc = $$$ADDSC(sc,..SendRequestSync(""diashenrique
.messageviewer.Operation.Consumer"",request,.response))")
        Do stream.WriteLine("   return sc")
        Set method.Implementation = stream
        Set sc = $$$ADDSC(sc,method.%Save())

        Set sc = $$$ADDSC(sc,$System.OBJ.Compile(classname,"fck-dv"))
    }
    Catch (err) {
        Set sc = $$$ADDSC(sc,err.AsStatus())
    }
    Return sc
}
```

Visual Studioコードの使用

%Dictionaryパッケージを使用してクラスを作成するのは困難な場合があり、読みにくくもありますが、非常に便利です。 コードの可読性を良くしてアプローチをもう少しわかりやすくするために、Visual Studioコードを使用して新しいリクエスト、ビジネスサービス、およびビジネスオペレーションクラスを作成することにします。

- diashenrique.messageviewer.Message.SimpleMessage.cls
- diashenrique.messageviewer.Operation.ConsumeMessageClass.cls
- diashenrique.messageviewer.Service.SendMessage.cls

```
    Class diashenrique.messageviewer.Message.SimpleMessage Extends Ens.Request [ Inhe
ritance = left, ProcedureBlock ]
    {
     Property ClassMessage As %String;
```

```
    }

    Class diashenrique.messageviewer.Operation.ConsumeMessageClass Extends Ens.Busine
ssOperation [ Inheritance = left, ProcedureBlock ]
    {
     Method Consume(input As diashenrique.messageviewer.Message.SimpleMessage, ByRef
output As Ens.Response) As %Status
    {
        Set sc = $$$OK
        $$$TRACE(pRequest.ClassMessage)
        Return sc
    }
    XData MessageMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
    {


           Consume


     }
    }

    Class diashenrique.messageviewer.Service.SendMessage Extends Ens.BusinessService
[ ProcedureBlock ]
    {
    Method OnProcessInput(input As %Library.AbstractStream, ByRef output As %Stream.O
bject) As %Status
    {
        Set tSC = $$$OK
        // ????????????
        Set request = ##class(diashenrique.messageviewer.Message.SimpleMessage).%New(
)
        // ??????????????????????
        Set request.ClassMessage = input
        // ????????????????????????????????????????
        Set tSC = ..SendRequestSync("diashenrique.messageviewer.Operation.ConsumeMess
ageClass",request,.output)
        Quit tSC
    }
    }
```

コードの可読性の観点では、大きな差があります！

相互運用性プロダクションの作成

相互運用性プロダクションを仕上げましょう。 これを行うには、プロダクションクラスを作成してから、それをビジネスオペレーションとサービスクラスに関連付けます。

```
Set sc = $$$ADDSC(sc,..CreateProduction()) quit:$$$ISERR(sc)

ClassMethod CreateProduction(purge As %Boolean = 0) As %Status [ Private ]
{
    New $Namespace
    Set $Namespace = ..#NAMESPACE
    Set sc = $$$OK
    Try {
```

```
         #; create new production
        Set class = ##class(%Dictionary.ClassDefinition).%New(..#PRODUCTION)
        Set class.ProcedureBlock = 1
        Set class.Super = "Ens.Production"
        Set class.GeneratedBy = $ClassName()
        Set xdata = ##class(%Dictionary.XDataDefinition).%New()
        Set xdata.Name = "ProductionDefinition"
        Do xdata.Data.Write("<Production Name=""_..#PRODUCTION_""" LogGeneralTraceEv
ents=""true""></Production>")
        Do class.XDatas.Insert(xdata)
        Set sc = $$$ADDSC(sc,class.%Save())
        Set sc = $$$ADDSC(sc,$System.OBJ.Compile(..#PRODUCTION,"fck-dv"))
        Set production = ##class(Ens.Config.Production).%OpenId(..#PRODUCTION)
        Set item = ##class(Ens.Config.Item).%New()
        Set item.ClassName = "diashenrique.messageviewer.Service.REST"
        Do production.Items.Insert(item)
        Set sc = $$$ADDSC(sc,production.%Save())
        Set item = ##class(Ens.Config.Item).%New()
        Set item.ClassName = "diashenrique.messageviewer.Operation.Consumer"
        Do production.Items.Insert(item)
        Set sc = $$$ADDSC(sc,production.%Save())
        Set item = ##class(Ens.Config.Item).%New()
        Set item.ClassName = "diashenrique.messageviewer.Service.SendMessage"
        Do production.Items.Insert(item)
        Set sc = $$$ADDSC(sc,production.%Save())
        Set item = ##class(Ens.Config.Item).%New()
        Set item.ClassName = "diashenrique.messageviewer.Operation.ConsumeMessageClas
s"
        Do production.Items.Insert(item)
        Set sc = $$$ADDSC(sc,production.%Save())
    }
    Catch (err) {
        Set sc = $$$ADDSC(sc,err.AsStatus())
    }
    Return sc
}
```

プロダクションク
ラスをビジネスオペレーションとサービスクラスに関連付けるために、クラス**Ens.Config.Item**を使用します。
これは、クラスの作成に%Dictionaryパッケージを使用したのか、VS
Code、Studio、またはAtelierを使用したかに関係なく使用できます。

いずれにしても、達成できました！ コードを使用して相互運用性プロダクションを作成できました。

ただし、このコードの元の目的を忘れてはいけません。拡張メッセージビューワの機能を示すプロダクションとメッセージを作成するという目的です。
以降のクラスメソッドを使用して、両方のビジネスサービスを実行し、メッセージを生成します。

%Net.HttpRequestを使用したメッセージの生成:

```
ClassMethod GenerateMessages() As %Status [ Private ]
{
    New $Namespace
    Set $Namespace = ..#NAMESPACE
    Set sc = $$$OK
    Try {
        Set action(0) = "/demoiris/send/message"
```

```
        Set action(1) = "/demoiris/send/something"
        For i=1:1:..#LIMIT {
            Set content = { }
            Set content.Message = "Hi, I'm just a random message named "_$Random(3000
0)
            Set content.Something = "Hi, I'm just a random something named "_$Random(
30000)
            Set httprequest = ##class(%Net.HttpRequest).%New()
            Set httprequest.SSLCheckServerIdentity = 0
            Set httprequest.SSLConfiguration = ""
            Set httprequest.Https = 0
            Set httprequest.Server = "localhost"
            Set httprequest.Port = 9980
            Set serverUrl = action($Random(2))
            Do httprequest.EntityBody.Write(content.%ToJSON())
            Set sc = httprequest.Post(serverUrl)
            Quit:$$$ISERR(sc)
        }
    }
    Catch (err) {
        Set sc = $$$ADDSC(sc,err.AsStatus())
    }
    Return sc
}
```

EnsDirectorを使用したメッセージの生成:

```
ClassMethod GenerateUsingEnsDirector() As %Status [ Private ]
{
    New $Namespace
    Set $Namespace = ..#NAMESPACE
    Set sc = $$$OK
    Try {
        For i=1:1:..#LIMIT {
            Set tSC = ##class(Ens.Director).CreateBusinessService("diashenrique.messa
geviewer.Service.SendMessage",.tService)
            Set message = "Message Generated By CreateBusinessService "_$Random(1000)
            Set tSC = tService.ProcessInput(message,.output)
            Quit:$$$ISERR(sc)
        }
    }
    Catch (err) {
        Set sc = $$$ADDSC(sc,err.AsStatus())
    }
    Return sc
}

}
```

コードは以上です。
完全なプロジェクトは、https://github.com/diashenrique/iris-message-viewerをご覧ください。

プロジェクトの実行

では、プロジェクトの実際の動作を確認しましょう。 まず、git cloneまたはgit pullで、任意のローカルディレクトリにリポジトリを作成します。

git clone https://github.com/diashenrique/iris-message-viewer.git

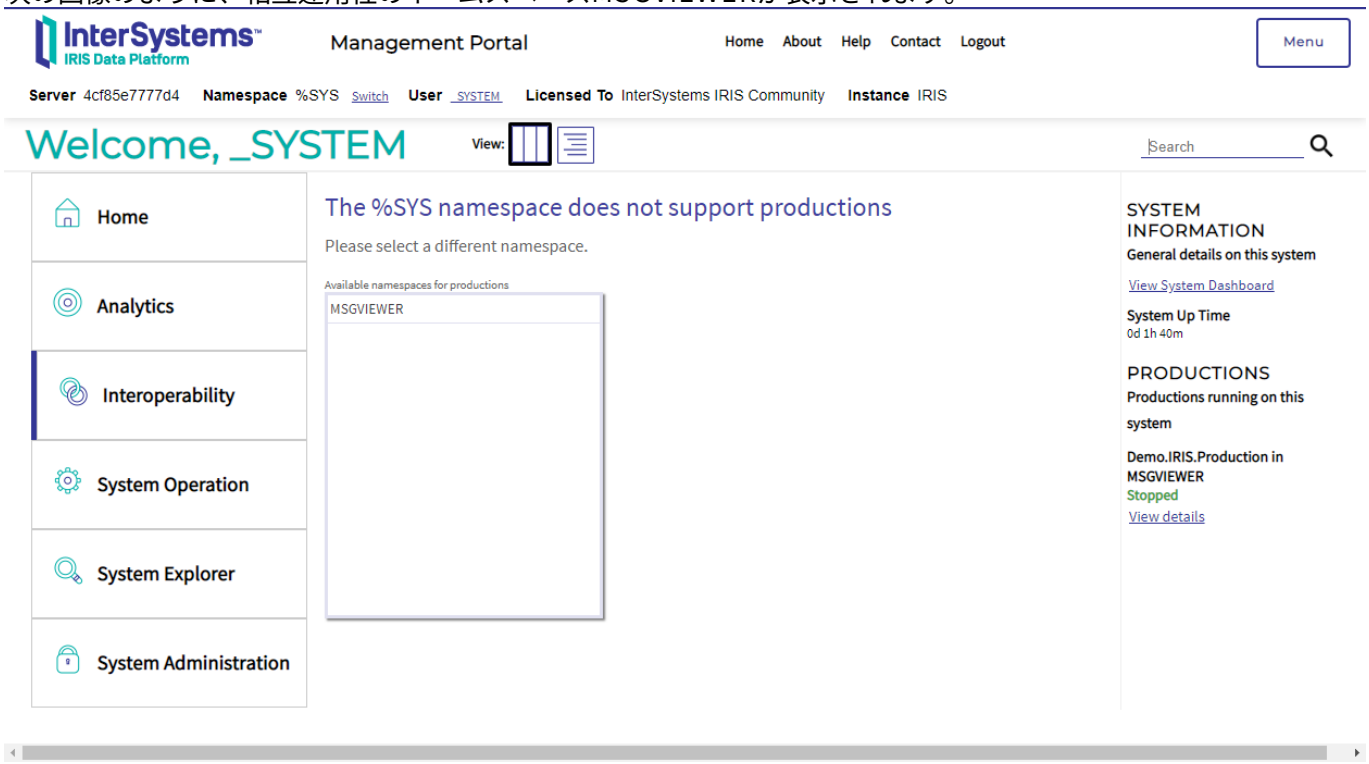
次に、このディレクトリでターミナルを開き、次を実行します。

docker-compose build

最後に、プロジェクトでIRISコンテナを実行します。

docker-compose up -d

さらに、http://localhost:52773/csp/sys/UtilHome.cspを使用して管理ポータルにアクセスします。
次の画像のように、相互運用性のネームスペースMSGVIEWERが表示されます。



そしてこれが、私たちの愛らしいプロダクションです。2つのビジネスサービスと2つのビジネスオペレーションがあります。

非常にたくさんのメッセージがあります。

カスタムメッセージビューワですべてが稼働しているので、その機能を見てみましょう。

拡張メッセージビューワ

相互運用性プロダクションに有効になっているネームスペースのみが表示されることに注意してください。

http://localhost:52773/csp/msgviewer/messageviewer.csp

拡張メッセージビューワには、さまざまなフィルタの作成、nレベルへの列のグループ化、Excelへのエクスポートなどを行える機能と柔軟性が備わっています。

さまざまなフィルタを使用して、必要な結果を得ることができます。
また、Shiftキーを押しながら列のヘッダーをクリックすると、複数の並べ替えを使用することも可能です。

データグリッドをExcelにエクスポートすることもできるのです！

さらに、フィルタビルダーオプションを使用して、複雑なフィルタを作成することができます。

使用できる任意の列に対してデータをグループ化し、必要なnレベルを使用して情報をまとめることができます。
デフォルトでは、このグループはDate Created（作成日）フィールドを使用して作成されます。

また、列を選択できる機能があります。 次のページには、Ens.MessageHeaderのすべての列があります。デフォルトの列のみが初期ビューに表示されていますが、 「Column
Chooser」（列選択）ボタンを使って、ほかの列を選択することができます。

すべてのグループはワンクリックで折りたたみと展開が可能です。

SessionId（セッションID）フィールドの情報には、ビジュアルトレース機能へのリンクがあります。

必要に応じて、メッセージを再送することができます。 必要なメッセージを選択し、Resend
をクリックするだけで再送信は完了です。 この機能には、次のクラスメソッドが使用されています。

##class(Ens.MessageHeader).ResendDuplicatedMessage(id)

最後に、前述のように、データグリッドをExcelにエクスポートすることができます。

Excelの結果には、キャッシュサーバーページ（CSP）に定義されているものと同じフォーマット、コンテンツ、およびグループが表示されます。

追伸: この問題への取り組みで大いに助けてくれた@Renan.Lourencoに、特に深くお礼申し上げます。

#Docker #ObjectScript #InterSystems Package Manager (IPM) #ビジネスオペレーション #ビジネスサービス #メッセージ検索 #相互運用性 #Ensemble #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/%E5%88%A5%E3%81%AEiris%E3%83%A1%E3%83%83%E3%82%BB%E3%83%BC%E3%82%B8%E3%83%93%E3%83%A5%E3%83%BC%E3%83%AF%E3%82%92%E4%BD%9C%E6%88%90%E3%81%99%E3%82%8B