



walk

plano

bê

Descritivo técnico dos comandos e funções da linguagem POSXML

PosXML 2.0 - 11 de abril/2008

Apresentação

O Presente documento tem por objetivo discorrer sobre os argumentos técnicos, funcionalidades, comandos e funções que compõem a estrutura da linguagem POSXML. Utilizada para criar aplicativos para o framework WALK, a linguagem possibilita um rápido desenvolvimento de aplicações complexas com uma sintaxe simples e intuitiva.

A linguagem

A linguagem aqui apresentada se refere a um conjunto de comandos, funções e regras lógicas que pode ser usada em terminais de captura de transações eletrônicas, neste documento denominados simplesmente de Terminal POS.



A estrutura do PosXML

A linguagem POSXML segue o padrão aberto XML. Amplamente difundido atualmente na indústria de software, o padrão XML é totalmente flexível e possibilita rápida expansão. Todas as regras de formatação aplicadas ao XML continuam existindo no POSXML. Alguma dessas são descritas abaixo:

- A linguagem POSXML é organizada e estruturada em forma de “tags” com níveis e subníveis de comandos que formam a estrutura lógica do aplicativo POSXML. Uma “tag” pode ser identificada quando está entre os sinais de menor e maior, deve-se obrigatoriamente possuir uma “tag” de abertura e uma “tag” de fechamento. Exemplo:

```
<?xml version="1.0"?>
```

```
<aplicativoposxml xmlns="http://tempuri.org/posxml.xsd">
```

```
...  
...  
...
```

```
</aplicativoposxml>
```

- Todo aplicativo POSXML deve obrigatoriamente possuir uma “tag pai” ou “tag root” e possuir um nome válido, obedecendo ao critério de abertura e fechamento da mesma. Abaixo da “tag root”, devemos obrigatoriamente ter uma “tag” chamada “pagina”. Esta tag significa um bloco da aplicação que vai ser executado conforme solicitado. A “pagina” principal é a primeira a ser executada, e a única que fica armazenada no terminal. Nela devemos escrever toda a lógica de nosso programa. Ao fazermos qualquer modificação na “pagina principal” e o terminal conectar ao host, a mensagem “WALK: Aplicativo Modificado” aparecerá na tela.

As demais páginas além da principal podem ser utilizadas para escrever tickets de impressão ou lógicas de confirmação. Lembre-se: Apenas a página principal fica armazenada no terminal, as demais são compiladas em tempo de execução e enviadas pelo host para o terminal.

A execução se inicia na página principal. Após executar a seqüência de comandos, se assim for especificado o terminal vai se conectar ao host. Cabe o host especificar o retorno contendo o nome da próxima página a ser executada do arquivo.xml. Assim é possível navegar entre as páginas segundo instruções do host autorizador. Como disse, a pagina principal inicia sua execução imediata no terminal sem nenhum tipo de conexão. Apenas se a página inicial do host estiver diferente da página inicial do terminal, é que esta vai ser atualizada.

```
<aplicativoposxml xmlns="http://tempuri.org/posxml.xsd">
```

```
  <pagina nome= "principal">
```

```
    <display linha="1" coluna="0" mensagem="Aplicativo PosXML" />
```

```
    <display linha="2" coluna="0" mensagem="Plano Be Tecnologia"/>
```

```
    <esperatecla />
```

```
    <limpadisplay />
```

```
    <conectar />
```

```
  </pagina>
```

```
</aplicativoposxml>
```



- A linguagem POSIXML é case - sensitive. Portanto existem diferenças entre palavras com letras minúsculas e palavras com letras maiúsculas. Devemos ficar atentos ao escrever aplicativos, principalmente ao nomear funções e variáveis. No exemplo abaixo a função que vai ser chamada é a segunda, porque “mostramensagem” não é igual a “mostraMensagem”. Exemplo:

```
<aplicativoposxml xmlns="http://tempuri.org/posxml.xsd">
  <pagina nome= "principal">
    <chamafuncao nome="mostramensagem" />

    <funcao nome="mostraMensagem">
      <display linha="0" coluna="0" mensagem=" Mensagem 1" />
      <esperatecla />
    </funcao>

    <funcao nome="mostramensagem">
      <display linha="0" coluna="0" mensagem=" Mensagem 2" />
      <esperatecla />
    </funcao>
  </pagina>
</aplicativoposxml>
```

- Todos os caracteres reservados na linguagem XML, continuam existindo em POSIXML e devem ser escritos da maneira correta. Os símbolos a seguir devem ser escritos da seguinte maneira:

<	<
>	>
&	&
'	'
"	"

Por exemplo para mostrarmos uma mensagem com o símbolo de menor , a sintaxe ficaria assim:

```
<aplicativoposxml xmlns="http://tempuri.org/posxml.xsd">
  <pagina nome= "principal">
    <chamafuncao nome="mostramensagem" />

    <funcao nome="mostramensagem">
      <display linha="0" coluna="0" mensagem="&lt; = menor" />
      <esperatecla />
    </funcao>
  </pagina>
</aplicativoposxml>
```

- Blocos de comentários podem ser utilizados com as tags <!-- -->. Exemplo:

```
<aplicativoposxml xmlns="http://tempuri.org/posxml.xsd">
  <pagina nome= "principal">
    <!-- função que mostra uma mensagem no display -->
    <funcao nome="mostramensagem">
      <display linha="0" coluna="0" mensagem="&lt; = menor" />
      <esperatecla />
    </funcao>
  </pagina>
</aplicativoposxml>
```



Conceito de variáveis e memória

O framework que é o responsável por executar o aplicativo compilado PosXML, ao ser instalado no terminal assume toda a responsabilidade de gerenciamento de memória e dispositivos do mesmo. Então para fazer uma aplicação rodar no terminal, basta escrevermos um arquivo no formato PosXML, e enviarmos ao terminal.

Como toda boa linguagem de programação, temos o conceito de tipos de variáveis também em PosXML.

As variáveis podem ser do tipo string e inteiro. Cada instrução da linguagem exige um tipo de variável, que está descrito na explicação do funcionamento das mesmas. Podemos converter variáveis do tipo inteiro para string e vice-versa. Basta utilizarmos as instruções [inttostring](#) e [stringtoint](#).

Também podemos utilizar variáveis do tipo double (ponto flutuante, decimal) utilizando para isso variáveis do tipo string tendo a casa decimal separada por . (ponto) e não por, (vírgula).

Temos um limite de 256 variáveis por aplicação PosXML, sendo 256 do tipo string com tamanho máximo de 1k (1023 caracteres, sendo que um é utilizado automaticamente pela plataforma para alocar o '\0'), e 256 do tipo inteiro.

Ao alocarmos uma variável através da instrução [variavelint](#) ou [variavelstr](#) esta ficará na memória da aplicação até que o terminal seja desligado. Por isso devemos ter o cuidado de definir no começo de nossa aplicação todas as variáveis utilizadas e limpa-las para não ocorrer erros no decorrer de nosso aplicativo.

As variáveis são compartilhadas entre páginas, portanto podemos utilizar variáveis alocadas na página principal nas páginas subseqüentes, como páginas de tickets de impressão e outras lógicas.

Ao desligarmos o terminal ou ocorrer uma atualização da página principal do aplicativo, todas as variáveis são desalocadas e a memória da aplicação PosXML é totalmente limpa.

Atualmente temos um limite de memória de 32K para carregamento de paginas PosXML compiladas. Podemos ter em um mesmo aplicativo, várias paginas sendo que o único impedimento da plataforma é que esta não ultrapasse 32K compilada.

Toda instrução PosXML que tiver um parâmetro cujo o nome contenha a palavra variável, por ex.: `variavelretorno`; devemos passar obrigatoriamente uma variável para aquele parâmetro.

A passagem de variáveis é feita com a seguinte nomenclatura: `$(nomedavariavel)`. Exemplo:

```
<variavelstr valor="" variavel="buffer" />
<display linha="0" coluna="0" mensagem="$(buffer)" />
```

Quando no nome do parâmetro não tiver a palavra variável podemos passar uma constante ou uma variável. A passagem de constantes é feita como em toda linguagem de programação.

```
<display linha="0" coluna="0" mensagem="Hello World" />
```

Em variáveis string, ainda é possível passar para a instrução apenas um trecho indexado da variável. Como em outras linguagens de programação, utilizando ponteiros de variáveis.

```
<variavelstr valor="Hello World" variavel="buffer" />
<variavelint valor="0" variavel="index" />
<display linha="0" coluna="0" mensagem="$(buffer[$(index)])" />
<!-- Isto mostraria no display: H -->
```

```
<variavelstr valor="Hello World" variavel="buffer" />
<variavelint valor="1" variavel="index" />
<display linha="0" coluna="0" mensagem="$(buffer[$(index)])" />
<!-- Isto mostraria no display: e -->
```

```
<variavelstr valor="Hello World" variavel="buffer" />
<variavelint valor="1" variavel="index" />
<display linha="0" coluna="0" mensagem="$(!buffer[$(index)])" />
<!-- Isto mostraria no display: ello World -->
```

Note neste último exemplo que especificamos, "!" Para mostramos o caractere da posição definida em `index`, até o final da string. Quando não especificamos "!", mostramos apenas o caractere definido na posição de `index`.



Sistema de Arquivo

O framework WALK possui um sistema de arquivo que consegue manipular arquivos WALK dbfile. Com este sistema de arquivo podemos criar e trabalhar com arquivos no formato “key=buffer\nkey=buffer\n”.

Basicamente seria o formato de arquivos texto em ambiente Unix, com a quebra de linha sendo um \n. As instruções [editaarquivo](#), [learquivo](#) e [learquivobyindex](#) podem ser utilizadas para trabalhar com este tipo de arquivo baseado em chaves e valores, possibilitando um meio rápido e eficiente para armazenar dados estáticos no terminal. O próprio arquivo de configuração do terminal é um arquivo WALK dbfile.

Configuração

O framework WALK possui várias variáveis de configuração, permitindo configurar qualquer parâmetro do terminal através da linguagem PosXML, como se fossem variáveis de ambiente utilizadas pelo aplicativo e por dispositivos do terminal como modem e impressora. O arquivo contendo todas as variáveis de configuração tem o nome de config.dat e tem os seguintes parâmetros de configuração.

sn_walk : serial number do WALK.
nomeaplicativo : nome do aplicativo que o WALK, vai buscar na pasta do WALK Server (obrigatoriamente 10 letras).
primeiravez
numerodestepos : número do pos.
tiposcartao : 0 Default. Aceita trilha 2. Digitacao ou passe. 2 Aceita trilha 2. Nao permite digitacao. 4 Aceita trilha 1 e trilha 2. Digitacao ou passe. 255 Aceita trilha 1 e trilha 2. Nao permite digitacao.

debug : nivel de debug da aplicacao 0,1,2,3.
retentativas : quantidade de tentativas de conexao com o dispositivo configurado.
qtdtentativasenvio : numero de pacotes que serão enviados até haver a confirmação pelo WALK Server.
sn_terminal : serial number do terminal.
model : modelo do terminal.
withssl : 0 não utiliza SSL, 1 utiliza SSL.
myip : ip do terminal.
mygateway : gateway que vai ser utilizado pelo terminal 192.168.0.1 se formos utilizar PPP.
dnsprimario : 192.168.0.1 se formos utilizar PPP.
dnssecundario : 192.168.0.1 se formos utilizar PPP.
iphost : ip do WALK Server.
portahost : porta do WALK Server.
subnet : mascara de rede. “255.255.255.255”.
uclmedia : Tipo de comunicação suportada no terminal. Pode ser GPRS, ETHERNET ou LANDLINE.
uclapn : APN da rede da operadora GPRS.
uclprotocol Pode ser TCP, SDLC ou X28 (RENPAK).
uclphoneno : número em que o terminal vai discar.
uclusername : nome de usuário do RAS ou APN (PPP).
uclpassword : password do usuário do RAS ou APN (PPP).
uclportamodemext : /dev/com1 ou /dev/com2.
uclvelocidademodem : 1200 , 2400 , 9600 ou 14400.
gprs_pin : pin do sim card GPRS.
autooffmodem : desligar modem automaticamente após conexão, 0 nunca desliga.
ipwalkweb : ip do WALK Web.
portawalkweb porta do WALK Web.
sslwalkweb 0 não utiliza SSL, 1 utiliza SSL.
timeoutatualizacaowalkweb : timeout de checagem se há atualização no WALK Web.
crcpaginawalkserver : CRC do WALK Server.
keypaperfeed : Valor da tecla que executa o Paperfeed.
keyalpha : Valor da tecla Alfa-Numérica.
keypound : Valor da tecla Pound ‘#’.
keystar : Valor da tecla star ‘*’.
touchscreen : Habilita ou desabilita o touchscreen.
renpac_nua : Número do X28.
renpac_niu : Número do X25.
renpac_psw : Senha do X25.



Qualquer um desses parâmetros pode ser configurado através da instrução [editaarquivo](#), apontando para `config.dat`, e utilizando o nome desses parâmetros como chave. É possível configurar qualquer tipo de media, retentativas, entre outros parâmetros e fazer lógicas de reconexão de backup através da linguagem PosXML, utilizando as instruções [editaarquivo](#) e [preconexao](#).

As especificações

A seguir a descrição detalhada de cada função, comando e instrução lógica contida na linguagem POSXML. Como toda linguagem de programação, a PosXML é compilada em um formato reduzido para que o terminal com o framework WALK devidamente instalado consiga interpretar e executar seus comandos.

Cada instrução é transformada pelo compilador , em um “*byte code*”, reduzindo consideravelmente o tamanho do aplicativo após compilado.

Outro ponto importante a salientar , é a diferença entre variáveis do tipo inteiro, e variáveis string. Como toda linguagem de programação em ambiente interpretado, existe a diferenciação e a definição de tipos. Devemos ficar atentos ao tipo de variável que o parâmetro da instrução pede, ou mesmo se há necessidade de passar uma variável ou uma constante como explicado em [Conceito de variáveis e memória](#).

A linguagem POSXML, é uma linguagem de programação básica, que está em constante evolução. Hoje temos suporte, a if, funções, tipos de variáveis, operações matemáticas com inteiros e ponto flutuantes. Segue abaixo descritivo das instruções existentes.



As instruções

A seguir as instruções existentes até o momento na linguagem PosXML que está em constante evolução e por este motivo novas instruções surgem a cada versão. Caso você execute algum aplicativo contendo uma instrução não suportada pela versão do WALK instalada no terminal ocorrerá o seguinte erro: "WALK: Byte Mode Error...", dizendo que aquele "byte code" daquela instrução ainda não existe nesta versão.

display		predial	substring
limpdisplay	editarquivo	conectar	somastring
menu	learquivo	preconexao	stringtoint
mostrabitmap	learquivobyindex	desligamodem	inttostring
	filesystem.space	network.send	convert.toint
digitainteiro	filesystem.filesize	network.receive	string.length
digitavalor	filesystem.listfiles	network.httprequest	string.charat
digitaformato	excluiarquivo	network.webservicenew	string.trim
digitaopcao		network.webserviceaddparameter	string.find
digitadecimal	enviaarquivo	network.webserviceconnect	string.replace
	baixaarquivo	network.checkprssignal	string.substring
espera	unzipfile	network.ping	string.elements
esperatecla		network.hostdisconnect	string.insertat
esperateclatimeout	ajustadatahora		string.removeat
letecla	pegadatahora	iso8583.iniciatabelacampos	string.replaceat
	incrementadatahora	iso8583.iniciamensagem	string.elementat
imprima	diferencadatahora	iso8583.analisamensagem	
imprimagrande		iso8583.finalizamensagem	if
imprimabitmap	pegacartao	iso8583.inserecampo	else
imprimacodigodebarras	pegacartaoevariavel	iso8583.pegacampo	while
		iso8583.transactmessage	break
paperfeed			exit
verificapapelimprensa	openserialport	smartcard.inicialeitoeitor	
	readserialport	smartcard.cartaoinsertado	execute
system.reinicia	writeserialport	smartcard.transmitAPDU	ultimapagina
system.checkbattery	closeserialport	smartcard.fechaleitor	
system.beep	pegaserialevariavel		funcao
system.readcard		variavelint	
	conversaointeiro	variavelstr	chamafuncao
playmusic	operacaomatematica	guardavariavel	
		limpvariaveis	
		variavelintoperador	



walk

plano

bê

display

Sintaxe utilizada

```
<display linha="n" coluna="n" mensagem="xxxxxxxx" />
```

Parâmetros

linha : Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

Pra que serve

O comando **display** é usado para exibir no visor (display) do terminal POS uma mensagem específica na posição linha e coluna.

Exemplo

```
<display linha="1" coluna="1" mensagem="POSXML" />
```

limpdisplay

Sintaxe utilizada

```
<limpdisplay />
```

Pra que serve

O comando **limpdisplay** é usado para apagar todas as mensagens exibidas no visor (display) do terminal POS.

Exemplo

```
<limpdisplay />
```

digitainteiro

Sintaxe utilizada

```
<digitainteiro variavel="xxx" linha="n" coluna="n" mensagem="xxxxxx" minimo="n" maximo="n" />
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (inteiro)

linha: Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

mínimo: Número mínimo a ser digitado. (inteiro)

máximo: Número máximo a ser digitado. (inteiro)

Pra que serve

O comando **digitainteiro** permite a digitação de dados numéricos através do teclado do terminal POS. Uma mensagem é exibida no visor do terminal em linha e coluna especificadas e este fica aguardando a digitação da informação solicitada. Pode se configurar os intervalos mínimo e máximo do valor numérico digitado, que por sua vez é armazenado em variável de memória do terminal POS.

Exemplo

```
<digitainteiro variavel="$(CEP)" linha="1" coluna="1" mensagem="Digite CEP:" minimo="0" maximo="999999" />
```



walk

plano

bê

digitavalor

Sintaxe utilizada

```
<digitavalor variavel="$ (xxx)" linha="n" coluna="n" mensagem="xxxxxxx" />
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (inteiro)

linha: Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

Pra que serve

O comando **digitavalor** aguarda o operador do terminal digitar um valor monetário através do teclado do terminal. Exibe uma mensagem em linha e coluna especificadas. O valor digitado é armazenado em uma variável de memória do terminal POS. Possui máscara automática de pontos e vírgulas que vai mostrando automaticamente com pontos e vírgulas o valor digitado sem necessidade do operador digitar . (ponto) e , (virgula)

O valor armazenado na variável é do tipo inteiro e não contem pontos nem vírgulas. O valor armazenado ficará assim, seguindo a especificação ISO8583 para valores.

1 = 1 centavo

10 = 10 centavos

100 = 1 real

1000 = 10 reais

10000 = 100 reais

100000 = 1000 reais

1000000 = 10000 reais

Exemplo

```
<digitavalor variavel="$ (valor)" linha="1" coluna="1" mensagem="Valor da Compra:" />
```

digitaformato

Sintaxe utilizada

```
<digitaformato variavel="$ (var1)" linha="n" coluna="n" mensagem="xxxx" formato="9*A" />
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (string)

linha: Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

formato: Máscara para digitação dos dados: 9 : número * : senha A : letra. Os demais caracteres serão inseridos automaticamente. (string)

Pra que serve

O comando **digitaformato** aguarda o operador do terminal digitar uma informação que tenha formato específico ou máscara de entrada. Exibe uma mensagem em linha e coluna especificadas. A informação digitada no teclado do terminal é então armazenada em memória.

Exemplo

```
<digitaformato variavel="$ (Data)" linha="1" coluna="1" mensagem="Data da compra:" formato="99/99/9999" />
```



walk

plano

bê

digitaopcao

Sintaxe utilizada

```
<digitaopcao variavel="$ (Var)" linha="0" coluna="0" mensagem="Digite a opcao:" minimo="0" maximo="9" />
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (inteiro)

linha: Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

mínimo: Número mínimo a ser digitado. (inteiro)

máximo: Número máximo a ser digitado. (inteiro)

Pra que serve

Praticamente a mesma instrução que `digitainteiro` com a diferença que ao digitarmos o número máximo especificado na instrução, a instrução automaticamente pula para a próxima, como que se estivéssemos pressionado enter ou OK.

Exemplo

```
<digitaopcao variavel="$ (Var)" linha="0" coluna="0" mensagem="Digite a opcao:" minimo="0" maximo="9" />
```

esperatecla

Sintaxe utilizada

```
<esperatecla />
```

Pra que serve

O comando `esperatecla` aguarda o operador do terminal pressionar qualquer tecla no terminal para continuar alguma operação.

Exemplo

```
<esperatecla />
```

imprimagrande imprima

Sintaxe utilizada

```
<imprimagrande mensagem="xxxxxxx" />
```

```
<imprima mensagem="xxxxxxx" />
```

Parâmetros

mensagem: Mensagem a ser impressa pela impressora do terminal. (string)

Pra que serve

O comando `imprima` e `imprimagrande` envia para a impressora do terminal uma mensagem especificada e que por sua vez é impressa em papel. Independente da impressora ser térmica, matricial de impacto, etc. A impressão é feita com tamanho de fonte dupla, ou seja, duas vezes maior que o tamanho normal. Já a instrução `imprima`, imprime com um tamanho de fonte normal.

Exemplo

```
<imprimagrande mensagem="Plano Bê Tecnologia" />
```

```
<imprima mensagem="xxxxxxx" />
```

paperfeed

Sintaxe utilizada

```
<paperfeed />
```

Pra que serve

O comando `paperfeed` envia para a impressora do terminal POS um comando para que o papel salte várias linhas até o necessário suficiente para que seja destacado sem cortar o conteúdo impresso.

Exemplo

```
<paperfeed />
```



conectar

Sintaxe utilizada

```
<conectar />
```

Pra que serve

O comando **conectar** prepara o dispositivo de conexão, seja um modem analógico ou não, e envia todas as informações coletadas pelo operador até o sistema atendedor e que irá receber tais informações coletadas.

Exemplo

```
<conectar />
```

preconexao

Sintaxe utilizada

```
<preconexao variavelstatus ="var1" />
```

Parâmetros

variavelstatus: Variável que indica o status da conexão. (inteiro)

Pra que serve

O comando **preconexao** inicializa o dispositivo de conexão e retorna na variável especificada se conseguiu conectar no host ou não. Se a variável for igual a -1, ocorreu um erro de conexão. Se a variável for igual a 0 o terminal está conectado ao host. Lembrando que esta instrução não envia nenhuma informação ao host, apenas faz a conexão. Para enviar as variáveis coletadas devemos chamar a instrução [conectar](#).

Exemplo

```
<variavelint valor="-1" variavel="iConectado" />
<preconexao variavelstatus="$(iConectado)" />
<if variavel="$(iConectado)" operador ="igual" valor="-1">
  <display linha="0" coluna="0" mensagem=" Erro de conexao" />
</if>
<if variavel="$(iConectado)" operador =" igual" valor="0">
  <display linha="0" coluna="0" mensagem=" Conectado ..." />
</if>
<esperatecla />
```

desligamodem

Sintaxe utilizada

```
<desligamodem />
```

Pra que serve

O comando **desligamodem** cancela as operações do dispositivo de conexão, encerrando a conexão aberta com o host (atendedor) e desliga o modem.

Exemplo

```
<desligamodem />
```



pegacartao

Sintaxe utilizada

```
<pegacartao primeiramensagem="xxx" segundamensagem="xxx" minimo="n" maximo="n" />
```

Parâmetros

primeiramensagem: Primeira mensagem que vai ser exibida no display. (string)

segundamensagem: Segunda mensagem que vai ser exibida no display, após o usuário pressionar OK. (string)

minimo: Número mínimo de caracteres a serem digitados. (inteiro)

maximo: Número máximo de caracteres a serem digitados. (inteiro)

Pra que serve

O comando **pegacartao** fica aguardando o operador do terminal POS passar um cartão magnético em sua leitora de trilhas ou a digitação dos números de um determinado cartão. Pode exibir duas mensagens, uma para ser exibida quando o terminal pedir que o operador passe o cartão na leitora e a outra para ser exibida quando o operador pede para digitar o cartão. Pode-se determinar o número mínimo e máximo de bytes que podem ser digitados. O conteúdo do cartão digitado ou capturado da tarja magnética é armazenado na memória que vai ser enviada ao host com a seguinte sintaxe : <cartao>0000000000000000 se foi passado ou <cartao>D000000000000000 se foi digitado. A configuração de trilhas a serem analisadas e se é permitido que o operador digite o número do cartão é definido na no parâmetro de configuração [tiposcartao](#). Se configurarmos o terminal de modo que não seja permitida a digitação do cartão , quando o operador pressionar qualquer tecla do terminal , ao invés de passar o cartão, o aplicativo se encerrará.

Exemplo

```
<pegacartao primeiramensagem="Passe cartao ou tecla OK" segundamensagem="Digite cartão com 16 digitos" minimo="16" maximo="16" />
```

pegacartaovariavel

Sintaxe utilizada

```
<pegacartaovariavel primeiramensagem="xxx" segundamensagem="xxx" minimo="n" maximo="n" variavel="$(var)" />
```

Parâmetros

primeiramensagem: Primeira mensagem que vai ser exibida no display. (string)

segundamensagem: Segunda mensagem que vai ser exibida no display, após o usuário pressionar OK. (string)

minimo: Número mínimo de caracteres a serem digitados. (inteiro)

maximo: Número máximo de caracteres a serem digitados. (inteiro)

variavel: Variável onde ficará armazenado o valor do cartão. (string)

Pra que serve

O comando **pegacartaovariavel** fica aguardando o operador do terminal POS passar um cartão magnético em sua leitora de trilhas ou a digitação dos números de um determinado cartão. Pode exibir duas mensagens, uma para ser exibida quando o terminal pedir que o operador passe o cartão na leitora e a outra para ser exibida quando o operador pede para digitar o cartão. Pode-se determinar o número mínimo e máximo de bytes que podem ser digitados. O conteúdo do cartão digitado ou capturado da tarja magnética é armazenado na variável especificada. Se foi digitado o primeiro caractere da variável vai ser 'D'. A configuração de trilhas a serem analisadas e se é permitido que o operador digite o número do cartão é definido na no parâmetro de configuração [tiposcartao](#). Se configurarmos o terminal de modo que não seja permitida a digitação do cartão , quando o operador pressionar qualquer tecla do terminal , ao invés de passar o cartão, o aplicativo se encerrará.

Exemplo

```
<pegacartaovariavel primeiramensagem="Passe o cartao ou tecla OK" segundamensagem="Digite o Cartao:" minimo="4" maximo="40" variavel="$(PIN)" />
```



if

Sintaxe utilizada

```
<if variavel="$(xxx)" operador="..." valor="xxxx">  
</if>
```

Parâmetros

variavel: Variável para comparação. (string ou inteiro)

operador: Operador de comparação, que pode ser : menor, maior, igual, diferente, maiorigual, menorigual. (string)

valor: Valor para comparação. (string ou inteiro)

Pra que serve

A instrução if é um termo utilizado para especificar uma tomada de decisão ou comparação lógica dentro do programa POSXML. Outros comandos e instruções podem ser contidos dentro do argumento if e /if. Os comandos e instruções dentro do argumento são executados caso a expressão seja verdadeira.

A instrução processa a operação entre variavel e o valor especificado, e entra na execução da próxima instrução dentro de if, caso seja verdadeiro o resultado.

Todos os operadores (igual/diferente/menor/menor/menor/menor/menor/menor/menor/menor) no caso de variável inteiro realizam as operações padrões de comparação de qualquer linguagem de programação.

Já os operadores (maior/menor/menor/menor/menor/menor/menor/menor/menor/menor) no caso de variável string, converterá a variável para ponto flutuante e realizarão a comparação. Os operadores (igual/diferente) seguem a mesma lógica de comparação de qualquer linguagem de programação.

Exemplo

```
<if variavel="$(TipoServico)" operador="igual" valor="1">  
  <guardavariavel variavel="$(TipoServico)" />  
  <pegacartao primeiramensagem="1. Cartao ou (OK)" segundamensagem="Digite Cartao:" minimo="15" maximo="16" />  
  <conectar />  
</if>
```

guardavariavel

Sintaxe utilizada

```
<guardavariavel variavel="$(nomeVariavel)" />
```

Parâmetros

variavel: Variável que vai ser armazenada no buffer que será enviado ao terminal. (string ou inteiro)

Pra que serve

A instrução **guardavariavel** é um termo de atribuição, responsável pelo armazenamento e enfileiramento do conteúdo de uma variável para ser enviado ao host (atendedor). O conteúdo é armazenado em memória do Terminal POS, sendo que cada vez que chamamos a instrução guarda variável , o conteúdo da variável é guardado em um buffer e separado por virgula. Ao chamarmos a instrução conectar , o terminal envia o buffer completo para o host.

Exemplo

```
<guardavariavel variavel="$(numeroCartao)" />
```

limpavariaveis

Sintaxe utilizada

```
<limpavariaveis />
```

Pra que serve

A instrução **limpavariaveis** é um comando que apaga o conteúdo da variável que contém o buffer a ser enviado ao host. O buffer separado por ',' montado por [guardavariavel](#) é limpo ao chamarmos esta instrução.

Exemplo

```
<limpavariaveis />
```



funcao

Sintaxe utilizada

```
<funcao nome="xxxxxx">
</funcao>
```

Parâmetros

nome: Função a ser chamada. (string)

Pra que serve

A instrução **funcao** é responsável por nomear uma função ou bloco de código contendo outras instruções, comandos e regras lógicas. Não podemos declarar blocos de funções dentro de funções. A execução da função só se dará , após a chamada pela instrução [chamafuncao](#)

Exemplo

```
<funcao nome="Imprime_Cupom">
  <imprimagrande mensagem="Plano Bê" />
  <imprimagrande mensagem="*****" />
  <imprima mensagem=" " />
  <imprima mensagem="Olá, isto é um teste" />
  <paperfeed />
</funcao>
```

chamafuncao

Sintaxe utilizada

```
<chamafuncao nome="xxxxxx" />
```

Parâmetros

nome: Função a ser chamada para execução. (função)

Pra que serve

A instrução **chamafuncao** é um termo de desvio e é responsável por fazer uma chamada e executar uma função contendo um bloco de comandos, conforme descrito acima.

Exemplo

```
<chamafuncao nome="Imprime_Cupom" />
```

variavelint

Sintaxe utilizada

```
<variavelint valor="n" variavel="xxxxxx" />
```

Parâmetros

valor: Valor da variável. (inteiro)

variável: Nome da variável. (variavel)

Pra que serve

A instrução **variavelint** é um termo de atribuição e serve para armazenar um determinado valor numérico e inteiro em uma variável especificada.

Exemplo

```
<variavelint valor="1" variavel="Parcelas" />
```



variavelstr

Sintaxe utilizada

```
<variavelstr valor="xxxx" variavel="xxxxxx" />
```

Parâmetros

valor: Valor da variável. (string)

variável: Nome da variável. (variável)

Pra que serve

A instrução **variavelstr** é um termo de atribuição e serve para armazenar um determinado valor alfanumérico (que contenha letras e números) em uma variável especificada.

Exemplo

```
<variavelstr valor="Plano Bê" variavel="Nome" />
```

substring

Sintaxe utilizada

```
<substring posicao="n" variavelorigem="$ (xxxxx)" variaveldestino="$ (yyyyy)" caracter="," variavelretorno="$ (ret)" />
```

Parâmetros

posicao: Indica a partir de que posicao vamos copiar a string até encontrar o caracter especificado. A primeira posicao é 0. (inteiro)

variavelorigem: Variável onde vai ser feito o substring. (string)

variaveldestino: Variável onde vai ficar armazenado o resultado do substring. (string)

caracter: Caractere delimitador. (string)

variavelretorno: Posição do caracter , se este foi encontrado. Se não foi encontrado o caracter retorna -1. (inteiro)

Pra que serve

A instrução **substring** é a instrução responsável por copiar um trecho de uma variável origem do tipo "string" para uma variável também do tipo string.

Exemplo

```
<substring posicao="1" variavelorigem="$ (ContemTudo)" variaveldestino="$ (ContemParte)" caracter="," variavelretorno="$ (ret)" />
```

stringtoint

Sintaxe utilizada

```
<stringtoint variavelstr="$ (Var1)" variavelint="$ (Var2)" />
```

Parâmetros

variavelstr: Variável que vai ser convertida. (string)

variavelint: Variável que vai armazenar o valor inteiro. (inteiro)

Pra que serve

A instrução **stringtoint** é a função responsável por converter o conteúdo de variáveis do tipo string para o tipo inteiro, sempre capturando um trecho da variável string informado no intervalo entre posição inicial e posição final. A conversão é feita e o conteúdo convertido é armazenado então em uma outra variável do tipo inteiro.

Exemplo

```
<stringtoint variavelstr="$ (Var1)" variavelint="$ (Var2)" />
```



inttostring

Sintaxe utilizada

```
<inttostring variavelint="$ (Var1)" variavelstr="$ (Var2)" />
```

Parâmetros

variavelint: Variável que vai ser convertida. (inteiro)

variavelstr: Variável que vai armazenar o valor string. (string)

Pra que serve

A instrução **inttostring** é a função responsável por converter o conteúdo de variáveis do tipo inteiro para o tipo string. A conversão é feita e o conteúdo convertido é armazenado então em uma outra variável do tipo string.

Exemplo

```
<inttostring variavelint="ValorInteiro" variavelstr="ValorStr" />
```

menu

Sintaxe utilizada

```
<menu variavel="$ (var1)" opcoes="MENU\1. primeira linha\2. segunda linha 2 \3. terceira linha"/>
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (inteiro)

opcoes: Opções do menu delimitadas por '\'. (string)

Pra que serve

A instrução **menu** constrói um menu na tela do terminal, conforme o número de linhas do mesmo. Na constante string opções para cada barra '\' existente, vai ser uma quebra de linha. A primeira opção antes da primeira barra será o título do menu e estará em todas as telas mesmo rolando para mais opções. Quando o número de opções atinge o tamanho máximo do número de linhas, vai aparecer ao lado do título um sinal de '+' indicando que é possível rolar para outra tela. É possível mostrar a outra tela, pressionado (enter, ou OK).

O usuário pode digitar de 0 a 9, ou seja, temos a possibilidade de ter 10 opções de menu. O valor digitado fica armazenado em variavel.

Exemplo

```
<menu variavel="$ (var1)" opcoes="MENU\1. primeira linha\2. segunda linha 2 \3. terceira linha"/>
<if variavel="$ (var1)" operador="igual" valor="2">
  <display linha="0" coluna="0" mensagem="Vc escolheu a segunda linha" />
  <esperatecla />
</if>
```

espera

Sintaxe utilizada

```
<espera milisegundos="1000" />
```

Parâmetros

milisegundos: Milisegundos que a instrução vai aguardar antes de continuar a execução. (inteiro)

Pra que serve

O comando **espera** aguarda o tempo especificado em milisegundos até que possa continuar com a execução do programa. O aplicativo aguardará o tempo especificado mesmo que operador pressione qualquer tecla.

Exemplo

```
<espera milisegundos="1000" />
```



imprimalogo

Sintaxe utilizada

```
<imprimalogo index="1" />
```

Parâmetros

index: Indica o número do logo a ser impresso pelo terminal. (inteiro)

Pra que serve

O comando **imprimalogo** envia para a impressora do terminal um código que possibilita a impressão de logos previamente carregados no terminal. Cada logo possui um código de identificação e este pode ser um número de 1 a 10.

Exemplo

```
<imprimalogo index="1" />
```

pegaserialevariavel

Sintaxe utilizada

```
<pegaserialevariavel primeiramensagem="PASSE DOC. NA LEITORA" segundamensagem="DIGITE NUM. DOC:" minimo="4" maximo="40" variavel="$(LEITORA)" />
```

Parâmetros

primeiramensagem: Primeira mensagem que vai ser exibida no display. (string)

segundamensagem: Segunda mensagem que vai ser exibida no display, após o usuário pressionar OK. (string)

mínimo: Número mínimo de caracteres a serem digitados. (inteiro)

máximo: Número máximo de caracteres a serem digitados. (inteiro)

variavel: Variável onde ficará armazenado o valor vindo do periférico da porta serial. (string)

Pra que serve

O comando **pegaserialevariavel** fica aguardando a porta do PIN PAD do terminal enviar um buffer no protocolo STX e ETX. Pode exibir duas mensagens, uma para ser exibida quando o terminal aguarda o buffer do periférico e outra quando o operador não quer esperar a resposta do periférico e tecla enter OK. Pode-se determinar o número mínimo e máximo de bytes que podem ser digitados. O buffer capturado do periférico é armazenado na variável especificada. Se foi digitado o primeiro caractere da variável vai ser 'D'.

Podemos utilizar qualquer periférico de leitura externa, plugado na porta do PINPAD , comunicando no protocolo STX / ETX , 2400 bps, 7E1 .

Exemplo

```
<pegaserialevariavel primeiramensagem="PASSE DOC. NA LEITORA" segundamensagem="DIGITE NUM. DOC:" minimo="4" maximo="40" variavel="$(LEITORA)" />
```

variavelintoperador

Sintaxe utilizada

```
<variavelintoperador operador="++" variavelorigem="$(var1)" />  
<variavelintoperador operador="--" variavelorigem="$(var1)" />
```

Parâmetros

operador: Operador de incremento '++' ou decremento '--'. (string)

variavelorigem: Variável que vai ser incrementada ou decrementada. (inteiro)

Pra que serve

Permite adicionar +1 ou decrementar -1 na variável inteiro especificado em variavelorigem. Muito útil em situações que desejamos utilizar laços ou fazer contadores.

Exemplo

```
<!-- incrementa mais 1 na variável counter -->  
<variavelint valor="0" variavel="$(counter)" />  
<variavelintoperador operador="++" variavelorigem="$(counter)" />
```



learquivo

Sintaxe utilizada

```
<learquivo nomearquivo="$(var1)" chave="$(var2)" variaveldestino="$(var3)" >
```

Parâmetros

nomearquivo: Nome do arquivo em disco a ser aberto. (string)

chave: Chave do valor, que queremos ler do arquivo. (string)

variaveldestino: Variável onde ficará armazenado o valor lido. (string)

Pra que serve

Permite ler um arquivo no formato WALK dbfile ('chave=valor\nchave=valor\n'). Se por algum motivo o arquivo não existir, o nome da chave não existir no arquivo, ou qualquer outro erro no sistema de arquivos, vardestino será completado com um espaço em branco (' ').

Pode-se ler arquivos criados por instruções PosXML ou mesmo arquivos criados em outros programas e carregados no terminal de alguma maneira, desde que sigam o formato citado.

Exemplo

```
<variavelstr valor="config.dat" variavel="filename" />
<variavelstr valor="senhatecnica" variavel="key" />
<variavelstr valor=" " variavel="buffer" />
<learquivo nomearquivo="$(filename)" chave=$(key)" variaveldestino="$(buffer)" />
```

editaarquivo

Sintaxe utilizada

```
<editaarquivo nomearquivo="$(var1)" chave="$(var2)" valor="$(var3)" >
```

Parâmetros

nomearquivo: Nome do arquivo a ser editado ou criado. (string)

chave: Chave do valor, que queremos editar no arquivo. (string)

valor: Valor que será atribuído a chave especificado. (string)

Pra que serve

Permite escrever um arquivo no formato WALK dbfile ('chave=valor\nchave=valor\n').

Exemplo

```
<variavelstr valor="config.dat" variavel="filename" />
<variavelstr valor="senhatecnica" variavel="key" />
<variavelstr valor="12345 " variavel="buffer" />
<editaarquivo nomearquivo="$(filename)" chave="$(key)" valor="$(buffer)" />
```

excluirarquivo

Sintaxe utilizada

```
<excluirarquivo nomearquivo="$(var1)" />
```

Parâmetros

nomearquivo: Nome do arquivo a ser excluído. (string)

Pra que serve

Permite excluir um arquivo do sistema de arquivo do POS.

Exemplo

```
<variavelstr valor="config.dat" variavel="filename" />
<excluirarquivo nomearquivo="$(filename)" />
```



ajustadatahora

Sintaxe utilizada

```
<ajustadatahora datahora="yyyyMMddhhmmss" />
```

Parâmetros

datahora: Data/hora no formato yyyyMMddhhmmss. (string)

Pra que serve

Permite atualizar o relógio interno do terminal. Devemos passar o ano, mês, dia, hora, minuto, segundo 'datahora' no formato 'yyyyMMddhhmmss'.

Exemplo

```
<ajustadatahora datahora="20060101000000" />
```

pegadatahora

Sintaxe utilizada

```
<pegadatahora formato="yMdHms" variaveldestino="$ (var1)" />
```

Parâmetros

formato: Formato da data que vai ser armazenada. (string)

variaveldestino: Variável contendo a data no formato especificado. (string)

Pra que serve

Permite colocar o valor de data/hora atuais na variável string , variaveldestino , no formato especificado em formato. Em formato, qualquer caractere de formatação de data vai ser substituído pelo valor do relógio do terminal. Os caracteres reservados para formatação de data são:

yy: Ano com 4 dígitos

y: Ano com 2 dígitos

M: mês com 2 dígitos

d: dia com 2 dígitos

h: hora com 2 dígitos

m: minuto com 2 dígitos

s: segundos com 2 dígitos

Exemplo

```
<variavelstr valor=" " variavel="buffer" />
```

```
<pegadatahora formato="Data Hoje : M/d/yy h:m:s" variaveldestino="$ (buffer)" />
```

verificapapelimprensa

Sintaxe utilizada

```
<verificapapelimprensa variavelresultado="$ (var1)" />
```

Parâmetros

variavelresultado: Variável contendo o resultado da verificação do papel. (inteiro)

Pra que serve

Permite verificar se o compartimento de papel ainda contém papel. Útil para avisar quando o papel acabou e não deixar o operador prosseguir com a operação, enquanto não trocar o papel. Se ainda houver papel no compartimento variavelresultado será igual a 1. Se o papel acabou variavelresultado ficará com o valor 0.

Exemplo

```
<variavelint valor="-1" variavel="temPapel" />
```

```
<verificapapelimprensa variavelresultado="temPapel" />
```

```
<if variavel="$ (temPapel)" operador="igual" valor="0">  
  <display linha="0" coluna="0" mensagem=" Papel acabou ... " />
```

```
</if>
```

```
<if variavel="$ (temPapel)" operador="igual" valor="1">  
  <display linha="0" coluna="0" mensagem=" Ainda tem papel ... " />
```

```
</if>
```



walk

plano

bê

ultimapagina

Sintaxe utilizada

```
<ultimapagina />
```

Pra que serve

Permite recarregar e executar a última página de comandos recebido pelo terminal, enviados pelo host. Esta última página sempre fica armazenada na memória do terminal até que se desligue o mesmo. Quando chamada, esta instrução executa a última página, e em seguida a execução do walk termina. Muito útil, para escrevermos lógicas de reimpressão caso a última página seja um ticket enviado para o terminal.

Exemplo

```
<ultimapagina />
```

digitadecimal

Sintaxe utilizada

```
<digitadecimal variavel="$(xxx)" linha="n" coluna="n" mensagem="xxxxxxx" />
```

Parâmetros

variavel: Variável onde ficará armazenado o valor digitado. (string)

linha: Linha que vai ser exibida a mensagem. (inteiro)

coluna: Coluna que vai ser exibida a mensagem. (inteiro)

mensagem: Mensagem que vai ser exibida no display. (string)

Pra que serve

O comando `digitadecimal` aguarda o operador do terminal digitar um valor com ponto (.) através do teclado do terminal utilizando a tecla * para inserir o ponto. Exibe uma mensagem em linha e coluna especificadas. O valor digitado é armazenado em uma variável string de memória do terminal POS, incluindo o ponto.

Exemplo

```
<variavelstr valor="1.00" variavel="valor" />
```

```
<digitadecimal variavel="$(valor)" linha="1" coluna="1" mensagem="Kilogramas:" />
```

somastring

Sintaxe utilizada

```
<somastring valor1="hello " valor2="world" variaveldestino="$(var3)" />
```

Parâmetros

valor1: Valor a ser concatenado. (string)

valor2: Valor a ser concatenado. (string)

variaveldestino: Variável contendo o resultado da concatenação. (string)

Pra que serve

Concatena duas strings, especificadas em valor1 e valor2, e após juntar os dois valores coloca o resultado na variável especificada em variaveldestino.

Exemplo

```
<variavelstr valor="AAA" variavel="var1" />
```

```
<variavelstr valor="BBB" variavel="var2" />
```

```
<variavelstr valor=" " variavel="var3" />
```

```
<somastring valor1="$(var1)" valor2="$(var2)" variaveldestino="$(var3)" />
```



operacaomatematica

Sintaxe utilizada

```
<operacaomatematica valor1="var1" operador="+ - * / ^ %" valor2="var2" variaveldestino="$(var3)" />
```

Parâmetros

valor1: Primeiro valor da operação matemática. (string ou inteiro)

operador: Operador da operação matemática, sendo : +, -, *, /, %, ^ . (string)

valor2: Segunda valor da operação matemática. (string ou inteiro)

variaveldestino: Variável contendo o resultado da operação matemática. (string ou inteiro)

Pra que serve

Realiza uma operação matemática entre valor1 e valor2 e coloca seu resultado em variaveldestino. Para realizar operações com pontos flutuantes devemos utilizar em variaveldestino uma variável do tipo string. Devemos sempre obdecer a regra de utilizar o mesmo tipo de variáveis nos parâmetros valor1, valor2, e variaveldestino.

Os operadores matemáticos são:

+ : soma

- : subtração

* : multiplicação

/ : divisão

^ : exponenciação

% : resto da divisão

Exemplo

```
<variavelstr valor="1.2" variavel="var1" />
```

```
<variavelstr valor="3.4" variavel="var2" />
```

```
<variavelstr valor=" " variavel="var3" />
```

```
<operacaomatematica valor1="$(var1)" operador="+" valor2="$(var2)" variaveldestino="$(var3)" />
```

imprimacodigodebarras

Sintaxe utilizada

```
<imprimacodigodebarras horizontal="0" numero="123456" />
```

Parâmetros

horizontal: Coloque 1 se você deseja que o código de barras seja impresso na horizontal. (inteiro)

numero: Número do código de barras a ser impresso. (string)

Pra que serve

Imprime o código de barras no formato I25 , especificado na variável string numero. O formato I25 é o padrão adotado pela FEBRABAN ficando possível imprimir boletos de cobrança e qualquer outro título. O parâmetro numero deve ter o numero de caracteres par, pois é exigido pelo formato I25. Exemplo: "1234" funcionaria e imprimiria o código de barras normalmente. "123" não funcionaria, pois não conseguiríamos codificar no formato exigido pelo padrão I25.

Exemplo

```
<imprimacodigodebarras horizontal="0" numero="123456" />
```

esperateclatimeout

Sintaxe utilizada

```
<esperateclatimeout segundos="10" />
```

Parâmetros

segundos: Segundos que a instrução vai aguardar antes de continuar a execução. (inteiro)

Pra que serve

O comando esperateclatimeout é basicamente igual a instrução [esperatecla](#). Aguarda o operador do terminal pressionar qualquer tecla no terminal para continuar alguma operação, com a diferença que podemos especificar em segundos o tempo que a instrução irá aguardar até que seja pressionado uma tecla para prosseguir a ação.

Exemplo

```
<esperateclatimeout segundos="10" />
```




openserialport

Sintaxe utilizada

```
<openserialport porta="COM1" velocidade="115200" configuração="A8N1" variavelretorno="$ret" />
```

Parâmetros

porta: Nome da porta serial. Pode ser: COM1,COM2,COM3,COM4,COM5. (string)

velocidade: Velocidade da porta serial. Pode ser:

300,600,1200,2400,4800,9600,19200,38400,57600,115200,12000,14400,28800,33600. (string)

configuracao: Configuracao da porta serial. Pode ser: A7E1,A7N1,A7O1,A8E1,A8N1,A8O1. (string)

variavelretorno: Retorna -2 , se algum parametro de configuração está incorreto. Qualquer valor menor ou igual a zero, ocorreu algum erro no processo de abertura da porta serial. Se abriu com sucesso retorna a handle para manipular a porta serial através das instruções:readserialport,writeserialport e closeserialport. (inteiro)

Pra que serve

Abre a porta serial para comunicação com periféricos externos.

Exemplo

```
<variavelint valor="0" variavel="ret" />
```

```
<openserialport porta="COM1" velocidade="115200" configuração="A8N1" variavelretorno="$ret" />
```

readserialport

Sintaxe utilizada

```
<readserialport variavelhandle="$var1" variavelbuffer="$var2" bytes="10" timeout="2000" variavelretorno="$ret" />
```

Parâmetros

variavelhandle: Variavel retornada em openserialport. Deve ser diferente e maior que zero. (inteiro)

variavelbuffer: Variavel que armazenará o buffer lido pela porta serial. (string)

bytes: Quantidade de bytes que vai ser lido da porta serial. (inteiro)

timeout: Tempo em milisegundos que a instrução vai esperar para receber alguma coisa da porta serial. (inteiro)

variavelretorno: Quantidade de bytes lidos da porta serial. Se retornou -1, é porque não foi lida nenhuma informação e o tempo especificado em timeout foi atingido. (inteiro)

Pra que serve

Lê um buffer capturado pela porta serial que está plugada a um dispositivo externo, previamente aberta com a instrução [openserialport](#).

Exemplo

```
<variavelint valor="0" variavel="ret" />
```

```
<variavelint valor="0" variavel="readBytes" />
```

```
<variavelstr valor="" variavel="buf" />
```

```
<openserialport porta="COM1" velocidade="115200" configuração="A8N1" variavelretorno="$ret" />
```

```
<readserialport variavelhandle="$ret" variavelbuffer="$buf" bytes="10" timeout="2000" variavelretorno="$readBytes" />
```

writeserialport

Sintaxe utilizada

```
<writeserialport variavelhandle="$var1" buffer="Hello World" />
```

Parâmetros

variavelhandle: Variavel retornada em openserialport. Deve ser diferente e maior que zero. (inteiro)

buffer: Buffer que vai ser escrito na porta serial. (string)

Pra que serve

Escreve um buffer na porta serial, previamente aberta com a instrução [openserialport](#).

Exemplo

```
<variavelint valor="0" variavel="ret" />
```

```
<variavelint valor="0" variavel="readBytes" />
```

```
<variavelstr valor="" variavel="buf" />
```

```
<openserialport porta="COM1" velocidade="115200" configuração="A8N1" variavelretorno="$ret" />
```

```
<writeserialport variavelhandle="$ret" buffer="Hello World" />
```



closeserialport

Sintaxe utilizada

```
<closeserialport variavelhandle="$$(var1)" />
```

Parâmetros

variavelhandle: Variavel retornada em [openserialport](#). Deve ser diferente e maior que zero. (inteiro)

Pra que serve

Fecha a porta serial serial, previamente aberta com a instrução [openserialport](#).

Exemplo

```
<variavelint valor="0" variavel="ret" />
<variavelint valor="0" variavel="readBytes" />
<variavelstr valor="" variavel="buf" />
<openserialport porta="COM1" velocidade="115200" configuração="A8N1" variavelretorno="$$(ret)" />
<closeserialport variavelhandle="$$(ret)" />
```

string.length

Sintaxe utilizada

```
<string.length valor="Hello World" variavelretorno="$$(var1)" />
```

Parâmetros

valor: String que vai ser contada os caracteres ate o fim. (string)

variavelretorno: Tamanho da string. (string)

Pra que serve

Retorna o tamanho da string especificada.

Exemplo

```
<variavelint valor="0" variavel="tamanho" />
<string.length valor="Hello World" variavelretorno="$$(tamanho)" />
```

smartcard.inicialeit

Sintaxe utilizada

```
<smartcard.inicialeit slot="1" variavelretorno="$$(var1)" />
```

Parâmetros

slot: inteiro (customer = 1 , merchant1 = 2 , merchant2 = 3, merchant3 = 4)

variavelretorno: inteiro (0: erro ao iniciar leitor ou ler cartão, 1: leitor iniciado e ATR do cartão lido com sucesso)

Pra que serve

Inicia o leitor de smart card do terminal e tenta iniciar o cartão no slot informado, verificando o ATR do cartão.

Exemplo

```
<variavelint valor="0" variavel="ret" />
<smartcard.inicialeit slot="1" variavelretorno="$$(ret)" />
```

smartcard.cartaoinserto

Sintaxe utilizada

```
<smartcard.cartaoinserto slot="1" variavelretorno="$$(var1)" />
```

Parâmetros

slot: inteiro (customer = 1 , merchant1 = 2 , merchant2 = 3, merchant3 = 4)

variavelretorno: inteiro (0: cartão não está inserido, 1: cartão está inserido no leitor do terminal)

Pra que serve

Verifica se o cartão está inserido no leitor de smart card do terminal Pos.

Exemplo

```
<variavelint valor="0" variavel="ret" />
<smartcard.cartaoinserto slot="1" variavelretorno="$$(ret)" />
```



smartcard.fechaleitor

Sintaxe utilizada

```
<smartcard.fechaleitor slot="1" variavelretorno="$${var1}" />
```

Parâmetros

slot: inteiro (customer = 1 , merchant1 = 2 , merchant2 = 3, merchant3 = 4)
variavelretorno: inteiro (0: erro ao fechar leitor, 1: leitor fechado com sucesso)

Pra que serve

Finalizar o processo de comunicação com o cartão , e fecha o leitor de smart card do terminal Pos.

Exemplo

```
<variavelint valor="0" variavel="ret" />  
<smartcard.fechaleitor slot="1" variavelretorno="$${ret}" />
```

exit

Sintaxe utilizada

```
<exit />
```

Pra que serve

Encerra imediatamente a execução do aplicativo PosXML.

Exemplo

```
<variavelint valor="0" variavel="i" />  
<while variavel="$${i}" operador="menor" valor="10">  
  <display linha="0" coluna="0" mensagem=" Loop ..." />  
  <variavelintoperador operador="++" variavelorigem="$${i}" />  
  <exit /> <!-- sai da aplicaticacao -->  
</while>
```

enviarquivo

Sintaxe utilizada

```
<enviarquivo nomearquivo="teste.dat" caminhoremoto="\teste.dat" variavelretorno="$${var1}" />
```

Parâmetros

nomearquivo: string (Nome do arquivo que está na memória do terminal que desejamos enviar para o Walk Server 2)
caminharemoto: string (Nome do arquivo que será criado na pasta de upload do Walk Server 2. Deve iniciar com '\')
variavelretorno: inteiro
1: arquivo enviado com sucesso,
0: erro no encode do arquivo para base 64,
-1: arquivo não encontrado,
-2: erro na conexão no envio do buffer,
-3: erro no recebimento da confirmação de recebimento do buffer do arquivo pelo servidor,
-4: erro no pacote enviado para o servidor,
-5: erro na conexão com o servidor,
-6: erro na transferência do header ou do footer

Pra que serve

Envia um arquivo do terminal para o WALK Server, em um protocolo de comunicação próprio semelhante ao FTP. Se no envio não ocorrer nenhum erro o arquivo vai estar disponível no caminho remoto especificado, dentro da pasta de upload do WALK Server.

Exemplo

```
<enviarquivo nomearquivo="teste.dat" caminhoremoto="\teste.dat" variavelretorno="$${ret}" />
```



baixaarquivo

Sintaxe utilizada

```
<baixaarquivo nomearquivo="teste.dat" caminhoremoto="\teste.dat" excluiaaposdownload="0" variavelretorno="\$(var1)" />
```

Parâmetros

nomearquivo: string (Nome do arquivo que vai ser gravado na memória do terminal)

caminhoremoto: string (Nome do arquivo que está na pasta de download do Walk Server 2. Deve iniciar com '\')

excluiaaposdownload: inteiro (0: não exclui o arquivo da pasta do Walk Server 2 após o download, 1: exclui o arquivo da pasta de download do Walk Server 2)

variavelretorno: inteiro

1: arquivo baixado com sucesso,

0: erro no encode do arquivo para base 64,

-1: arquivo não encontrado no servidor ou não foi possível encoda-lo,

-2: erro na conexão no envio do buffer,

-3: erro no recebimento do buffer de conteúdo do arquivo no servidor,

-4: erro no pacote recebido do servidor,

-5: erro na conexão com o servidor,

-6: erro na transferência do header ou do footer,

-7: erro no envio da confirmação de download completo

Pra que serve

Baixa um arquivo do WALK Server para o terminal, em um protocolo de comunicação próprio semelhante ao FTP. O terminal vai buscar o arquivo dentro da pasta de downloads do WALK Server.

Exemplo

```
<baixaarquivo nomearquivo="teste.dat" caminhoremoto="\teste.dat" excluiaaposdownload="0" variavelretorno="\$(ret)" />
```

playmusic

Sintaxe utilizada

```
<playmusic nomearquivo="music.dat" />
```

Parâmetros

nomearquivo: string (Nome do arquivo contendo uma música no formato RTTL)

Pra que serve

Toca a música especificada no arquivo definido em nomearquivo. Este arquivo tem que ser no formato RTTL.

Exemplo

```
<playmusic nomearquivo="music.dat" />
```



learquivobyindex

Sintaxe utilizada

```
<learquivobyindex nomearquivo="teste.db" index="0" variavelchave="$ (var1)" variavelvalor="$ (var2)" variavelretorno="$ (var3)" />
```

Parâmetros

nomearquivo: string (Nome do arquivo formato WALK dbfile)

index: inteiro (Index da chave que queremos analisar. Começa em 0, e vai aumentando de acordo com a quantidade de itens no arquivo)

variavelchave: string (Variavel que vai armazenar o valor da chave do item no index especificado)

variavelvalor: string (Variavel que vai armazenar o valor do item no index especificado)

variavelretorno: inteiro (0: item não encontrado, 1: item encontrado)

Pra que serve

Permite ler um arquivo no formato WALK dbfile , ex: 'chave="valor"\nchave="valor"\n'. Especificamos o index da chave que queremos pesquisas, e se a mesma existir no arquivo, variavelchave, e variavelvalor serão preenchidos com seus respectivos valores.

Se por algum motivo o arquivo não existir, o nome da chave não existir no arquivo, ou qualquer outro erro no sistema de arquivos, variavelretorno será igual a 0. Do contrário será igual a 1.

Pode-se ler arquivos criados por instruções PosXML ou mesmo arquivos criados em outros programas e carregados no terminal de alguma maneira, desde que sigam o formato citado.

Exemplo

```
<variavelstr valor="" variavel="key" />
```

```
<variavelstr valor="" variavel="value" />
```

```
<variavelint valor="0" variavel="ret" />
```

```
<learquivobyindex nomearquivo="teste.db" index="0" variavelchave="$ (key)" variavelvalor="$ (value)" variavelretorno="$ (ret)" />
```

incrementatahora

Sintaxe utilizada

```
<incrementatahora datahora="20070424164602" dias=5" formatoretorno="d/M/yy h:m:s" variavelretorno="$ (var)" />
```

Parâmetros

datahora: string (Data e hora no formato yyyyMMddhhmmss, onde yyyy (ano com 4 dígitos), MM (mês com 2 dígitos), dd (dia com 2 dígitos), hh (hora com 2 dígitos), mm (minutos com 2 dígitos) e ss (segundos com 2 dígitos))

dias: inteiro (Número de dias que vai ser adicionado a data especificada)

formatoretorno: string (Formato da data modificada. Os caracteres reservados são: yy (ano com 4 dígitos) ou y (ano com 2 dígitos) , M (mês) , d (dia), h (hora), m (minutos), s (segundos))

variavelretorno: string (Váriavel que vai armazenar a data com os dias acrescentados no formato especificado)

Pra que serve

Permite incrementar (somar) dias a uma data pré - estabelecida. Basta criar uma variável do tipo string que receberá o valor de retorno dos dias incrementados. Em datahora será informada a data base, com a qual você deseja efetuar o cálculo; em dias, o número de dias que serão acrescidos na data base; o formatoretorno recebe o formato como a nova data calculada será exibida; e por último, a variavelretorno, criada anteriormente, que receberá a data calculada.

Exemplo

```
<variavelstr valor="0" variavel="var"/>
```

```
<incrementatahora datahora="20070501165807" dias="5" formatoretorno="d/M/y h:m:s" variavelretorno="$ (var)"/>
```

```
<display coluna="0" linha="0" mensagem="$ (var)"/>
```



diferencadatahora

Sintaxe utilizada

```
<diferencadatahora datahora1="20070425121402" datahora2="20070423102578" variavelretorno="$ (var)"/>
```

Parâmetros

datahora1: string (Data e hora no formato yyyyMMddhhmmss, onde yyyy (ano com 4 dígitos), MM (mês com 2 dígitos), dd (dia com 2 dígitos), hh (hora com 2 dígitos), mm (minutos com 2 dígitos) e ss (segundos com 2 dígitos)). Esta data é a maior. Caso seja especificada uma data menor que datahora2, o resultado em segundos será negativo)

datahora2: string (Data e hora no formato yyyyMMddhhmmss)

variavelretorno: inteiro (Diferença em segundos entre as duas datas especificadas)

Pra que serve

Permite calcular a diferença entre duas datas especificadas em datahora1 e datahora2. Basta criar uma variável do tipo inteiro, que receberá o resultado da diferença das duas datas na forma de segundos, para ser exibida. Se a diferença entre as duas datas ultrapassar uma hora, a função indicará erro de datetime.

Exemplo

```
<variavelint valor="0" variavel="var"/>
<variavelstr valor="" variavel="var2"/>
<diferencadatahora datahora1="20070510155135" datahora2="20070510155030" variavelretorno="$ (var)"/>
<inttostring variavelint="$ (var)" variavelstr="$ (var2)"/>
<display coluna="0" linha="0" mensagem="$ (var2)"/>
```

mostrabitmap

Sintaxe utilizada

```
<mostrabitmap nomearquivo="$ (nomearquivo)" variavelretorno="$ (ret)"/>
```

Parâmetros

nomearquivo: string (Nome do arquivo bitmap que vai ser mostrado no display)

variavelretorno: inteiro (1: bitmap mostrado no display corretamente. 0: arquivo bitmap não encontrado. -1: bitmap com tamanho inválido. -2: bitmap não é monocromático)

Pra que serve

Permite mostrar um bitmap monocromático no visor do terminal, que pode ser usado para animações, layout, entre outras finalidades. Para isso, é necessário salvar o bitmap na pasta download e baixar o arquivo com a função baixaraquivo.

Exemplo

```
<variavelint valor="1" variavel="ret"/>
<variavelstr valor="teste.bmp" variavel="arquivobmp"/>
<baixaraquivo caminhoremoto="\teste.bmp" excluiaaposdownload="0" nomearquivo="$ (arquivobmp)" variavelretorno="$ (ret)"/>
<mostrabitmap nomearquivo="$ (arquivobmp)" variavelretorno="$ (ret)"/>
<esperatecla/>
```

imprimabitmap

Sintaxe utilizada

```
<imprimabitmap nomearquivo="$ (nomearquivo)" variavelretorno="$ (ret)"/>
```

Parâmetros

nomearquivo: string (Nome do arquivo bitmap que vai ser impresso)

variavelretorno: inteiro (1: bitmap impresso corretamente. 0: arquivo bitmap não encontrado. -1: bitmap com tamanho inválido. -2: bitmap não é monocromático)

Pra que serve

Permite imprimir um bitmap monocromático, após o mesmo ter sido baixado da pasta download.

Exemplo

```
<variavelint valor="1" variavel="ret"/>
<variavelstr valor="teste.bmp" variavel="arquivobmp"/>
<baixaraquivo caminhoremoto="\teste.bmp" excluiaaposdownload="0" nomearquivo="$ (arquivobmp)" variavelretorno="$ (ret)"/>
<imprimabitmap nomearquivo="$ (arquivobmp)" variavelretorno="$ (ret)"/>
<esperatecla/>
```



letecla

Sintaxe utilizada

```
<letecla milisegundos="$(milisegs)" variavelretorno="$(ret)"/>
```

Parâmetros

milisegundos: inteiro (Milisegundos que a instrução vai aguardar antes de continuar a execução)

variavelretorno: string (Tecla pressionada pelo usuário. As teclas padrões que todos terminais possuem são: KEY_0 , KEY_1, KEY_2, KEY_3, KEY_4, KEY_5, KEY_6, KEY_7, KEY_8, KEY_9, KEY_POUND, KEY_STAR, KEY_CLEAR, KEY_CANCEL, KEY_ENTER. As teclas opcionais que variam de terminal pra terminal podem ir de KEY_EXTRA1 até KEY_EXTRA99)

Pra que serve

O comando letecla é basicamente igual à instrução esperateclatimeout. Ele aguarda o operador do terminal pressionar qualquer tecla no terminal para continuar alguma operação, com a diferença que em variavelretorno, temos a tecla que foi pressionada pelo operador.

Exemplo

```
<variavelint valor="2000" variavel="segs"/>
<variavelstr valor="" variavel="ret"/>
<variavelstr valor="KEY_1" variavel="tecla"/>
<display coluna="0" linha="0" mensagem="aperte uma tecla"/>
<letecla milisegundos="$(segs)" variavelretorno="$(ret)"/>
<if operador="igual" variavel="$(ret)" valor="$(tecla)">
<display coluna="0" linha="1" mensagem="numero 1 digitado!"/>
<esperatecla/>
</if>
```

unzipfile

Sintaxe utilizada

```
<unzipfile nomearquivo="$(nomearq)" variavelretorno="$(ret)"/>
```

Parâmetros

nomearquivo: string (nome do arquivo que vai ser unzipado)

variavelretorno: inteiro (0: descompressão feita com sucesso. -1: erro na descompressão)

Pra que serve

Permite descompactar um arquivo compactado no formato zip. Os arquivos dentro do .zip serão extraídos para o mesmo diretório onde se encontra o arquivo zipado especificado. Para isto, o arquivo deve estar dentro da pasta download, para que possa ser baixado pela função baixaarquivo e então ser unzipado.

Exemplo

```
<variavelint valor="0" variavel="ret"/>
<variavelstr valor="teste.zip" variavel="nomearq"/>
<baixaarquivo caminhoremoto="\teste.zip" excluiaposdownload="0" nomearquivo="$(nomearq)" variavelretorno="$(ret)"/>
<unzipfile nomearquivo="$(nomearq)" variavelretorno="$(ret)"/>
```

else

Sintaxe utilizada

```
<if operador="igual" variavel="$(num)" valor="1">
<else/>
.. condição lógica ..
</if>
```

Pra que serve

O comando else é utilizado dentro de um if, como uma condição lógica (se não).

Exemplo

```
<variavelint valor="0" variavel="num"/>
<if operador="igual" variavel="$(num)" valor="1">
<display coluna="0" linha="0" mensagem="Variavel igual a 1"/>
<esperatecla/>
<else/>
<display coluna="0" linha="0" mensagem="Variavel diferente de 1"/>
<esperatecla/>
</if>
```



conversaointeiro

Sintaxe utilizada

```
<conversaointeiro numero="$ (num)" base="$ (base)" tamanhoretorno="$ (tamanho)" variavelretorno="$ (ret)"/>
```

Parâmetros

numero: inteiro (Número que vai ser convertido)

base: inteiro (2: binário, 8: octal, 16: hexadecimal)

tamanhoretorno: inteiro (Tamanho do resultado. Se o número convertido for menor, então vai ser completado com zeros a esquerda)

variavelretorno: string (Variável onde ficará armazenado o número convertido)

Pra que serve

Permite converter um número inteiro em um número binário, octal ou hexadecimal.

Exemplo

```
<variavelint valor="12" variavel="num"/>
```

```
<variavelint valor="2" variavel="base"/>
```

```
<variavelint valor="10" variavel="tamanho"/>
```

```
<variavelstr valor="" variavel="ret"/>
```

```
<conversaointeiro numero="$ (num)" base="$ (base)" tamanhoretorno="$ (tamanho)" variavelretorno="$ (ret)"/>
```

```
<display coluna="0" linha="0" mensagem="$ (ret)"/>
```

```
<esperatecla/>
```

iso8583.iniciatabelacampos

Sintaxe utilizada

```
<iso8583.iniciatabelacampos nomearquivo="$ (arquivo)" variavelretorno="$ (ret)"/>
```

Parâmetros

nomearquivo: string (Nome do arquivo contendo o formato e o tamanho de todos os 128 campos da norma ISO8583)

variavelretorno: inteiro (0: tabela de campos iniciada corretamente, -802: não foi possível iniciar a tabela de campos)

Pra que serve

Inicia a tabela de campos ISO8583 de acordo com os campos especificados no arquivo definido no parâmetro nomearquivo.

Esta função deve ser chamada antes de iniciarmos o trabalho com as mensagens ISO8583.

iso8583.iniciamensagem

Sintaxe utilizada

```
<iso8583.iniciamensagem formato="$ (formato)" identificador="$ (id)" variavelmensagem="$ (var_msg)" variavelretorno="$ (ret)"/>
```

Parâmetros

formato: string (Formato da mensagem que desejamos montar, podendo ser : "ASCII" ou "BCD")

identificador: string (Identificador da mensagem com 4 dígitos, por exemplo: "0200" ou "0400")

variavelmensagem: string (Variável onde ficará armazenada a mensagem que está sendo montada)

variavelretorno: inteiro (0: mensagem iniciada com sucesso, -803: buffer overrun, -801: invalid parameter)

Pra que serve

Inicia a montagem de uma mensagem que será enviada ao host autorizador.



iso8583.analisamensagem

Sintaxe utilizada

```
<iso8583.analisamensagem formato="$(formato)" tamanho="$(tam)" variavelmensagem="$(var_msg)" variavelidentificador="$(var_id)" variavelretorno="$(ret)"/>
```

Parâmetros

formato: string (Formato da mensagem que desejamos analisar, podendo ser : "ASCII" ou "BCD")

tamanho: inteiro (Tamanho da mensagem que vamos analisar)

variavelmensagem: string (Variável onde está localizada a mensagem que será analisada)

variavelidentificador: string (Variável onde ficará armazenado o identificador da mensagem, se a análise foi realizada com sucesso)

variavelretorno: inteiro (0: mensagem analisada com sucesso, -806: erro na mensagem que foi analisada)

Pra que serve

Inicia o processo de análise e decomposição dos campos da mensagem especificada.

iso8583.finalizamensagem

Sintaxe utilizada

```
<iso8583.finalizamensagem variaveltamanho="$(tam)" variavelretorno="$(ret)"/>
```

Parâmetros

variaveltamanho: inteiro (Variável onde ficará armazenado o tamanho da mensagem finalizada, pronta para ser enviada)

variavelretorno: inteiro (0: mensagem finalizada com sucesso, -803: buffer overrun, -802: function error)

Pra que serve

Finaliza a montagem de uma mensagem a ser enviada ao host autorizador, iniciada por iso8583.iniciamensagem.

iso8583.inserecampo

Sintaxe utilizada

```
<iso8583.inserecampo numerocampo="$(n_campo)" tipo="$(tipo)" valor="$(val)" variavelretorno="$(ret)"/>
```

Parâmetros

numerocampo: inteiro (Número do campo que vamos inserir na mensagem. Podemos especificar de 2 a 128)

tipo: string (Tipo do campo que vamos inserir na mensagem, podendo ser: "string" ou "integer")

valor: string ou inteiro (Valor do campo)

variavelretorno: inteiro (0: campo inserido com sucesso, -801: invalid parameter, -802: function error, -803: buffer overrun, -804: field error)

Pra que serve

Inserir um campo na mensagem a ser enviada, iniciada por iso8583.iniciamensagem. Os campos devem ser inseridos sempre em ordem crescente.

iso8583.pegacampo

Sintaxe utilizada

```
<iso8583.pegacampo numerocampo="$(n_campo)" tipo="$(tipo)" variavelvalor="$(val)" variavelretorno="$(ret)"/>
```

Parâmetros

numerocampo: inteiro (Número do campo que vamos pegar da mensagem. Podemos especificar de 2 a 128)

tipo: string (Tipo do campo que vamos pegar da mensagem, podendo ser: "string" ou "integer")

variavelvalor: string ou inteiro (Variável onde será armazenado o valor do campo)

variavelretorno: inteiro (0: campo inserido com sucesso, -801: invalid parameter, -802: function error, -803: buffer overrun, -805: no field, -806: msg error)

Pra que serve

Extrai um campo de uma mensagem previamente analisada através da iso8583.analizamensagem. Os campos devem ser extraídos sempre em ordem crescente.



string.charat

Sintaxe utilizada

```
<string.charat string="$ (str)" character_index="$ (index)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (Uma string)

character_index: inteiro (Index indicando uma posição na string)

variavelretorno: string (Character da posição especificada)

Pra que serve

Extrai um character da string especificada.

string.trim

Sintaxe utilizada

```
<string.trim string="$ (str)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (Uma string)

variavelretorno: string (String sem espaços em branco na esquerda ou na direita)

Pra que serve

Elimina os espaços em branco da esquerda e da direita da string especificada.

string.find

Sintaxe utilizada

```
<string.find string="$ (str)" substring="$ (sub_str)" start="$ (index)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vamos procurar pela substring especificada)

substring: string (Valor que vamos procurar na string especificada)

start: inteiro (Index indicando uma posição para iniciarmos a procura na string especificada)

variavelretorno: inteiro (-1: substring não encontrada, > -1: posição na string da substring encontrada)

Pra que serve

Procura por uma substring na string especificada.

string.replace

Sintaxe utilizada

```
<string.replace original_string="$ (str_orig)" old_substring="$ (str_val)" new_substring="$ (str_new)" variavelretorno="$ (ret)"/>
```

Parâmetros

original_string: string (String original)

old_substring: string (Valor que vai ser substituído na string original pela nova substring especificada)

new_substring: string (Valor que vai ser colocado no lugar do valor antigo, na string especificada)

variavelretorno: string (Variável na onde ficará armazenada a nova string com os valores especificados substituídos)

Pra que serve

Substitui todas as ocorrências da substring especificada na string original.



string.substring

Sintaxe utilizada

```
<string.substring string="$ (str)" start="$ (index)" length="$ (tamanho)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (Uma string)

start: inteiro (Index para a posição em string de onde desejamos iniciar a cópia da substring)

length: inteiro (Número de caracteres que desejamos copiar)

variavelretorno: string (Substring copiada de string)

Pra que serve

Permite pegar um pedaço da string especificada.

string.elements

Sintaxe utilizada

```
<string.elements string="$ (str)" delimiter="$ (dlmt)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vai ser dividida)

delimiter: string (Character que vai ser procurado na string e que vai ser utilizado para dividir a string em vários elementos, criando um "array")

variavelretorno: inteiro (Número de elementos divididos por delimiter encontrados na string especificada)

Pra que serve

Retorna quantos elementos existem na string especificada, separadas pelo character delimiter. Mesmo quando o character especificado em delimiter não existe na string, teremos 1 elemento.

string.insertat

Sintaxe utilizada

```
<string.insertat string="$ (str)" string_to_be_inserted="$ (str_insert)" element_index="$ (index)" delimiter="$ (dlmt)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vai ser dividida)

string_to_be_inserted: string (String que vai ser inserida em determinada posição da string especificada)

element_index: inteiro (Index para determinada posição no "array" criado pela string separada por delimiter)

delimiter: string (Character que vai ser procurado na string e que será utilizado para dividir a string em vários elementos, criando um "array")

variavelretorno: string (String original com o novo elemento inserido)

Pra que serve

Permite inserir um elemento em um index específico na string especificada, separada pelo character delimiter.

string.replaceat

Sintaxe utilizada

```
<string.replaceat string="$ (str)" new_element="$ (new_elemt)" element_index="$ (index)" delimiter="$ (dlmt)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vai ser dividida)

new_element: string (String que vai ser inserida em determinada posição da string especificada)

element_index: inteiro (Index para determinada posição no "array", criado pela string separada por delimiter)

delimiter: string (Character que vai ser procurado na string e que será utilizado para dividir a string em vários elementos, criando um "array")

variavelretorno: string (String original com o elemento especificado substituído)

Pra que serve

Permite substituir um elemento em um index específico na string especificada separada pelo character delimiter.



string.elementat

Sintaxe utilizada

```
<string.elementat string="$ (str)" element_index="$ (index)" delimiter="$ (dlmt)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vai ser dividida)

element_index: inteiro (Index para determinada posição no "array", criado pela string separada por delimiter)

delimiter: string (Character que vai ser procurado na string e que será utilizado para dividir a string em vários elementos, criando um "array")

variavelretorno: string (Elemento especificado)

Pra que serve

Retorna um elemento em um index específico na string especificada, separada pelo character delimiter.

string.removeat

Sintaxe utilizada

```
<string.removeat string="$ (str)" element_index="$ (index)" delimiter="$ (dlmt)" variavelretorno="$ (ret)"/>
```

Parâmetros

string: string (String que vai ser dividida)

element_index: inteiro (Index para determinada posição no "array", criado pela string separada por delimiter)

delimiter: string (Character que vai ser procurado na string e que será utilizado para dividir a string em vários elementos, criando um "array")

variavelretorno: string (String original com o elemento especificado removido)

Pra que serve

Remove um elemento em um index específico na string especificada, separada pelo character delimiter.

network.send

Sintaxe utilizada

```
<network.send buffer="$ (buf)" tamanho="$ (tamanho)" variavelretorno="$ (ret)"/>
```

Parâmetros

buffer: string (Buffer que será enviado para o host atualmente conectado)

tamanho: inteiro (Tamanho do buffer que será enviado)

variavelretorno: inteiro (1: buffer enviado com sucesso, 0: erro no envio do buffer)

Pra que serve

Envia um buffer para o host que atualmente estamos conectados. É necessário já estar conectado a um host com a instrução pré-conectar.

network.receive

Sintaxe utilizada

```
<network.receive variavelbuffer="$ (buf)" tamanhomaximo="$ (tamanho_max)" variavelbytesrecebidos="$ (bytes)" variavelretorno="$ (ret)"/>
```

Parâmetros

variavelbuffer: string (Variável que irá armazenar o buffer recebido do host que atualmente estamos conectados)

tamanhomaximo: inteiro (Tamanho máximo do buffer que desejamos receber do host)

variavelbytesrecebidos: inteiro (Número de bytes recebidos do host)

variavelretorno: inteiro (1: buffer recebido com sucesso, 0: erro no recebimento do buffer)

Pra que serve

Recebe um buffer do host que atualmente estamos conectados. É necessário já estar conectado a um host com a instrução pré-conectar.



system.reinicia

Sintaxe utilizada

```
<system.reinicia/>
```

Pra que serve

Reinicia o terminal.

network.httprequest

Sintaxe utilizada

```
<network.httprequest url="$${str_url}" campos="$${str_campos}" savetofile="$${str_file}" variavelresponse="$${str_response}"  
variavelstatusCode="$${str_status}" variavelretorno="$${ret}">
```

Parâmetros

url: string (Url do host, contendo o nome do arquivo que vamos acessar, sem 'http://')

campos: string (Campos que vamos enviar no 'POST' no formato: "variable=value&variable2=value2". Se este campo estiver vazio, a funcao realizará um 'GET' na url especificado ao invés de 'POST'. Também podemos especificar neste campo, um arquivo a ser enviado. Para isso utilizado a nomenclatura 'file:nomedoarquivo' e em seguida as outras váriaveis do 'POST'. Só podemos especificar um arquivo por vez, e 'file:' deve ser especificado sempre como primeiro campo. Exemplo: 'file:inicio.posxml'&variable=value&variable2=value2)

savetofile: string (Nome do arquivo que vamos salvar a resposta do servidor HTTP. Se esta variavel estiver vazia a resposta então será colocada em variavelresponse.)

variavelresponse: string (Variavel que irá armazenar a resposta do servidor HTTP. Se especificamos o parâmetro savetofile, então está variavel terá um valor vazio)

variavelstatusCode: string (Código de status do protocolo HTTP retorno do servidor)

variavelretorno: inteiro

-9 : arquivo de upload invalido

-8 : download incompleto

-7 : erro ao iniciar gravação do arquivo

-6 : nao conseguiu conectar no servidor

-5 : tamanho maximo da resposta do servidor excedido. Mesmo assim copia para variavelresponse

-4 : erro no header recebido do servidor

-3 : url invalida

-2 : erro no recebimento do post

-1 : erro no envio do post

0 : se especificado savetofile, arquivo baixo com sucesso

>0 : tamanho da resposta do servidor que está armazenado em variavelresponse)

Pra que serve

Permite realizar um 'POST' ou 'GET' do protocolo HTTP. Possibilita baixar e enviar arquivos, receber e enviar informações de um servidor HTTP.

filesystem.filesize

Sintaxe utilizada

```
<filesystem.filesize nomearquivo="$${nome}" variavelretorno="$${ret}"/>
```

Parâmetros

nomearquivo: string (Nome do arquivo)

variavelretorno: inteiro (Tamanho do arquivo em bytes)

Pra que serve

Retorna o tamanho do arquivo especificado.

filesystem.space

Sintaxe utilizada

```
<filesystem.space dir="$${dir}" tipo="$${tipo}" variavelretorno="$${ret}"/>
```

Parâmetros

dir: string (Diretório que vamos analisar. Pode ser: 'I:' ou 'F:')

tipo: string (Tipo da informação sobre a memória. Pode ser: 'free', 'used', 'total' ou 'countfiles')

variavelretorno: inteiro (Espaço em bytes ou número de arquivos)

Pra que serve

Permite verificar o tamanho total da memória, o espaço utilizado, o espaço livre e o número de arquivos na memória.



filesystem.listfiles

Sintaxe utilizada

```
<filesystem.listfiles dir="$dir" nomearquivolista="$arquivo" variavelretorno="$ret"/>
```

Parâmetros

dir: string (Diretório que vamos analisar. Pode ser: 'I:' ou 'F:')

nomearquivolista: string (Nome do arquivo que irá conter o nome e o tamanho dos arquivos do diretório especificado)

variavelretorno: inteiro (-1: diretório vazio, 0: listagem de arquivos criada)

Pra que serve

Lista os arquivos do diretório especificado e salva em um arquivo no formato 'Walk Db File'.

convert.toint

Sintaxe utilizada

```
<convert.toint base="16" numero="FF" variavelretorno="$num" />
```

Parâmetros

base: string (2:binário, 10:decimal, 16:hexadecimal)

numero: string string (Número que vai ser convertido)

variavelretorno: inteiro (Variavel onde ficará armazenado o número convertido)

Pra que serve

Permite converter uma string contendo um número binário, decimal ou hexadecimal em um número inteiro.

network.webservicenew

Sintaxe utilizada

```
<network.webservicenew method="$method" soapAction="$soapAction" url="$url" xmlns="$xmlns"/>
```

Parâmetros

method: string (É o nome do method existente no WebService)

soapAction: string (soapAction é 'http://' + url + o nome do method)

url: string (Url do host, contendo o nome do arquivo que vamos acessar, sem 'http://')

xmlns: string (É o namespace utilizado no method no WebService)

Pra que serve

Esta função cria em memória uma nova estrutura XML para realizar uma requisição em um determinado WebService.

network.webserviceaddparameter

Sintaxe utilizada

```
<network.webserviceaddparameter method="$method" type="$type" variable="$variable" value="$value" variavelretorno="$variavelretorno"/>
```

Parâmetros

method: string (É o nome do method existente no WebService)

type: string (É o tipo da variável que será adicionada no method)

variable: string (É o nome da variável que será adicionada ao method)

value: string (Valor da variável adicionada ao method)

variavelretorno: inteiro (:-3 Tipo inválido, :-2 Estrutura XML inválida, :-1 Arquivo de envio não encontrado, :0 Sucesso.)

Pra que serve

Função que adiciona parâmetros dentro da estrutura XML criados na memória anteriormente.



network.webserviceconnect

Sintaxe utilizada

```
<network.webserviceconnect variavelresposta="$variavelresposta" variavelretorno="$variavelretorno" savetofile="$savetofile"/>
```

Parâmetros

Variavelresposta: string (Variável que irá armazenar a resposta do Webservice. Se especificarmos o parâmetro savetofile, então esta variável terá um valor vazio)

Variavelretorno: inteiro (

: -8 Method não encontrado,
: -7 Tamanho máximo da resposta do servidor excedido,
: -6 Erro no header recebido do servidor,
: -5 Erro no recebimento do post,
: -4 Erro no envio do post,
: -3 Não conseguiu conectar no servidor,
: -2 Estrutura xml inválida,
: -1 Arquivo de envio/configuração não encontrado,
: 0 Sucesso.)

Savetofile: string (Nome do arquivo que vamos salvar a resposta do Webservice. Se esta variável estiver vazia a resposta então será colocada em variavelresponse.)

Pra que serve

Essa função é responsável por conectar no Webservice, enviar a estrutura XML com os dados e receber a resposta.

predial

Sintaxe utilizada

```
<predial opcao="1" variavelstatus="$status" />
```

Parâmetros

opcao: inteiro (:1 O comando realiza uma pré-discagem. :2 O comando checa se a discagem foi completada.)

variavelstatus: inteiro (Variável que indica o status da discagem. :0 Discado :-1 Erro na discagem :-2 Valor inválido na variável opcao.)

Pra que serve

O comando efetua uma pré-discagem. Lembrando que esta instrução não conecta ou envia informação para o host, apenas faz a discagem.

network.checkgprssignal

Sintaxe utilizada

```
<network.checkgprssignal variavelstatus="$status" />
```

Parâmetros

variavelstatus: inteiro (Porcentagem do nível do sinal.)

Pra que serve

O comando network.checkgprssignal checa o status da conexão. Caso o terminal tenha suporte a conexão GPRS.

system.checkbattery

Sintaxe utilizada

```
<system.checkbattery variavelstatus="$status" />
```

Parâmetros

variavelstatus: inteiro (Porcentagem do nível do sinal.)

Pra que serve

O comando system.checkbattery checa o status da bateria do terminal. Caso o terminal possua bateria.



network.ping

Sintaxe utilizada

```
<network.ping variavelretorno="$$(retorno)" />
```

Parâmetros

variavelretorno: inteiro (:0 Sucesso. :-1 Sem conexão com o servidor. :-2 Falha no envio.)

Pra que serve

O comando network.ping checa o status do host.

system.beep

Sintaxe utilizada

```
<system.beep />
```

Pra que serve

O comando system.beep executa um sinal sonoro curto.

system.readcard

Sintaxe utilizada

```
<system.readcard variaveltecla="$$(tecla)" variavelcartao="$$(cartao)" timeout="5000" variavelretorno="$$(retorno)" />
```

Parâmetros

variaveltecla: string (Tecla pressionado pelo usuário. As teclas padrões que todos terminais possuem são: KEY_0, KEY_1, KEY_2, KEY_3, KEY_4, KEY_5, KEY_6, KEY_7, KEY_8, KEY_9, KEY_CLEAR, KEY_CANCEL, KEY_ENTER. As teclas opcionais que variam de terminal para terminal pode ir de KEY_EXTRA1 até KEY_EXTRA99.)

variavelcartao: string (Variável onde ficará armazenado o valor do cartão.)

timeout: inteiro (Tempo em milisegundos que a instrução vai aguardar antes de continuar a execução.)

variavelretorno: inteiro (:0 Botão digitado ou cartão passado com sucesso. :-1 Erro na leitura do cartão. :-2 Passou o tempo especificado na variavel timeout e nenhuma ação ocorreu.)

Pra que serve

Essa função espera que um cartão seja passado ou uma tecla seja pressionada, o tempo esperado é especificado em timeout. Caso nenhuma ação seja tomada, os valores de retorno padrão são: variaveltecla="KEY_CANCEL", variavelcartao="0".

network.hostdisconnect

Sintaxe utilizada

```
<network.hostdisconnect />
```

Pra que serve

Essa função efetua a desconexão entre o terminal e o host, mas o terminal ainda permanece conectado à rede.



iso8583.transactmessage

Sintaxe utilizada

```
<iso8583.transactmessage channel="NONE" header="$header" trailer="$trailer" isomsg="$isomsg" variavelresposta="$  
(resposta)" variavelretorno="$retorno" />
```

Parâmetros

channel: string ("NONE": nenhum size acrescentado "NAC": size de 2 bytes binários (x01 x00 ou 0000-0001 0000-0000)

"NCC": size de 2 bytes no formato BCD (x02 x56) "RAW": size de 4 bytes no formato binário (x00 x00 x01 x00 ou 0000-0000 0000-0000 0000-0001 0000-0000) "ASCII": size de 4 bytes no formato ASCII (x30 x32 x35 x36 ou "0256")

header: string (Header que será acrescentado ao início da mensagem ISO 8583 antes do envio.)

trailer: string (Trailer que será acrescentado ao início da mensagem ISO 8583 antes do envio.)

isomsg: string (Mensagem ISO 8583.)

variavelresposta: string (Mensagem ISO 8583 de resposta do autorizador.)

variavelretorno: inteiro (:> 0 Tamanho da mensagem de resposta. :-1 Channel desconhecido ou não implementado. :-2 Não conseguiu conectar ao host ou fazer a discagem. :-3 Falha no envio da mensagem. :-4 Falha ao receber o tamanho da mensagem de resposta. :-5 Falha ao receber a mensagem de resposta.)

Pra que serve

A instrução iso8583.transactmessage efetua a discagem, conecta ao host, acrescenta o size da mensagem, acrescenta o header e o trailer da mensagem ISO 8583 de acordo com os parâmetros previamente configurados, envia a mensagem ao host e faz 10 tentativas de recebimento da mensagem de resposta.