[Form] Quais são as funções nativas do AgilityFlow disponíveis no JavaScript para utilizar em um Formulário? formContext

O módulo `formContext` fornece uma série de funções utilitárias para manipulação de formulários, campos, datas, mensagens e outras funcionalidades no contexto de um formulário personalizado. Abaixo estão as funcionalidades disponíveis, organizadas por categoria.

Verificações de Contexto

parentFrame

Identifica e armazena a referência do iframe pai.

```
formContext.parentFrame
```

Com essa opção você pode ter acesso ao contexto do form pai que abriu o form atual por exemplo. Sendo assim você poderia utilizar todas as funções disponíveis no form pai, exemplo:

```
formContext.parentFrame.formContext.field.getValue("xxx")
```

isMobile()

Verifica se o dispositivo é móvel.

- **Retorno:**
  - `true` se for um dispositivo móvel.
  - `false` caso contrário.
- **Exemplo:**

```
if (formContext.isMobile()) {
    console.log("Dispositivo móvel detectado.");
}
```

---

isPublicPortal()

Verifica se o formulário está sendo exibido em um portal público.

- **Retorno:**
  - `true` se for um portal público.
  - `false` caso contrário.
- **Exemplo:**

```
if (formContext.isPublicPortal()) {
    console.log("Formulário em portal público.");
}
```

---

isPublicForm()

Verifica se o formulário é público (mesma lógica de `isPublicPortal`).

- **Retorno:**
  - `true` se for um formulário público.
  - `false` caso contrário.
- **Exemplo:**

```
if (formContext.isPublicForm()) {
    console.log("Formulário público detectado.");
}
```

Redirecionamento

redirectTo(url)

Redireciona para uma URL especificada.

- **Parâmetros:**
  - `url` (String): URL para redirecionamento.
- **Exemplo:**

```
formContext.redirectTo("https://www.exemplo.com");
```

Abrir Lightbox

openLightBox(link, tamanholightbox, titulo)

Abre uma lightbox com o conteúdo da URL especificada.

- **Parâmetros:**
  - `link` (String): URL a ser carregada na lightbox.
  - `tamanholightbox` (String): Tamanho da lightbox. Valores possíveis: `'p'`, `'m'`, `'g'`, `'fullscreen'`, `'fullscreen-headerless'`.
  - `titulo` (String): Título da lightbox.
- **Exemplo:**

```
formContext.openLightBox("https://www.exemplo.com", "m", "Título da Lightbox");
```

Carregamento de Arquivos JavaScript

addJavascriptFile(options)

Adiciona um arquivo JavaScript à página de forma assíncrona e executa um callback após o carregamento.

- **Parâmetros:**
  - `options` (Object): Objeto com as propriedades:
    - `file_url` (String): URL do arquivo JavaScript.
    - `callback_onload` (Function): Função a ser executada após o carregamento do arquivo.
- **Exemplo:**

```
formContext.addJavascriptFile({
    file_url: "https://www.exemplo.com/script.js",
    callback_onload: function() {
        console.log("Script carregado!");
    }
});
```

Busca de CEP

`buscaCEP.configurar(idCampoCep, idCamposPreenchiveis, callbacks)`

Configura a busca de CEP para campos específicos.

- **Parâmetros:**
  - `idCampoCep` (String): ID do campo de CEP.
  - `idCamposPreenchiveis` (String): ID dos campos a serem preenchidos com os dados do CEP.
  - `callbacks` (Object): Callbacks para eventos relacionados à busca de CEP.
- **Exemplo:**

```
formContext.buscaCEP.configurar("cep", "endereco", {
    onSuccess: function(data) {
        console.log("CEP encontrado:", data);
    }
});
```

Carregamento de Componentes CSHTML

loadCsHtmlComponent(componentName, options)

Carrega um componente CSHTML via AJAX.

- **Parâmetros:**
  - `componentName` (String): Nome do componente.
  - `options` (Object): Opções adicionais para o carregamento.
- **Exemplo:**

```
function loadPartial() {


    var cshtmlPartialName = "[coloque aqui o nome do partial cshtml que vc fez as regras de atualizar]";
    formContext.loadCsHtmlComponent(cshtmlPartialName,
        {
            placeholderId: 'result_cshtml', //parametro opcional, se o cshtml retornar algum html e vc quiser, vc pode colocar o retorno dentro de algum placeholder, div, etc..
            extraData: {
                idRegistro: $('#guidRegistro').val(),
                nome: $('#nome').val(),
                email: $('#email').val()
            },
            onSuccess: function (result) {
                formContext.msg.success("Executado com sucesso", "Confirmação");
            },
            onError: function (error) {
                formContext.msg.error("Execução não realizada", "Erro");
                console.log('error', error)
            }

        }
    )
}
```

Manipulação de URLs

url.getBaseUrl()

Retorna a URL base do sistema.

- **Retorno:**
  - String com a URL base.
- **Exemplo:**

```
console.log("URL base:", formContext.url.getBaseUrl());
```

---

url.getStaticFileUrl()

Retorna a URL base para arquivos estáticos.

- **Retorno:**
  - String com a URL de arquivos estáticos.
- **Exemplo:**

```
console.log("URL de arquivos estáticos:", formContext.url.getStaticFileUrl());
```

Idioma Atual

currentLanguage.get()

Retorna o idioma atual do sistema.

- **Retorno:**
  - String com o código do idioma (ex: `'pt-BR'`, `'en-US'`).
- **Exemplo:**

```
console.log("Idioma atual:", formContext.currentLanguage.get());
```

---

currentLanguage.isEnglish()

Verifica se o idioma atual é inglês.

- **Retorno:**
  - `true` se for inglês.
  - `false` caso contrário.
- **Exemplo:**

```
if (formContext.currentLanguage.isEnglish()) {
    console.log("Idioma atual é inglês.");
}
```

---

currentLanguage.getIIF_TextFromCurrentLanguage(text_ptBR, text_ENG, text_ESP)

Retorna o texto correspondente ao idioma atual.

- **Parâmetros:**
  - `text_ptBR` (String): Texto em português.
  - `text_ENG` (String): Texto em inglês.
  - `text_ESP` (String): Texto em espanhol.
- **Retorno:**
  - String com o texto no idioma atual.
- **Exemplo:**

```
const texto = formContext.currentLanguage.getIIF_TextFromCurrentLanguage("Olá", "Hello", "Hola");
console.log(texto); // Retorna "Olá" se o idioma for português.
```

Mensagens

O módulo `formContext.msg` fornece funções para exibir e ocultar mensagens de alerta, erro, sucesso e aviso no sistema. Abaixo estão as funcionalidades disponíveis.

Ocultar Mensagens

hide(forcarFechar)

Oculta todas as mensagens exibidas.

- **Parâmetros:**
  - `forcarFechar` (Boolean): Força o fechamento das mensagens.
- **Exemplo:**

```
formContext.msg.hide(true); // Oculta todas as mensagens, forçando o fechamento.
```

Exibir Mensagens

show(type, msg, title, options)

Exibe uma mensagem de alerta, erro, sucesso ou aviso.

- **Parâmetros:**
  - `type` (String): Tipo da mensagem. Valores possíveis:
    - `'warning'`: Mensagem de aviso.
    - `'error'`: Mensagem de erro.
    - `'success'`: Mensagem de sucesso.
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
  - `options` (Object): Opções adicionais para personalização da mensagem.
- **Exemplo:**

```
formContext.msg.show("success", "Operação realizada com sucesso!", "Sucesso", { timeout: 5000 });
```

---

warning(msg, title, options)

Exibe uma mensagem de aviso.

- **Parâmetros:**

- `msg` (String): Texto da mensagem.
- `title` (String): Título da mensagem.
- `options` (Object): Opções adicionais.

- **Exemplo:**

```
formContext.msg.warning("Atenção: Este campo é obrigatório.", "Aviso");
```

---

error(msg, title)

Exibe uma mensagem de erro.

- **Parâmetros:**
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
- **Exemplo:**

```
formContext.msg.error("Ocorreu um erro ao salvar o formulário.", "Erro");
```

---

success(msg, title, options)

Exibe uma mensagem de sucesso.

- **Parâmetros:**
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
  - `options` (Object): Opções adicionais.
- **Exemplo:**

```
formContext.msg.success("Formulário salvo com sucesso!", "Sucesso", { timeout: 3000 });
```

Funções Legadas de Mensagens

As funções abaixo são mantidas para compatibilidade com versões anteriores, mas é recomendado utilizar as funções acima (`show`, `warning`, `error`, `success`).

hideMsgs()

Oculta todas as mensagens exibidas.

- **Exemplo:**

```
formContext.msg.hideMsgs();
```

---

showMsgWarning(msg, title, options)

Exibe uma mensagem de aviso (legado).

- **Parâmetros:**
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
  - `options` (Object): Opções adicionais.
- **Exemplo:**

```
formContext.msg.showMsgWarning("Atenção: Este campo é obrigatório.", "Aviso");
```

---

showMsgError(msg, title)

Exibe uma mensagem de erro (legado).

- **Parâmetros:**
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
- **Exemplo:**

```
formContext.msg.showMsgError("Ocorreu um erro ao salvar o formulário.", "Erro");
```

---

showMsgSuccess(msg, title, options)

Exibe uma mensagem de sucesso (legado).

- **Parâmetros:**
  - `msg` (String): Texto da mensagem.
  - `title` (String): Título da mensagem.
  - `options` (Object): Opções adicionais.

- **Exemplo:**

```
formContext.msg.showMsgSuccess("Formulário salvo com sucesso!", "Sucesso", { timeout: 3000 });
```

Exemplos de Uso de Mensagem

Exibindo uma mensagem de sucesso:

```
formContext.msg.success("Dados salvos com sucesso!", "Sucesso", { timeout: 5000 });
```

Exibindo uma mensagem de erro:

```
formContext.msg.error("Erro ao processar a solicitação.", "Erro");
```

Ocultando todas as mensagens:

```
formContext.msg.hide(true);
```

Manipulação de Números

`number.convertString_toNumber(strValueToConvert, qtdCasasDecimais)`

Converte uma string formatada em número.

- **Parâmetros:**
  - `strValueToConvert` (String): Valor em formato de string.
  - `qtdCasasDecimais` (Number): Quantidade de casas decimais.
- **Retorno:**
  - Número convertido.
- **Exemplo:**

```
const numero = formContext.number.convertString_toNumber("1.000,50", 2);
console.log(numero); // Retorna 1000.50
```

---

number.convertNumber_toStringFormatted(numberToConvert, qtdCasasDecimais)

Converte um número em uma string formatada de acordo com o idioma.

- **Parâmetros:**
  - `numberToConvert` (Number): Número a ser convertido.
  - `qtdCasasDecimais` (Number): Quantidade de casas decimais.
- **Retorno:**
  - String formatada.
- **Exemplo:**

```
const texto = formContext.number.convertNumber_toStringFormatted(1000.50, 2);
console.log(texto); // Retorna "1.000,50" em português.
```

Manipulação de Datas

isCurrentMonthAndYear(data)

Verifica se a data fornecida pertence ao mês e ano atuais.

- **Parâmetros:**
  - `data` (String): Data no formato `DD/MM/YYYY`.
- **Retorno:**
  - `true` se a data for do mês e ano atuais.
  - `false` caso contrário.
- **Exceções:**
  - Lança um `TypeError` se a data estiver em um formato inválido.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.validation.isCurrentMonthAndYear("25/10/2023"); // Retorna true se outubro de 2023 f
```

---

compare(date1, compare, date2)

Compara duas datas com base no operador especificado.

- **Parâmetros:**
  - `date1` (String): Primeira data no formato `DD/MM/YYYY`.
  - `compare` (String): Operador de comparação. Valores possíveis:
    - `'greater'` ou `'>'`
    - `'greater-or-equal'` ou `'>='`
    - `'less'` ou `'<'`
    - `'less-or-equal'` ou `'<='`
    - `'equal'` ou `'=='`
  - `date2` (String): Segunda data no formato `DD/MM/YYYY`.
- **Retorno:**
  - `true` ou `false`, dependendo da comparação.
- **Exceções:**
  - Lança um `TypeError` se as datas estiverem em formato inválido ou se o operador de comparação não for reconhecido.
- **Exemplo:**
  javascript
  Copy
  ```
  formContext.datetime.validation.compare("25/10/2023", "greater", "20/10/2023"); // Retorna true.
  ```

---

compareDatetime(datetime1, compare, datetime2)

Compara duas datas e horários com base no operador especificado.

- **Parâmetros:**
  - `datetime1` (String): Primeira data e horário no formato `DD/MM/YYYY HH:mm`.
  - `compare` (String): Operador de comparação. Valores possíveis:
    - `'greater'` ou `'>'`
    - `'greater-or-equal'` ou `'>='`
    - `'less'` ou `'<'`
    - `'less-or-equal'` ou `'<='`
    - `'equal'` ou `'=='`
  - `datetime2` (String): Segunda data e horário no formato `DD/MM/YYYY HH:mm`.

- **Retorno:**
  - `true` ou `false`, dependendo da comparação.
- **Exceções:**
  - Lança um `TypeError` se os dados estiverem em formato inválido ou se o operador de comparação não for reconhecido.
- **Exemplo:**
  javascript
  Copy
  `formContext.datetime.validation.compareDatetime("25/10/2023 14:30", "greater", "25/10/2023 12:00"); // Re`

---

compareToday(compare, date2)

Compara a data atual com uma data fornecida.

- **Parâmetros:**
  - `compare` (String): Operador de comparação. Valores possíveis:
    - `'greater'` ou `'>'`
    - `'greater-or-equal'` ou `'>='`
    - `'less'` ou `'<'`
    - `'less-or-equal'` ou `'<='`
    - `'equal'` ou `'=='`
  - `date2` (String): Data no formato `DD/MM/YYYY`.
- **Retorno:**
  - `true` ou `false`, dependendo da comparação.
- **Exemplo:**
  javascript
  Copy
  `formContext.datetime.validation.compareToday("less", "30/10/2023"); // Retorna true se a data atual for mer`

Adição de Tempo

day(data, days)

Adiciona dias a uma data.

- **Parâmetros:**
  - `data` (String): Data no formato `DD/MM/YYYY` ou `DD/MM/YYYY HH:mm`.
  - `days` (Number): Número de dias a serem adicionados.

- **Retorno:**
  - Data resultante no mesmo formato da entrada.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.add.day("25/10/2023", 5); // Retorna "30/10/2023".
```

---

month(data, months)

Adiciona meses a uma data.

- **Parâmetros:**
  - `data` (String): Data no formato `DD/MM/YYYY` ou `DD/MM/YYYY HH:mm`.
  - `months` (Number): Número de meses a serem adicionados.
- **Retorno:**
  - Data resultante no mesmo formato da entrada.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.add.month("25/10/2023", 2); // Retorna "25/12/2023".
```

---

year(data, years)

Adiciona anos a uma data.

- **Parâmetros:**
  - `data` (String): Data no formato `DD/MM/YYYY` ou `DD/MM/YYYY HH:mm`.
  - `years` (Number): Número de anos a serem adicionados.
- **Retorno:**
  - Data resultante no mesmo

---

hour(data, hours)

Adiciona horas a uma data e horário.

- **Parâmetros:**

- `data` (String): Data e horário no formato `DD/MM/YYYY HH:mm`.
- `hours` (Number): Número de horas a serem adicionadas.

- **Retorno:**
  - Data e horário resultante no formato `DD/MM/YYYY HH:mm`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.add.hour("25/10/2023 14:30", 3); // Retorna "25/10/2023 17:30".
```

---

minute(data, minutes)

Adiciona minutos a uma data e horário.

- **Parâmetros:**
  - `data` (String): Data e horário no formato `DD/MM/YYYY HH:mm`.
  - `minutes` (Number): Número de minutos a serem adicionados.
- **Retorno:**
  - Data e horário resultante no formato `DD/MM/YYYY HH:mm`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.add.minute("25/10/2023 14:30", 15); // Retorna "25/10/2023 14:45".
```

Obtenção de Datas

getLastDayOfCurrentMonth()

Retorna o último dia do mês atual.

- **Retorno:**
  - Data no formato `DD/MM/YYYY`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.getLastDayOfCurrentMonth(); // Retorna "31/10/2023" se outubro for o mês atual.
```

---

getLastDayOfMonth(data)

Retorna o último dia do mês da data fornecida.

- **Parâmetros:**
  - `data` (String): Data no formato `DD/MM/YYYY`.
- **Retorno:**
  - Data no formato `DD/MM/YYYY`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.getLastDayOfMonth("25/10/2023"); // Retorna "31/10/2023".
```

Data e Horário Atuais

getDateNow()

Retorna a data e horário atuais.

- **Retorno:**
  - Objeto `Date` representando a data e horário atuais.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.getDateNow(); // Retorna a data e horário atuais.
```

---

getFormattedDateTime()

Retorna a data e horário atuais formatados.

- **Retorno:**
  - String no formato `DD/MM/YYYY HH:mm`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.getFormattedDateTime(); // Retorna "25/10/2023 14:30".
```

getFormattedDate()

Retorna a data atual formatada.

- **Retorno:**
  - String no formato `DD/MM/YYYY`.
- **Exemplo:**
  javascript
  Copy

```
formContext.datetime.getFormattedDate(); // Retorna "25/10/2023".
```

Manipulação de Campos

O módulo `formContext.field` fornece funções para manipular campos em formulários, como verificar assinaturas, controlar campos obrigatórios, gerenciar selects, ocultar/exibir campos, habilitar/desabilitar campos e definir/obter valores. Abaixo estão as funcionalidades disponíveis.

Campo de Assinatura

signature.isDefined(campokey)

Verifica se uma assinatura foi preenchida em um campo específico.

- **Parâmetros:**
  - `campokey` (String): ID do campo de assinatura.
- **Retorno:**
  - `true` se a assinatura foi preenchida.
  - `false` caso contrário.

- **Exemplo:**

```
if (formContext.field.signature.isDefined("assinatura")) {
    console.log("Assinatura preenchida.");
}
```

Campos Obrigatórios

required.ignoreRequired(campokey)

Ignora a validação de campo obrigatório para um campo específico.

- **Parâmetros:**
  - `campokey` (String): ID do campo.
- **Exemplo:**

```
formContext.field.required.ignoreRequired("campo1"); // Ignora a validação de campo obrigatório para "camp
```

---

required.unsetIgnoreRequired(campokey)

Remove a configuração de ignorar validação de campo obrigatório para um campo específico.

- **Parâmetros:**
  - `campokey` (String): ID do campo.
- **Exemplo:**

```
formContext.field.required.unsetIgnoreRequired("campo1"); // Remove a configuração de ignorar validação p
```

Campos Select Personalizados

customSelect.setDefaultStyle(selector)

Aplica o estilo padrão do sistema a um select personalizado.

- **Parâmetros:**
  - `selector` (String): Seletor do campo select.
- **Exemplo:**

```
formContext.field.customSelect.setDefaultStyle("meuSelect"); // Aplica o estilo padrão ao select com ID "meu
```

Manipulação de Itens em Campos Select Personalizados

select.items.disable(campokey, valuesToDisable)

Desabilita itens específicos em um campo select.

- **Parâmetros:**
  - `campokey` (String): ID do campo select.
  - `valuesToDisable` (String ou Array): Valor(es) do(s) item(ns) a ser(em) desabilitado(s).
- **Exemplo:**

```
formContext.field.select.items.disable("meuSelect", "valor1"); // Desabilita o item com valor "valor1".
formContext.field.select.items.disable("meuSelect", ["valor1", "valor2"]); // Desabilita os itens com valores "v
```

---

select.items.enable(campokey, valuesToDisable)

Habilita itens específicos em um campo select.

- **Parâmetros:**
  - `campokey` (String): ID do campo select.
  - `valuesToDisable` (String ou Array): Valor(es) do(s) item(ns) a ser(em) habilitado(s).
- **Exemplo:**

```
formContext.field.select.items.enable("meuSelect", "valor1"); // Habilita o item com valor "valor1".
formContext.field.select.items.enable("meuSelect", ["valor1", "valor2"]); // Habilita os itens com valores "valo
```

---

select.items.disableAll(campokey)

Desabilita todos os itens de um campo select.

- **Parâmetros:**
  - `campokey` (String): ID do campo select.
- **Exemplo:**

```
formContext.field.select.items.disableAll("meuSelect"); // Desabilita todos os itens do select.
```

---

select.items.enableAll(campokey)

Habilita todos os itens de um campo select.

- **Parâmetros:**
  - `campokey` (String): ID do campo select.
- **Exemplo:**

```
formContext.field.select.items.enableAll("meuSelect"); // Habilita todos os itens do select.
```

Visibilidade de Campos

hide(campokey)

Oculta um campo específico.

- **Parâmetros:**
  - `campokey` (String ou Array): ID do campo ou lista de IDs.
- **Exemplo:**

```
formContext.field.hide("campo1"); // Oculta o campo com ID "campo1".
formContext.field.hide(["campo1", "campo2"]); // Oculta os campos com IDs "campo1" e "campo2".
```

---

show(campokey)

Exibe um campo específico.

- **Parâmetros:**

- `campokey` (String ou Array): ID do campo ou lista de IDs.
- **Exemplo:**

```
formContext.field.show("campo1"); // Exibe o campo com ID "campo1".
formContext.field.show(["campo1", "campo2"]); // Exibe os campos com IDs "campo1" e "campo2".
```

---

hideAll()

Oculta todos os campos do formulário.

- **Exemplo:**

```
formContext.field.hideAll(); // Oculta todos os campos.
```

---

showAll()

Exibe todos os campos do formulário.

- **Exemplo:**

```
formContext.field.showAll(); // Exibe todos os campos.
```

Habilitação/Desabilitação de Campos

disable(campokey)

Desabilita um campo específico.

- **Parâmetros:**
  - `campokey` (String ou Array): ID do campo ou lista de IDs.
- **Exemplo:**

```
formContext.field.disable("campo1"); // Desabilita o campo com ID "campo1".
formContext.field.disable(["campo1", "campo2"]); // Desabilita os campos com IDs "campo1" e "campo2".
```

---

enable(campokey)

Habilita um campo específico.

- **Parâmetros:**
  - `campokey` (String ou Array): ID do campo ou lista de IDs.
- **Exemplo:**

```
formContext.field.enable("campo1"); // Habilita o campo com ID "campo1".
formContext.field.enable(["campo1", "campo2"]); // Habilita os campos com IDs "campo1" e "campo2".
```

---

disableAll()

Desabilita todos os campos do formulário.

- **Exemplo:**

```
formContext.field.disableAll(); // Desabilita todos os campos.
```

---

enableAll()

Habilita todos os campos do formulário.

- **Exemplo:**

```
formContext.field.enableAll(); // Habilita todos os campos.
```

Definição e Obtenção de Valores de Campos

setValue(campoKey, value, options)

Define o valor de um campo.

- **Parâmetros:**
  - `campoKey` (String): ID do campo.
  - `value` (String): Valor a ser definido.
  - `options` (Object): Opções adicionais.
- **Exemplo:**

```
formContext.field.setValue("campo1", "Valor exemplo"); // Define o valor do campo "campo1".
```

---

getValue(campoKey)

Obtém o valor de um campo.

- **Parâmetros:**
  - `campoKey` (String): ID do campo.
- **Retorno:**
  - Valor do campo.
- **Exemplo:**

```
const valor = formContext.field.getValue("campo1"); // Obtém o valor do campo "campo1".
```

---

getText(campoKey)

Obtém o texto exibido de um campo.

- **Parâmetros:**
  - `campoKey` (String): ID do campo.
- **Retorno:**
  - Texto do campo.
- **Exemplo:**

```
const texto = formContext.field.getText("campo1"); // Obtém o texto do campo "campo1".
```

---

getInt(campoKey)

Obtém o valor de um campo como número inteiro.

- **Parâmetros:**

    - `campoKey` (String): ID do campo.
- **Retorno:**
    - Valor inteiro.
- **Exemplo:**

```
const valorInteiro = formContext.field.getInt("campo1"); // Obtém o valor inteiro do campo "campo1".
```

---

getDecimal(campoKey)

Obtém o valor de um campo como número decimal.

- **Parâmetros:**
    - `campoKey` (String): ID do campo.
- **Retorno:**
    - Valor decimal.
- **Exemplo:**

```
const valorDecimal = formContext.field.getDecimal("campo1"); // Obtém o valor decimal do campo "campo1"
```

---

getFloat(campoKey)

Obtém o valor de um campo como número de ponto flutuante.

- **Parâmetros:**
    - `campoKey` (String): ID do campo.
- **Retorno:**
    - Valor de ponto flutuante.
- **Exemplo:**

```
const valorFloat = formContext.field.getFloat("campo1"); // Obtém o valor de ponto flutuante do campo "cam
```

---

getNumber(campoKey)

Obtém o valor de um campo como número (inteiro ou decimal, dependendo da máscara do campo).

- **Parâmetros:**

    - `campoKey` (String): ID do campo.
- **Retorno:**
    - Valor numérico.
- **Exemplo:**

```
const valorNumerico = formContext.field.getNumber("campo1"); // Obtém o valor numérico do campo "camp
```

Manipulação do Formulário

O objeto `formContext.form` gerencia diversas funcionalidades relacionadas ao formulário, incluindo salvamento, rascunhos, ações e manipulação da toolbar.

Estrutura do Objeto formContext.form

- `formContext.form`: Contém funcionalidades relacionadas ao formulário.
    - `save()`: Salva o formulário.
    - `saveDraft()`: Salva o formulário como rascunho.
    - `undoDraft()`: Desfaz um rascunho.
    - `getFormActionOnSubmit()`: Obtém o tipo de envio do formulário.
    - `invalidateForm(options)`: Inválida o submit do formulário.
    - `toolbar`: Manipula a toolbar do formulário.
    - `disableNativeSuccessMsg()`: Desabilita mensagens nativas de sucesso.
    - `forceToSetStateFormToChanged()`: Força o status do formulário para "alterado".
    - `isNew()`: Verifica se o formulário é novo.
    - `isDraft()`: Verifica se o formulário é um rascunho.
    - `action`: Define os tipos de ações possíveis no formulário.

Funções e Exemplos de Uso do formContext.form

`save()`

Salva o formulário.

```
formContext.form.save();
```

saveDraft()

Salva o formulário como rascunho.

```
formContext.form.saveDraft();
```

undoDraft()

Desfaz um rascunho.

```
formContext.form.undoDraft();
```

getFormActionOnSubmit()

Obtém o tipo de envio do formulário.

```
var action = formContext.form.getFormActionOnSubmit();
console.log(action);
```

invalidateForm(options)

Inválida o submit do formulário.

```
formContext.form.invalidateForm({ message: "Erro ao enviar!" });
```

toolbar.hide()

Esconde a toolbar.

```
formContext.form.toolbar.hide();
```

toolbar.show()

Exibe a toolbar.

```
formContext.form.toolbar.show();
```

toolbar.remove()

Remove a toolbar.

```
formContext.form.toolbar.remove();
```

disableNativeSuccessMsg()

Desabilita mensagens nativas de sucesso.

```
formContext.form.disableNativeSuccessMsg();
```

forceToSetStateFormToChanged()

Força o status do formulário para "alterado".

```
formContext.form.forceToSetStateFormToChanged();
```

isNew()

Verifica se o formulário é novo.

```
if (formContext.form.isNew()) {
    console.log("Formulário novo");
}
```

isDraft()

Verifica se o formulário é um rascunho.

```
if (formContext.form.isDraft()) {
    console.log("Formulário em rascunho");
}
```

Funções e Exemplos de Uso do formContext.form.action

action.isSave(formAction)

Verifica se a ação é de salvar.

```
if (formContext.form.action.isSave(1)) {
    console.log("Ação de salvar detectada");
}
```

action.isSaveDraft(formAction)

Verifica se a ação é de salvar como rascunho.

```
if (formContext.form.action.isSaveDraft(2)) {
    console.log("Ação de salvar rascunho");
}
```

action.isDiscardDraft(formAction)

Verifica se a ação é de descartar o rascunho.

```
if (formContext.form.action.isDiscardDraft(3)) {
    console.log("Ação de descartar rascunho");
}
```

action.isApproveFromWorkflow(formAction)

Verifica se a ação é de aprovar no workflow.

```
if (formContext.form.action.isApproveFromWorkflow(4)) {
    console.log("Ação de aprovação no workflow");
}
```

action.isRejectFromWorkflow(formAction)

Verifica se a ação é de rejeitar no workflow.

```
if (formContext.form.action.isRejectFromWorkflow(5)) {
    console.log("Ação de rejeição no workflow");
}
```

action.isReturnFromWorkflow(formAction)

Verifica se a ação é de retorno no workflow.

```
if (formContext.form.action.isReturnFromWorkflow(6)) {
    console.log("Ação de retorno no workflow");
}
```

action.isDelete(formAction)

Verifica se a ação é de deleção.

```
if (formContext.form.action.isDelete(7)) {
    console.log("Ação de deleção detectada");
}
```

action.isSaveChildForm(formAction)

Verifica se a ação é de salvar um formulário filho.

```
if (formContext.form.action.isSaveChildForm(8)) {
    console.log("Ação de salvar formulário filho");
}
```

action.isDeleteChildForm(formAction)

Verifica se a ação é de deletar um formulário filho.

```
if (formContext.form.action.isDeleteChildForm(9)) {
    console.log("Ação de deletar formulário filho");
}
```

Manipulação do Formulário Filho (Child Form)

formContext.childForm

datatable.refresh(tableid_relacaoFormularioPaiFilhoId)

Atualiza a tabela do formulário filho que está no formulário pai.

Exemplo de Uso:

```
formContext.childForm.datatable.refresh("tabela123");
```

---

datatable.totalRows(tableid_relacaoFormularioPaiFilhoId)

Retorna o número total de linhas da tabela do formulário filho.

Exemplo de Uso:

```
var total = formContext.childForm.datatable.totalRows("tabela123");
console.log("Total de linhas:", total);
```

---

datatable.getAllChildIds(tableid_relacaoFormularioPaiFilhoId)

Retorna todos os IDs dos formulários filhos presentes na tabela.

Exemplo de Uso:

```
var ids = formContext.childForm.datatable.getAllChildIds("tabela123");
console.log("IDs dos formulários filhos:", ids);
```

---

datatable.getAllChildIdsOrdered(tableid_relacaoFormularioPaiFilhoId)

Retorna todos os IDs dos formulários filhos respeitando a ordenação atual.

Exemplo de Uso:

```
var orderedIds = formContext.childForm.datatable.getAllChildIdsOrdered("tabela123");
console.log("IDs ordenados:", orderedIds);
```

---

datatable.getResultJson(tableid_relacaoFormularioPaiFilhoId)

Retorna um JSON com os resultados apresentados na tabela do formulário filho.

Exemplo de Uso:

```
var result = formContext.childForm.datatable.getResultJson("tabela123");
console.log("Resultados da tabela:", result);
```

---

datatable.setOrder(tableid_relacaoFormularioPaiFilhoId, items_sorted)

Reordena os itens da tabela filha conforme a lista ordenada de IDs.

Exemplo de Uso:

```
var sortedItems = [
    { formid: "123", draftid: "" },
    { formid: "456", draftid: "" },
    { formid: "789", draftid: "" }
];
formContext.childForm.datatable.setOrder("tabela123", sortedItems);
```

---

blockLinkEditOpening()

Bloqueia a abertura do link ao clicar na linha do grid.

Exemplo de Uso:

```
formContext.childForm.blockLinkEditOpening();
```

---

unlockLinkEditOpening()

Desbloqueia a abertura do link ao clicar na linha do grid.

Exemplo de Uso:

```
formContext.childForm.unlockLinkEditOpening();
```

Painéis formContext.panel

O módulo `formContext.panel` fornece funções para manipular painéis (grupo de informações), como exibir, ocultar e controlar o estado de colapso (aberto/fechado). Abaixo estão as funcionalidades disponíveis.

Exibir e Ocultar Painéis

hide(panelId)

Oculta um painel específico.

- **Parâmetros:**
  - `panelId` (String): ID do painel a ser ocultado.
- **Exemplo:**

```
formContext.panel.hide("painel1"); // Oculta o painel com ID "painel1".
```

---

show(panelId)

Exibe um painel específico.

- **Parâmetros:**
  - `panelId` (String): ID do painel a ser exibido.
- **Exemplo:**

```
formContext.panel.show("painel1"); // Exibe o painel com ID "painel1".
```

Expandir e Recolher Painéis

O módulo `collapse` permite controlar o estado de colapso (aberto/fechado) de um painel.

collapse.toggle(panelId)

Alterna o estado de colapso de um painel (abre se estiver fechado e fecha se estiver aberto).

- **Parâmetros:**
  - `panelId` (String): ID do painel.
- **Exemplo:**

```
formContext.panel.collapse.toggle("painel1"); // Alterna o estado de colapso do painel com ID "painel1".
```

---

collapse.isCollapsed(panelId)

Verifica se um painel está colapsado (fechado).

- **Parâmetros:**
  - `panelId` (String): ID do painel.
- **Retorno:**
  - `true` se o painel estiver colapsado.
  - `false` se o painel estiver expandido.
- **Exemplo:**

```
const estaColapsado = formContext.panel.collapse.isCollapsed("painel1");
if (estaColapsado) {
    console.log("O painel está fechado.");
} else {
    console.log("O painel está aberto.");
}
```

---

collapse.close(panelId)

Fecha (colapsa) um painel.

- **Parâmetros:**
  - `panelId` (String): ID do painel.
- **Exemplo:**

```
formContext.panel.collapse.close("painel1"); // Fecha o painel com ID "painel1".
```

---

collapse.open(panelId)

Abre (expande) um painel.

- **Parâmetros:**
  - `panelId` (String): ID do painel.
- **Exemplo:**

```
formContext.panel.collapse.open("painel1"); // Abre o painel com ID "painel1".
```

Exemplos de Uso de funções de painéis

Ocultando e exibindo um painel:

```
formContext.panel.hide("painel1"); // Oculta o painel.
formContext.panel.show("painel1"); // Exibe o painel.
```

Alternando o estado de colapso de um painel:

```
formContext.panel.collapse.toggle("painel1"); // Alterna entre aberto e fechado.
```

Verificando se um painel está colapsado:

```
if (formContext.panel.collapse.isCollapsed("painel1")) {
    console.log("O painel está fechado.");
} else {
    console.log("O painel está aberto.");
}
```

Fechando e abrindo um painel:

```
formContext.panel.collapse.close("painel1"); // Fecha o painel.
formContext.panel.collapse.open("painel1"); // Abre o painel.
```

Revision #15
Created 5 March 2025 12:38:43 by agilityflow
Updated 5 March 2025 13:44:13 by agilityflow