

# Expressões BPM

**A partir da versão 12.24, os processos de negócio (BPM) contam com um novo conceito: Propriedades definidas por expressão.**

Isso traz dinamismo para a colaboração de processos sem depender do PEX.

## 1 - O que é uma expressão?

Expressão pode ser entendida como uma **fórmula matemática** que vai retornar um valor, para **representar o estado de uma propriedade**. A expressão pode ser uma média de valores ou uma verificação se algum campo foi informado, por exemplo.

## 2 - Motivações

Os principais objetivos de implementar as expressões foram:

- Dar uma alternativa ao atual evento ao sair de campos (PEX), que fosse mais performática e mais simples.
- Avançar no conceito de Low-Code dentro do DOX.

## 3 - Identificador

Para possibilitar as expressões, foi criado o campo “Identificador”, cujo propósito é auto-explicativo: ser um **identificador único** para um campo de uma atividade.

Atualmente, nos eventos ao sair, é comum encontrar linhas de código parecidas com essa:

```
LValorAtual := aoFormularios.GetJSON('1').GetStr('TEXT0');
```

Nessa abordagem fica difícil entender o que está acontecendo analisando apenas essa linha de código: Afinal, **ao que corresponde o aoFormulario “1” e ao que corresponde o “TEXT0”?**

Partindo para uma abordagem em que o **campo pode ser identificado por um nome**, ao invés de um número, o código fica muito mais compreensível.

### 3.1 - Gerando o identificador

Ao atualizar o sistema para a versão 12.24 e abrir uma atividade de um processo já cadastrado, uma nova coluna vai aparecer na listagem de campos: a coluna “identificador”, já preenchida, de acordo com o “Código” de cada campo, da seguinte forma: \_1, \_2, \_3, etc. Isso porque esse campo não pode ficar vazio, nem pode se repetir na atividade e para usuários que não vão utilizar expressões, esse campo vai ser indiferente.

Na imagem abaixo é possível visualizar essa nova coluna:Image not found or type unknown

Ao criar um novo processo e adicionar campos manualmente, **o identificador vai ser preenchido automaticamente de acordo com o “Título” informado**, sendo possível alterá-lo manualmente também.

## 4 - Propriedades com expressão

Atualmente, as seguintes propriedades de um campo podem ter expressão:

- Visível
- Obrigatório
- Somente leitura
- Título
- Observação
- Valor

Vale destacar que essas propriedades vão estar disponíveis de acordo com o tipo do campo (texto, inteiro, grade, etc.) no cadastro do mesmo.

## 5 - Acessando propriedades

Dentro de uma expressão, para acessar uma propriedade de um campo do formulário, precisamos digitar o seguinte padrão:

```
IDENTIFICADOR.propriedadeSelecionada
```

Perceba o formato desse padrão:

- **Identificador** do campo (que é maiúsculo)
- **Ponto** para separar campo e propriedade
- Propriedade do campo em **camel case**

Exemplos nesse formato:

- ENDERECO.valor
- CEP.visivel
- NOME.somenteLeitura
- NUMERO.observacao
- CPF.titulo
- DATANASCIMENTO.obrigatorio

OBS: Muito cuidado ao digitar letras maiúsculas e minúsculas, pois a expressão é CASESENSITIVEL!

## 5.1 - Exemplo acessando propriedades

imagine um processo com 3 campos: "Número 1", "Número 2" e "Total", conforme imagem abaixo:

[image-1640292444178.png](#)

Image not found or type unknown

Imagine que queremos exibir a soma dos dois número no total. Podemos fazer isso via evento ao sair ou através de uma expressão. Através do PEX, precisaríamos adicionar o código nos eventos ao sair dos campos "Número 1" e "Número 2":

```
aoFormularios.GetJSON(' 3' )  .SetInt(' TEXT0' ,  
aoFormularios.GetJSON(' 1' ).GetInt(' TEXT0' ) +  
aoFormularios.GetJSON(' 2' ).GetInt(' TEXT0' ) );
```

Para fazer o mesmo através de expressões, o código é similar. Precisamos somar o valor do "Número 1" e do "Número 2". O identificador desses campos é respectivamente "NUMERO1" e "NUMERO2". Seguindo o formato para acessar campos e propriedades, temos a seguinte expressão para somar o valor dos dois campos:

```
NUMERO1.valor + NUMERO2.valor
```

Agora basta selecionar que o valor do campo "Total" é definido por expressão e colocar a expressão criada, conforme imagem abaixo:

[image-1640292546549.png](#)

Image not found or type unknown

## 6 - Linguagem de programação

As expressões são executadas em **JavaScript**. Caso você entenda sobre programação, você pode imaginar a expressão como o retorno de uma função da seguinte forma:

```
function executaExpressao() {  
    return expressao;  
}
```

**\*\*\*\* Isso significa que você pode retornar qualquer valor (Number, String, Boolean), mas caso você tente fazer um *if*, um *switch* ou tentar definir uma *var* no início da sua expressão, ela não vai funcionar.**

## 7 - Mudanças no cadastro

Ao criar ou editar um campo em um processo é possível ver que a tela está bem diferente. Na imagem abaixo é possível comparar o antes e depois do cadastro de um mesmo campo:

**Antes:**

image-1640292641471.png

Image not found or type unknown

**Agora:**

image-1640292664543.png

Image not found or type unknown

As seguintes alterações foram feitas:

- Propriedades booleanas (visível, obrigatório e somente leitura) passaram de caixas de seleção para um menu de seleção com as opções: sim, não e expressão.
- Propriedades textuais (título e observação) ganharam um menu de seleção com as opções: fixo e expressão.
- A propriedade valor ganhou um menu de seleção com as opções: informado e expressão.

Dessa forma é possível definir as propriedades como já era possível além de adicionar uma nova possibilidade, através de expressões. Caso o usuário selecione a opção “expressão” no menu de seleção, um campo para a entrada da expressão vai aparecer ao lado.

### 7.1 - Assistente

Para ajudar o usuário a montar suas expressões, foi criado um assistente que vai trazer grande parte do que é necessário:

- Os campos do processo e as propriedades disponíveis de acordo com o tipo de campo
- Operadores matemáticos
- Operadores lógicos
- Operadores relacionais
- Conversões para tipos de dados

Para acessar o assistente, é necessário focar no campo da expressão e pressionar **Ctrl + Espaço**. O assistente deve aparecer na tela dessa forma:

image-1640292781397.png

Image not found or type unknown

image-1640292799128.png

Image not found or type unknown

Ao selecionar alguma opção do assistente, o modelo de código já vai ser inserido na entrada da expressão. Ao selecionar um modelo de operação pronto, como por exemplo o modelo de “Adição”, a expressão montada vai ser:

```
/*campo1*/ + /*campo2*/
```

Nesse modelo, o “campo1” e “campo2” são apenas indicações de que você deve selecionar algum valor, como o valor de algum campo do formulário.

**ATENÇÃO:** Não se engane pelos caracteres `/* */`, as expressões **NÃO interpretam variáveis**. Nesse modelo, ao deixar o cursor de digitação em cima do “campo1” ou do “campo2” e abrir o assistente novamente, o texto `/*campo*/` do modelo vai sumir, para que algum valor possa ser selecionado a partir do assistente.

Utilizando expressão:

## 8 - Exemplo de uso das expressões

Imagine o seguinte formulário:

image-1640292989027.png

Image not found or type unknown

Esse formulário possui as seguintes regras:

- Caso o valor pago seja menor que a dívida, o campo “Em aberto” deve ficar visível e exibir o valor em aberto da dívida;
- Caso o valor pago seja maior que a dívida, o campo “Troco” deve ficar visível e exibir o valor que foi pago a mais.

### De que forma podemos expressar matematicamente essas regras?

O campo “Em aberto”:

- Vai ficar visível **SE**: a dívida **menos** o pagamento **for maior que 0**
- Vai exibir como valor: a dívida **menos** o pagamento

O campo “Troco”:

- Vai ficar visível **SE**: o pagamento menos a dívida for maior que 0
- Vai exibir como valor: o pagamento menos a dívida

A partir do momento que sabemos como definir essas expressões, basta passar para o cadastro do campo.

Na imagem abaixo, as expressões do campo “Em aberto”:

[image-1640293022346.png](#)

Image not found or type unknown

Já na imagem abaixo, as expressões do campo “Troco”:

[image-1640293053079.png](#)

Image not found or type unknown

## 9 - O que é possível fazer com expressões?

As tabelas abaixo foram montadas para exibir o que funciona ou não nas expressões, como um guia para quem já está familiarizado com o PEX e quer começar a montar expressões nos seus processos:

### Funciona;

- Acessar campos do formulário
- Definir regras para a própria propriedade que possui a expressão

### Não funciona

- Acessar variáveis
- Alterar outras propriedades deste e de outros campos do formulário
- Alterar propriedades do formulário / atividade
- Fazer requisições HTTP para retornar definir uma propriedade
- Interagir com o banco de dados
- Interagir com o sistema de arquivos
- Adicionar uma mensagem para o usuário

Para os programadores: Pense no **PEX** como **programação imperativa** e nas expressões como **programação declarativa**:

**No PEX, ao sair um campo qualquer, determinadas ações são realizadas**, por exemplo: alterar o valor de um campo, deixar outro campo invisível, deixar aquele campo obrigatório, etc.

**Já as expressões, podem ser pensadas como regras que definem uma propriedade**, por exemplo: este campo vai ser visível quando o valor do outro campo for menor que 100; este campo vai ser obrigatório caso outro campo tenha algum valor informado.

São duas formas de pensar diferentes. Por exemplo, na expressão da propriedade visível de um campo, não é possível definir a propriedade somente leitura desse mesmo campo.

## 10 - Executando as expressões

Após configurar um processo com as expressões, é possível criar uma nova instância do processo normalmente.

Ao entrar na atividade para colaborar, as expressões já vão ser executadas, e a cada vez que um campo tiver seu valor alterado, as expressões serão executadas novamente. Com isso, para o usuário, **as expressões sempre vão estar sempre refletindo o estado do formulário.**

É nesse momento que uma outra diferença do PEX para as expressões fica clara: as expressões vão ser executadas quase que instantaneamente, sem mostrar uma mensagem de “Aguarde...” para o usuário. Isso é possível porque **as expressões são executadas diretamente no navegador do usuário do Portal.** Isso permite uma performance muito maior e uma execução praticamente instantânea, além de aumentar a confiabilidade da execução, sem riscos de instabilidades no sistema.

### 10.1 - Erro!

Erros acontecem. Seja porque o identificador de um campo mudou e as expressões não foram atualizadas, seja por conta de um erro de sintaxe, seja porque as expressões são circulares.

#### **Há muitos jeitos de errar!**

Quando um erro acontece na execução de uma expressão, a execução das demais expressões é totalmente interrompida e uma mensagem de erro é exibida para o usuário.

Para os modeladores de processo: quando uma mensagem de erro aparecer pro usuário no Portal, a mensagem de erro original vai ser logada no console das ferramentas do desenvolvedor.

Na maioria dos navegadores as ferramentas do desenvolvedor podem ser acessadas através do **botão F12.**

Os erros das expressões são logados apenas no console do navegador, ou seja: no log do processo, que é acessível pelo Estúdio, esses erros não vão estar disponíveis para visualização.

Os exemplos de mensagem de erro abaixo:

[image-1640293252439.png](#)

Image not found or type unknown

### 10.2 - Expressões circulares

A maioria dos erros pode ser entendida facilmente, mas as expressões circulares exigem uma atenção especial. Uma expressão circular é uma expressão que depende dela mesma para ser executada.

Imagine um formulário que possua dois campos: CPF e CNPJ. Suponha que o campo “visível” de

ambos esteja definido por expressão, conforme as imagens a seguir:

[image-1640293290790.png](#)

Image not found or type unknown

[image-1640293312263.png](#)

Image not found or type unknown

[image-1640293337488.png](#)

Image not found or type unknown

Esse caso vai resultar em um erro circular. Isso porque para definir a visibilidade do campo CPF, que é definida por expressão, o Portal vai buscar a visibilidade do campo CNPJ. A visibilidade do campo CNPJ é definida por expressão, por isso o Portal vai tentar buscar a visibilidade do campo CPF. É um loop infinito, porque a expressão depende dela mesma.

Esse exemplo é mais simples, mas pode acontecer com 1, 2 ou n campos definidos por expressão.

## 10.2 - Mas e o PEX?

**O objetivo das expressões não é substituir o PEX**, mas oferecer uma alternativa para os casos mais simples.

Os eventos ao sair vão continuar sendo necessários para situações mais complexas e para ações assíncronas, como requisições HTTP, interações com o banco de dados ou com o sistema de arquivos.

**Eventos ao sair e expressões foram pensados para serem usados em conjunto.** Nos eventos ao sair de campos é possível ler e escrever as propriedades definidas por expressão, **porém ao sobrescrever o valor de uma propriedade definida por expressão, ela deixa de ser definida por expressão e assume o valor definido no PEX.**

**Utilizando PEX e expressões**

## 11 - Exemplo integrando PEX e Expressões

Imagine um processo que possui campos para informar um endereço, conforme a imagem a seguir:

[image-1640293524913.png](#)

Image not found or type unknown

Este processo possui um campo chamado “Endereço completo” que concatena as informações do endereço. Utilizando expressões é possível definir o que o valor do “Endereço completo” recebe,



conforme mostrado abaixo:

[image-1640293551328.png](#)

Image not found or type unknown

Se essa regra fosse implementada via evento ao sair, seria necessário copiar o código que define o valor do “Endereço completo” ao sair dos campos “Endereço” e “Número”, **duplicando o código e aumentando a chance de bugs**, caso a regra que define o formato de exibição mude.

Imagine agora que foram adicionados os campos “Estado” e “Município”. Para alterar a informação exibida no “Endereço completo”, basta alterar a expressão para incluir esses campos. Se fossemos fazer isso via PEX, seria necessário alterar o código já existente do “Endereço” e “Número”, além de incluir no “Estado” e no “Município”.

Perceba que o processo possui um campo CEP. Podemos utilizar um evento ao sair desse campo para buscar via requisição HTTP as informações do CEP informado e sobrescrever as informações dos campos “Endereço”, “Número”, “Estado” e “Município”, deixando esses campos somente leitura.

Veja como nesse exemplo conseguimos extrair o melhor das expressões, através de uma regra simples que define o valor de um campo, e do PEX, para buscar informações através de uma requisição HTTP.

## 12 - Aplicando boas práticas

As boas práticas das expressões são as mesmas que se aplicam para o **JavaScript**. As boas práticas em geral reduzem a complexidade ou redundância do código. Alguns exemplos são:

Boas práticas:

- CPF.visivel
- !CPF.visivel
- IDADE.valor === 20
- NUM1.valor || NUM2.valor
- NUM.valor || 10
- true
- CNPJ.somenteLeitura
- NUM1.valor > 10 && NUM2.valor > 10
- ['joão', 'pedro'].includes(NOME.valor)

Má prática:

- CPF.visivel === true
- CPF.visivel === false
- IDADE.valor == 20
- NUM1.valor ? NUM1.valor : NUM2.valor

- `NUM.valor !== null ? NUM.valor : 10`
- `1 === 1`
- `CNPJ.somenteLeitura ? true : false`
- `(NUM1.valor > 10) && (NUM2.valor > 10)`
- `NOME.valor === 'joão' || NOME.valor === 'pedro'`

---

Revisão #3

Criado 23 December 2021 17:37:43 por Nicolly Andrielly

Atualizado 11 March 2022 16:26:00 por Nicolly Andrielly