AppVision

Manuel de programmation

12/01/2016

Rev 1

AppVision is a registered brand to **PRYSM**.

Windows is a registered brand to **Microsoft Corporation**.

Made in France.

CONTENU

INTRODUCTION

Bienvenue dans l'interface de programmation **AppVision**, logiciel de supervision multi protocole.

Ce document est destiné aux utilisateurs qui ont en charge la mise en place de script dans le logiciel **AppVision**.

La programmation du superviseur est utilisée dans les contextes suivants :

- Les scripts,
- L'animation/commande des synoptiques et des symboles,

Le langage C# est utilisé pour programmer les scripts.

LES OPERANDES

Les informations du superviseur (variables, groupes, régions, utilisateurs, …) sont accessibles à travers des opérandes. Un opérande est une chaîne de caractères permettant d'identifier de façon unique une information du superviseur, par exemple :

$V.A.Z001#Ack : désigne la propriété « attente d'acquittement » de la variable « A.Z001 »,

Le nom d'un opérande commence toujours par le caractère $.

Liste des opérandes :

Opérande	Description
$V.{nom}#{propriété}	Propriété d'une variable
$V.{nom}[{numéro}]	Elément d'une variable buffer
$G.{nom}#{propriété}	Propriété d'un groupe de variables
$A.{nom}#{propriété]	Propriété d'une région
$S#{propriété}	Propriété du serveur
$U#{propriété}	Propriété de l'utilisateur connecté

LES OPERANDES RELATIFS AUX VARIABLES

Opérande	Type	Description
$V.{nom}	Texte/Numérique	Valeur de la variable
$V.{nom}#Ack	Numérique	Attente d'acquittement si > 0
$V.{nom}#Alarm	Numérique	Etat d'alarme si > 0
$V.{nom}#AreaDesc	Texte	Description de la région
$V.{nom}#AreaName	Texte	Nom de la région
$V.{nom}#Date	Date	Date du dernier changement d'état
$V.{nom}#Desc	Texte	Description de la variable
$V.{nom}#Id	Numérique	Id de la variable
$V.{nom}#Info	Texte	Information complémentaire
$V.{nom}#Instructions	Texte	Consignes
$V.{nom}#IsAlarmMasked	0/1	Etat de masquage
$V.{nom}#IsLocked	0/1	Etat de forçage
$V.{nom}#MaxSeverity	Numérique	Etat de gravité
$V.{nom}#Name	Texte	Nom de la variable
$V.{nom}#PreviousDate	Date	Date du l'avant dernier changement d'état
$V.{nom}#PreviousValue	Texte/Numérique	Avant dernier changement d'état
$V.{nom}#Quality	0-255	Qualité
$V.{nom}#Report	0/1	Attente de rapport
$V.{nom}#Sound	Texte	Son
$V.{nom}#State	Texte	Etat de la variable
$V.{nom}#Symbol	Texte	Symbole
$V.{nom}#Synoptic	Texte	Synoptique

$V.{nom}#Parameters.{pnom}	Texte	Paramètre pnom
$V.{nom}#VideoLink	Texte	Liaison vidéo
$V.{nom}#CustomField1	Texte	Champ supplémentaire 1
…		
$V.{nom}#CustomField5	Texte	Champ supplémentaire 5

Pour les variables buffer, il est possible d'accéder à ses éléments en lecture/écriture avec la syntaxe suivante :

$V.{nom_variable_buffer}[{numéro}]

par exemple :
$V.Moteur.Total = $V.Moteur.BuffEtat[0] + $V.Moteur.BuffEtat[1] ;
$V.Moteur.BuffCmd[0] = 2;

LES OPERANDES RELATIFS AUX REGIONS

Opérande	Type	Description
$A.{nom}#OrValue	Numérique	OR des valeurs des variables
$A.{nom}#AndValue	Numérique	AND des valeurs des variables
$A.{nom}#Alarm	0/1	Etat d'alarme
$A.{nom}#Ack	0/1	Attente d'acquittement
$A.{nom}#Report	0/1	Attente de rapport
$A.{nom}#Desc	Texte	Description
$A.{nom}#Name	Texte	Nom
$A.{nom}#Id	Numérique	Id
$A.{nom}#AlarmCount	Numérique	Compteur d'alarme
$A.{nom}#AlarmEndCount	Numérique	Compteur d'alarme en attente de fin
$A.{nom}#AckCount	Numérique	Compteur d'attente d'acquittement
$A.{nom}#ReportCount	Numérique	Compteur d'attente de rapport
$A.{nom}#LockedCount	Numérique	Compteur d'attente de forçage
$A.{nom}#MaskedCount	Numérique	Compteur d'attente de masquage
$A.{nom}#MaxSeverity	Numérique	Etat de gravité
$A.{nom}#Parameters.{pnom}	Texte	Paramètre pnom
$A.{nom}#CustomField1	Texte	
…		
$A.{nom}#CustomField5	Texte	

LES OPERANDES RELATIFS AUX GROUPES

Opérande	Type	Description
$G.{nom}#OrValue	Numérique	OR des valeurs des variables
$G.{nom}#AndValue	Numérique	AND des valeurs des variables
$G.{nom}#Alarm	0/1	Etat d'alarme

$G.{nom}#Ack	0/1	Attente d'acquittement
$G.{nom}#Report	0/1	Attente de rapport
$G.{nom}#Desc	Texte	Description
$G.{nom}#Name	Texte	Nom
$G.{nom}#Id	Numérique	Id
$G.{nom}#AlarmCount	Numérique	Compteur d'alarme
$G.{nom}#AlarmEndCount	Numérique	Compteur d'alarme en attente de fin
$G.{nom}#AckCount	Numérique	Compteur d'attente d'acquittement
$G.{nom}#ReportCount	Numérique	Compteur d'attente de rapport
$G.{nom}#LockedCount	Numérique	Compteur d'attente de forçage
$G.{nom}#MaskedCount	Numérique	Compteur d'attente de masquage
$G.{nom}#MaxSeverity	Numérique	Etat de gravité
$G.{nom}#Parameters.{pnom}	Texte	Paramètre pnom
$G.{nom}#CustomField1	Texte	
…		
$G.{nom}#CustomField5	Texte	

LES OPERANDES RELATIFS AU SERVER

Opérande	Type	Description
$S#Debug	Numérique	Niveau de debug
$S#Version	Texte	Version du serveur
$S#Project	Texte	Description du projet

LES OPERANDES RELATIFS A L'UTILISATEUR CONNECTE

ATTENTION : ces opérandes ne pourront pas être utilisés dans un contexte de script serveur.

Opérande	Type	Description
$U#Name	Texte	Login
$U#Fullname	Texte	Nom – prénom
$U#Areas.{nom}	0/1	Filtre région 'nom'
$U#Groups.{nom}	0/1	Filtre groupe 'nom'
$U#Rights.{nom}	0/1	Droit utilisateur cf. liste des droits
$U#Options.{nom}	Texte	Option utilisateur cf. liste des options
$U#Parameters.{nom}	Texte	Paramètre 'nom'

Liste des droits utilisateur :

Droit utilisateur	Type	Description
Server.AcknowledgeAlarm	0/1	

AcknowledgeAlarm.All	0/1	
.ReportAlarm	0/1	
.ReportAlarm.Update	0/1	
.MaskAlarm	0/1	
.SetVariable	0/1	
.LockVariable	0/1	
.ChangePassword	0/1	
.AddUserMessage	0/1	
.ProtocolDownload	0/1	
.ProtocolRefresh	0/1	
.Maintenance	0/1	
Client.Print	0/1	
.Export	0/1	
.Quit	0/1	
.UnlockForm	0/1	
.LoadEnv	0/1	
.SaveEnv	0/1	
.ChangeEnv	0/1	
Configure.Synoptic	0/1	
.Variable	0/1	
.Timetable	0/1	
.Holiday	0/1	
.User	0/1	
.Profile	0/1	
.Script	0/1	
.Instruction	0/1	
.Area	0/1	
.Group	0/1	
.Protocol	0/1	
.Mailing	0/1	
.Option	0/1	
.Tool	0/1	
Add.Timetable	0/1	
Display.Synoptic	0/1	
.AlarmsList	0/1	
.StatesList	0/1	
.EventsList	0/1	
.VariablesLockedMasqued	0/1	
.EventsHistory	0/1	
.AlarmsHistory	0/1	
.Graphs	0/1	
.Statistics	0/1	
.Timetable	0/1	
.UserMessage	0/1	
.Left.Area	0/1	

.Left.Synoptic	0/1	
.Left.Layout	0/1	
.Left.Standard	0/1	
Custom.B1	0/1	
…	0/1	
Custom.B8	0/1	

Liste des options utilisateur :

Option utilisateur	Type	Description
DisplayFrontOnAlarm	0/1	
PriorityLevel	Numérique	
TempoDisconnect	Numérique	
WebHomePage	Texte	
SoundOnAlarm	0/1/2	
DisplayInstructionOnAlarm	0/1/2	
CommuteSynoptic	0/1/2	
DisplayDateTimeUtc	0/1	

Le profil utilisateur :

Opérande	Type	Description
$U#Profile.Name	Texte	
$U#Profile.Description	Texte	

LES CLASSES ET LES INTERFACES DE PROGRAMMATION

Le module de programmation du superviseur propose un ensemble de classes et d'interfaces :

Interfaces:

- IVariableManager
- IAreaManager
- IGroupManager
- …

Classes:

- VariableRow
- VariableState
- AreaRow
- AreaState
- …

L'ensemble des classes et interfaces utilisées pour la programmation du superviseur est documentée dans le fichier ***AppVisionAPI.chm*** situé dans le répertoire *Documentations*.

LES SCRIPTS SERVEUR

Les scripts serveur sont exécutés par le processus serveur du superviseur.

Les scripts serveur sont visibles dans la page *Scripts* du configurateur.

Lors de la création d'un nouveau script serveur, automatiquement une classe est créée héritant de la classe *ScriptServerBase* :

```
public class ScriptServerScript1 : Prysm.AppVision.Script.ScriptServerBase
{
    public override void OnStart() { }
    public override void OnStop() { }
    public override void OnIdle() { }
    public override void OnAreaStateChanged(AreaState obj) { }
    public override void OnGroupStateChanged(GroupState obj) { }
    public override void OnVariableStateChanged(VariableState obj) { }
    public override void OnProtocolStateChanged(ProtocolState obj) { }
    public override void OnAlarmChanged(AlarmInfo al) { }
}
```

Cette classe est créée avec l'implémentation des méthodes virtuelles de *ScriptServerBase* :

- OnStart : exécuté lors de l'activation de la supervision
- OnStop : exécuté lors de la désactivation de la supervision
- OnIdle : exécuté en permanence

- OnAreaStateChanged : exécuté à chaque changement d'état de région
- OnGroupStateChanged : exécuté à chaque changement d'état de groupe
- OnVariableStateChanged : exécuté à chaque changement d'état de variable
- OnProtocolStateChanged : exécuté à chaque changement d'état de protocole
- OnAlarmChanged : exécuté à chaque changement d'état d'alarme

La classe *ScriptServerBase* possède une propriété *Server* qui permet d'interagir avec le serveur :

```
IServer Server ;
```

L'interface *IServer* est documentée dans le fichier ***AppVisionAPI.chm.***

LES REFERENCES

Il est possible de rajouter des références sur des librairies .NET externes, elles seront listées sous le nœud *Références*.

LA GENERATION

La phase de génération permet de compiler l'ensemble des fichiers sources constituant le script serveur.

Les erreurs de compilation sont affichées lors de cette phase.

LES SCRIPTS CLIENT

Les scripts client sont exécutés par le processus client du superviseur.

Les scripts client sont visibles dans l'onglet *Scripts* de la page *Synoptiques* du configurateur.

Lors de la création d'un nouveau script client, automatiquement une classe est créée héritant de la classe *ScriptClientBase* :

```
public class Script3 : Prysm.AppVision.Script.ScriptClientBase
{
    public override void OnOpen() { }
    public override void OnClose() { }
    public override void OnLogin() { }
    public override void OnLogout() { }
    public override void OnVariableStateChanged(VariableState st) { }
    public override void OnAreaStateChanged(AreaState st) { }
    public override void OnGroupStateChanged(GroupState st) { }
    public override void OnProtocolStateChanged(ProtocolState st) { }
    public override void OnAlarmChanged(AlarmInfo al) { }
    public override void OnEventChanged(EventRow ev) { }
  public override void OnKeyDown(Key key) { }
}
```

Cette classe est créée avec l'implémentation des méthodes virtuelles de *ScriptClientBase* :

- OnOpen : exécuté lors du démarrage du client
- OnClose : exécuté lors de l'arrêt du client

- OnLogin : exécuté lors de la connexion d'un utilisateur
- OnLogout : exécuté lors de la déconnexion d'un utilisateur
- OnAreaStateChanged : exécuté à chaque changement d'état de région
- OnGroupStateChanged : exécuté à chaque changement d'état de groupe
- OnVariableStateChanged : exécuté à chaque changement d'état de variable
- OnProtocolStateChanged : exécuté à chaque changement d'état de protocole
- OnAlarmChanged : exécuté à chaque changement d'état d'alarme
- OnEventChanged : exécuté lors d'un nouvel événement de la supervision
- OnKeyDown : exécuté lors de l'appui d'une touche sur le clavier

La classe *ScriptClientBase* possède 2 propriétés *Client* et *Server* qui permettent d'interagir avec le client et le serveur :

```
IClient Client ;
IServer Server ;
```

Les interfaces *IClient* et *IServer* sont documentées dans le fichier ***AppVisionAPI.chm.***

LES REFERENCES

Il est possible de rajouter des références sur des librairies .NET externes, elles seront listées dans l'onglet *Références*.

LA GENERATION

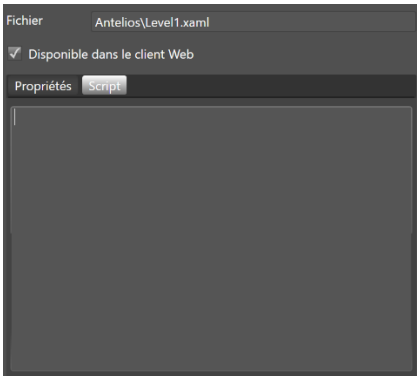
La phase de génération permet de compiler l'ensemble des fichiers sources constituant le script client ainsi que les synoptiques.

Les erreurs de compilation sont affichées lors de cette phase.

LES SCRIPTS SYNOPTIQUE ET SYMBOLE

Les scripts synoptique et symbole sont exécutés par le processus client du superviseur.

Les scripts synoptique et symbole sont visibles dans les propriétés du synoptique ou du symbole dans l'onglet *Script* :



Il est possible d'implémenter les méthodes virtuelles suivantes :

```
public virtual void OnLoaded() { }
public virtual void OnUnloaded() { }
public virtual void OnVariableStateChanged(VariableState state) { }
public virtual void OnAlarmChanged(AlarmInfo alarm) { }
```

- OnLoaded : exécuté lors du chargement du synoptique ou du symbole
- OnUnloaded : exécuté lors du déchargement du synoptique ou du symbole
- OnVariableStateChanged : exécuté à chaque changement d'état de variable
- OnAlarmChanged : exécuté à chaque changement d'état d'alarme

Par exemple :

```
public override void OnLoaded()
{
	this.Visibility=Visibility.Collapsed;
}
public override void OnVariableStateChanged(VariableState st)
{
	if( ("$V."+st.Name)=="%1")
	{
		…
	}
}
```

L'OUVERTURE ET LE CHARGEMENT DE DOCUMENT

CHARGEMENT D'UN DOCUMENT

object Client.Load(string document, params string[] args)

object Client.Load(int numWindow, string document)

Charge un document dans une fenêtre ou dans la fenêtre active si numWindow n'est pas fourni.

document : nom du document

args : paramètres du document

Exemple :

```
Client.Load("Hall.xaml")          : charge un synoptique dans la fenêtre active.
Client.Load(2,"Hall.xaml")        : charge un synoptique dans une fenêtre.
```

Liste des documents :

Nom de document	Description
@HomePage	Page d'accueil
@AlarmsList	Liste des alarmes en cours
@EventsList	Liste des événements en temps réel
@AlarmsHistory	Historique des alarmes
@EventsHistory	Historique des événements
@Timetables	Les programmes horaires
@VariableStates	Table d'état des variables
@Profiles	Gestion des utilisateurs
@UserMessages	Les messages utilisateur
@LockedMaskedVariables	Les variables forcées ou masquées
*.xaml	Affiche un synoptique
http*	Affiche une page web
{nom_du_type}	Affiche un contrôle externe

Les paramètres de @AlarmsList :

Paramètre	Description
$A.{nom}	Filtre sur une région
$G.{nom}	Filtre sur un groupe
Severity>{0-100}	Filtre gravité
Severity>={0-100}	Filtre gravité
Severity<{0-100}	Filtre gravité
Severity<={0-100}	Filtre gravité
WaitAck={true/false}	Filtre attente d'acquittement
WaitReport={true/false}	Filtre attente de rapport
WaitEnd={true/false}	Filtre attente de fin

Exemple :

```
Client.Load("@AlarmsList", "$A.Site1",
      "Severity>=50", "Severity<80", "WaitAck=false");
```

Les paramètres de @EventsList :

Paramètre	Description
$A.{nom}	Filtre sur une région
$G.{nom}	Filtre sur un groupe

Exemple :

```
Client.Load("@EventsList", "$A.Site1");
```

Les paramètres de @AlarmsHistory :

Paramètre	Description
$A.{nom}	Filtre sur une région
$G.{nom}	Filtre sur un groupe
Severity>{0-100}	Filtre gravité
Severity>={0-100}	Filtre gravité
Severity<{0-100}	Filtre gravité
Severity<={0-100}	Filtre gravité
DateStart={date}	Date début de recherche dans l'historique
DateEnd={date}	Date fin de recherche dans l'historique

Exemple :

```
Client.Load("@AlarmsHistory", "$A.Site1",
      "DateStart=20/02/2015", "DateEnd=21/02/2015");

Client.Load("@AlarmsHistory", "$A.Site1",
      "DateStart=20/02/2015 08:00", "DateEnd=20/02/2015 10:00");
```

Les paramètres de @EventsHistory :

Paramètre	Description
$A.{nom}	Filtre sur une région
$G.{nom}	Filtre sur un groupe
DateStart={date}	Date début de recherche dans l'historique
DateEnd={date}	Date fin de recherche dans l'historique

Exemple :

```
Client.Load("@EventsHistory", "$A.Site1",
      "DateStart=20/02/2015", "DateEnd=21/02/2015");

Client.Load("@EventsHistory", "$A.Site1",
      "DateStart=20/02/2015 08:00", "DateEnd=20/02/2015 10:00");
```

Les paramètres de @Timetables :

Paramètre	Description
$V.{nom}	Filtre sur un seul programme horaire
$G.{nom}	Filtre sur un groupe

Exemple :

```
Client.Load("@Timetables", "$V.TT1);
```

Les paramètres de @VariableStates :

Paramètre	Description
$A.{nom}	Filtre sur une région
$G.{nom}	Filtre sur un groupe

Exemple :

```
Client.Load("@VariableStates", "$A.Site1");
```

Les paramètres d'un synoptique :

Paramètre	Description
$V.{nom}	Affiche un synoptique en centrant sur une variable
Parameters={parameters}	Affiche un synoptique comportant des paramètres

Exemple :

```
Client.Load("Maintenance.xaml", "$V.A.E001");
```

CHARGEMENT D'UNE PRESENTATION

object Client.Load(string layoutName)

object Client.Load(int numWindow, string layoutName)

Cette méthode permet d'afficher une présentation dans une fenêtre.

layoutName : nom de la présentation

Une présentation est constituée de plusieurs frames qui ont chacune un nom :

Nom de présentation	Nom des frames
LayoutBottom	main, bottom
LayoutLeft	main, left
LayoutLeftTop	main, left, top
LayoutTop	main, top

Exemple :

```
Client.Load("LayoutLeft")        : charge une présentation dans la fenêtre active.
Client.Load(2," LayoutLeft")     : charge une présentation dans une fenêtre.
```

Le contenu des frames est chargé avec la méthode qui suit.

CHARGEMENT D'UNE FRAME

object Client.Load(int numWindow, string frameName, string document, params string[] args)

Charge un document dans une frame.

Retourne le contrôle chargé dans la frame.

numWindow : numéro de la fenêtre en configuration multi écran si > 0 ou fenêtre active si = 0

frameName : nom de la frame

document : nom du document

args : paramètres du document

Exemple :

```
Client.Load(0,"main","Hall.xaml");
```

CHARGEMENT D'UN ENVIRONNEMENT UTILISATEUR

object Client.Load(string fichier)

Cette méthode permet de charger un environnement utilisateur multi fenêtre dans l'interface du client.

Exemple :

```
Client.Load("Test.env");
```

OUVERTURE D'UNE NOUVELLE FENETRE

object Client.Open(string document, params string[] args)

Ouvre un document dans une nouvelle fenêtre.

Exemple :

```
Client.Open("")                  : ouvre une fenêtre vide.
Client.Open("","TopMost")        : ouvre une fenêtre au 1er plan.
Client.Open("LayoutLeft")        : ouvre une fenêtre avec une présentation.
Client.Open("Hall.xaml")         : ouvre une fenêtre avec un synoptique.
```

La méthode Client.Open retourne un objet Window, il est utilisable pour contrôler la fenêtre, par exemple :

```
var wnd = Client.Open("Hall.xaml","TopMost") as Window;
if(wnd!=null )
{
       wnd.Left = 40;
       wnd.Top = 40;
       wnd.Width = 300;
       wnd.Height = 300;
}
```

Les fenêtres sont identifiées par un nombre :

0 : fenêtre active
1 : 1ère fenêtre
2 : 2éme fenêtre
…

ANNEXE 1 - LES FICHIERS PCS

Par défaut l'ajout d'un nouveau script serveur ou client créé un fichier avec l'extension .pcs.

Il s'agit d'un fichier en langage C# destiné à un pré compilateur.

L'intérêt de ce format est de pouvoir saisir les opérandes directement sans les placer dans une *string* de façon à simplifier l'écriture des scripts.

Par exemple :

```
$V.A.G01.Cmd = 1 ;

$V.Synthese = $V.Alarm1 | $V.Alarm2;
```

Le pré compilateur du superviseur remplace ces instructions par du code C#.

Note : il est possible d'ajouter un script serveur ou client en C# natif, il suffit pour ceci de remplacer l'extension .pcs par .cs lors de la création.

HEADER ET FOOTER

Exemple de *Header* :

```
using System;
using System.IO;
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using Prysm.AppVision.Data;
using Prysm.AppVision.Common;
using Prysm.AppVision.SDK;
using Prysm.AppVision.Script;

namespace Prysm.AppVision.Server
{
```

Il s'agit d'une section qui permet de déclarer tous les namespace utilisés par les fichiers .pcs.

La section *Header* est ajoutée automatiquement au début de tous les fichiers .pcs.

La section *Footer* est ajoutée automatiquement à la fin de tous les fichiers .pcs.