# PowerDataStruct_V2

A biblioteca PowerDataStruct_V2 tem a função de varrer a estrutura de dados da aplicação, e gerar uma saída em formato XML. Este padrão de estrutura serve de base para diversas outras bibliotecas que envolvem o uso de listagens ou tabulares.

## Formato XML

O XML é um padrão de linguagem mundialmente conhecido, tendo como um dos principais propósitos o compartilhamento de informações através da internet. Algumas vantagens:

- É baseado em texto simples;
- Pode representar as estruturas de dados relevantes da computação: listas, registros, árvores;
- A sintaxe restrita e requerimentos de parsing tornam os algoritmos de análise mais eficientes e consistentes;
- É editável, devido à popularidade do XML nos dias de hoje, com diferentes níveis de automação, em qualquer ambiente.

*Fonte: Wikipedia*

## Pré-requisitos

Para o uso desta biblioteca, é necessário ter instalado o *Microsoft Core XML Services (MSXML) 6.0*. Este componente é utilizado por diversos programas do Windows, como por exemplo o Internet Explorer. Portanto na grande maioria dos casos, ele já se encontrará instalado na máquina, mas caso seja necessário ele pode ser baixado através do site da Microsoft:

http://www.microsoft.com/en-us/download/details.aspx?id=3988

## Padrão XML da biblioteca

O padrão XML usado por esta biblioteca é como demonstrado no exemplo abaixo. Cada objeto do Elipse Power tem o seu respectivo nó, cujo nome equivale ao tipo do objeto (PowerSubstation, PowerBreaker, etc). Cada nó possui por padrão o atributo PathName, e pode ter outros atributos, conforme desejado. Os objetos são organizados de forma hierárquica, da mesma forma que ocorre na estrutura de dados do Elipse Power.

```
- <PowerSubstation PathName="SE2">
    - <PowerBreaker PathName="SE2.[5212]" BaseVoltage="13.8">
        - <PowerTerminal PathName="SE2.[5212].Terminal1">
            - <PowerDiscreteMeasurement PathName="SE2.[5212].Terminal1.TerminalState1" MeasurementType="*TerminalState">
                <PowerDiscreteSource PathName="SE2.[5212].Terminal1.TerminalState1.Operator"/>
                <PowerDiscreteSource PathName="SE2.[5212].Terminal1.TerminalState1.TopologyProcessor"/>
              </PowerDiscreteMeasurement>
          </PowerTerminal>
          ...
```

## PowerDataStruct_V2

A biblioteca é composta basicamente por um XObject chamado PowerDataStruct_V2, que é responsável por rodar o script que varre a aplicação, e disponibilizar o resultado em uma de suas propriedades.

| Propriedade | Descrição |
|---|---|
| Value | Armazena o resultado da varredura. |
| Run | Ao receber um pulso, dispara a execução do script de varredura. |
| AutoRun | Indica se a varredura deve ser executada automaticamente quando o objeto for iniciado. Caso seja configurada como FALSE, o usuário deve dar um pulso manualmente na propriedade Run, para iniciar o processo. |
| ExportToFile | Indica qual o caminho do arquivo XML, quando uma exportação for feita. |
| Export | Ao receber um pulso, exporta o resultado obtido para um arquivo em disco, cujo caminho é configurável em 'ExportToFile'. |
| SearchFilter | Tipos de objetos que devem ser ignorados na varredura. Deve ser uma string separada por ponto-e-vírgula (;). Exemplo: *PowerConfig;IODriver;DataServer* |
| RootReference | Objeto de dados que deve ser varrido. Deixar em branco para varrer toda a aplicação. |
| CustomProperties | Lista de propriedades que o usuário deseja adicionar na estrutura XML, na forma de atributos. Deve ser uma string separada por ponto-e-vírgula (;). Exemplo: *ShortName;MeasurementType;BaseVoltage* |

## Configurando o objeto

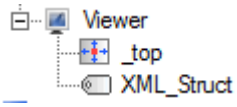
Para utilizar a biblioteca, basta adicioná-la no domínio e instanciar um objeto PowerDataStruct_V2 em uma pasta de dados. Todas as outras configurações são opcionais.

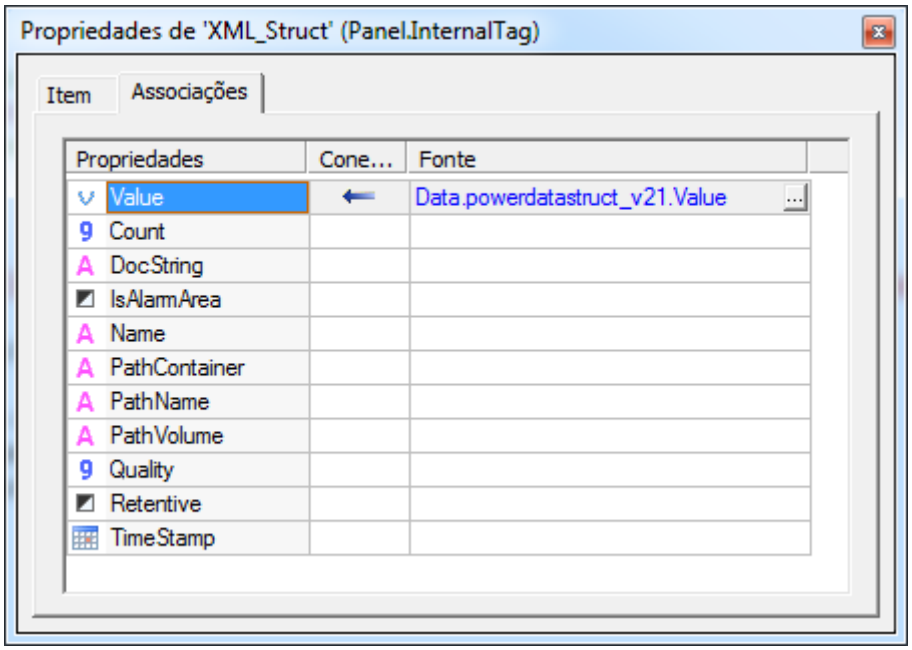
Algumas considerações importantes:

- O tempo de varredura é proporcional ao tamanho e à complexidade do aplicativo.
- Utilizar a propriedade *AutoRun* em TRUE aumentará o tempo de inicialização do domínio.
- O usuário pode colocar qualquer propriedade no *CustomProperties*. Durante a varredura, caso seja detectado que a propriedade não existe no objeto corrente, ela é simplesmente ignorada.
- Recomendamos que a varredura seja feita sobre toda a aplicação, portanto que se deixe a propriedade *RootReference* em branco. Ela serve para casos específicos, onde se deseja obter a estrutura de dados de apenas um objeto específico.
- O XML é "case sensitive", portanto difere letras maiúsculas e minúsculas.

## Viewer vs. Servidor

Ao desenvolver interfaces de tela baseadas nesta biblioteca, sugerimos que seja criado um tag interno no Viewer, que armazene o resultado da varredura através de uma associação simples; e que os controles de tela apontem para este tag interno, ao invés de apontar para o XObject.




Desta forma, diminui-se os acessos do Viewer ao servidor, o que tende a melhorar a performance geral do aplicativo.

## Desenvolvendo XControls

Recomendamos o uso desta biblioteca como base para o desenvolvimento de XControls que envolvam buscas de coleções de objetos de dados, como por exemplo, tabulares, listas, etc. A performance das telas tende a ser bem melhor, pois a busca é feita em um texto estruturado, e não em objetos individuais do servidor.

Para desenvolver XControls baseados nessa estrutura XML:

- Esta parte inicial sempre deve estar presente. Ela serve para carregar o resultado obtido pelo objeto de dados:

```
set docxml = createobject("MSXML2.DOMDocument.6.0")
docxml.async = false

'pode-se carregar tanto do XObject quanto do tag interno do Viewer
set obj = Application.GetObject("Data.PowerDataStruct_V2")
'set obj = Application.Item("XML_Struct")
docxml.loadXML(obj.Value)
set rootNode = docxml.SelectSingleNode("root")
...
```

- Para buscar uma coleção de nós de um determinado tipo, pode-se utilizar a função *GetElementsByTagName*:

```
...
tipo = "PowerBreaker"
set ObjCol = rootNode.GetElementsByTagName(tipo)
for i=0 to ObjCol.length-1
        set node = ObjCol.item(i)
        MsgBox node.getAttribute("PathName") 'poderia ser Name, ShortName, etc
Next
```

- Para buscar um objeto pelo seu PathName (semelhante ao método GetObject), pode-se utilizar a função *SelectSingleNode*:

```
...
objPath = "SE1.Bay1.Breaker1"
set node = rootNode.SelectSingleNode("//*[@PathName='"& objPath &"']")
If TypeName(node) = "IXMLDOMElement" then
        MsgBox "Object found!"
Else
        MsgBox "Object not found!"
End If
```

- Mesclando os dois exemplos acima, é possível buscar uma coleção de objetos dentro de um nó específico (medidas dentro de um equipamento, por exemplo):

```
objPath = "SE1.Bay1.Breaker1"
set node = rootNode.SelectSingleNode("//*[@PathName='"& objPath &"']")
tipo = "PowerAnalogMeasurement"
set ObjCol = node.getElementsByTagName(tipo)
for i=0 to ObjCol.length-1
        set node = ObjCol.item(i)
        MsgBox node.getAttribute("PathName") 'poderia ser Name, ShortName, etc
next
```

- O método **SelectNodes** abre um leque infinito de possibilidades de buscas por nós e coleções, baseado em expressões “XPath”. O exemplo abaixo faz uma busca por medidas analógicas que pertençam a disjuntores de tensão nominal 13.8 kV:

```
MyXPath = "//PowerBreaker[@BaseVoltage = 13.8]//PowerAnalogMeasurement"
set NodeCol = rootNode.SelectNodes(MyXPath)
for i=0 to NodeCol.length-1
        set node = NodeCol.item(i)
        MsgBox node.getAttribute("PathName") 'poderia ser Name, ShortName, etc
next
```

OBS.: Note que, para o correto funcionamento do exemplo acima, a propriedade BaseVoltage deve ser adicionada como atributo na estrutura XML.

## Referências

Todos os métodos e propriedades do *MSXML2.DOMDocument* estão disponíveis para uso, e estão documentados no site da MSDN:

http://msdn.microsoft.com/en-us/library/ms757828(v=vs.85).aspx

A sintaxe do formato *XPath* pode ser estudada neste tutorial do W3Schools:

http://www.w3schools.com/XPath/xpath_intro.asp

## Informações adicionais

A biblioteca foi desenvolvida no Elipse Power v4.5.243.