Programação em C# - Na API de integração (POST)

As informações descritas abaixo referem-se as APIs de **POST**

> IMPORTANTE: as variáveis e métodos descritos aqui só funcionarão na programação C# na **API de integração**. Para a programação C# nas Regras de Negócio de um formulário clique aqui

O agilityflow permite a customização da API em C#, além de já disponibilizar diversas bibliotecas e funções para facilitar sua programação, incluindo acesso a dados, validações, envio de e-mail, notificação, entre outras.

> Para cada requisição na API é criada uma transação de banco de dados, essa transação é única em toda execução da API e é gerenciada automaticamente pelo Agilityflow, caso você prefira, você pode fazer esse gerenciamento seguindo esses passos, veja mais detalhes aqui

Abaixo mostramos alguns exemplos, se o material abaixo não for suficiente , entre em contato com nossa equipe.

Variáveis disponíveis

Variável	**Tipo**	
content	string	Conteúdo enviado no Body da API No caso se ser enviado uma string com o JSON, por exemplo no formato: `{ nome: 'José', email: 'jose@xxxx.com'}` Você pode converter essa string para JSON e utilizá-la como object `var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`

Classe JsonHelper e outras para auxiliar o tratamento de JSON

Para utilizar execute `JsonHelper.NomeDoMetodo(....)`

Método	**Retorno**	

HasProperty(dynamic json, string nomePropriedade)	bool
	(true or false)
		Testa se o JSON tem uma determinada propriedade.
		Por exemplo, a sua API espera receber o JSON
		no formato:
		`{ nome: 'José', email: 'jose@xxxx.com'}`
		
		Porém recebe o JSON formato:
		`{ nomecompleto: 'José da Silva', idade : '21'}`
		
		Como a propriedade NOME e EMAIL não existem no JSON recebido, o seu programa pode dar erro. Para evitar o erro, utilize o método HasProperty(...) Como no exemplo abaixo:
		
		`if (json != null)`
		`{`
		
		`if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") ){`
		
		`var nome = json["nome"].ToString();`
		`var email= json["email"].ToString();`
		
		`}`
		
		`}`

`Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`	dynamic	converte o conteúdo string para json
		
		`var content = "{ nome: 'José', email: 'jose@xxxx.com'}";`
		`var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);`
		
		`var nome = json["nome"].ToString();`

Métodos Disponíveis no contexto da API

Método	**Retorno**	
await ApiContext.GetDataTableAsync (string sql, params DbParameter[] parameters)	DataTable	Recuperar informações dos bancos de dados através de uma query sql. Veja exemplo

await ApiContext.SaveEntityAsync(Guid estruturaformularioid_ASerCriadoOuAtualizado, DataDictionary campos_e_valores)	Guid	Salva as informações de um formulário no bancos de dados. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Caso seja necessário, utilizar o .ToString() para formatar um campo decimal, não use. Utilize a conversão usando o string.Format, como no exemplo abaixo: `string.Format(new System.Globalization.CultureInfo("en-gb"), "{0:F2}", json["valor"]);` Veja exemplo
await ApiContext.ApproveEntityAsync (Guid estruturaformularioid_ASerAprovado, DataDictionary campos_e_valores)	Guid	Avança uma etapa em um formulário com Workflow. As regras de usuário aprovador, permissão de aprovação e responsabilidade por uma etapa devem ser validadas antes desse método ser executado. Obs: Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas. Caso seja necessário, utilizar o .ToString() para formatar um campo decimal, não use. Utilize a conversão usando o string.Format, como no exemplo abaixo: `string.Format(new System.Globalization.CultureInfo("en-gb"), "{0:F2}", json["valor"]);` Veja exemplo
await ApiContext.DeleteEntityAsync(Guid formularioId)	void	deletar um formulario
ApiContext.GetEnvironmentVariable(string nomeVariavel) ou **ApiContext.GetEnvironmentVariable_Text**(string nomeVariavel) **	string	Recupera o Valor de uma variável de ambiente. ** Recupera a Descrição (texto), no caso de variáveis do Tipo "Que

await ApiContext.LogAsync(string log, string logTipo)	void	Cria um log na tabela de log geral, o log pode ser consultado entrando na Área de Customização e clicando na opção "Log Geral >> consultar Log" ** Além do Log Padrão da API que o sistema gera, você pode gerar esses Logs customizados para o seu próprio controle.
await ApiContext.DbTransaction_BeginTransactionAsync() **await ApiContext.DbTransaction_CommitAsync()** **await ApiContext.DbTransaction_RollbackAsync()**	void	Para gerenciar a transação de banco de dados manualmente. Leia os detalhes aqui mais abaixo
ApiContext.GetUrlBase()	void	Retorna a URL base da sua aplicação exemplo: https://suaempresa.agilityflow.io/ *A url sempre virá com uma barra no final /

Recuperando dados do banco de dados através de uma query SQL

```
//parametros da query
var paramsQuery = new List<SqlParameter>();
paramsQuery.Add(new SqlParameter("@param1", "xxx"));
paramsQuery.Add(new SqlParameter("@param2", "yyy"));
paramsQuery.Add(new SqlParameter("@param3", "zzz"));

//query usando (nolock) nas tabelas para evitar problema com a transação que está ativa
var sql = "select column1,column2,column3 from table (nolock) where column1 = @param1 or  column2 = @param2 or  column3 = @param3 ";

//executar no banco de dados
var dt = await ApiContext.GetDataTableAsync(sql, paramsQuery.ToArray());
```

```
//percorrendo as linhas de retorno da tabela
foreach (DataRow dr in dt.Rows)
{
  if (dr["column1"] != DBNull.Value){
  var xx = dr["column1"].ToString();
  }
}
```

Salvando os dados recebidos através do método SaveEntityAsync

* Para cadastrar valores numéricos, o números devem estar no formato com ponto no separador decimal: 999999.99 sem utilização de virgulas.

```
public void Execute(){

  //opcional para registrar no log de controle
  await ApiContext.LogAsync("Entrou da API", "log_api_xpto");

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");



        if (json != null)
        {

            //verifica se o JSON recebido contém a propriedade "nome", "email" e "cel" que são necessários p
cadastro
            if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") &&
JsonHelper.HasProperty(json, "perfil")){

                var values = new DataDictionary();
                values.Add("nome", json["nome"].ToString());
                values.Add("email", json["email"].ToString());

                //no caso do campo do formulário ser uma tabela associativa (tabela relacional N:N)
                var idCampoTabelaAssociativa = "70b6f362-2587-4fd2-a2fb-148bd0caf437";
```

```
                if (json["perfil"].ToString() == "todos_os_perfis" ) {

                  var idPerfilAdmin = "51cf02f1-3787-4dca-8a2c-e219a5ce1298";
                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfisConcatenadosComVirgula = idPerfilAdmin + "," + idPerfilColaborador + ",";

                  //adiciona os perfis
                  values.Add(idCampoTabelaAssociativa, idPerfilAdmin); //adiciona perfil de admin
                  values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                  values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfisConcatenadosComVirgula);// perfis concatenados com virgula
                  values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                  //----------------------------------------

                }
                else if (json["perfil"].ToString() == "apenas_colaborador" ) {


                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfilConcatenadoComVirgula = idPerfilColaborador + ",";

                  //adiciona o perfil
                  values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                  values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfilConcatenadoComVirgula);// perfis concatenados com virgula
                   values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se
fosse pra remover, era só colocar os perfis concatenados com virgula
                  //----------------------------------------


                }

                //salva a informação no banco de dados
                await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values);

              }

          }
//opcional para registrar no log de controle
```

```
  await ApiContext.LogAsync("Saiu da API", "log_api_xpto");

   }
```

Aprovando uma etapa através do método ApproveEntityAsync

```
public void Execute(){

  //opcional para registrar no log de controle
  await ApiContext.LogAsync("Entrou da API", "log_api_xpto");

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");



        if (json != null)
        {

            //verifica se o JSON recebido contém a propriedade "nome", "email" e "cel" que são necessários p
cadastro
            if (JsonHelper.HasProperty(json, "nome") && JsonHelper.HasProperty(json, "email") &&
JsonHelper.HasProperty(json, "perfil")){

                var values = new DataDictionary();
                values.Add("nome", json["nome"].ToString());
                values.Add("email", json["email"].ToString());
                values.Add("id", json["id"].ToString());

                //no caso do campo do formulário ser uma tabela associativa (tabela relacional N:N)
                var idCampoTabelaAssociativa = "70b6f362-2587-4fd2-a2fb-148bd0caf437";
                if (json["perfil"].ToString() == "todos_os_perfis" ) {

                  var idPerfilAdmin = "51cf02f1-3787-4dca-8a2c-e219a5ce1298";
                  var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                  var idPerfisConcatenadosComVirgula = idPerfilAdmin + "," + idPerfilColaborador + ",";

                  //adiciona os perfis
```

```
                    values.Add(idCampoTabelaAssociativa, idPerfilAdmin); //adiciona perfil de admin
                    values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                    values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfisConcatenadosComVirgula);// perfis concatenados com virgula
                    values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                    //----------------------------------------


                }
                else if (json["perfil"].ToString() == "apenas_colaborador" ) {



                    var idPerfilColaborador = "f999f103-c775-4245-92d3-034cb3ded5e4";
                    var idPerfilConcatenadoComVirgula = idPerfilColaborador + ",";

                    //adiciona o perfil
                    values.Add(idCampoTabelaAssociativa, idPerfilColaborador); //adiciona perfil de colaborador
                    values.Add("tabela_associativa_added-
"+idCampoTabelaAssociativa,idPerfilConcatenadoComVirgula);// perfis concatenados com virgula
                    values.Add("tabela_associativa_removed-"+idCampoTabelaAssociativa,"");//em branco, se fosse
pra remover, era só colocar os perfis concatenados com virgula
                    //----------------------------------------



                }

                //salva a informação no banco de dados
                await ApiContext.ApproveEntityAsync(idEstruturaFormulario_PESSOA, values);


            }


        }
//opcional para registrar no log de controle
    await ApiContext.LogAsync("saiu da API", "log_api_xpto");


    }
```

Recuperar os valores das "Variáveis de Ambiente"

Para saber mais sobre variáveis de ambiente entre em Variáveis de ambiente

```
var valorDaVariavel = ApiContext.GetEnvironmentVariable("var_nomeDaVariavel");
```

Recuperar a Descrição (texto), no caso de variáveis do Tipo "Query com Id + Descrição":

```
var textDaVariavel = ApiContext.GetEnvironmentVariable_Text("var_nomeDaVariavel");
```

Envio de e-mail

Responsável pelo envio de e-mail

Método	**Retorno**	
EmailSender.SendToGrupoUsuario(Guid grupoUsuarioId, string subject, string htmlContent)	void	Enviar um e-mail para um determinado Grupo de Usuario
EmailSender.SendEmail(string toEmail, string subject, string htmlContent)	void	Enviar um e-mail
EmailSender.SendEmail(string toEmail, string subject, string htmlContent, string ccEmail)	void	Enviar um e-mail, com copia (CC)

Enviar e-mail para um Grupo de Usuário

Antes de iniciar, crie um "Grupo de Usuário" no menu "Segurança e Acesso". Depois de criado, associe os usuários que receberão e-mail ao novo grupo de usuário que você acabou de criar.

Pegue o ID desse novo Grupo de usuário. Você vai precisar dele para enviar o e-mail.

```
var grupoUsuarioId = Guid.Parse("af7c9275-0e23-4b64-a433-f238bb457005"); //substitua o ID "af7c9275-0e23-4b64-a433-f238bb457005" pelo Id do grupo de usuario que você deseja enviar o e-mail
var assunto = "Novo E-mail para um Grupo de usuario ";

var html = "Olá Grupo, <BR><BR> Teste para envio de e-mail para um Grupo de Usuário no AgilityFlow!";

EmailSender.SendToGrupoUsuario(grupoUsuarioId, assunto, html);
```

Enviar e-mail

```
var htmlContent = "<strong>Olá,</strong> teste envio de e-mail.";
var emailTo = "john@email.com";
var subject = "Subject do E-mail";

EmailSender.SendEmail(emailTo, subject, htmlContent);
```

Método principal para utilização da Programação em C# na api de post

A única regra é que o método se chame Execute, não tenha parâmetros de entrada e o retorno seja void.

Exemplo:

```
public void Execute(){



  //aqui você pode programar



}
```

Gerenciar as transações de Banco de dados manualmente

Caso você escolha por gerenciar as transactions de banco de dados manualmente, você precisará obrigatoriamente utilizar as 3 funções abaixo:

> Atenção: Configuração recomendada apenas para especialista.

1. **ApiContext.DbTransaction_BeginTransaction()** Para abrir uma transação, antes de iniciar as alterações no banco de dados
2. **ApiContext.DbTransaction_Commit()** Para finalizar e confirmar com sucesso as alterações no banco de dados
3. **ApiContext.DbTransaction_Rollback()** Para finalizar e desfazer as alterações no banco de dados

Exemplo de uso:

> Atenção: Utilize as transactions sempre dentro de TRY e CATCH

> Podem ser abertas uma ou mais transações na mesma API. Você ficará responsável por todas as aberturas e encerramentos das transações criadas.

```
public async Task RunAsync(){

        //converte o conteúdo enviado no body da API para JSON
        var json = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(content);

        //guarda em uma variável o ID da  E S T R U T U R A  do formulário de pessoa
        var idEstruturaFormulario_PESSOA = Guid.Parse("0b56b66a-4f6f-4ded-ad04-016d7c0724e1");

        try
        {

          await ApiContext.DbTransaction_BeginTransactionAsync(); //ABRE A TRANSAÇÃO, DENTRO DO TRY CATCH

          var values = new DataDictionary();
          values.Add("nome", "exemplo1_PRIMEIRA_transacao");

          await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values); //salva a informação no banco de dados

          await ApiContext.DbTransaction_CommitAsync(); //CONFIRMA A TRANSAÇÃO, DENTRO DO TRY CATCH

        }
        catch
        {
          await ApiContext.DbTransaction_RollbackAsync(); //DESFAZ A TRANSAÇÃO, DENTRO DO CATCH em caso de erro
          throw;
        }


  □□□//mais código
```

```
   //...
   //mais código
   //...
    //mais código
   //...
   //mais código
   //...




   try
          {

           await ApiContext.DbTransaction_BeginTransactionAsync(); //ABRE A TRANSAÇÃO, DENTRO DO TRY
CATCH

           var values2 = new DataDictionary();
           values2.Add("nome", "exemplo2_SEGUNDA_transacao");

           await ApiContext.SaveEntityAsync(idEstruturaFormulario_PESSOA, values2); //salva a informação no
banco de dados

           await ApiContext.DbTransaction_CommitAsync(); //CONFIRMA A TRANSAÇÃO, DENTRO DO TRY CATCH

          }
          catch
          {
           await ApiContext.DbTransaction_RollbackAsync(); //DESFAZ A TRANSAÇÃO, DENTRO DO CATCH em
caso de erro
           throw;
          }
   


    }
```

Revision #48
Created 6 December 2019 15:53:19 by agilityflow
Updated 5 March 2025 12:33:44 by agilityflow