



TOTVS

EDUCAÇÃO EMPRESARIAL

**FORMAÇÃO
E CERTIFICAÇÃO
TOTVS**



Sumário

1. Objetivo.....	8
2. Introdução.....	8
3. Características da Tecnologia	8
3.1. Hardware Minimo recomendável	9
3.1.1. Recomendações	9
4. Camadas do Protheus	10
4.1. Servidor de Aplicação - Application Server.....	10
4.2. Terminais Thin - Smart Client Protheus	10
4.3. Base de Dados	10
4.4. Repositórios de RPO's.....	11
5. Instalação do Protheus 12	11
5.1. Instalando Central de Ajuda.....	16
5.2. Servidor de Licenças	19
5.2.1. Instalação	19
5.2.2. Como configurar o Aplicador de Licenças.....	22
5.2.3. Gestão de Instalações.....	23
5.2.4. Gestão de Licenças.....	24
5.2.5. Licenças Tradicionais	25
5.2.6. Licenças Corporativas	25
5.3. Parâmetros do atalho AppServer.....	26
5.4. Parâmetros do Atalho SmartClient.....	27
5.5. Arquivos de Configurações	28
5.6. Configuração do appserver.ini	29
5.6.1. Environment	29
5.6.2. DbAccess	30
5.6.3. Drivers	30
5.6.4. Servernetwork	30
5.6.5. HTTP	31
5.6.6. FTP	31
5.6.7. ONSTART	31
5.6.8. License Server	31
5.6.9. License Client	32
5.6.10. General	32
5.6.11. Service	32
5.6.12. Exemplo do Appserver.ini.....	32
5.7. TOTVS DBAccess	35
5.7.1. Instalação do TOTVS DBAccess	35
5.7.2. Configurar a Conexão com o DBAccess no Protheus	37
5.7.3. ODBC – Open Database Connectivity	38
5.7.4. Utilizando o TOTVS DBAccess	39

5.7.5. TOTVSDBMonitor	39
5.7.6. Usuário	42
5.7.7. Rastrear	42
5.7.8. Encerrar	43
5.7.9. Atividades	43
5.7.10. Locks	44
5.7.11. Mensagens	45
5.7.12. Configurações	46
5.7.13. Configurando o drive de ODBC no dbAccess	47
5.8. Configuração do TOTVS Smart Client	48
5.9. Acessando o Protheus	49
5.10. Totvs Development Studio (DevStudio)	51
5.10.1. Instalação	52
5.10.2. Iniciando o TOTVS Developer Studio	55
5.10.3. Perspectivas	58
5.10.4. Nomenclatura dos arquivos	58
5.10.5. Atualização de BUILD	59
5.10.6. Aplicações de Patchs	61
5.10.7. Log's dos Repositórios	63
5.10.8. Desfragmentar Reppositório	66
5.10.9. Compatibilizador - UPDDISTR	68
5.11. TOTVS Wizard - Assistente de Configuração	74
5.11.1. Configurações de Ambientes	75
5.11.2. Configurações de Serviços	77
5.11.3. Configuração do Balanceamento de Carga Load Balance	78
5.11.4. Teste de Balanceamento	80
5.12. TOTVS Smart Client	81
5.12.1. Atualização do TOTVS Smart Client automática	82
5.12.2. SmartClient Activex	82
5.13. C-tree Server	83
5.13.1. Instalação C-tree Server	83
5.13.2. Configurações do C-tree Server	87
5.14. Configuração Portal	88
6. Modulo Configurador	90
6.1. Estrutura dos Diretórios	91
6.2. Famílias de Arquivos	91
6.3. Arquivos, Tabelas e Campos	94
6.3.1. Perguntas (SX1)	94
6.3.2. Arquivos (Sx2)	97
6.3.3. Campos (SX3)	98
6.4. Índices (SIX)	106
6.4.1. Atualização dos dicionários de dados	107
6.4.2. Atualização agendada dos dicionários de dados	109

6.4.3. Pasta (SXA).....	110
6.4.4. Grupos de Campos (SXG)	112
6.4.5. Tabelas Genéricas (SX5)	113
6.4.6. Consultas - Padrão (SXB)	114
6.4.7. Parâmetros (Sx6)	118
6.4.8. Gatilhos (SX7)	120
6.5. Menus e Senhas.....	121
6.6. Política de Segurança.....	124
6.6.1. Usuários	130
6.7. Privilégios	138
6.7.1. Config. Perguntas.....	141
6.8. Papel de Trabalho	142
6.9. Repositório de Imagens	145
6.10. Embedded Audit Trail	146
7. Desenvolvimento	149
7.1. Perspectiva TOTVS Developer	149
7.2. Iniciando um Projeto	150
7.2.1. Configuração de Includes.....	151
7.2.2. Novo Projeto TOTVS.....	153
7.2.3. Novo Arquivo Fonte.....	154
7.2.4. Perfil de Execução dos Programas	155
7.2.5. Depuração de Arquivo Fonte (Debug).....	157
7.2.6. Preferências de configurações	163
7.2.7. Criação de um Programa	165
7.2.8. Linhas de Programa	165
7.2.9. Operadores da linguagem ADVPL	174
7.2.10. Ordem de Precedência dos Operadores	176
7.2.11. Operação de Macro Substituição	176
7.2.12. Conversões entre tipos de variáveis	177
7.2.13. Manipulação de strings	178
7.2.14. Verificação de tipos de variáveis	180
7.3. Estruturas Básicas de Programação.....	181
7.3.1. Repetição de Comandos	181
7.3.2. Desviando a Execução	183
7.4. ARRAYS E BLOCOS DE CÓDIGO	185
7.4.1. Arrays	185
7.4.2. Blocos de Código	191
7.5. Funções	194
7.5.1. Tipos e escopos de funções	194
7.5.2. User Function().....	195
7.5.3. Static Function()	195
7.5.4. Main Function()	196
7.5.5. Passagem de parâmetros entre funções	196

7.5.6. Passagem de parâmetros por conteúdo	196
7.6. Diretivas De Compilação	198
7.7. Acesso e manipulação na bases de dados	199
7.7.1. Funções de manipulação de dados genéricos	199
7.7.2. Estrutura dos registros (informações)	200
7.8. Diferenciação entre variáveis e nomes de campos	204
7.9. Controle de numeração sequencial	205
7.9.1. Semáforos	205
7.9.2. Funções de controle de semáforos e numeração sequencial	205
7.10. Customização de parâmetros – Configurador	206
7.11. Pontos de Entrada – Conceitos	207
7.12. Interfaces Visual	208
7.12.1. Sintaxe e componentes das interfaces visuais	209
7.13. Interfaces padrões para atualizações de dados	211
7.13.1. Modelo1	211
7.13.2. EnchoiceBar	220
7.13.3. TcBrowser	220
7.13.4. Modelo 2	222
7.13.5. Modelo3	223
7.14. Utilizando Querys no Protheus	226
7.14.1. Embedded SQL	229
8. Arquitetura MVC	230
8.1. Principais funções da aplicação em AdvPL utilizando o MVC	231
8.1.1. O que é a função ModelDef?	231
8.1.2. O que é a função ViewDef?	231
8.1.3. O que é a função MenuDef?	232
8.1.4. Novo comportamento na interface	232
8.2. Aplicações com Browses (FWMBrowse)	233
8.2.1. Construção básica de um Browse	233
8.2.2. Legendas de um Browse (AddLegend)	234
8.2.3. Filtros de um Browse (SetFilterDefault)	234
8.2.4. Desabilitação de detalhes do Browse (DisableDetails)	235
8.2.5. Construção de aplicação ADVPL utilizando MVC	236
8.2.6. Criando o MenuDef	236
8.2.7. Construção da função ModelDef	237
8.2.8. Construção de uma estrutura de dados (FWFormStruct)	237
8.2.9. Criação de componente no modelo de dados (AddFields)	238
8.2.10. Criação de um componente na interface (AddField)	239
8.2.11. Descrição dos componentes do modelo de dados (SetDescription)	239
8.2.12. Finalização de ModelDef	239
8.2.13. Construção da função ViewDef	240
8.2.14. Exibição dos dados (CreateHorizontalBox / CreateVerticalBox)	240
8.2.15. Relacionando o componente da interface (SetOwnerView)	241

8.2.16. Finalização da ViewDef	241
8.2.17. Carregar o modelo de dados de uma aplicação já existente (FWLoadModel)	242
8.2.18. Carregar a interface de uma aplicação já existente (FWLoadView)	242
8.2.19. Carregar a menu de uma aplicação já existente (FWLoadMenudef)	242
8.3. Instalação do Desenhador MVC	243
8.3.1. Criação de um novo fonte Desenhador MVC	247
8.4. Tratamentos para o Modelo de dados (Model)	256
8.4.1. Retorno da operação que está sendo realizada (GetOperation)	256
8.4.2. Retorno do componente do modelo de dados (GetModel)	257
8.4.3. Componente do modelo de dados Ativo FWModelActive()	257
8.4.4. Carregar o modelo de dados (FWLoadModel)	257
8.5. Obtendo os valores do modelo de dados	258
8.5.1. Atribuindo valores no modelo de dados	258
8.5.2. Mensagens exibidas na interface	259
8.6. Manipulação do Modelo de dados	260
8.6.1. Gravação manual de dados (FWFormCommit)	261
8.6.2. Cancelamento da gravação de dados	262
8.6.3. Validação da Ativação do Modelo de Dados	262
8.6.4. Método de desativação do Modelo de Dados	263
8.6.5. Tratamentos de estrutura de dados(FWFormStruct)	263
8.6.6. Código para a estrutura (FWBuildFeature)	269
8.7. Manipulação do componente AddFields	269
8.8. Manipulação do componente FormGrid	271
8.8.1. Criação de relação entre as entidades do modelo (SetRelation)	274
8.8.2. Quantidade de linhas do componente de grid	274
8.8.3. Status da linha de um componente de grid	275
8.8.4. Adição uma linha a grid	275
8.8.5. Apagando e recuperando uma linha	276
8.8.6. Guardando e restaurando o posicionamento do grid	276
8.8.7. Validação de linha duplicada (SetUniqueLine)	277
8.8.8. Campo Incremental (AddIncrementField)	277
8.9. Ações com a interce View	277
8.9.1. Método na SetViewAction	278
8.9.2. Ação de interface do campo (SetFieldAction)	278
8.9.3. Metodo SetAfterOkButton	279
8.9.4. Metodo SetViewCanActivate	279
8.9.5. Metodo SetAfterViewActivate	279
8.9.6. Adicionando botão na tela	280
8.9.7. Criação de pastas (CreateFolder)	280
8.9.8. Validação das pasta SetVldFolder	281
8.9.9. Outros objetos (AddOtherObjects)	281
8.9.10. Carregar a interface de uma aplicação já existente (FWLoadView)	282
8.10. Criação de campos de total ou contadores (AddCalc)	282

8.11. FWExecView	283
8.12. Pontos de entrada no MVC	285
8.13. Rotina automática	290
9. Relatórios Gráficos	292
9.1.1. Pergunte()	295
9.1.2. AjustaSX1()	295
9.1.3. PutSX1()	297
9.1.4. ParamBox()	298
9.2. Rotinas automáticas	302
9.2.1. MsExecAuto	302
9.2.2. Prepare Environment	303
9.2.3. RpcSetEnv	304
9.3. Manipulação de arquivos texto	304
10. Webservices	309
10.1. O Que É Um Webservices Wsdl	309
10.1.1. O Que É Um Xml	310
10.1.2. O Que É Soap	311
10.1.3. O Servidor Protheus como um servidor WEBSERVICES	311
10.1.4. Configurando servidor de WEBSERVICES	311
10.1.5. WSINDEX - Índice de Serviços	317
10.1.6. Estrutura do Serviço	319
10.1.7. Tipos básicos de dados - 'Server'	320
10.1.8. Estruturas - Tipos complexos	320
10.1.9. Codificando o serviço	321
10.1.10. TWsdlManager	332
10.2. Webservices MVC	334

1. Objetivo

A Formação Programação ADVPL tem por objetivo explorar em forma de conhecimento toda a contribuição do sistema ERP TOTVS nas organizações. De forma prática e objetiva, o curso visa a qualidade da gestão em processos e projetos organizacionais, relacionado à Administração e customizações no Protheus.

2. Introdução

A realidade empresarial atual é caracterizada por um cenário mundial globalizado e competitivo, com rápidos avanços da tecnologia de produção, informática e de telecomunicações, bem como em outras transformações que sugerem novas formas de percepção e interpretação das estruturas organizacionais (ANTUNES, 2000).

Segundo McGee e Prusak (1994), a concorrência entre as organizações, atualmente, baseia-se em sua capacidade de adquirir, tratar, interpretar e utilizar a informação de forma eficaz.

Dentro das empresas, o que se vê hoje é um novo cenário na área de Tecnologia da Informação (TI). Se antes os profissionais da área contribuíam apenas nas questões tecnológicas, hoje eles possuem um desafio de gestão e contribuem para os resultados dos negócios, tornando-se estratégicos. A importância de profissionais ligados a TI faz crescer o interesse das empresas em investirem nessa área e encontrar um perfil adequado a essa função.

3. Características da Tecnologia

O Protheus é uma tecnologia desenvolvida a partir do Sistema Advanced, que teve a Inteligência toda, dividida em quatro camadas ao saber: servidor de aplicação –Application Server (AppServer), Smart Client Protheus (SmartClient), repositório de funções (RPO) e banco de dados. Ou seja, uma aplicação Windows , se encarrega do gerenciamento das conexões da execução do código ADVPL e do acesso aos recursos de banco de dados através do Codebase, ADS, Btrieve, Ctree ou do TOTVS DBAccess .

É uma Aplicação cliente servidor, que realiza apenas a Interface com o usuário.

Principais Características da Tecnologia Protheus

- Possibilidade de grande variação de topologias de Redes e Processamentos Distribuídos;
- Baixo tráfego de rede Application Server e o Smart Client Protheus
- Utilização de configurações, possibilitando o uso de conexões simultâneas, através de protocolos diferentes e o acesso a diferentes repositórios de APO's e diretórios (O que permite o uso de Diferentes Idiomas e Interfaces acessando a mesma Base de Dados);
- Diferentes possibilidades de impressão de relatórios;

3.1. Hardware Mínimo recomendável

As configurações citadas, trata das necessidades mínimas e EXCLUSIVAS para utilização do Application Server e dos SGBDs (Sistema Gerenciador de Banco de Dados) homologados. No entanto, diante da utilização de outros aplicativos, as necessidades deverão ser melhor dimensionadas.

Banco de Dados	10 Usuários	15 Usuários	20 Usuários	30 Usuários	50 Usuários	
AS/400 (**)	Pentium D, 3,0 Ghz L2 4,0 MB	Pentium D, 3,0 Ghz L2 4,0 MB	Pentium D, 3,0 Ghz L2 4,0 MB	Pentium D, 3,0 Ghz L2 4,0 MB		Pentium D, 3,0 Ghz L2 4,0 MB
	2,0 GB RAM	2,0 GB RAM	2,0 GB RAM	2,0 GB RAM		3,0 GB RAM
	Ultra 4 SCSI (Aplicativo e File Server)	Ultra 4 SCSI (Aplicativo e File Server)	Ultra 4 SCSI (Aplicativo e File Server)	Ultra 4 SCSI (Aplicativo e File Server)		Ultra 4 SCSI (Aplicativo e File Server)
Oracle Windows,	Xeon 2.0 1X	Xeon 2.6 1X	Xeon 2.8 1X	Xeon 3.0 2X	Xeon 3.0 2X	Pentium D, 3,0 Ghz L2 4,0 MB
Oracle LINUX,	Dual Core L2 4,0 MB	Dual Core L2 4,0 MB	Dual Core L2 4,0 MB	Dual Core L2 4,0 MB	Dual Core L2 4,0 MB	3,0 GB RAM
MSSql Server,	2,0 GB RAM	2,0 GB RAM	3,0 GB RAM	4,0 GB RAM	4,0 GB RAM	Ultra 4 SCSI
Informix Windows,	Ultra 4 SCSI	Ultra 4 SCSI	Ultra 4 SCSI Controladora com cache e bateria backup.	HD SAS 15K	HD SAS 15K	
Informix LINUX,				Controladora com cache e bateria backup.	Controladora com cache e bateria backup.	

3.1.1. Recomendações

Em sites com mais de 250 usuários, utilize o Memory Files por meio do c-tree Enterprise Server. Ao utilizar os ambientes Call Center (SIGATMK) e Front Loja (SIGAFRT), do produto Microsiga Protheus, deve-se utilizar mais 10 MB de memória RAM por usuário no servidor de aplicação.

Para sites com mais de 50 usuários ou base de dados maior que 5 GB, entre em contato conosco pelo endereço: www.totvs.com/customercenter.

Estação com processamento no servidor:

Pentium III 700 MHz - 128 MB RAM + Memória recomendada para operação do sistema operacional.

Estação com processamento local (Two Tier)

Pentium III 700 MHz - 256 MB RAM (Sem base de dados na estação) + Memória recomendada para operação do Sistema Operacional.

O disco rígido (HD - Hard Disk) deve ser adquirido em função das necessidades da empresa; ou seja, conforme os volumes de dados.

Linha privada (LP) de dados 48 kbps para uma sessão.

Linha privada (LP) de dados 15 kbps por sessão (utilize um Frame Relay puro), desde que esteja com mais de 5 usuários.

O tempo de resposta, do site remoto, deve ser inferior a 120 ms (tempo de resposta do comando Ping) em pacotes de 32 KB.

4. Camadas do Protheus

O Protheus é dividido em quatro camadas, para a operação.

4.1. Servidor de Aplicação - Application Server

O Protheus Application Server (AppServer.exe), é a aplicação encarregada da compilação e execução do código em (ADVPL), no qual o Protheus 12, foi escrito a partir da Versão 5.07.

Na Linguagem ADVPL, as rotinas são mantidas em RPO. Isso permite que as mesmas sejam carregadas e descarregadas dinamicamente da memória da máquina onde o servidor está sendo executado, de acordo com a necessidade de execução dos terminais (SmartClient.exe) conectados.

Isso facilita a atualização após correções de não conformidades ou criação de melhorias, pois apenas os RPO's modificados necessitam ser atualizados.

Desse modo, a performance é alta e não requer muito da máquina, para a execução do servidor.

4.2. Terminais Thin - Smart Client Protheus

O SmartClient, é a aplicação encarregada da interface com o usuário. Não existe processamento local, por isso o tráfego de rede entre o terminal e o servidor de aplicação é baixo, tratando apenas de comandos, para o desenho das telas e tratamento do teclado e mouse.

4.3. Base de Dados

O acesso aos dados é realizado pelo servidor de aplicação, através do padrão ISAM (Codebase), ADS (Para padrão DBF), ou do DBAccess (Para padrão SQL).

Para bases de dados (SQL), existe total suporte a Stored Procedures.

Nas versões antigas do Protheus, todas as bases de dados têm suporte a controle de transações, inclusive a base de dados (Padrão DBF).

O Protheus 12, permite a utilização de tecnologias de replicação de dados, como o CISASync ou o próprio MSSQL Server.

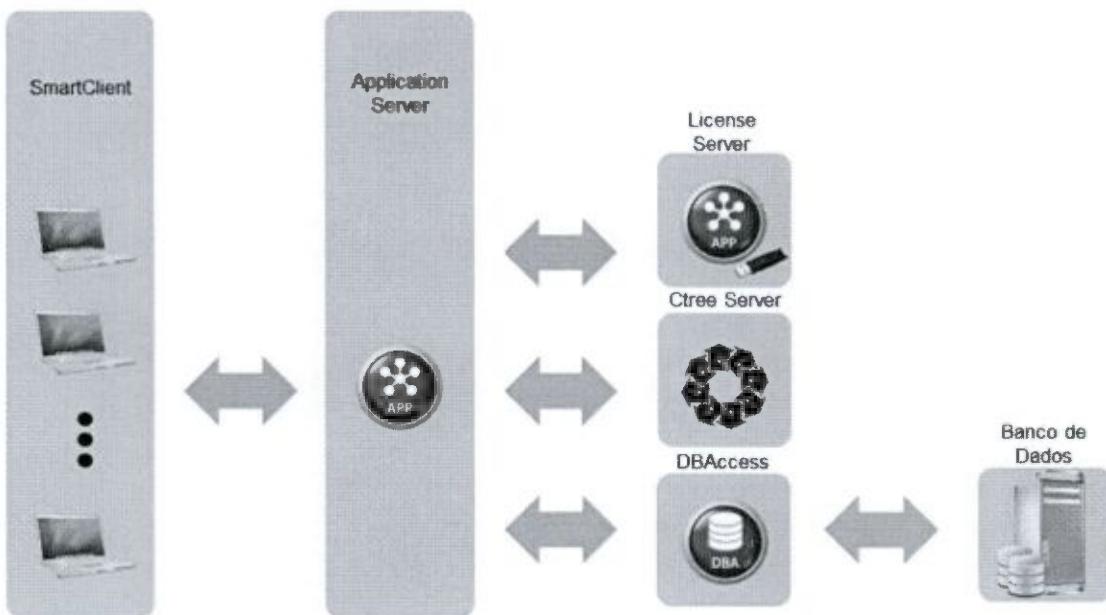
4.4. Repositórios de RPO's

É nesta parte do sistema que estarão os programas escritos em ADVPL, à serem acionados para a execução de determinadas tarefas.

É através dos repositórios de RPO's, que poderemos incluir novas customizações no sistema.

O repositório RPO, é um arquivo binário compilado, que por sua vez não pode ser editado ou modificado, pois tratam os programas desenvolvidos pela TOTVS.

Arquitetura Simplificada



5. Instalação do Protheus 12

Iremos instalar o Protheus, juntamente com todas as Ferramentas que o Protheus 12, disponibiliza.

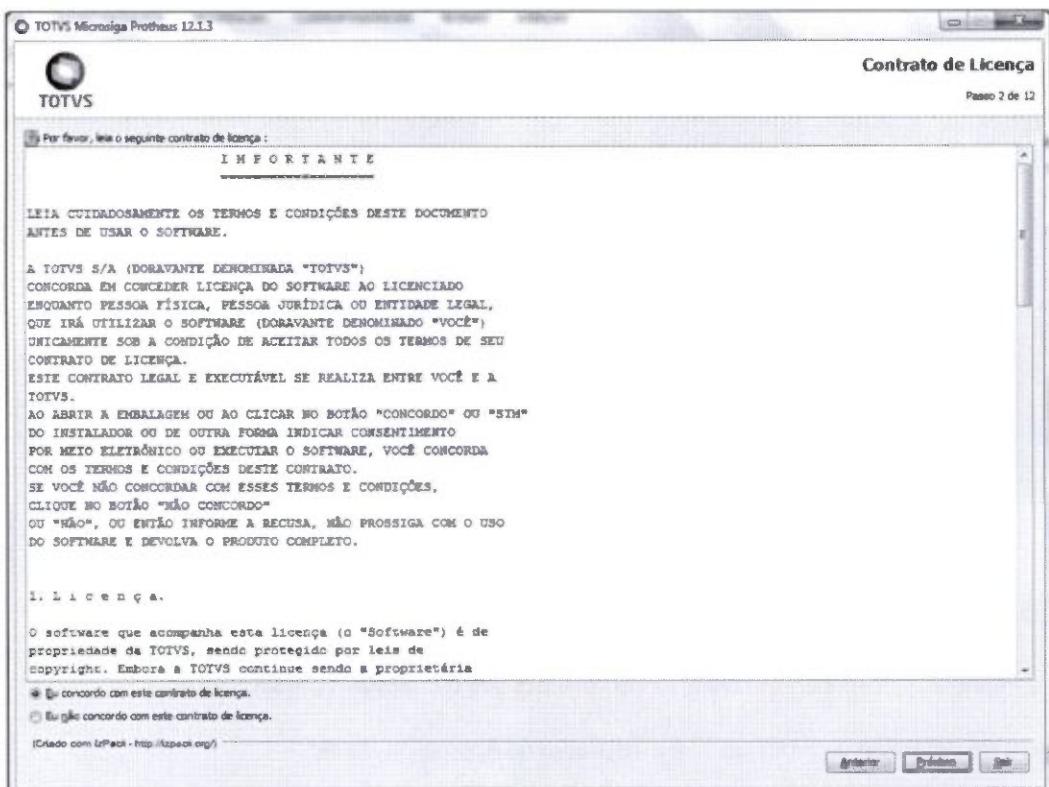
O primeiro passo para instalação do Protheus 12 é a instalação do Application Server.

A instalação do Application Server, assim como do SmartClient e das Ferramentas auxiliares (DevStudio, MPDump e Monitor) é realizada através do instalador do server, localizado no CD do Protheus 12. Coloque o CD-ROM no drive e aguarde a exibição da tela de abertura conforme a seguir.



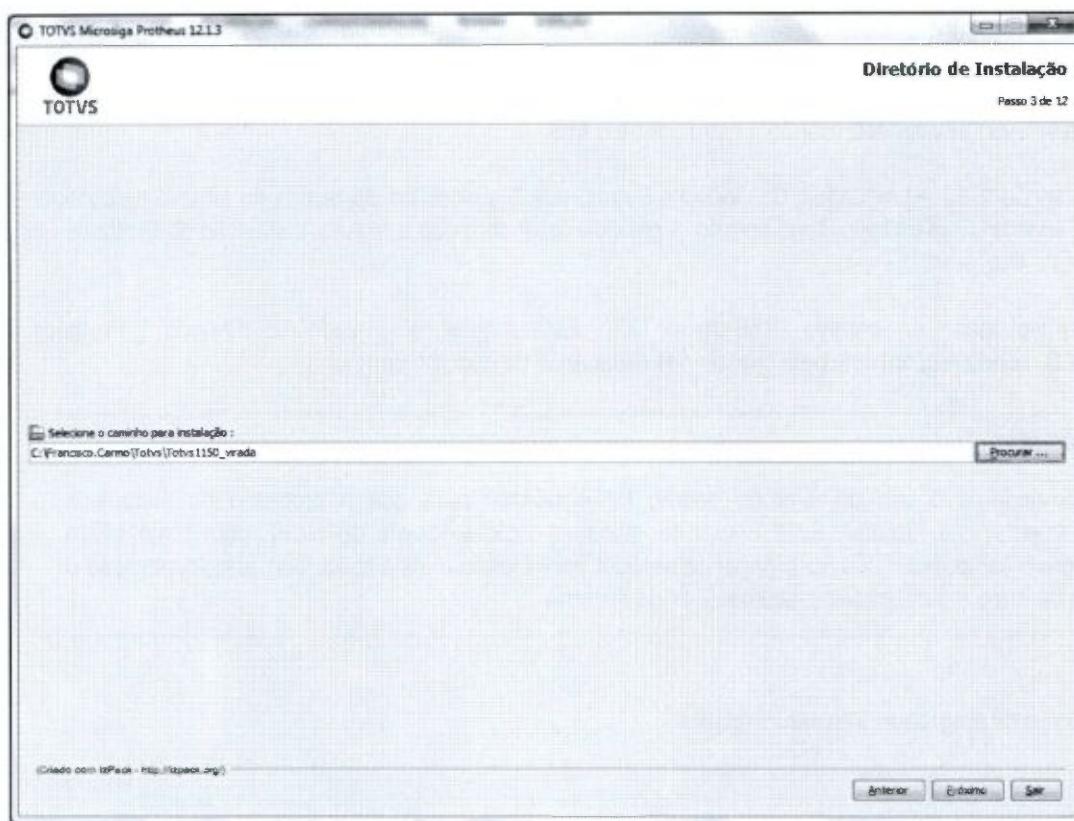
Na parte inferior da tela, são apresentados os idiomas disponíveis para apresentação do instalador do Protheus. Clique no ícone correspondente ao idioma que deseja utilizar.

Selecione a opção: "Eu concordo com este contrato de licença" para prosseguir na instalação.

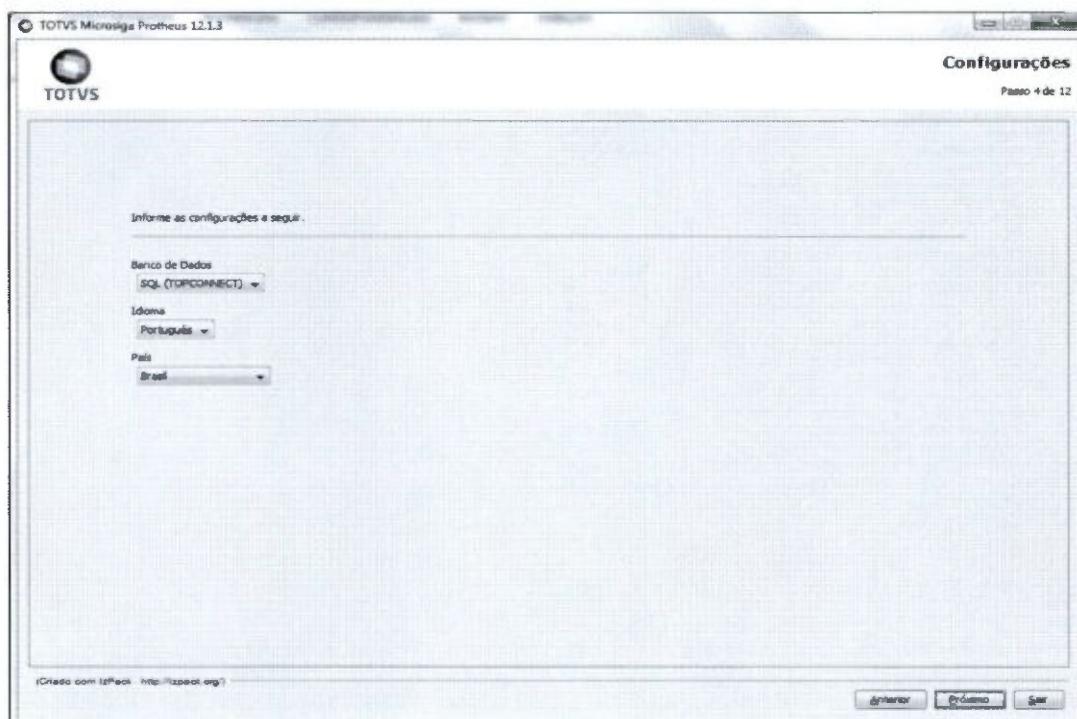


O instalador será iniciado. A janela de Bem-vindo é apresentada.

Formação Programação ADVPL



Clique no botão "Avançar" para prosseguir.



Nesta tela, deve-se selecionar:

- **SQL:** Necessário trabalhar com o serviço de DBAccess (integrado com todos os tipos de banco de dados disponíveis para esta aplicação).
- **c-tree:** Suas tabelas irão ficar com estrutura de c-tree.

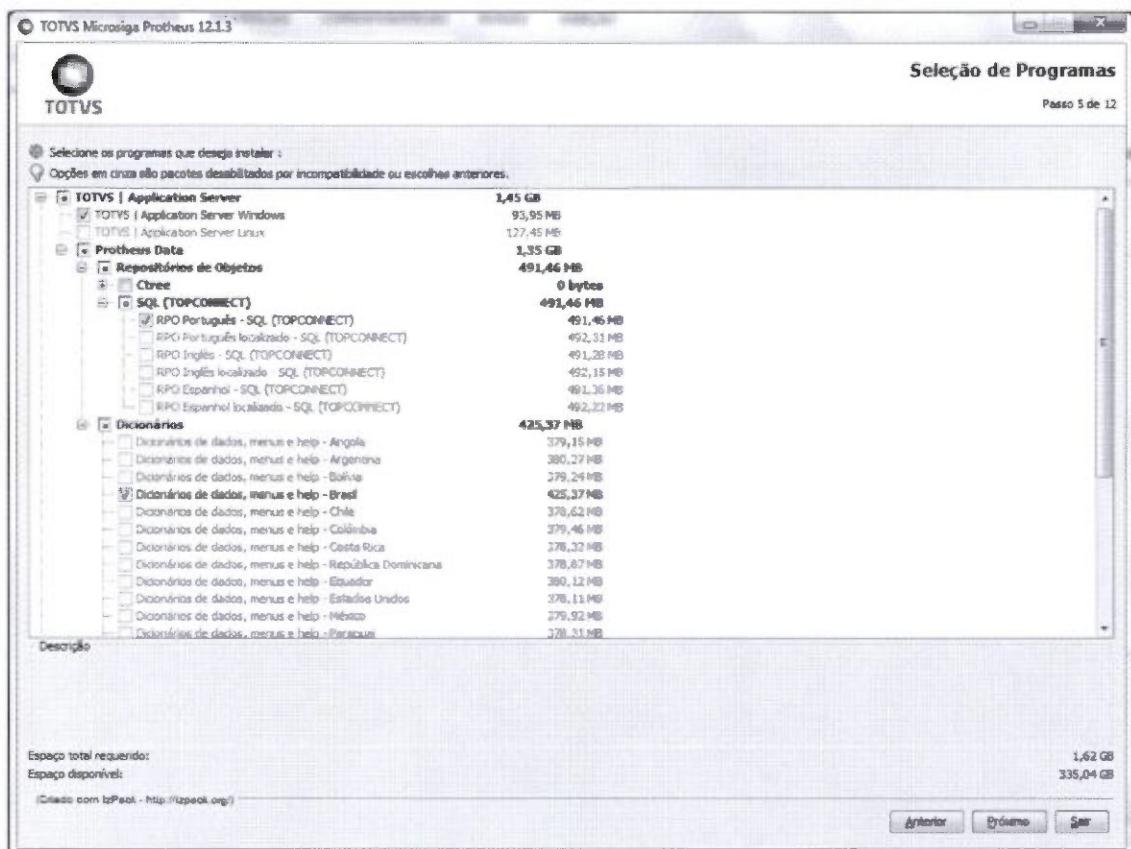
Selecionar a opção "SQL – Linguagem de Consulta Estruturada", através de alterações no arquivo de configuração do Application Server e cópia do repositório correto, é possível fazer com que a mesma instalação do Protheus utilize dois tipos de base de dados.

O país para instalação do arquivo "SXS<pais>.TXT". Este arquivo é gravado no diretório \ Protheus_Data \ SYSTEMLOAD, sendo responsável pela criação dos dicionários de dados e demais

Importante

Recomenda-se o uso do diretório padrão de instalação para que o processo de suporte e treinamento seja facilitado. Evite o uso de unidades lógicas (drives de rede), para tanto utilize sempre o computador que faz o papel de servidor para efetuar a instalação. Com isso, diminuirá o risco de erros em eventuais desinstalações do sistema

Informados quais os programas irão ser instalados:

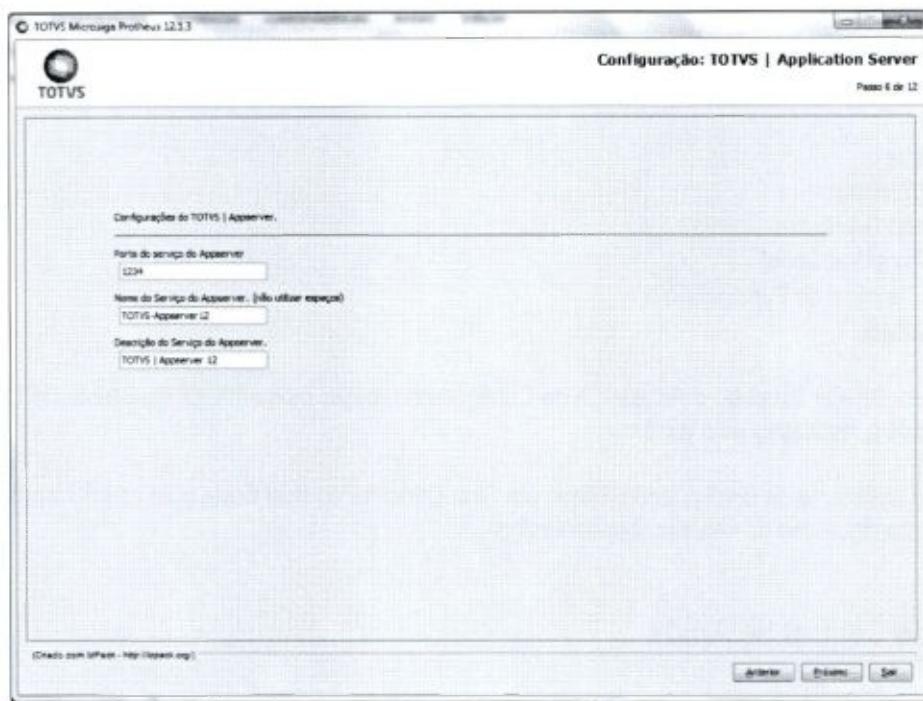


Formação Programação ADVPL

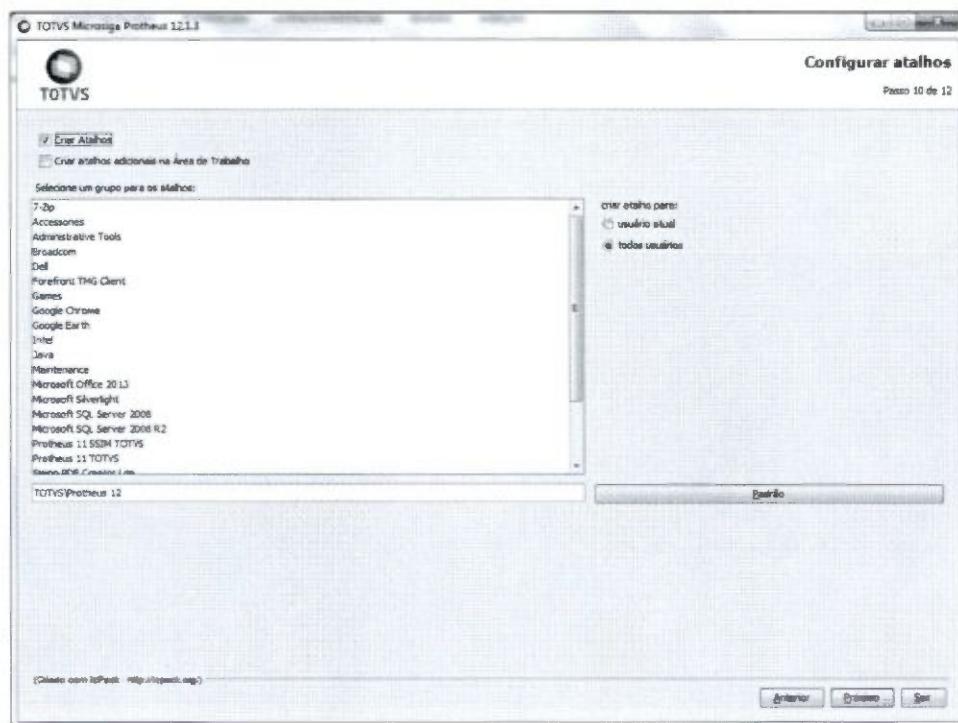


Informar a configurações do AppServer:

- **Porta:** Número da porta para acessar a aplicação
- **Nome do serviço:** Nome do serviço no Windows



Informe se deseja criar atalho dos programas instalados.



O programa de instalação prepara o Application Server com as configurações mínimas para a sua correta execução. Porém, pode-se configurá-lo através da manutenção do arquivo de configurações (AppServer.INI). As configurações são mantidas em diferentes seções de Environments, conforme a estrutura do arquivo de configurações.

No Windows, é possível configurar o arquivo appserver.ini, utilizando este Assistente de Configuração do Application Server.

Além disso, o assistente permite a instalação e configuração dos módulos Web do Microsiga Protheus:

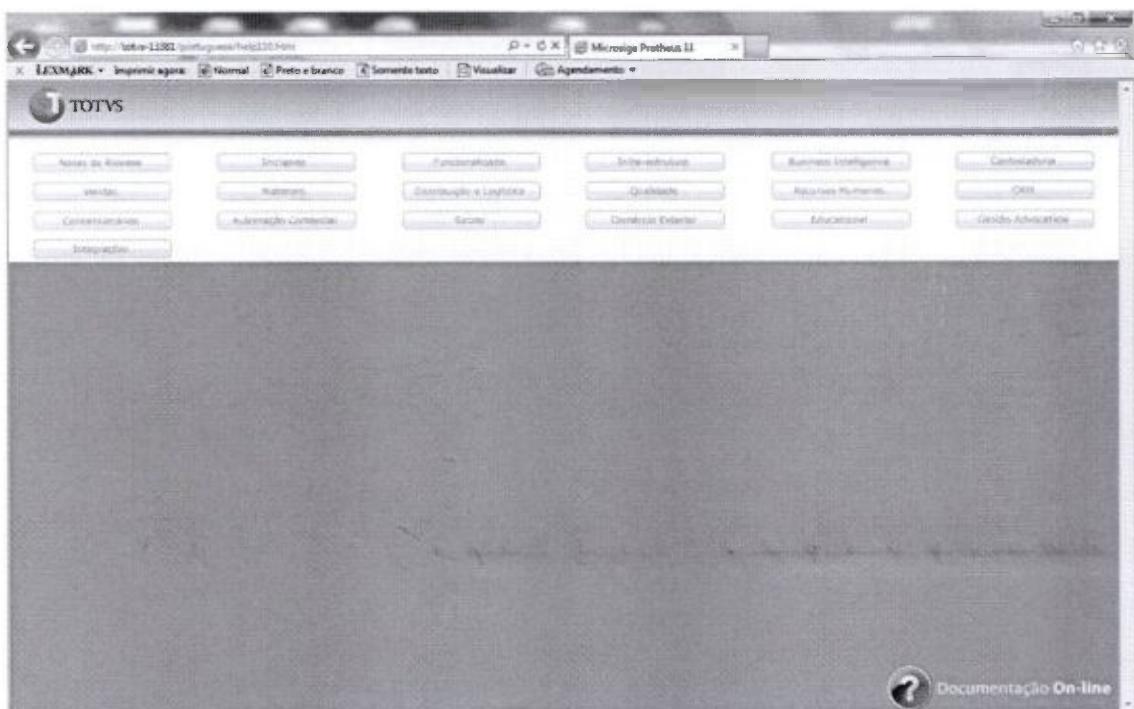
- Portal Protheus
- DW - Data Warehouse
- BSC- Balanced Scorecard
- WPS - WebPrint/WebSpool
- RH Online - Terminal do Funcionário e
- WS - Web Service.

Clique em Não, se não desejar executar o Assistente de Configuração, neste momento. O Assistente de Configuração pode ser acessado após a instalação pelo Wizard:

Se desejar executar o assistente de configuração, clique em Sim. Consulte ao final deste guia as informações sobre as seções do arquivo de configuração do servidor (appserver.ini).

5.1. Instalando Central de Ajuda

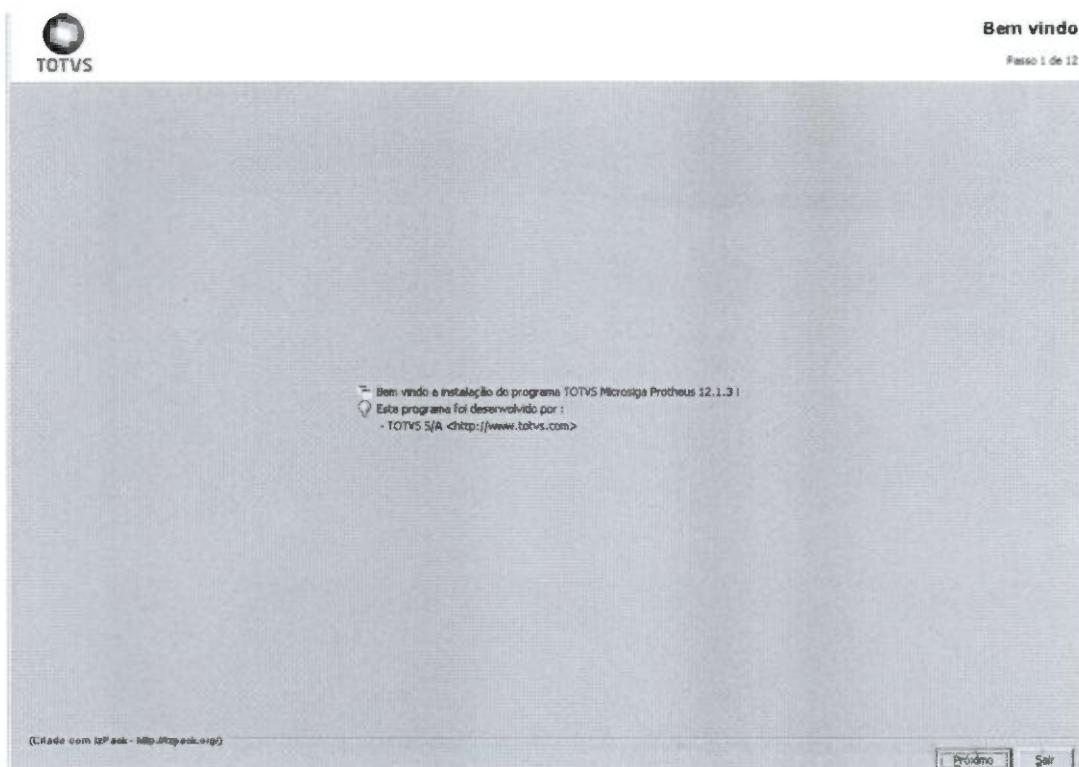
Para consultar a documentação on-line, é necessário instalar o Help do Protheus 12. A documentação é acessada a partir dos ambientes do Protheus, pela tecla[F1].



Coloque o CD-ROM do Help do Protheus 12 no drive e aguarde a exibição da tela de abertura conforme a seguir.



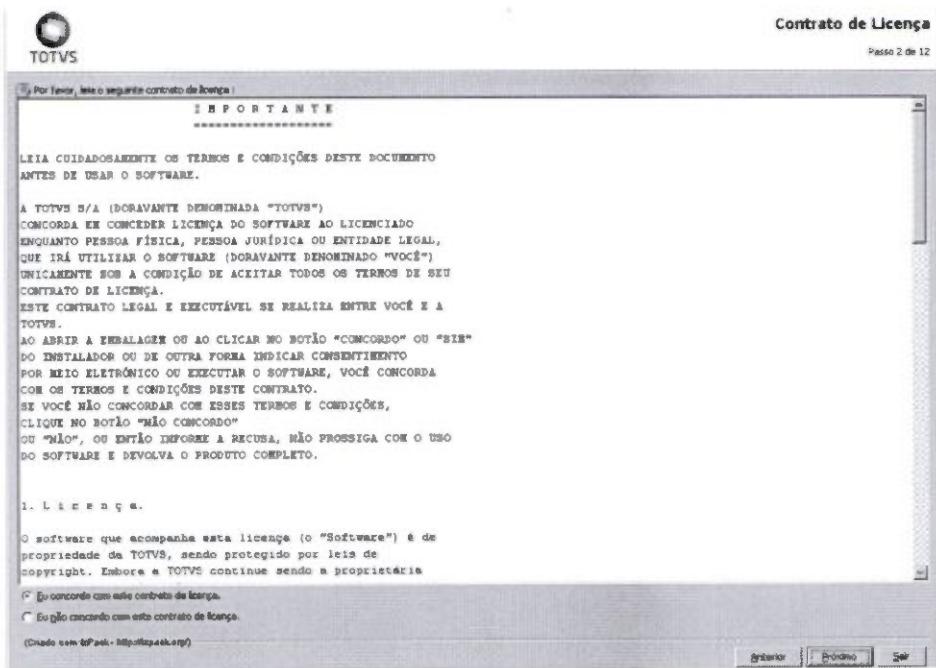
A instalação será iniciada. Janela de Bem-vindo é apresentada.



Clique no botão "Próximo" para prosseguir.

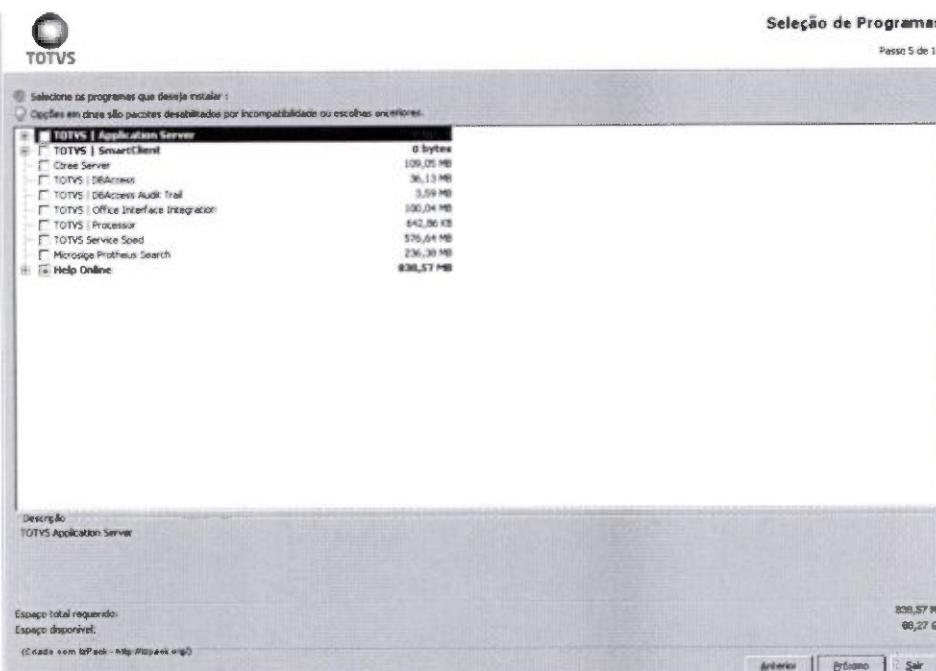
Na tela de Contrato de Licença, Leia o contrato de licença com atenção e clique no botão Sim para prosseguir.

Formação Programação ADVPL



Diretório de instalação dos Arquivos de Help.

Selecionar a opção "Help Online" para instalação.



Com a configuração da instalação do help, será iniciado o processo de cópia dos arquivos.

Este procedimento configura o acesso aos helps a partir do Application Server, caso o usuário queira utilizar outro servidor de help, poderá copiar os arquivos instalados para outro servidor e alterar o arquivo AppServer.INI, indicando a sua nova localização, através da configuração da chave "PATH" da seção HTTP e da chave HELPSERVER do ambiente utilizado. Para isto, pode ser utilizado o Wizard (Assistente de Configuração do TOTVS Application Server).

5.2. Servidor de Licenças

License Server é um recurso computacional da TOTVS que têm como objetivo realizar o controle das licenças de uso dos softwares e aplicações TOTVS. Através deste recurso é possível prover de forma eficiente e segura as licenças e liberações de utilização de módulos e/ou funcionalidades das aplicações TOTVS em conformidade com o contrato firmado entre o cliente e a TOTVS.

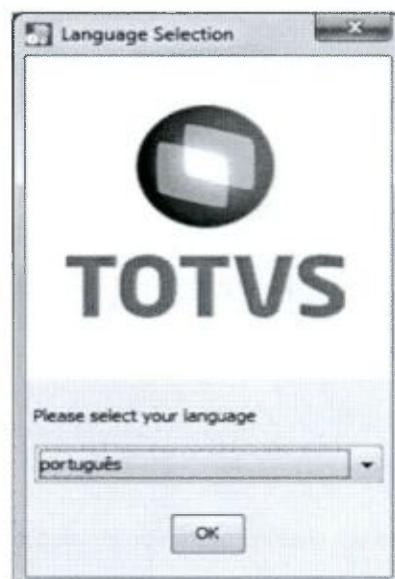
Cada instância do License Server identifica uma instalação física vinculada ao contrato de licenciamento do cliente. Para maximizar o uso do licenciamento TOTVS, recomenda-se a utilização de uma única instância do License Server, independentemente do ambiente de operação (Produção, Homologação e Teste). Caso opte-se por mais de uma instalação física do License Server, será necessário o registro da divisão das licenças do contrato entre as instalações físicas.

O License Server foi projetado para permitir uma configuração em alta disponibilidade do serviço e virtualização. Um sistema de alta disponibilidade (HA:High-Availability) é um sistema resistente a falhas de hardware, software e energia, cujo objetivo é manter os serviços disponibilizados o máximo de tempo possível. Para garantir a ausência de interrupções de serviço é necessário, muitas vezes, dispor de hardware redundante que entre em funcionamento automaticamente quando da falha de um dos componentes em utilização, provocando a interrupção momentânea do serviço durante a troca (Hot Swap).

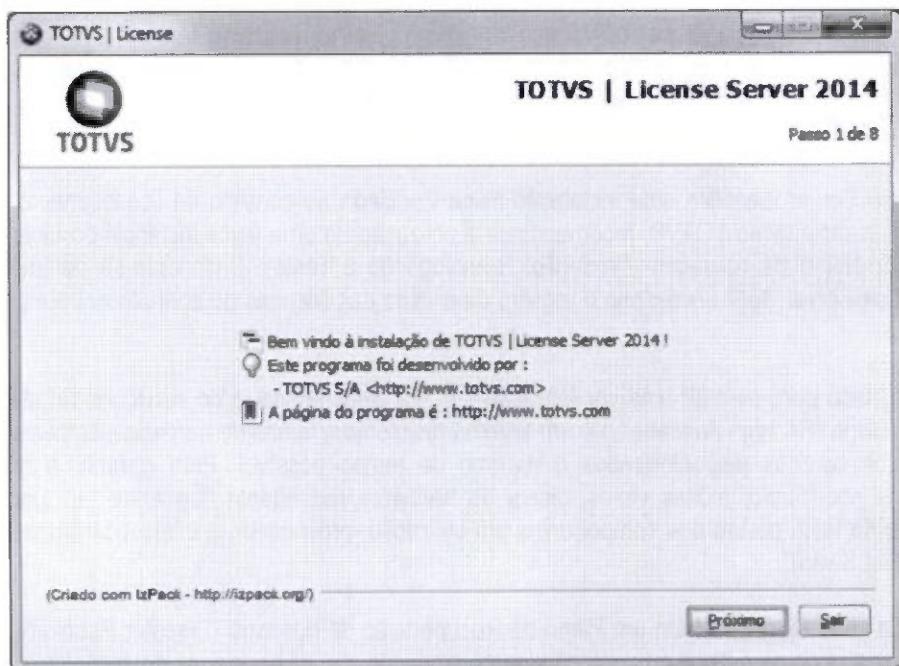
A TOTVS recomenda que seus clientes tenham um Plano de Recuperação de desastre (Disaster Recovery Plan), que contenha os procedimentos para permitir a recuperação ou continuação da infraestrutura de tecnologia de nossos softwares, na sequência de um desastre. Neste manual você irá encontrar nossa recomendação de plano. É imprescindível que o cliente conheça e simule a execução de nossas recomendações.

5.2.1. Instalação

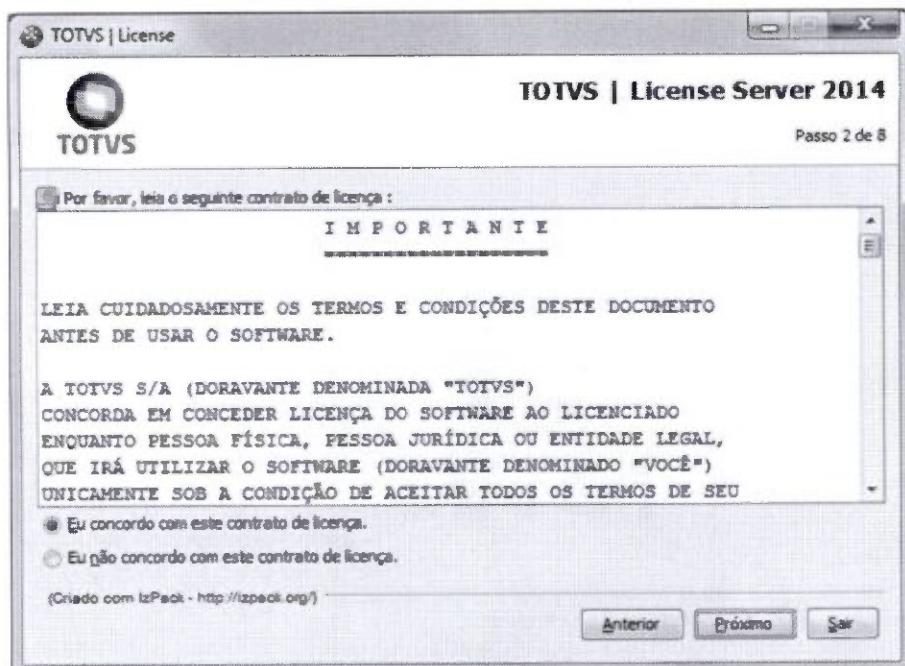
O instalador do TOTVS | License Server pode ser obtido no link Instaladores desta página (no canto superior direito) ou no Portal do Cliente, para maiores informações, consulte o Guia de Relacionamento e suporte. De posse do arquivo de instalação, execute o arquivo e selecione o Idioma de instalação, conforme mostra a figura abaixo:



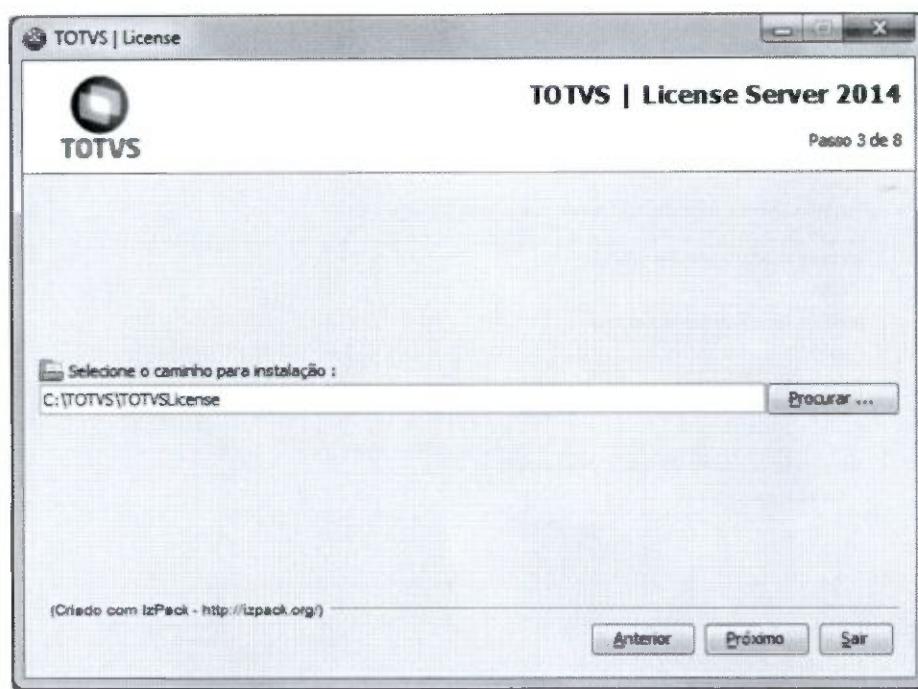
Após selecionar o idioma, será apresentada a tela abaixo:



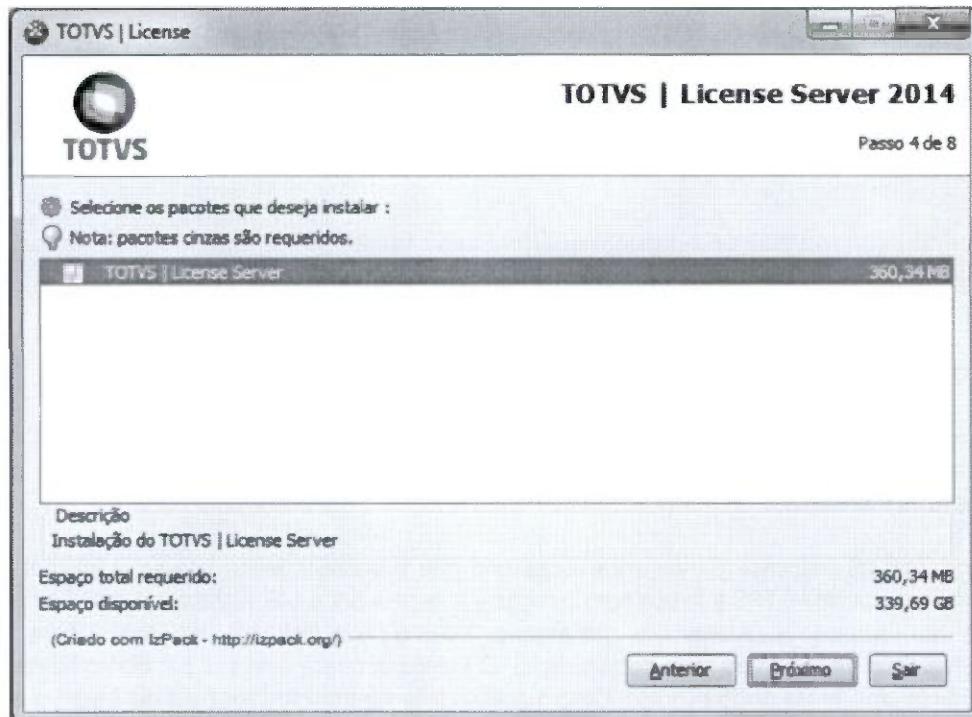
Prosseguindo será exibido o EULA do software TOTVS, estando de acordo, prossiga a instalação.



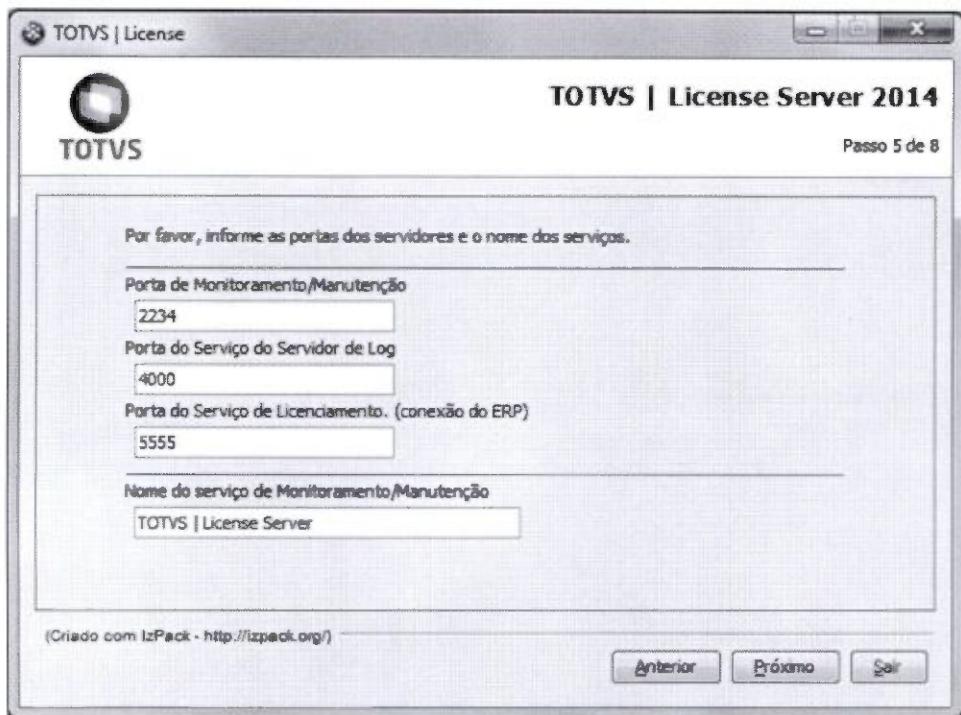
O instalador do TOTVS | License Server sugere um diretório padrão de instalação, que pode ser alterado pelo usuário sem prejuízo de uso.



Prosseguindo, é apresentado as informações do espaço requerido para a instalação. Lembre-se que o TOTVS | License Server possui um banco de dados temporário irá requerer mais espaço que o apresentado na instalação. Para maiores informações, consulte o capítulo Requisitos do Sistema.



Serviço de Licenciamento precisam estar disponível para os demais servidores da TOTVS, pois estas portas são utilizadas por eles para conexão com esta instalação.



Na tela seguinte será inicio o processo de instalação do TOTVS | License Server, aguarde o termino.

Após o termino da instalação será apresentada uma tela para a criação do atalho da interface de monitoramento do TOTVS | License Server.

Por ultimo, será executado o Monitor do TOTVS | License Server, onde o mesmo deverá ser configurado.

Os componentes instalados do TOTVS | License Server são:

- Serviço de Licenciamento - TOTVS | License Server. Este serviço é instalado automaticamente e colocado em modo automático de inicialização. Caso seja necessário, deve-se alterar a conta do usuário de Logon do Serviço.
- Atalho da interface de monitoramento do TOTVS | License Server.

5.2.2. Como configurar o Aplicador de Licenças

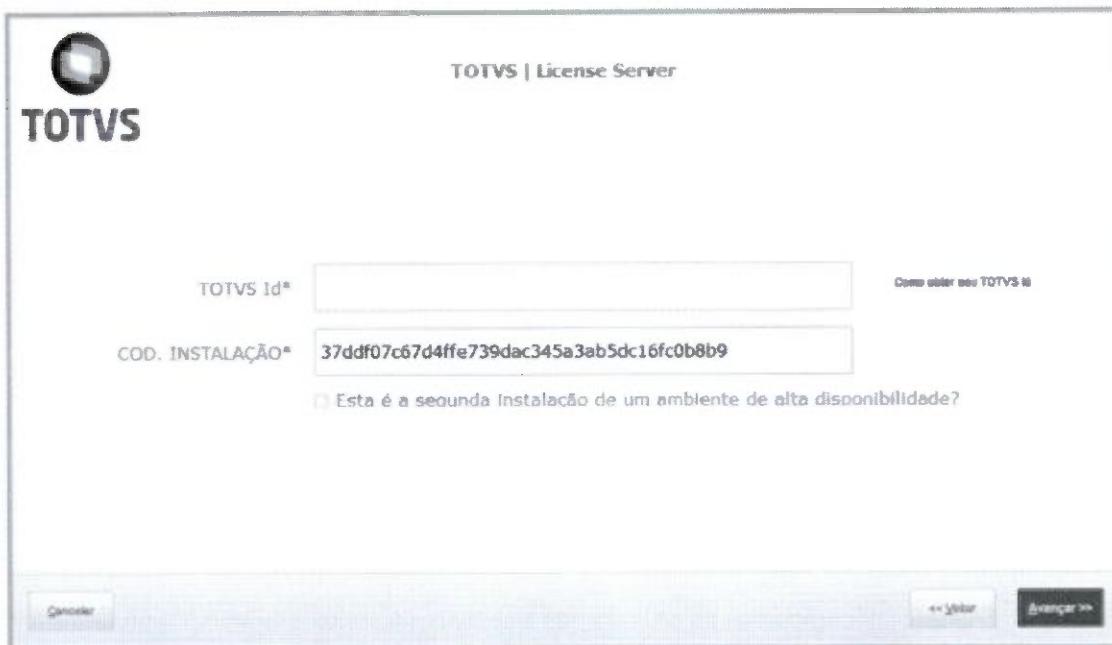
A configuração do TOTVS | License Server é realizada, logo após o termino na instalação. Se por algum motivo, não for possível se conectar a TOTVS. A configuração poderá ser realizada pelo atalho de monitoramento que foi criado durante o processo de instalação.

Para configurar o TOTVS | License Server será necessário que o servidor tenha acesso à internet. O Acesso é realizado através do protocolo HTTPS e toda a comunicação é feita pela porta 443. Caso a sua empresa possua firewall e/ou restrições de acesso à internet, é necessário que o administrador da rede libere o endereço <https://licenseservercloud.totvs.com.br/>, para comunicação. O tráfego ocorrerá uma vez por dia no horário configurado pela TOTVS, durante uma janela de 15 minutos. Caso o acesso seja interrompido por mais de 7 dias, a autorização de uso do sistema expirará e os usuários e/ou administradores dos software TOTVS serão notificados no Login do sistema, por mais 15 dias. Após este prazo, o sistema não estará mais disponível.

Formação Programação ADVPL



Ao iniciar o monitor do TOTVS | License Server, será exibida uma tela de boas vindas e na tela seguinte, será solicitado o TOTVS Id.



O TOTVS Id é a identificação do contrato do cliente. Para obter este número, o cliente deverá acessar o Portal do Cliente através do link <http://suporte.totvs.com/gestaodelicencas>.



Gestão de Licenças
Gerencie suas licenças
dinamicamente.

O Termo de Aceite é a peça fundamental para iniciar a utilização de seus sistemas TOTVS. Nele estarão presentes, para países como Brasil, Argentina e México, todas as informações necessárias para habilitar-se como cliente TOTVS dentro de seu respectivo contrato.

Abaixo, uma imagem ilustrativa da tela de Termo de Aceite Totvs, com destaque às abas contendo o país onde a empresa se encontra, os termos em si e os botões para aceitar (na coloração verde) ou rejeitar (coloração vermelha) o Termo de Aceite:

5.2.3. Gestão de Instalações

Todo cliente TOTVS tem, necessariamente, ao menos uma instalação em sua tela de instalações, a fim de facilitar o uso das instalações pelo cliente.

Nesta tela é possível criar novas instalações, através do botão "Incluir instalação", que deverá gerar um novo TOTVS ID em sua tela de instalações.

Abaixo, uma imagem ilustrativa da tela de Gestão de Instalações, com destaque para o botão da funcionalidade de incluir novas instalações, assim como alterar o bloqueio das mesmas.

TOTVS ID?

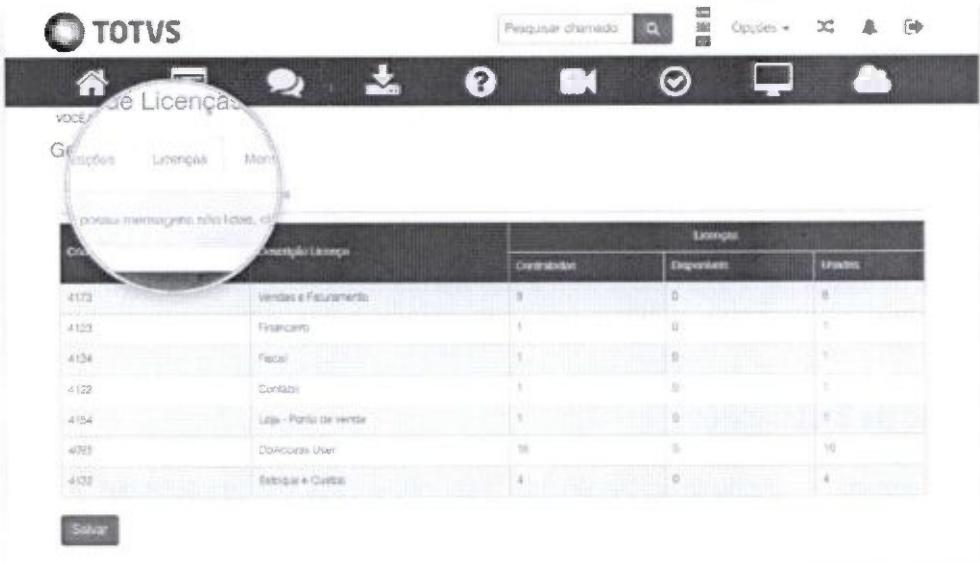


- TOTVS ID:** É uma identificação única de cada um dos ambientes instalados, que identifica de maneira eficaz, através (e inclusive) qualquer ambiente, assim como o cliente do qual pertence este ambiente, entre outras informações vitais para o bom funcionamento de seu ambiente.

5.2.4. Gestão de Licenças

O TOTVS ID é uma identificação única de cada um dos ambientes instalados, que identifica de maneira eficaz, através (e inclusive) qualquer ambiente, assim como o cliente do qual pertence este ambiente, entre outras informações vitais para o bom funcionamento de seu ambiente.

Abaixo, uma imagem ilustrativa desta tela, com destaque para seu acesso:



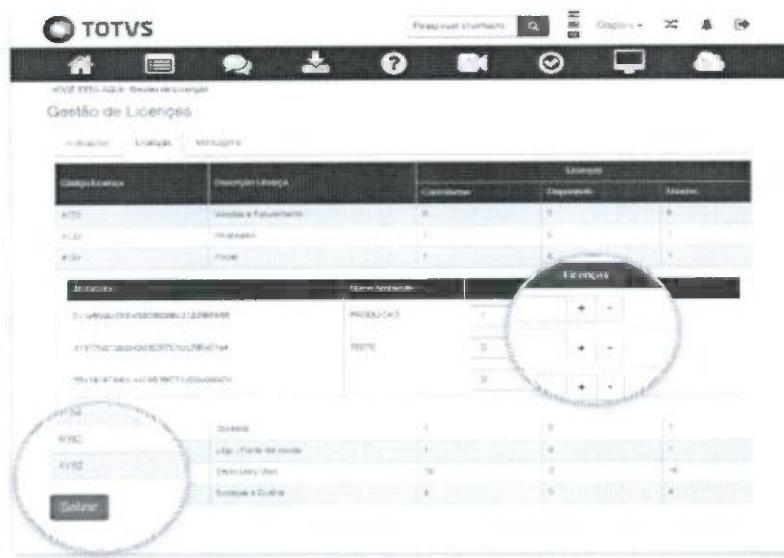
Formação Programação ADVPL



5.2.5. Licenças Tradicionais

As Licenças Tradicionais TOTVS são licenças contratadas pelo cliente com um determinado número de acessos para distribuição à critério do cliente.

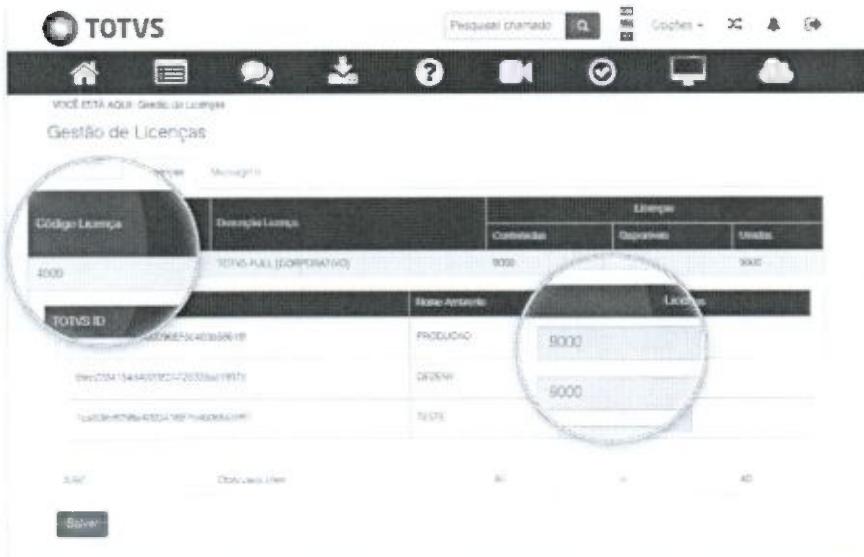
Abaixo, uma imagem ilustrativa das funcionalidades desta tela, com destaque aos controles para distribuir as licenças entre diferentes TOTVS ID, suas descrições e o botão de salvar, necessário para gravar a distribuição das licenças:



5.2.6. Licenças Corporativas

As Licenças Corporativas TOTVS possuem um diferencial que permite o uso por um número ilimitado de usuários nas instalações contratadas.

Abaixo, uma imagem ilustrativa da tela da tela de Gestão de Licenças para uma licença corporativa, com seu código em destaque:



5.3. Parâmetros do atalho AppServer

Para iniciarmos o (Aplicativo – AppServer.exe), devemos inicialmente entender seus (Parâmetros de Inicialização):

- **Console ou –Debug**

Executado como uma (Janela Console), as informações recebidas das conexões com o Application Server (AppServer) conectados são exibidas diretamente na tela do console do Application Server (AppServer), bem como informações de (Não Conformidades), e a execução do Application Server (AppServer) pode ser interrompida com o pressionamento da combinação de teclas [Ctrl]+[Break];

- **Install**

Se Application Server (AppServer), não for instalado como um (Serviço do NT), durante a Instalação, isto pode ser realizado, executando com a opção de (Linha de Comando). Ao optar por executar o Application Server (AppServer), como um (Serviço do NT), durante o processo de Instalação, o mesmo será adicionado à (Lista de Serviços do NT) e iniciado.

- **Remove**

Se Application Server (AppServer), foi instalado como serviço, pode ser removido com a instrução **-REMOVE**

O administrador poderá então (Iniciar ou Parar o Serviço), através do Item (Services), do (Painel de Controle do Windows NT) ou ainda removê-lo do serviço, através do Parâmetro (-Remove). Para removê-lo da (Lista de Serviços do NT), pode-se executá-lo com a opção de Linha de Comando).

Como Configurar os Parâmetros do Protheus 2 Server:

1. Acesse o Windows Explorer;
2. Localize a pasta C:\PROTHEUS12\BIN\APP SERVER \
3. Crie um Atalho do Arquivo APP SERVER.EXE, para a Área de Trabalho, do Windows;
4. Clique com o botão direito do mouse no Atalho e selecione a opção Propriedades;
5. Clique na pasta Atalho e observe que o caminho do atalho deve ser: C:\PROTHEUS12\BIN\APP SERVER\APP SERVER.EXE;
6. Acrescente o Parâmetro -Console, no Final do Atalho, para que o Protheus 12 Server, possa ativar a console do Protheus 12, informando os dados a seguir:

"C:\TOTVS 12\MICROSIGA\Protheus12\bin\APP SERVER\APP SERVER.EXE" –CONSOLE



7. Confira os dados e confirme a "Configuração do Parâmetro do Atalho do Application Server".

5.4. Parâmetros do Atalho SmartClient

O Smart Client (SmartClient), pode receber (Parâmetros de Inicialização), assim como o Application Server (AppServer).

Estes Parâmetros, servem para configurarmos o Smart Client (SmartClient) em sua Inicialização.

As opções de Linhas de Comando, do Smart Client (SmartClient) são as seguintes:

- **Q (Quiet)** – Indica que o TOTVS Smart Client (SmartClient), não deverá mostrar o Splash (Imagen de Apresentação) e a tela de identificação de Parâmetros Iniciais, necessita ser acompanhada da (Cláusula -P);
 - **P (Main Program)** – Identifica o Programa (APO) Inicial;
 - **E (Environment)** – Nome da Seção de Environment, no (Ini do Server), que será utilizada, para definições gerais;
 - **C (Connection)** – Nome da Seção de Conexão, que será utilizada, para a conexão ao TOTVS Application Server (AppServer);
 - **L (TOTVS Smart Client (SmartClient) Log File)** – Para Não Conformidades, que ocorram no TOTVS Smart Client (TotvsSmartClient), (Antes que este possa se conectar ao TOTVS Application Server (AppServer), é gerado um Arquivo de Log, no diretório de execução do TOTVS Smart Client (SmartClient). Este arquivo tem o nome definido pelo nome do executável (SmartClient), mais um Caracter de Underline (_), mais o Nome da Máquina em que o Smart Client (SmartClient) está sendo executado com a extensão (.LOG). Esta opção permite informar um nome específico para a geração deste Arquivo de Log, visando automatizações específicas que necessitem saber quando uma Não Conformidade, ocorreu no Smart Client (SmartClient). Por exemplo: Impossibilidade de Conexão;
 - **M (AllowMultiSession)** – Permite múltiplas instâncias (Cópias) do TOTVS Smart Client (SmartClient), na mesma máquina.
1. Acesse o Windows Explorer;
 2. Localize a pasta C:\PROTHEUS12\BIN\SMARTCLIENT;
 3. Crie um "Atalho" do Arquivo – SMARTCLIENT.EXE, para a Área de Trabalho, do Windows;
 4. Clique com o botão direito do mouse no Atalho Criado;
 5. Selecione a opção Propriedades;
 6. Clique na pasta Atalho e observe que o Caminho do Atalho, deve ser:
C:\PROTHEUS12\BIN\SMARTCLIENT\TOTVSSMARTCLIENT.EXE
 7. Acrescente os "Parâmetros", informando os dados a seguir:
C:\PROTHEUS12\BIN\SMARTCLIENT\SMARTCLIENT.EXE -Q -P=Sigacom -E=Environment -M;



8. Confira os dados e confirme a "Configuração dos Parâmetros do Smart Client (SmartClient)".

5.5. Arquivos de Configurações

A partir de agora, iremos conhecer um pouco mais sobre os Arquivos de Configuração do Protheus.

Dentre eles, aprofundaremos nosso conhecimento no arquivo (**AppServer.INI**), que é o Arquivo de Configuração, utilizado pelo Application Server (AppServer) e no arquivo (**SmartClient.INI**), que indica as configurações, para o Smart Client (SmartClient).

A saber, existem os seguintes Arquivos de Configuração no Protheus os quais estão localizados na pasta (\BIN), no diretório raiz, do Protheus:

- **AppServer.INI:** Através dele poderemos configurar praticamente todas as Funcionalidades disponíveis no Protheus.
- **SmartClient.INI:** Aqui realizaremos as configurações pertinentes ao TOTVS Smart Client (SmartClient), tais como: Direcionar a Conexão, Definir Protocolos Utilizados entre outros;
- **DevStudio.INI:** Neste arquivo ficam gravadas todas as opções do TOTVS Development Studio, tais como: Posição de Janelas, Fonte Utilizada, Últimos Arquivos Abertos, Preferências de Idioma, Diretório para Localização de Fontes entre outros. (Este arquivo não deve ser alterado manualmente);
- **Adslocal.CFG:** Este arquivo permite diversas configurações como: Número de Usuários, Número Máximo de Tabelas a serem abertas, Número de Workáreas disponíveis entre outros. (Este arquivo só é utilizado para instalações que possuam o "Parâmetro – LOCALFILES = ADS", no "Arquivo (AppServer.INI)");
- **DevStudio.CFG:** Aqui ficam guardadas todas as Configurações de Comunicação, realizadas no Totvs Development Studio. (Não deve ser alterado manualmente).

5.6. Configuração do appserver.ini

O arquivo appserver.ini é responsável pela configuração do Application Server. A sua atualização é realizada pelo Assistente de Configuração durante a instalação do Microsiga Protheus ou pela execução deste aplicativo a partir da pasta Protheus 12/Ferramentas/Assistente de Configuração.

5.6.1. Environment

As seções Environment contêm as informações dos diretórios de execução do Application Server, as informações do idioma, interface, diretórios e repositório e, opcionalmente, as informações para acesso ao DBAccess. Essas informações são identificadas para cada conexão, ou seja, cada Smart Client que se conectar ao Servidor deve informar qual a seção deseja utilizar. Assim, os diretórios para trabalho, o idioma, a interface e o repositório podem ser obtidos a partir das informações da seção definida. Uma seção padrão chamada Environment é criada na instalação e utilizada sempre que o Smart Client

Conectar-se sem informar uma seção pelos parâmetros de linha de comando.

- **SourcePath:** Identifica o diretório em que os repositórios de APOs são mantidos na máquina em que o Application Server está instalado. • *Onde está o RPO*
- **RootPath:** Identifica o diretório raiz do Microsiga Protheus®, a partir do qual todos os diretórios utilizados pelo Microsiga Protheus® serão criados. • *Raiz da instalação*
- **StartPath:** Identifica o diretório em que os arquivos de configuração do Microsiga Protheus estão. É o diretório inicial de execução do Sistema. Este diretório é criado a partir do diretório raiz, definido na chave anterior.
- **RpoDb:** Identifica o tipo de base de dados utilizada. Pode ser DBF, ADS ou SQL. Esta informação é utilizada pelo servidor para definir qual repositório será acessado. *LDB, ADS, SQL*
- **RpoLanguage:** Identifica a linguagem que será utilizada. Pode ser Portuguese, English ou Spanish. Essa informação também é utilizada pelo servidor para definição do repositório a ser acessado. RpoVersion Identifica a versão do Protheus.
- **AdsShare:** Indica o diretório compartilhado em que está instalado o servidor ADS. Este diretório é utilizado para fazer a conexão ao servidor ADS.
- **Trace:** Indica se deve ser feito o Log de warnings em arquivo. O arquivo de log (Trace.log) será criado no diretório: protheus12\bin\appserver. Valores possíveis: 1 (faz log), 0 (não faz log).
- **FilesOnDemand:** Indica se durante a abertura dos módulos, o Sistema efetuará a abertura pelo menu (padrão) ou por necessidade (abre e fecha arquivos conforme utilização). Os valores possíveis são: 1- Abre arquivos por demanda, 0 - Abre os arquivos a partir do menu.
- **LocalFiles:** Indica qual é a base de dados que será utilizada para abertura dos arquivos locais. Pode ser: ADS, ADSServer ou Ctree.
- **Localdbextension:** Define qual será a extensão default dos arquivos ISAM para os drivers DBFCDX, DBFCDXAX.
- **HelpServer:** Define o endereço do servidor HTTP de help online.

- **JumpSenhap:** Esta chave permite que o responsável da rede altere a segurança imposta pelo SenhaP, no caso de quebra ou perda do aparelho do Administrador do Sistema. Caso o conteúdo seja 1, o Sistema apresenta uma advertência sobre a abertura da segurança e permite acesso ao Sistema. Conteúdos possíveis = 0,1.
- **PictFormat:** Esta chave permite a criação de environments com composição de data no AMERICAN nesta chave. Conteúdos Possíveis =DEFAULT, AMERICAN DEFAULT=Default (Dia, Mês, Ano).

5.6.2. DbAccess

A seção DBAccess contém as informações utilizadas para o acesso à base de dados relacional por meio do DBAccess.

As informações desta seção são utilizadas para todas as conexões, a não ser que estejam definidas no environment das conexões, como detalhado a seguir. (Exceção: driver e ProtheusOnly). Database Identifica o nome da Database que deve ser utilizado para acesso via DBAccess à base de dados (MSSQL, ORACLE, DB2, SYBASE, INFORMIX e outros).

Para verificar a homologação dos bancos e suas versões homologadas consulte a página tdn.totvs.com (Inteligência Protheus > Engenharias > Engenharias de Banco > Bancos Homologados).

Server Identifica o nome ou o endereço IP do servidor.

Alias Identifica o alias utilizado no DBAccess para acesso à base de dados.

- **Protheus Only:** Se esta chave também estiver ativada, a conexão a este somente poderá ser feita Microsiga Protheus. Valores válidos: 0 ou 1 Default=0 (desligado)
- **PORT:** Porta de conexão Default = 7980. Opcionalmente, pode-se colocar estas 4 chaves no environment, acrescentando-se DB no início da chave.

5.6.3. Drivers

A seção Drivers define quais protocolos de conexão poderão ser utilizados para que os terminais conectem-se ao Application Server. Active Identifica os nomes das seções de configuração de protocolos ativos. Mais de uma seção pode ser informada separando-as por uma vírgula (,).

Deste modo, permitindo que existam conexões de terminais ao servidor efetuadas através de diferentes protocolos.

5.6.4. Servernetwork

Esta seção é utilizada no caso do uso de Balanceamento de Carga. Deve-se informar na chave a seguir quais são os servidores disponíveis na rede.

Servers Especificar os nomes das seções que contêm as informações dos servidores, separados por vírgula e na sequência de busca desejada. Dentro de cada nova seção de servidor, deve-se informar o nome, o tipo de conexão (TCP), a porta ou serviço e o número de conexões permitidas.

5.6.5. HTTP

Esta seção permite ao Application Server atuar como Servidor Http (Protocolo Internet); ou seja, pode fornecer páginas em HTML para um browser.

- **Enable:** 1 = Ligado e 0=Desligado
- **Path:** Local onde será o diretório raiz para as páginas.
- **Port:** Número da porta utilizada para a conexão http.
- **RPCTimeout:** Tempo limite para nova tentativa de conexão.
- **RPCEnv:** Nome do environment usado para a conexão.
- **RPCServer:** Nome da seção que indica a conexão do servidor para processamentos.

5.6.6. FTP

Esta seção permite ao Application Server atuar como Servidor FTP (Protocolo Internet).

- **Enable:** 1=Ligado e 0=Desligado.
- **Path:** Diretório raiz para os arquivos de FTP.
- **Port:** Define o número da porta de conexão FTP.

5.6.7. ONSTART

Nesta seção, determine a execução de uma função (sem interface) logo após o inicio de execução do Application Server.

Jobs Informe o(s) nome(s) de seção(s) para executar funções.

5.6.8. License Server

A seção License Server contém as informações sobre o Servidor de Licenças.

- **Enable:** Indica se o Application Server será um Servidor de Licenças. Valores válidos: 0 ou 1 Default= 0 (desligado)
- **Port:** Identifica a porta que será utilizada para a comunicação entre o Servidor de Licenças e o Application Server
- **ShowStatus:** Esta chave permite o controle de requisições, liberações de licença e semáforos pelas mensagens na tela de console. Valores válidos: 0 ou 1 Default = 1 -apresenta mensagens.
- **EnableNumber:** Identifica como será controlada a numeração no Application Server. Valores Válidos: 0-faz a numeração pelos arquivos SXE e SXF 1-controle pelo License Server (default)

5.6.9. License Client

A seção LicenseClient define as informações que serão utilizadas para o Application Server se conectar a um Servidor de Licenças.

- **Server:** É o nome ou número IP do computador onde está o Servidor de Licenças.
- **Port:** Identifica a porta que será utilizada para a comunicação entre o Application Server e o Servidor de Licenças. Portanto, deve ser o mesmo valor da chave Port na seção License server do Servidor de Licenças.

5.6.10. General

A seção General contém as informações globais sobre o Application Server, comuns a todos os ambientes.

- **CtreeMode:** Indica se o Ctree será utilizado em modo Local ou Servidor. Uma vez selecionado o modo pela chave todos os ambientes deverão utilizá-lo da mesma forma. Valores válidos: SERVER, LOCAL e BOUNDSERVER Default = LOCAL

5.6.11. Service

Utilize esta seção para informar o nome interno e externo do Serviço do Application Server no Windows Esta opção é útil quando for necessário utilizar mais de um Application Server, rodando como serviço na mesma máquina, pois permite ao usuário informar nomes diferentes.

- **Name:** Nome interno do Serviço.
- **Displayname:** Nome a ser exibido na janela de serviços do Windows

5.6.12. Exemplo do Appserver.ini

[HOMOLOGACAO]
; Nome da environment

SourcePath=D:\TOTVS\Protheus\apo
Localização do repositório de dados

RootPath= D:\TOTVS\Protheus\Protheus_Data
Localização do Protheus_Data (Pasta que antecede a system)

CTREERootPath=D:\TOTVS\Protheus\Protheus_Data
;utilizado com balance em ambiente com ctreeserver

StartPath='system'
;Pasta onde se encontra o dicionário de dados

X2_path=
; Utilizado quando a base de dados é dbf.

Formação Programação ADVPL



RpoDb=Top

Tipo do repositório de dados (c-tree, dbf,top)

RpoLanguage=portuguese

Linguagem do RPO

RpoVersion=120

Versão do repositório Versão 12

LocalFiles=c-tree

Dicionário em Ctree

Trace=0

Habilita a gravação de um arquivo log (Padrão=0) contendo as informações sobre todas as chamadas (wsstrace.log)

localdbextension=.dtc

Extensão do dicionário de dados (Ctree), na pasta system

PictFormat=DEFAULT

Esta chave atua na utilização de Picture @E, utilizada no Sistema em campos de entrada e saída de valores numéricos (@...SAY/GET...PICTURE'@E...'), bem como na função AdvPL Transform(), quando utilizada esta picture.

DateFormat=DEFAULT

ddmmmyyyy

TopMemoMega=1

Permite que as conexões SGBD (Sistema de Gerenciamento de Banco de Dados), realizadas através do TOTVS | DBAccess, utilizem campo "M" Memo com até 1000000 de bytes.

RegionalLanguage=BRA

;Região da Linguagem da aplicação

DBDataBase=MSSQL

;Tipo de Banco de dados utilizado

DBServer=127.0.0.1

; Servidor em que o DBAccess está localizado

DBALIAS=DADOSADV

; Nome do Alias criado para o Banco de Dados e configurado no DBAccess

DbPort=7890

;Porta do DBAccess

SPECIALKEY=PRDMT

A chave special key realiza o controle de numeração em colunas de tabelas que utilizam essa funcionalidade.

Helpserver=help.outsourcing.com.br/p12

Help da Aplicação

ConnectionTimeout=180

*-1 quando houver controle
pela read lock.*

Indica o tempo, em segundos, que a aplicação deve aguardar quando enviar um pacote de comunicação para o SmartClient solicitando retorno de dados.

InactiveTimeOut=180

Define o intervalo de tempo por inatividade, em segundos, para derrubar automaticamente a conexão entre o server e o client

MaxLocks=30000

Define o número máximo de locks simultâneos por conexão.

[SERVICE]

NAME=TOTVS-HOMOLOGACAO

DISPLAYNAME=.02 TOTVS P12 HOMOLOGACAO

;Nome do serviço no Windows

[Drivers]

Active=TCP

[TCP]

TYPE=TCPIP

Port=1122

[General]

CheckSpecialKey=0

;Utilizada se o controle de numeração for por HardLock e na utilização do SpecialKey.

InstallPath=C:\TOTVS\Protheus

Local de instalação do Protheus

CtreeMode=server

; Local do c-tree Server.

ServerMemoryLimit=1700

Determina o limite de alocação de memória (MB) residente.

ConnectionTimeout=180

Indica o tempo, em segundos, que o Server deve aguardar quando enviar um pacote de comunicação para o SmartClient solicitando retorno de dados.

ConsoleMaxSize=10485760

Define o tamanho padrão do arquivo (console.log) gravado pelo Server, quando utilizado como serviço, ISAPI ou quando a chave ConsoleLog está configurada como 1 na seção [General]. No exemplo, 10Mb

ConsoleLog=1

Determina a gravação de log das mensagens apresentadas no console do TOTVS | Application Server.

ConsoleFile=D:\TOTVS\Logs

Permite especificar um novo path e nome do arquivo para gravação do log de console do TOTVS | Application Server.

DebugThreadUsedMemory=1

Está chave habilita uma coluna no TOTVS | Monitor, onde será informada a quantidade de memória utilizada para cada processo apresentado no monitoramento.

IXBLOG=LOGRUN

Executa e armazena informações dos P.E. encontrados durante a execução do sistema.

IXBLOG=NORUN

Armazena as informações no LOG, porém NÃO executa o P.E.

CtreePreImg=1

Para uso com tabelas temporárias, é possível parametrizar o Server e o c-tree-Server em conjunto, viabilizando um acesso mais leve e mais rápido nas operações de inserção e atualização de registros. Testes realizados com a parametrização foram de 2 a 3 vezes mais rápidos.

Em conjunto, deve ser alterado o arquivo de configuração do c-tree Server (ctsvr.cfg), eliminando ou comentando as chaves COMPATIBILITY FORCE_WRITETHRU e COMPATIBILITY WTHRU_UPDFLG

[LICENSECLIENT]

server=127.0.0.1

port=5555

[SERVENETWORK]

SERVERS=SLV01,SLV02

MASTERCONNECTION=0

[SLV01]

SERVER=srvapp01

PORT=11001

CONNECTIONS=20

[SLV02]

SERVER=srvapp01

PORT=11002

CONNECTIONS=20

5.7. TOTVS DBAccess

Na seção [TOTVS DBACCESS] do arquivo de configuração do TOTVS Application Server, devem ser definidas as opções de conexão padrão dos ambientes (environments) configurados no Servidor. Estas configurações permitem a definição do banco de dados utilizado, alias, servidor e demais opções. Estas informações são apenas para ambientes que utilizam o repositório de objetos (APO) configurado para o TOTVS DBAccess como banco de dados principal.

Caso sejam utilizados mais de um ambiente com TOTVS DBAccess e exista a necessidade de estabelecer conexão com bancos de dados diferentes e/ou de estações com outro servidor TOTVS DBAccess, é possível configurar os parâmetros desta conexão com o TOTVS DBAccess na seção de configuração do próprio Ambiente (environment).

Para o DBAccess fazer comunicação com o bando de dados necessário cria uma ODBC fonte de dados do sistema.

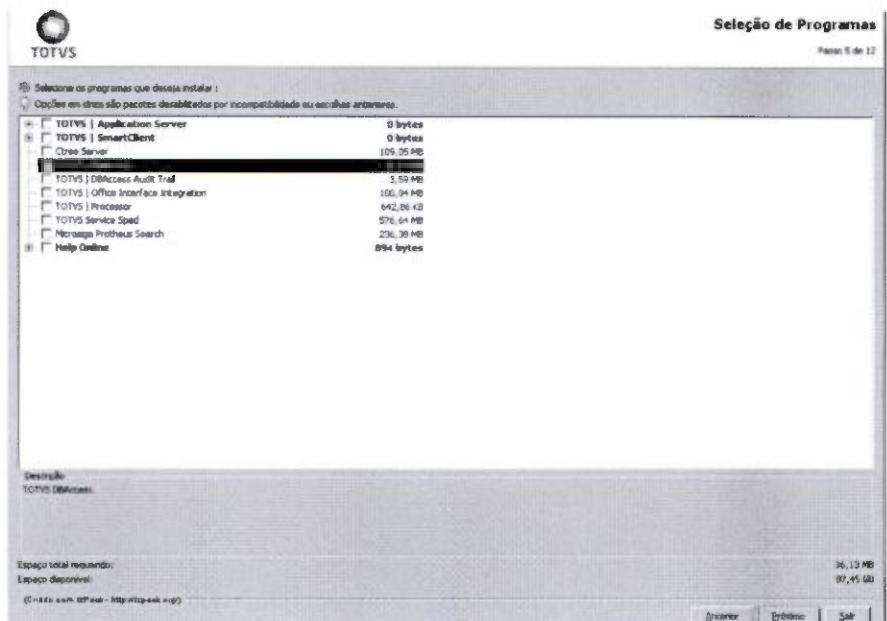
5.7.1. Instalação do TOTVS DBAccess

Coloque o CD-ROM no drive e aguarde a exibição da tela de abertura conforme a seguir.

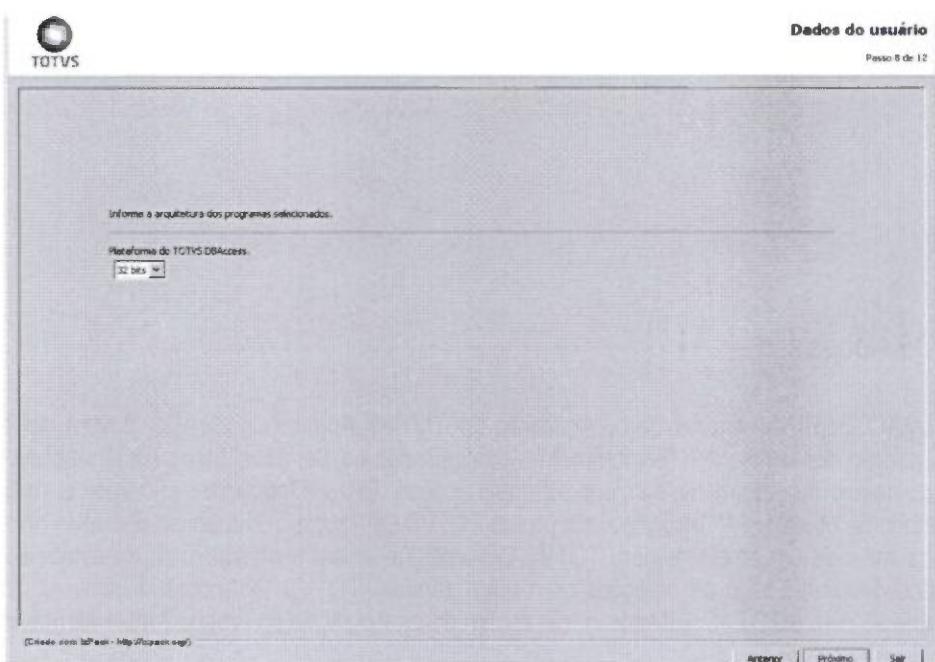
No inicio da tela, são apresentados os idiomas disponíveis para o instalador do Protheus.

Clique no ícone correspondente ao idioma que deseja utilizar.

Selecionar a opção TOTVS DBAccess.

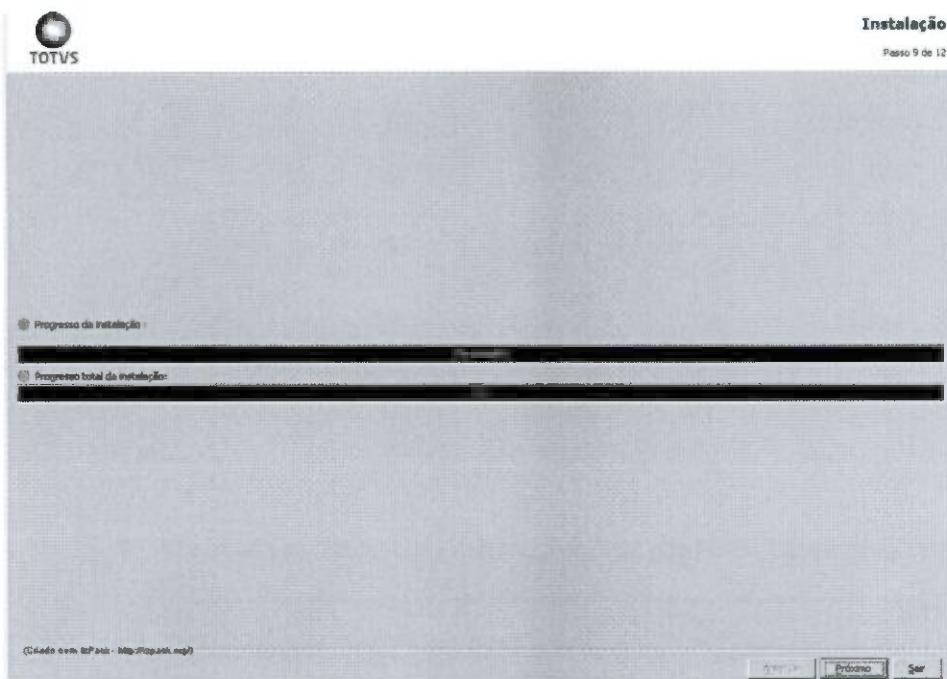


Informar a opção compatível com seu sistema operacional.



Selecionar o botão Próximo.

Formação Programação ADVPL



Após instalar o DBAccess, entrar no painel de controle, ferramentas administrativas e serviços. A seguir deixar o DBAccess Startado(automático).

5.7.2. Configurar a Conexão com o DBAccess no Protheus

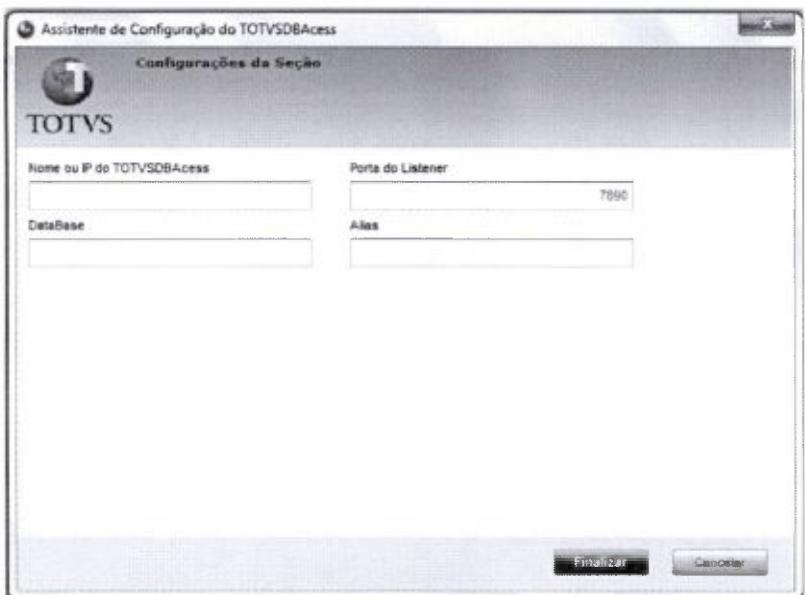
Selecione as seguintes opções "Iniciar" + "Programas" + "Protheus 12" + "Ferramentas" + "Assistente de Configuração do Servidor" ou no SmartClient digitar: **SRWIZARD**



Abra a opção "DBAccess" e clique no item "DBAccess".



Clique no ícone “Editar Configuração”; Preencha as “Configurações”, informando os dados a seguir:



- **Nome ou Ip do TotvsDbAccess:** Informar o nome ou ip onde esta instalado o dbAccess
- **Porta do Listener:** Numero da porta que foi instalado o dbAccess, porta default 7890
- **DataBase:** Nome do fabricante do bando de dados Exemplo: MSSQL,ORACLE e DB2
- **Alias:** Nome da ODBC.

Selecionar o botão "Finalizar";

5.7.3. ODBC – Open Database Connectivity

O ODBC é um padrão para acesso a sistemas gerenciadores de banco de dados independentemente de uma linguagem de programação, banco de dados e sistema operacional. Para criar um ODBC necessário.

1. No Windows, clique Start.

2. No Start Menu, clique Computer.
3. Em Computer, informar:
 - %systemdrive%\Windows\System32
 - %systemdrive%\Windows\SysWoW64 Para TOTVS | DBAccess 32 bits em sistemas operacionais 64 bits.
4. Em System32 ou SysWoW64, selecione e clique duas vezes no arquivo odbcad32.exe.
5. Na janela ODBC Data Source Administrator, acesse a guia System DSN e clique Add....
6. Na janela Create New Data Source, selecione o respectivo driver e clique Finish.
7. Na janela Create a New Data Source to SQL Server, preencha:
 - Name - Informe o nome da fonte de dados.
 - Server - Informe o nome do servidor do banco de dados.

Na janela Create a New Data Source to SQL Server, em How should SQL Server verify the authentication of the login ID?,

Selecione a opção With SQL Server authentication using a login ID and password entered by the user, marque Connect to SQL Server to obtain default settings for the additional configuration options.

Informe em Login ID o nome do usuário do banco de dados e, em Password, a senha do usuário do banco de dados.

Na janela Create a New Data Source to SQL Server, marque a opção Change the default database to e selecione o banco de dados.

Nessa mesma janela marque as opções Use ANSI quoted identifiers e Use ANSI nulls, paddings and warnings.

Na janela Create a New Data Source to SQL Server, marque a opção Perform translation for character data.

Na janela ODBC Microsoft SQL Server Setup.

5.7.4. Utilizando o TOTVS DBAccess

Todas a configurações e monitoramentos possíveis através do TOTVSDBAccess ocorrem neste monitor, permitindo gerenciar informações como:

- Usuários logados
- Bancos de dados em utilização
- Ambiente dos bancos
- Testes de conexão
- Monitores de índices
- Informações gerais, etc.

5.7.5. TOTVSDBMonitor

Acesse o ícone do programa criado no sistema operacional: "TOTVSDBAccess".

O sistema apresenta uma pequena janela referente às configurações de Monitor.

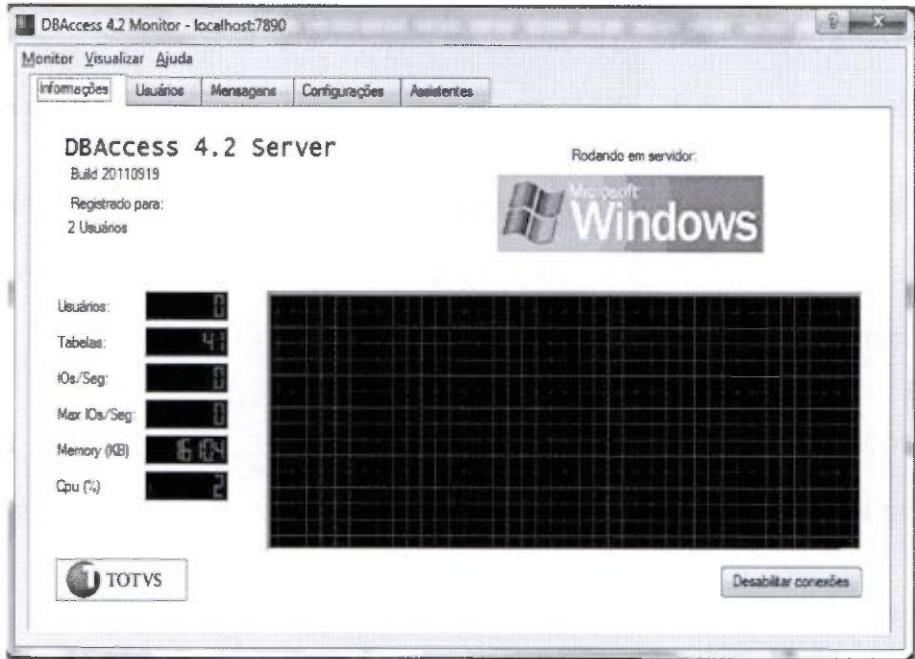


Servidor: Endereço do servidor de instalação do dbAccess

Porta: Porta configura no dbAccess porta padrão 7890

Após preencher os dados selecionar o botão OK .

A próxima tela apresenta as opções de gerenciamento de informações do DBAccess, subdividindo-as em pastas, sendo:



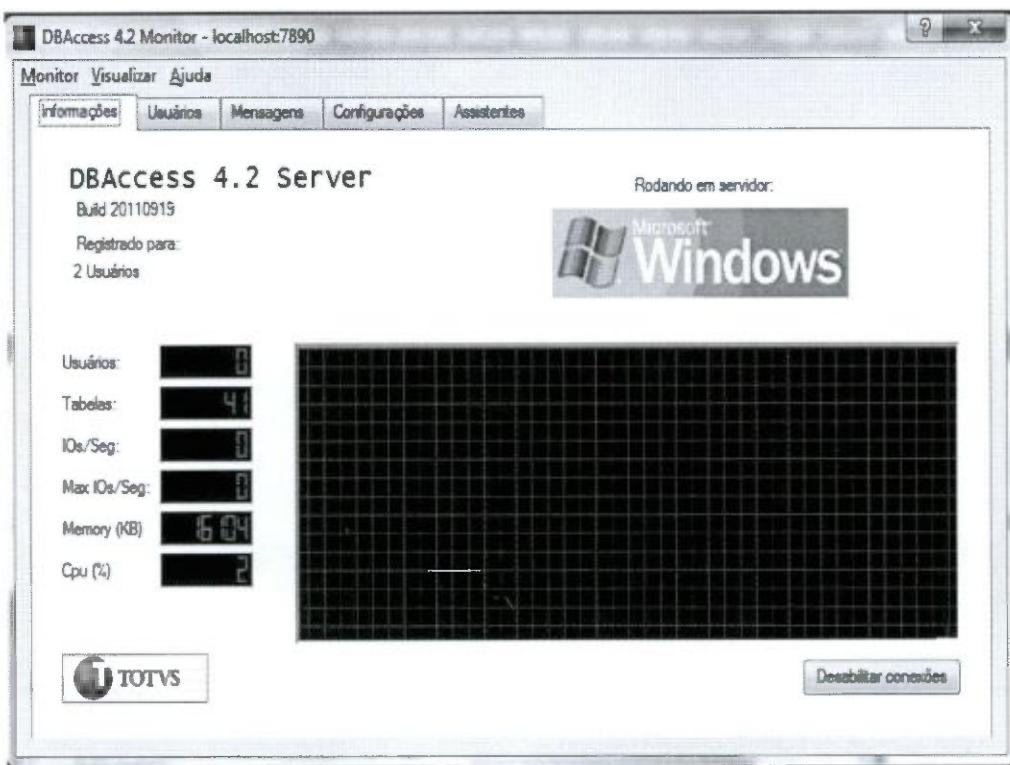
- **Informações:** Relação dos dados de registro do TOTVSDBAccess.
- **Usuários:** Relação dos usuários conectados ao TOTVSDBAccess.
- **Mensagens:** Apresentação de mensagens referentes a erros e ocorrências do banco de dados.
- **Configurações:** Configuração de acesso, senhas, usuários, chaves, para cada um dos bancos de dados suportados.

- **Assistentes:** Configuração e monitoramento de conexões e índices gerados.

Cada uma dessas pastas apresentam os campos necessários à informação dos principais dados, fundamentais ao gerenciamento oferecido pelo TOTVSDBAccess. Pasta "Informações".

Na abertura do TOTVSDBMonitor é apresentada a primeira pasta "Informações", com os dados de registro do TOTVSDBAccess, versão, quantidade de usuários conectados, tabelas em uso, servidor de base etc.

Os dados apresentados como "Informações" do TOTVSDBAccess, referem-se ao seguinte:



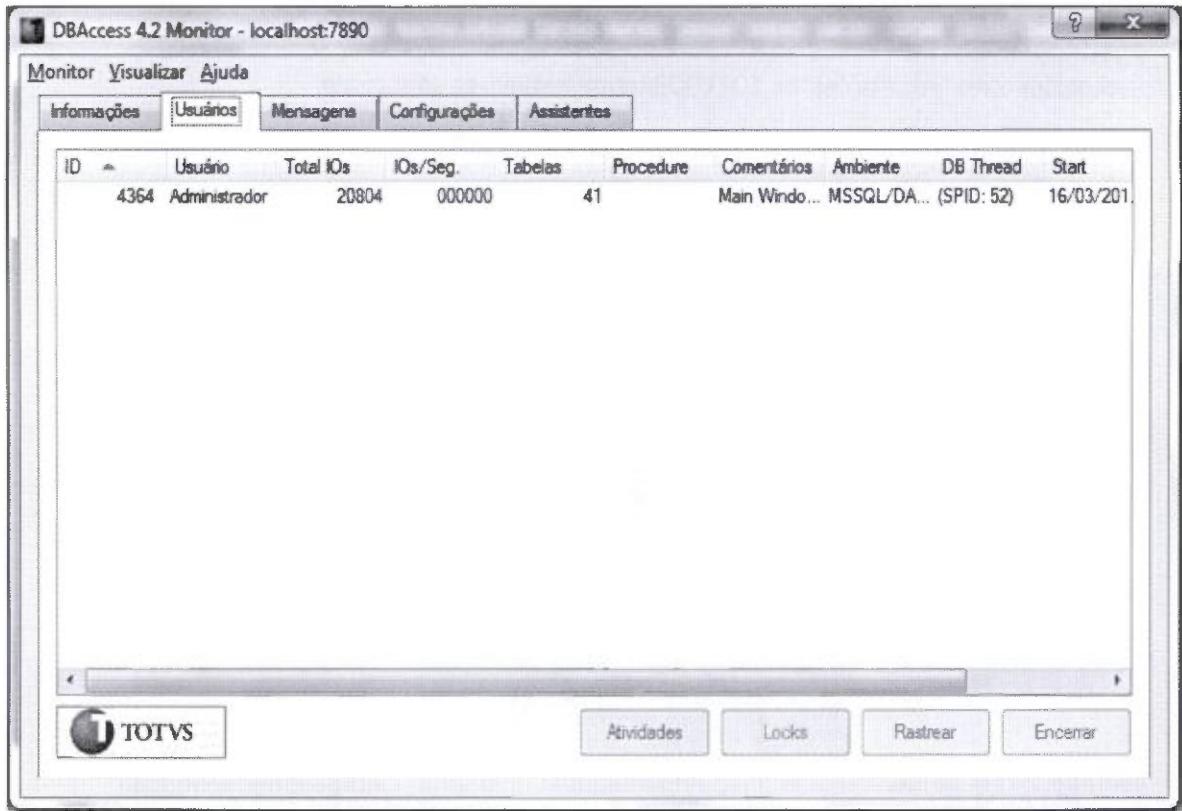
- Versão do TOTVSDBAccess e respectiva Build
- Número de usuários registrados para utilização
- Servidor em que atua o TOTVSDBAccess (o logotipo apresentado varia conforme o ambiente utilizado pelo servidor)

Possuindo as "Estatísticas", o monitor apresenta valores e gráficos referentes a:

- **Usuários:** Quantidade de usuários utilizando o banco de dados
- **Tabelas:** Quantidade de tabelas acessadas
- **IOs/Seg:** IOs por segundo (inputs/outputs = entradas e saídas), registrando a velocidade das informações
- **Max IOs/Seg:** Máximo de IOs por segundo, registrando a maior velocidade das informações
- O botão "Desabilitar conexões" deve ser utilizado para não permitir novas conexões no TOTVSDBAccess.

5.7.6. Usuário

Nesta guia estão relacionados todos os usuários conectados na base de dados. Para monitorar os usuários possui as opções:



- **ID:** Relaciona o número de identificação do usuário.
- **Usuário:** Relaciona o nome do usuário.
- **IOs:** Relaciona o número de entradas e saídas do usuário.
- **Tabela:** Relaciona o número de tabelas movimentadas pelo usuário.
- **Procedure:** Relaciona qual procedure está sendo executada.
- **Comentários:** Descreve a rotina sendo executada pelo usuário.

5.7.7. Rastrear

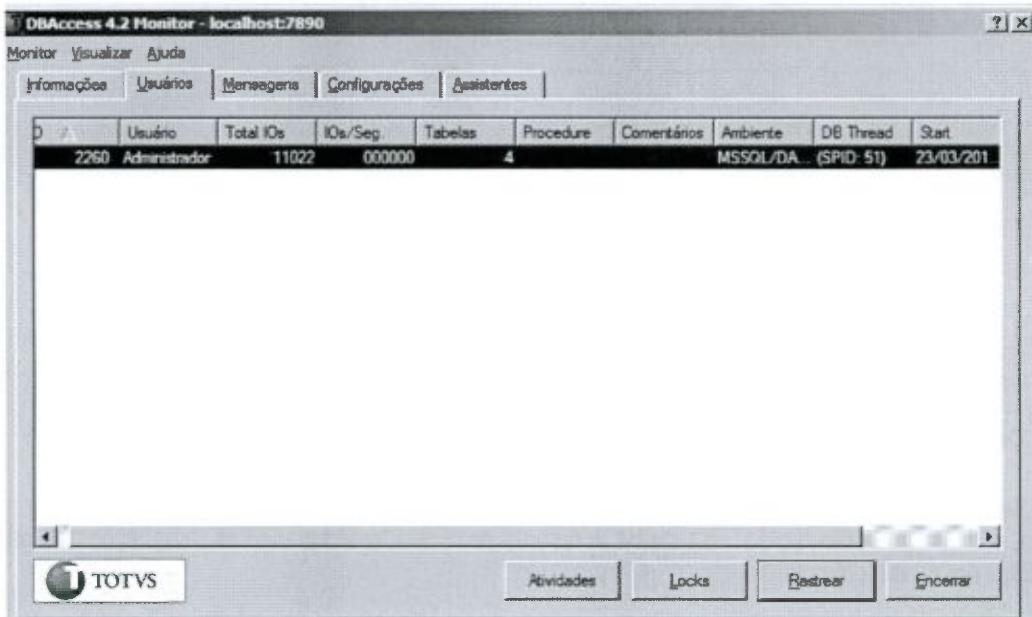
Para monitorar os processos em execução selecionar o usuário e clicar no botão "Rastrear".

Seq.	Tempo	Rotina	Query
1	0.000	Set Thread Trace	- Running at Thread ID [2260]
2	0.000	...	User [Administrador]
3	0.000	...	IO [11007]
4	0.000	...	Tables [3]
5	0.000	...	MaxTables [3]
6	0.000	...	Comment []
7	0.000	...	Status []
8	0.000	...	SP []
9	0.000	...	Traced [Yes]
10	0.000	...	InTran [No]
11	0.000	...	DBEnv [MSSQL/DADOS115]
12	0.000	...	DBThread [(SPID: 51)]
13	0.000	...	Started [23/03/2015 10:01:28]
14	0.000	...	LastIO []
15	0.000	...	IP [127.0.0.1]
16	0.000	...	RCV [30093]
17	0.000	...	SND [416854]

Listando todas atividades do usuário no banco de dados.

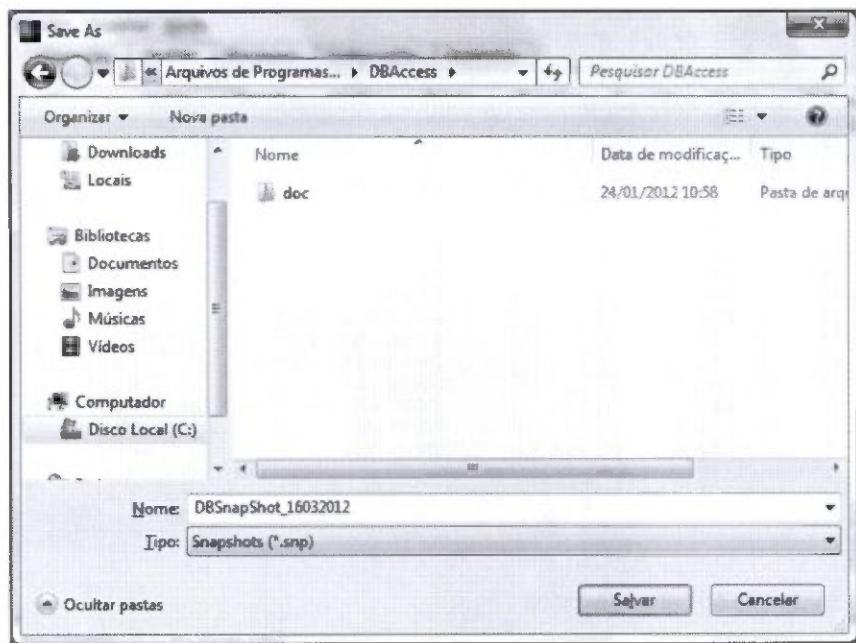
5.7.8. Encerrar

Para encerrar a conexão do usuário no banco de dados, clique no botão "Encerrar"



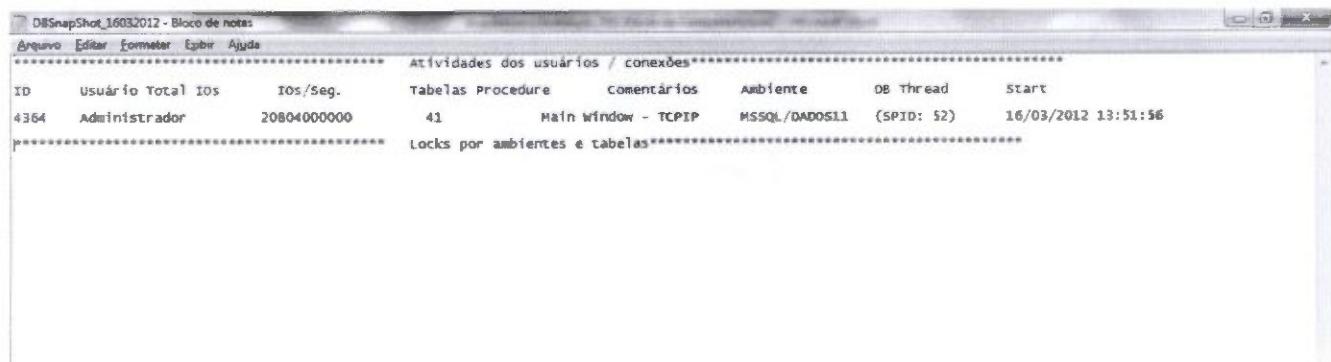
5.7.9. Atividades

O botão "Atividades" permite salvar as todas as atividades do usuário que foi selecionado, para gerar as atividades selecione o usuário desejado e selecione o botão "Atividades".



Será apresentada a tela para salvar as atividades do usuário, selecione o diretório desejado e clique no botão "Salvar".

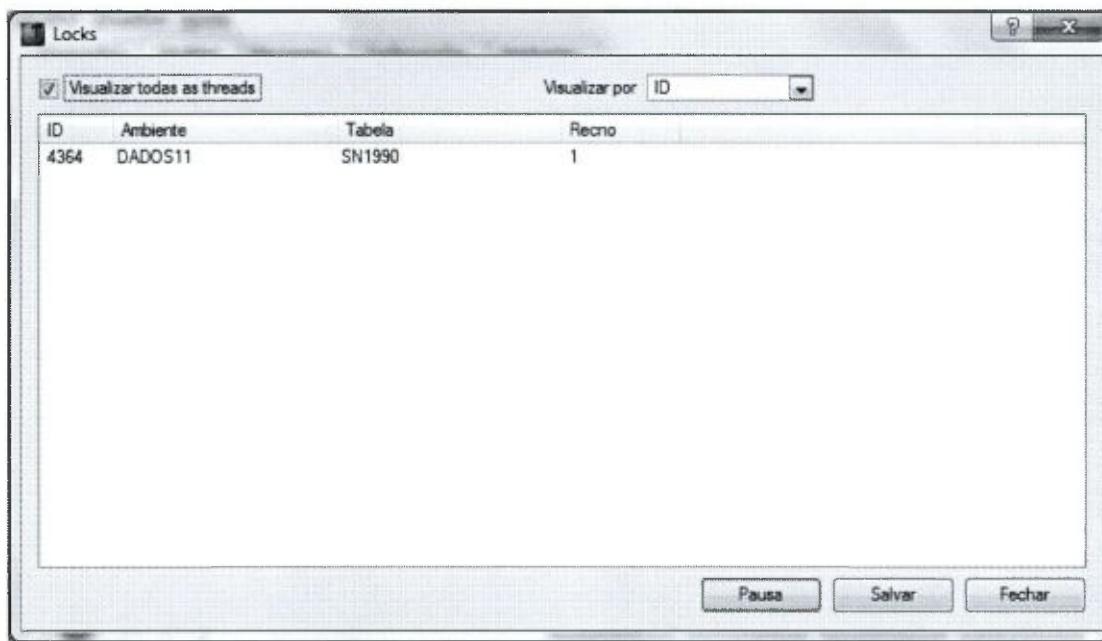
Observe no arquivo salvo as atividades que estavam sendo executada pelo usuário.



Atividades dos usuários / conexões								
ID	Usuário	Total I/Os	I/Os/Seg.	Tabelas	Procedure	Comentários	Ambiente	DB Thread
4364	Administrador	20804000000	20804000000	41	Main Window - TCPIP	MSSQL/DADOS11	(SPID: 52)	16/03/2012 13:51:56
Locks por ambientes e tabelas								

5.7.10. Locks

O botão "Locks" lista todos os registros que o usuário está trabalhando. Na pasta "Usuários", posicione o cursor sobre o usuário desejado, e clique no botão "Locks", o sistema apresenta nova tela, relacionando as informações de atividade do usuário, divididas em quatro colunas:



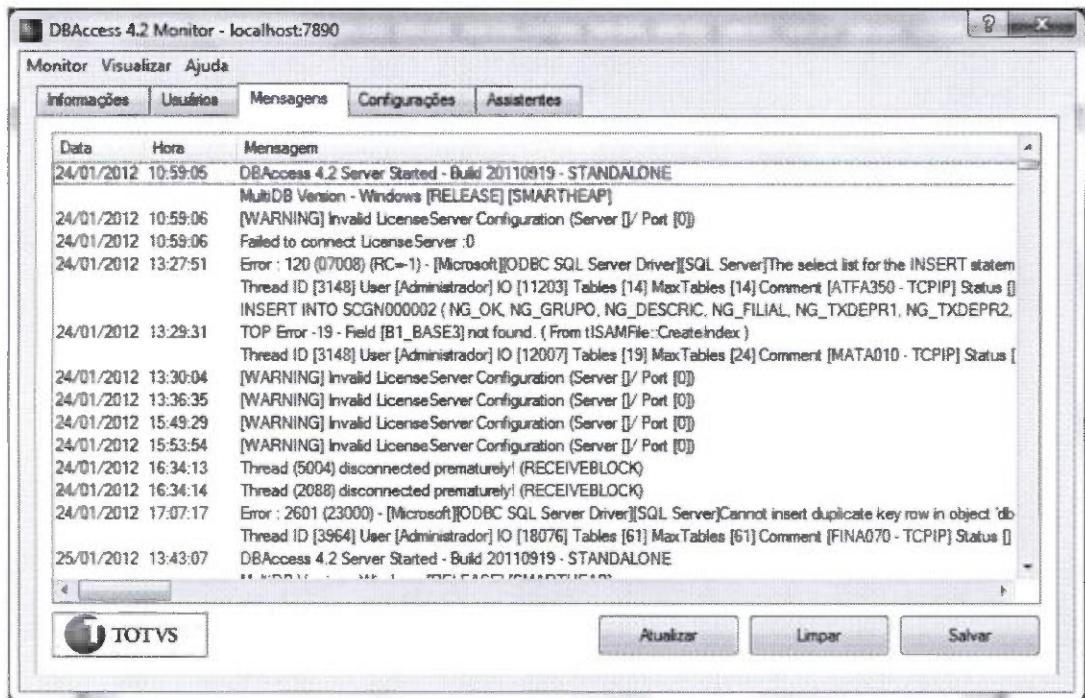
Visualizar por:

- **ID:** Relaciona os registros pela identificação do usuário.
- **Ambiente:** Relaciona o nome do ambiente TOTVSDBAccess.
- **Tabela:** Relaciona a tabela em uso.
- **Recno:** Relaciona o registro que está em uso para o usuário.

5.7.11. Mensagens

Na pasta "Mensagens" são apresentadas as mensagens de erro e ocorrências reportadas pelo servidor de banco de dados.

As informações são distribuídas em colunas que, por sua vez, relacionam os seguintes dados:



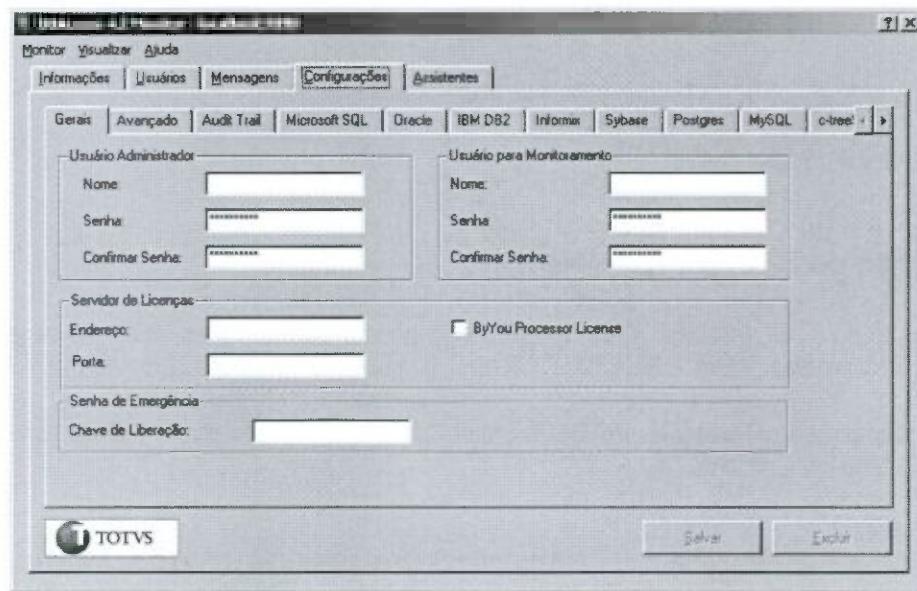
- **Data:** Apresenta a data da mensagem.
- **Hora:** Relaciona a hora da mensagem.
- **Mensagem:** Relaciona as mensagens de erro do banco.

Para controlar as informações, estão disponíveis os botões:

- **Atualizar:** Atualiza as mensagens, datas e horários apresentados
- **Limpar:** Apaga as mensagens apresentadas e respectivos horários e datas
- **Salvar:** Grava as mensagens geradas e respectivas datas e horários

5.7.12. Configurações

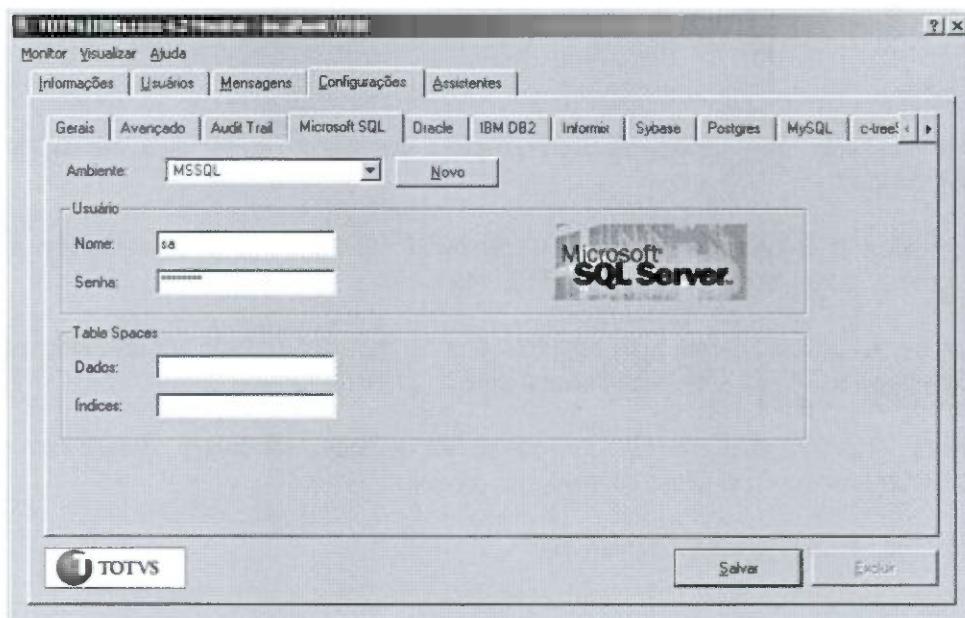
Na guia Gerais possui as configurações para fazer autenticação e servidor de licença.



- **Usuário Administrador:** Caso desejar colocar uma senha para acessar o Monitor do DBAccess só preencher os dados
- **Usuário Monitoramento:** Usuário limitado para monitoramento
- **Servidor de Licenças:** Informar o endereço e porta que foi configurado o hardLock
- **Senha de Emergência:** Senha liberada pela Totvs,

5.7.13. Configurando o drive de ODBC no dbAccess

Acessar a guia do respectivo banco instalado, iremos trabalhar com Microsoft SQL.



- **Ambiente:** Podemos criar um novo ambiente, Exemplo informar o nome do ADBC.
- **Usuário:** Usuário do banco de dados que possui de acessos administrador
- **Table Space:** Nome do table spaces

5.8. Configuração do TOTVS Smart Client

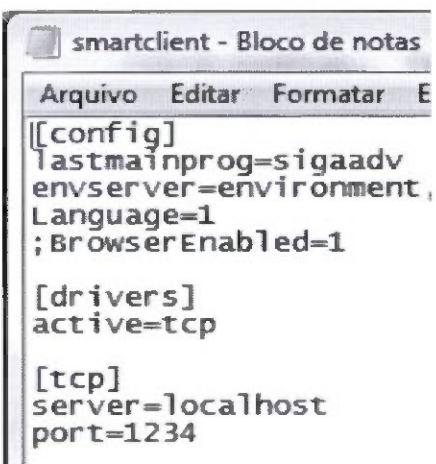
As Configurações do TOTVS Smart Client (TotvsSmartClient), podem ser alteradas, editando-se o (Arquivo-TotvsSmartClient.INI), localizado na (Pasta – \BIN\ SMARTCLIENT \), na raiz do diretório do Protheus 12.

Neste arquivo encontraremos as Configurações do TOTVS Smart Client, para conexão com o TOTVS TotvsAppServer, sendo o mesmo, configurado manualmente.

Acesse o "Windows Explorer";

Localizar o "Arquivo – SmartClient.INI", na "Pasta C:\TOTVS12\MICROSIGA\PROTHEUS12\BIN\ SMARTCLIENT \";

Edite o "Arquivo" e siga os "Parâmetros" a seguir:



```

smartclient - Bloco de notas
Arquivo Editar Formatar E
[config]
lastmainprog=sigaadv
envserver=environment,
Language=1
;BrowserEnabled=1

[drivers]
active=tcp

[tcp]
server=localhost
port=1234

```

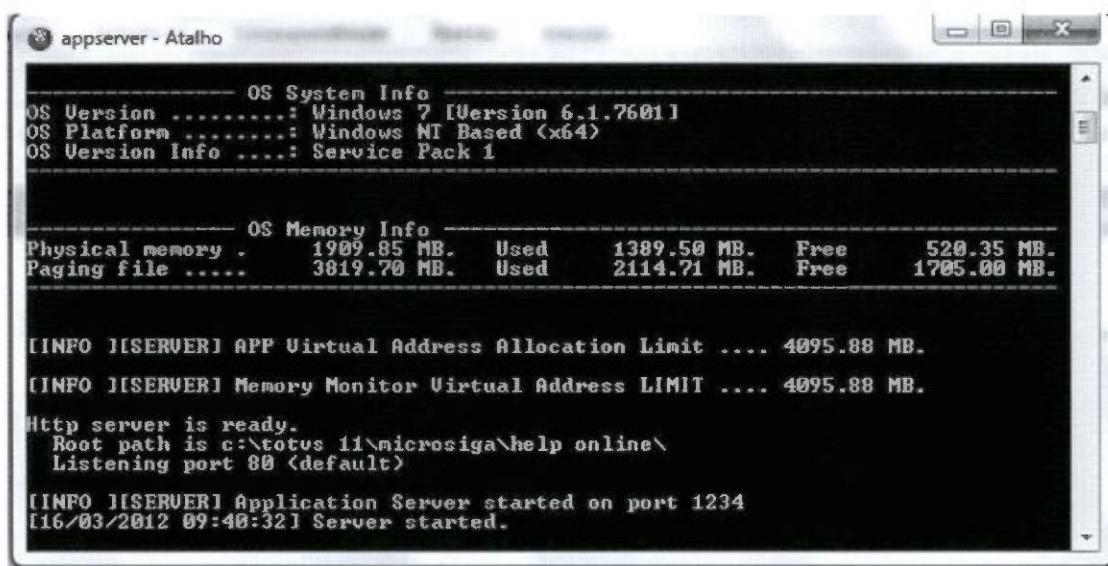
Vamos iniciar o entendimento do "Arquivo de Configuração", observando a estrutura básica do arquivo "TotvsSmartClient.INI" a seguir:

- **Config** – Identifica as "Informações Gerais de Configuração", do TOTVS SmartClient Contém a lista dos Últimos Programas Iniciais, executados no do TOTVS SmartClient.
- **Drivers** – A seção "Drivers", define quais os "Protocolos de Conexão" poderão ser utilizados, para que o Terminal conecte-se ao TOTVS Application Server e também qual é o "Nome do Serviço".
- **Nome Drivers** – Identifica qual é o "Servidor" que deverá ser localizado para a conexão com o TOTVS Application Server e também qual será a porta utilizada para essa "Conexão".

Confira os dados e confirme a "Configuração do SmartClient".

5.9. Acessando o Protheus

Execute o "Servidor do Protheus", através do ícone "AppServer" criado na área de trabalho com a instrução no atalho – console para aplicação iniciar;



```
appserver - Atalho

OS System Info
OS Version ..... : Windows 7 [Version 6.1.7601]
OS Platform ..... : Windows NT Based (x64)
OS Version Info ... : Service Pack 1

OS Memory Info
Physical memory . 1909.85 MB. Used 1389.50 MB. Free 520.35 MB.
Paging file ..... 3819.70 MB. Used 2114.71 MB. Free 1705.00 MB.

[INFO ][SERVER] APP Virtual Address Allocation Limit .... 4095.88 MB.
[INFO ][SERVER] Memory Monitor Virtual Address LIMIT .... 4095.88 MB.
Http server is ready.
Root path is c:\totvs 11\microsiga\help online\
Listening port 80 <default>
[INFO ][SERVER] Application Server started on port 1234
[16/03/2012 09:40:32] Server started.
```

Execute o "Client do Protheus 12", através do ícone "SmartClient";

Confira os dados e confirme os dados:

- **Parâmetros Iniciais:** Informar o modulo que deseja acessar Exemplo: SigaADV
- **Comunicação no cliente:** Informar o nome da comunicação criada no SmartCliente.ini
- **Ambiente no servidor:** Nome do Environment disponível



Dados para acessar o Protheus:

- **Usuário:** Administrador
- **Senha:** "Sem senha"



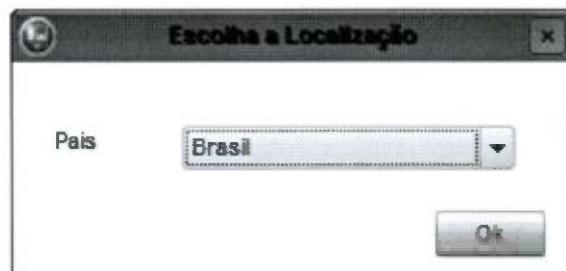
Informar os dados:

- **Data Base:** Data base do sistema, pode logar no sistema com datas diferentes
- **Empresa/Filia:** Informar os dados da empresa e filial que irá trabalhar
- **Ambiente:** Informar qual modulo deseja trabalhar



Após preencher os dados acessar o botão "Entrar". Os arquivos de "Helps dos Campos", serão atualizados;

Na janela "Localização", selecione "Brasil" e confirme;



Agora o Protheus, irá criar os Arquivos Customizadores".



Ao aparecer a janela "Diretório dos Arquivos de Dados", a partir do "Servidor", selecione o diretório "\Data\", confirme-a.

5.10. Totvs Development Studio (DevStudio)

A Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio), é a Ferramenta de Edição, Compilação e Depuração de Erros do Microsiga Protheus.

Com esta Ferramenta, podemos aplicar Paths que são as Correções e/ou Atualizações que a Microsiga Protheus, envia a seus clientes.

A Ferramenta de Desenvolvimento DevStudio está apta a reconhecer a sintaxe da Linguagem ADVPL, que é uma Linguagem Proprietária da Microsiga Protheus.

A Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio)é o único modo de compilar os Arquivos de Programas em APO's (RPO), para serem registrados no TOTVS Application Server (AppServer) .

Para a utilização da Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio), alguns pontos devem ter atenção especial:

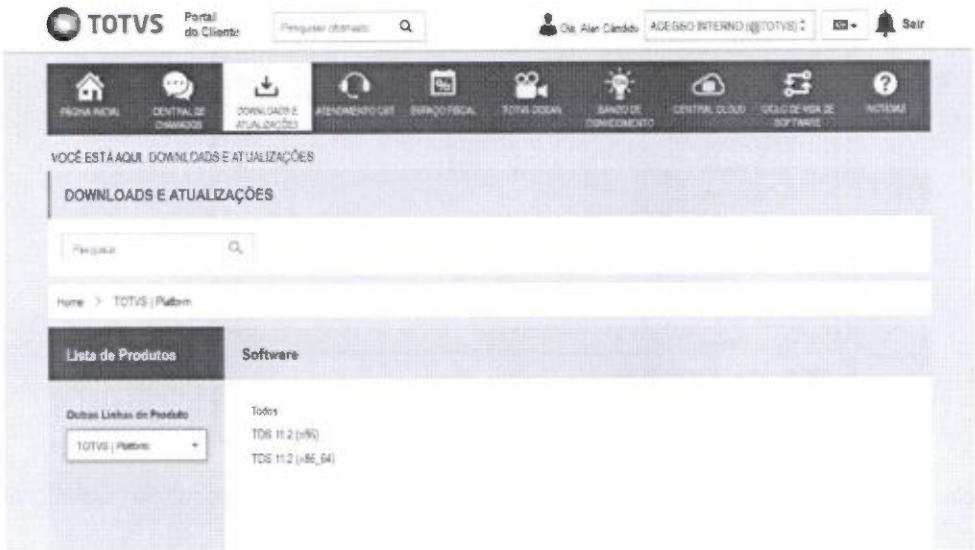
- A Instalação da Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio), deve ser realizada através do mesmo Programa de Instalação do TOTVS Application Server (AppServer);

Após a Ferramenta Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio), ter sido instalada.

Para iniciarmos Ferramenta de Desenvolvimento Protheus Totvs Development Studio (DevStudio), devemos ter necessariamente o TOTVS Application Server (AppServer).

5.10.1. Instalação

Acesse Suporte.totvs.com.br e fazer o login para acessar o portal. Acesse a área de downloads e informe os parâmetros de pesquisa: TOTVS | Platform.



Inicie o instalador do TOTVS | Developer Studio, conforme o seu sistema operacional.

Baixe o instalador adequado ao seu ambiente, arquitetura e versão desejadas, executando-o em seguida. Atente que existe versões específicas conforme o sistema operacional (Windows, Linux e Mac), arquitetura (32 ou 64 bits) e versão do produto.

Para iniciar a instalação, Acione o menu de contexto sobre o instalador e clique em "Run as Administrator". Selecione a opção "Sim" para descompactar a instalação.



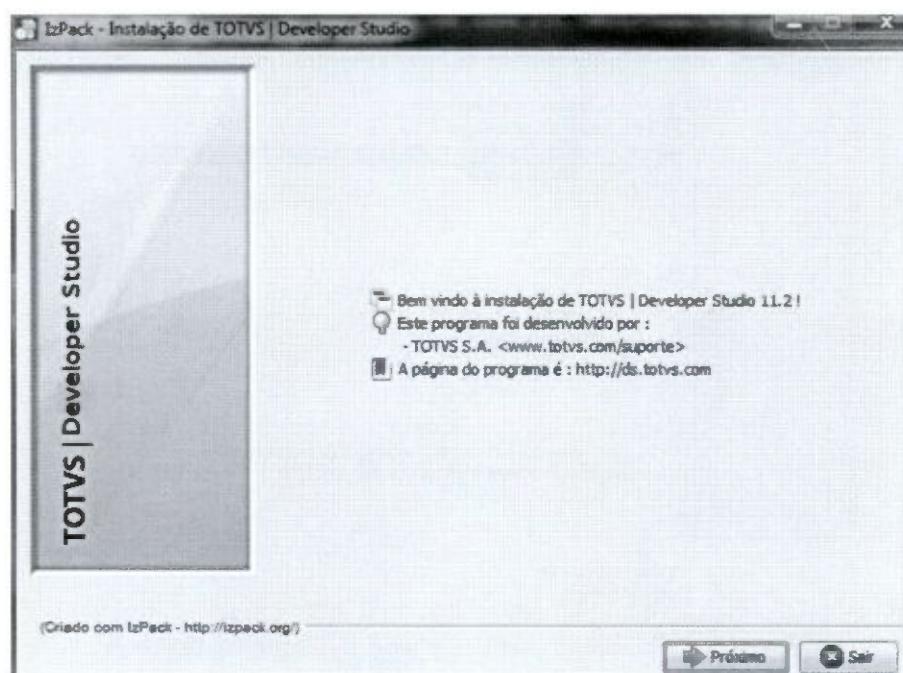
Formação Programação ADVPL



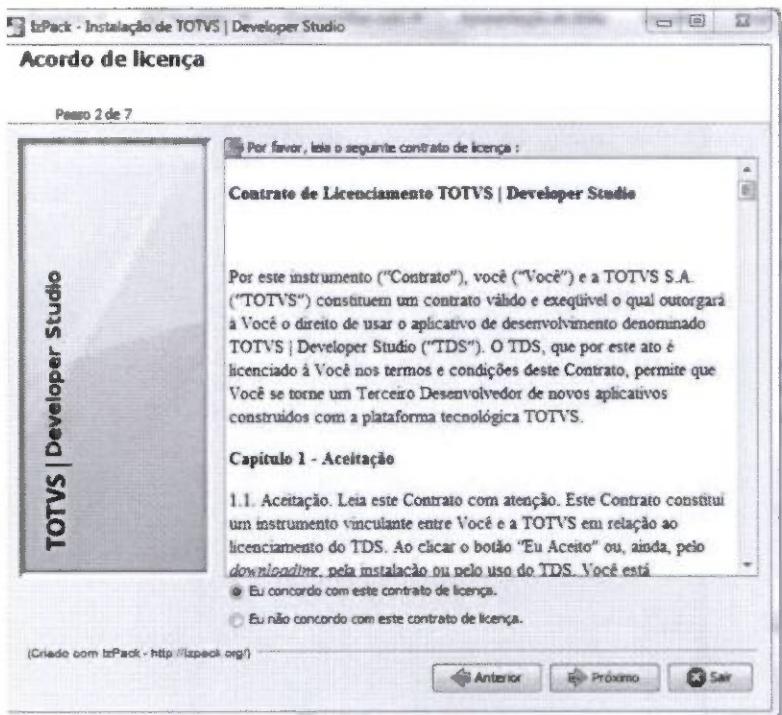
Selecione o idioma que deseja utilizar durante o processo de instalação.



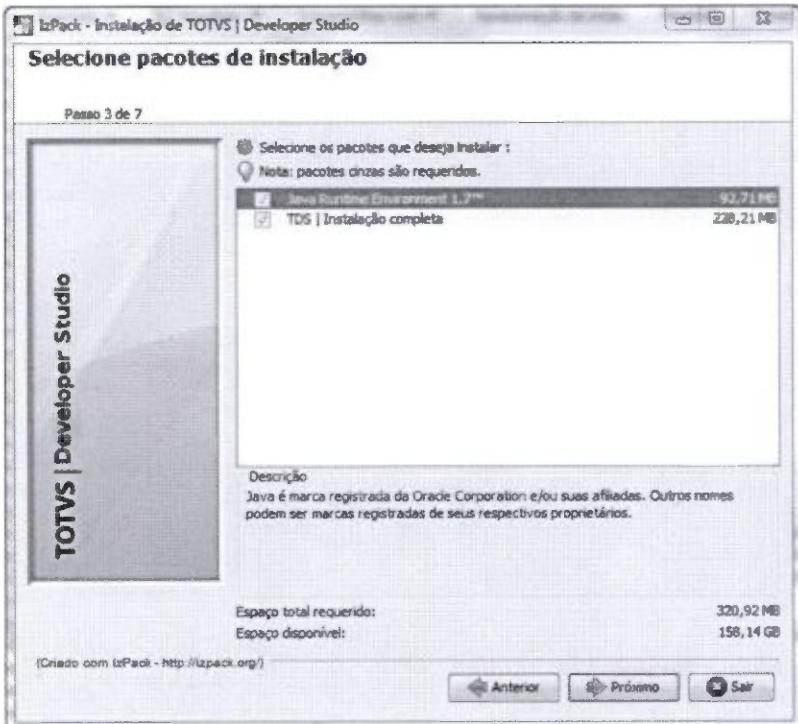
Com a tela de boas-vindas, que identifica o produto, avance acionando "Próximo".



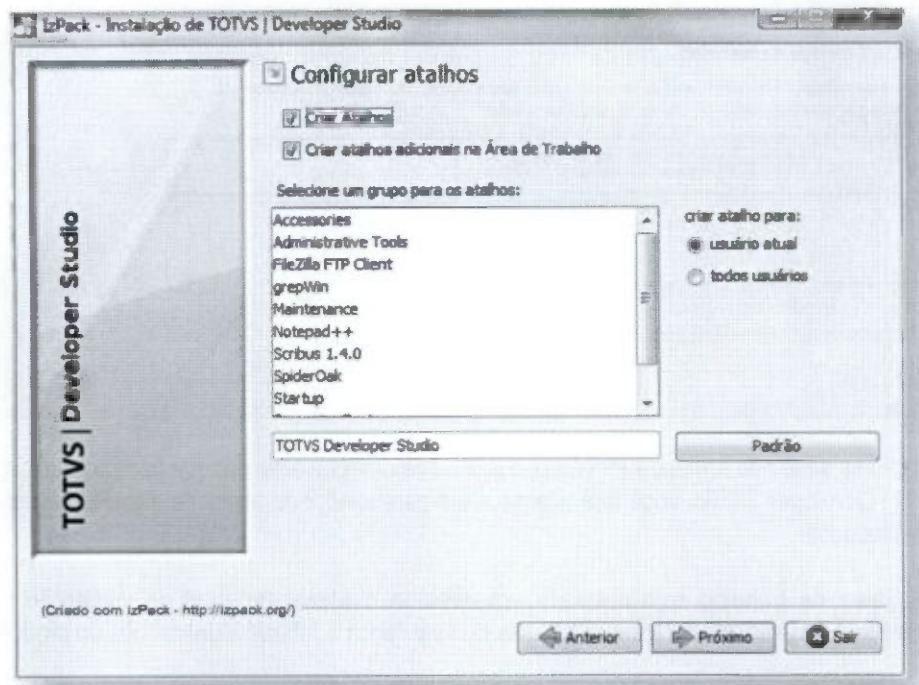
É apresentado nessa tela, o acordo de licença, leia atentamente o documento e se estiver de acordo com ele clique em "Eu concordo com esse contrato de licença".



Nessa tela é apresentado os pacotes que serão instalados junto com o TDS.



Selecione o local da instalação. Indique onde o menu e atalho para execução serão criados e avance, instalação foi concluída.



A instalação foi concluída.

5.10.2. Iniciando o TOTVS Developer Studio

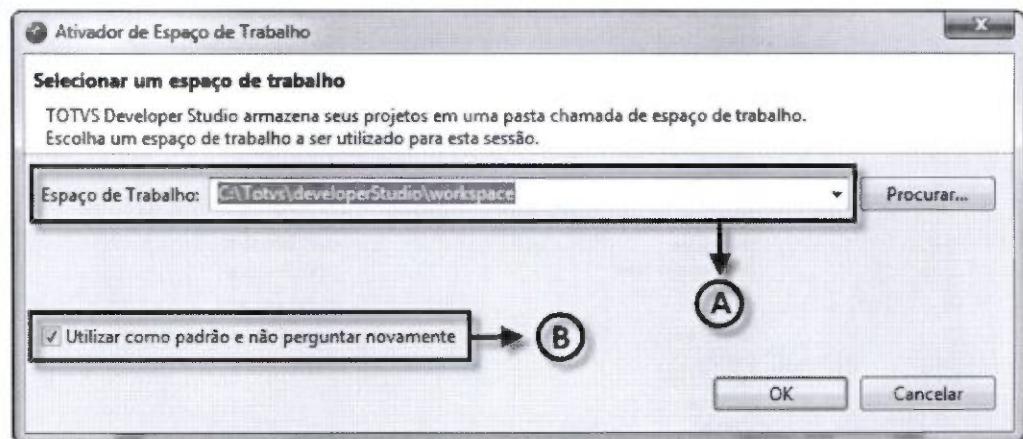
Para iniciar o uso do TOTVS | Developer Studio, execute o procedimento adequado ao seu sistema operacional. Assim que este iniciar, lhe será solicitado que informe o local da área de trabalho (*workspace*).

- **Área de trabalho (*workspace*)** – Local onde seus trabalhos podem ser armazenados e/ou acessados via ligação simbólica (*symbolic link*) ou fisicamente, informações de gerenciamento e organização dos fontes e outras informações necessárias ao funcionamento do TOTVS | Developer Studio.

Você pode manter quantas áreas de trabalho desejar, organizando seus trabalhos da forma que você achar mais confortável e natural. Por exemplo, mantendo uma área para cada produto/versão, cliente, projetos pessoais, etc.

Usando as ligações simbólicas (*symbolic links*), você pode compartilhar diretórios e arquivos, que são comuns a vários projetos.

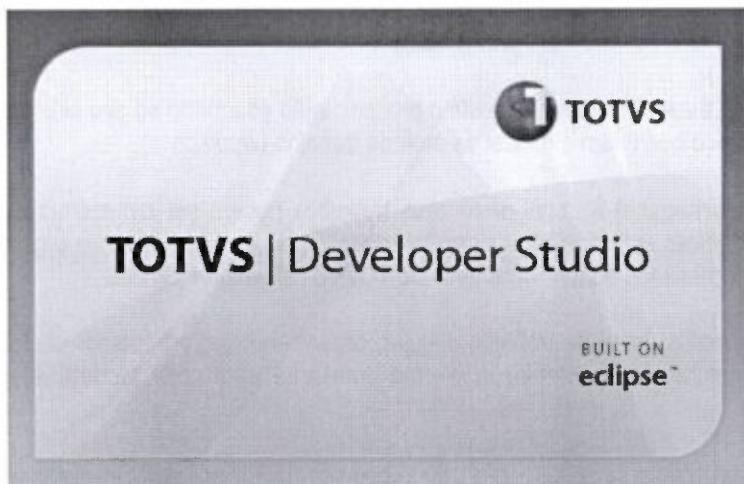
Na caixa de texto "workspace" (Tela-07- A), selecione um diretório para ser o ponto de partida (diretório raiz) da sua área. Logo abaixo, você tem uma caixa de marcação (Tela-07-B) na qual você pode optar para que esta seja a sua área de trabalho padrão. Se você a marcar, na próxima vez que entrar no TOTVS | Developer Studio, não lhe será solicitado a área. Caso o diretório selecionado não exista, este será criado.



NOTA: Utilize esta opção se você não pretende ter várias áreas ou se ela for a área em que você costuma trabalhar. Nos menus do TOTVS | Developer Studio, você terá acesso a um gerenciador de áreas de trabalho ou solicitar a troca da área em tempo de execução.

Recomenda-se que a área de trabalho seja colocada em diretório diferente do local de instalação do TOTVS | Developer Studio. Este procedimento facilitará manter cópias de segurança e futuras atualizações do produto.

Após a seleção da área de trabalho, aguarde alguns instantes enquanto o TOTVS | Developer Studio é inicializado e apresente a tela de boas-vindas.



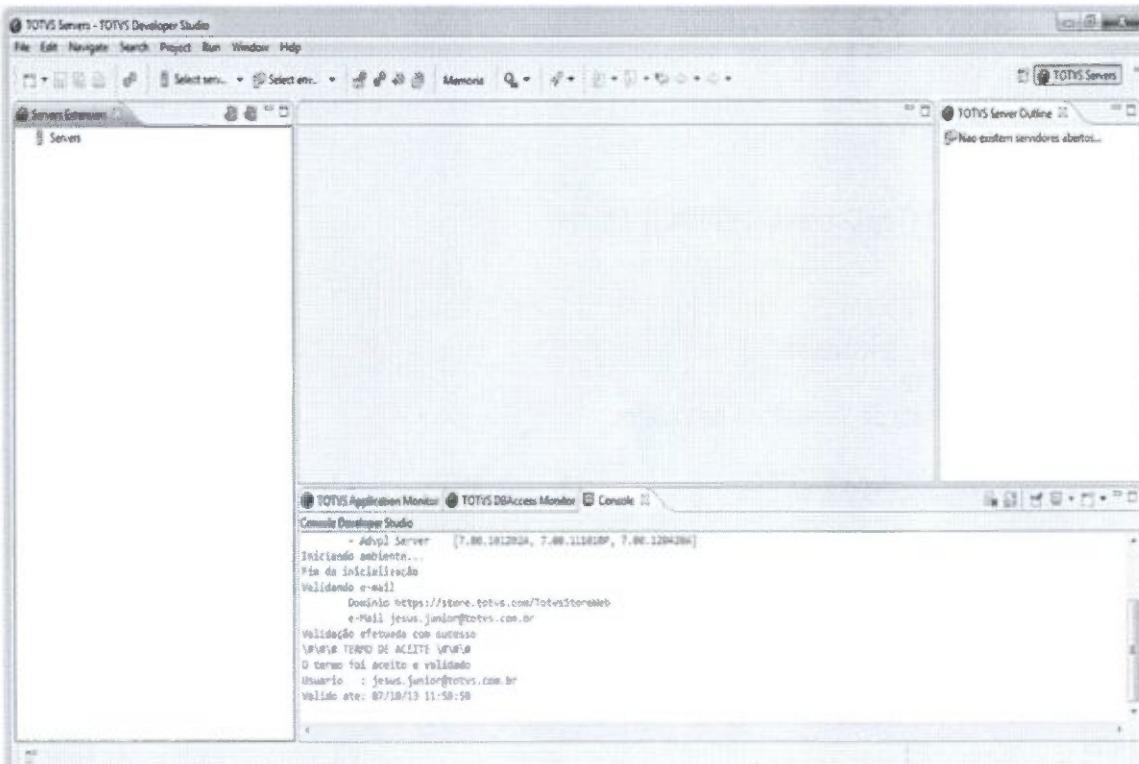
Você pode fechar a aba de boas-vindas acionando o X. Caso deseje acessar esta aba novamente, ação menu Ajuda | Boas-vindas.

Formação Programação ADVPL



Esta tela de boas-vindas, contém indicações de onde você pode obter documentação mais completa e com indicações das primeiras tarefas a serem executadas por você, para customizar o TDS a suas necessidades de desenvolvimento.

O TOTVS | Developer Studio é uma ferramenta de desenvolvimento integrada (*IDE*). Por isso, você possui uma vasta gama de funcionalidades, tais como edição de programas, execução, depuração (debug), análise de desempenho, configuração de servidores e muitas outras funcionalidades. Para ativá-las é necessário instalar adicionais ao seu TDS.



5.10.3. Perspectivas

As funcionalidades adicionadas ou estendidas por (*plugins*), são agrupadas em perspectivas (**perspectivas**). Ao instalar um dos adicionais do TOTVS | Developer Studio, ele virá com as perspectivas básicas para o desenvolvedor de produtos TOTVS.

- **Perspectivas (perspective)** – São "configurações" pré-definidas, que podem ser customizadas pelo desenvolvedor, que permite de uma forma rápida, modificar o seu ponto de vista (perspectiva) em relação ao trabalho que está sendo executado ou que seja necessário fazer na perspectiva, você tem uma ou mais visões, que se relacionam de alguma forma entre si e com o trabalho a ser executado.

Atualização e Backup do Protheus 12

As principais Nomenclaturas do Protheus 12, são referentes às rotinas de Atualização e desenvolvimento. São elas:

- **Build:** Versão completa do sistema com seus Executáveis, Dil's e RPO completo. O Build do sistema pode ser identificado através das seguintes opções "Ajuda" + "Sobre", dentro de qualquer Módulo do sistema, ou na Tela de Console do TOTVS Application Server (TotvsAppServer).
- **Repositórios:** Arquivos Binários Compilados, os quais contêm Instruções de Funcionamento, como Funções e Aplicações de todos os Módulos do ERP, utilizadas pelo Protheus 12 e seguem a seguinte nomenclatura:

5.10.4. Nomenclatura dos arquivos

Repositório de Objeto:

TTDP12.RPO

TT – Totvs;

D – Tipo de banco de Dados.

Exemplo:

D = Codebase, A=ADS, T=Top Connect, C=Ctree, B=Btrieve);

P – Idioma.

Exemplo:

P = Portuguese, E=English, S=Spanish

12 – Versão do Protheus 12.

RPO – Identifica que se trata do Repositório de Objetos:

Patch: Arquivos de Correções para o RPO.

Sua finalidade é a correção do Arquivo Binário que contém as funções utilizadas pelos Módulos do sistema, a fim de se evitar que seja necessário Atualizar o RPO por completo, sempre que uma Função for corrigida.

Sua Nomenclatura é a seguinte:

TTDP121.PAT

TT – Totvs ;

D – Tipo de banco de Dados.

Exemplo:

D = Codebase, A=ADS, T=Top Connect, C=Ctree, B=Btrieve);

P – Idioma

Exemplo:

P = Portuguese, E=English, S=Spanish

121 – Versão do Protheus 12.1.

PAT – Identifica que se trata do Arquivo de Correção do RPO.

5.10.5. Atualização de BUILD

A Atualização da Build consiste em atualizar todas as alterações realizadas, dentro da estrutura do Protheus 12, como por exemplo, as DLL's, os Executáveis, as Correções dos Utilitários, os Aplicativos e etc.

É recomendado que se atualize a Build a cada 3 (Três) meses, pois normalmente, esse é o período em que o possui novos arquivos atualizado, com exceção da Path, que não tem data exata para a Atualização.

Sequência de atualização do sistema:

1. Binário
2. RPO
3. Update
4. LIB
5. Demais Patchs dos módulos
6. Compatibilizadores dos Módulos
7. Compilar as Customizações (caso existam)

Importante:

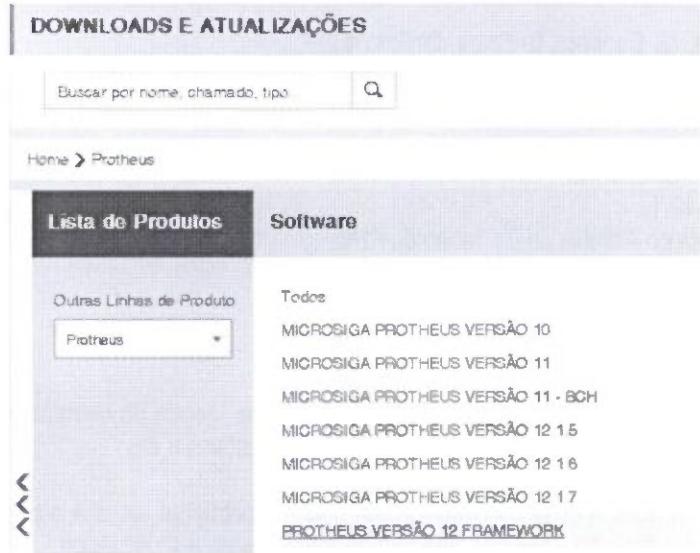
A atualização do RPO na versão 12 está atrelada à respectiva Release, ou seja, há um RPO específico para cada Release. Verifique a versão da sua Release antes de baixar um RPO (menu Ajuda/Sobre de qualquer módulo).

EFETUAR BACKUP DO DIRETÓRIO PRINCIPAL DO SISTEMA (C:\TOTVS12 por exemplo) ANTES DE INICIAR O PROCEDIMENTO.

Para as atualizações de build versão, acesse o site da TOTVS suporte.totvs.com.br, no link "Portal do Cliente". Para acesso aos clientes ativos, necessitando de usuário e senha.

Selecionar no menu a opção "Downloads".

Selecionar a pasta "Protheus", selecionar a versão que deseja fazer Downloads.



The screenshot shows a web interface for software downloads. At the top, there's a search bar with placeholder text "Buscar por nome, chamado, tipo" and a magnifying glass icon. Below the search bar, the URL "Home > Protheus" is visible. The main area has two tabs: "Lista de Produtos" (selected) and "Software". Under "Software", there's a dropdown menu "Outras Linhas de Produto" set to "Protheus". To the right, a list of software versions is shown under the heading "Todos": MICROSIGA PROTHEUS VERSÃO 10, MICROSIGA PROTHEUS VERSÃO 11, MICROSIGA PROTHEUS VERSÃO 11 - BCH, MICROSIGA PROTHEUS VERSÃO 12 1 5, MICROSIGA PROTHEUS VERSÃO 12 1 6, MICROSIGA PROTHEUS VERSÃO 12 1 7, and PROTHEUS VERSÃO 12 FRAMEWORK.

Faça o Download dos arquivos:

DOWNLOAD Binário (Appserver e SmartClient) PROTHEUS VERSÃO 12 FRAMEWORK:

1. Fazer um backup da pasta BIN
2. Descompactar o arquivo baixado
3. Copiar o conteúdo da pasta appserver do arquivo baixado para dentro da pasta Appserver do Protheus, substituindo os arquivos quando solicitar
4. Copiar o conteúdo da pasta SmartClient do arquivo baixado para dentro da pasta SmartClient do Protheus e de cada estação client, substituindo os arquivos quando solicitar
5. Copiar o conteúdo da pasta smartclientactivex do arquivo baixado para dentro da pasta SmartClientActivex do Protheus, substituindo os arquivos quando solicitar.

DOWNLOAD Repositório de Objetos - RPO:

1. MICROSIGA PROTHEUS VERSÃO 12.1.X (onde X é a versão da sua release)
2. ROTHEUS 12 TOPCONNECT PORTUGUES
3. Categoria: Repositório de objetos
4. Baixar o repositório para o Brasil (BRA-EUA-PAR-URU-TTP120)
5. Pare o serviço do Protheus
6. Na pasta APO, renomear o repositório anterior, ou mover para outra pasta
7. Copiar o novo arquivo para o diretório APO do sistema
8. Renomear o arquivo mantendo apenas os últimos 7 caracteres (Exemplo: ttp120.rpo)

DOWNLOAD Patch de Programa - LIB:

1. Linha Protheus
2. PROTHEUS VERSÃO 12 FRAMEWORK
3. PROTHEUS 12 TOPCONNECT PORTUGUES
4. Categoria: Patch de programa
5. No campo "Buscar", digite: LIB_P12

6. Baixar a LIB_P12
7. ATUALIZAR:
8. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data e efetuar a aplicação do mesmo via Dev Studio ou TOTVS Developer Studio.

Procedimento de Atualização da LIB:

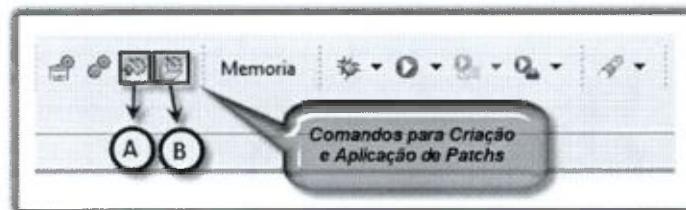
1. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data e efetuar a aplicação do mesmo via Dev Studio ou TOTVS Developer Studio.

5.10.6. Aplicações de Patchs

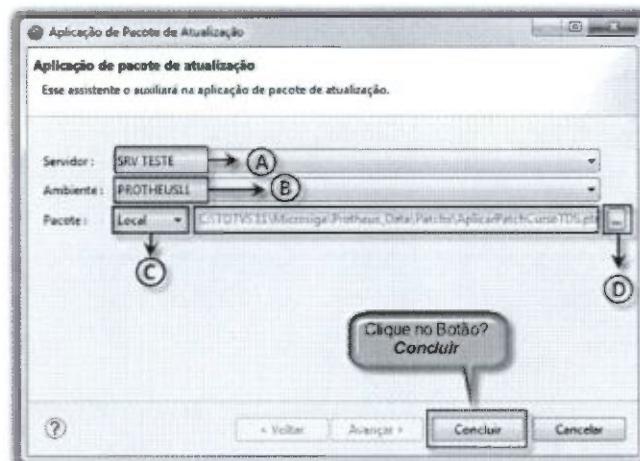
A opção Aplicações de Patchs é utilizada quando houver a necessidade da Atualização dos Programas existentes no RPO em uso pelo sistema.

Esse Processo poderá ser utilizado, para Atualizarmos os Programas gerados pela MICROSIGA ou simplesmente Programas Customizados pelo Cliente.

1. Acessando o botão  Aplicar Patch. Conforme figura (Tela 01-B)



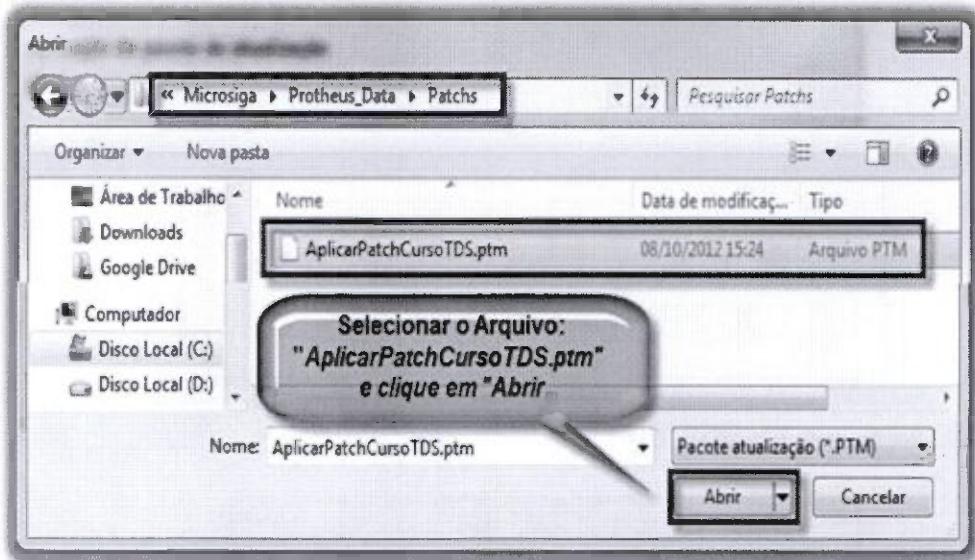
Comando para manipulação de Patchs (Tela -01)



Comando para manipulação de Patchs (Tela -02)

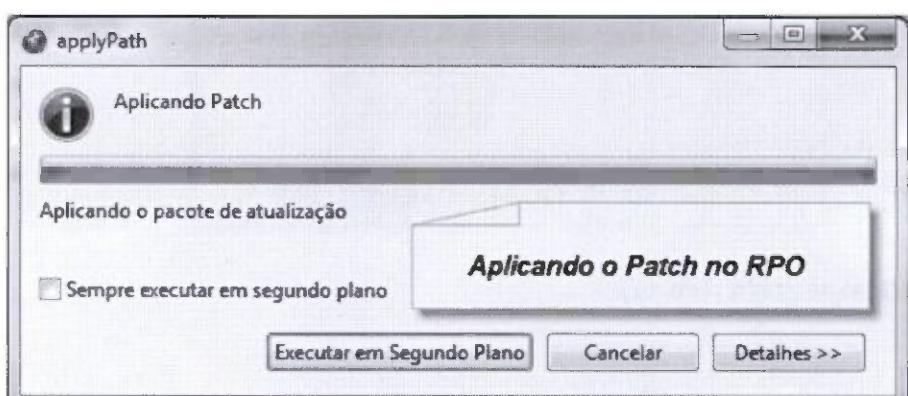
Execute os procedimentos abaixo:

- Selecione o Servidor "SRV TESTE" (Tela-02-A)
- Selecione o Ambiente "PROTHEUS12" (Tela-02-B)
- Selecione a opção "Local" (Tela-02-C)
- Clique no botão  (Tela-02-D) será exibido uma tela para selecionar o diretório, conforme figura(Tela-03-A)
- Selecione o diretório "Patchs" em C:\TOTVS 12\Microsiga\Protheus_Data\Patchs
- Selecione o Arquivo "AplicarPatchCursoTDS.ptm" (Tela-03)
- Clique no botão Abrir

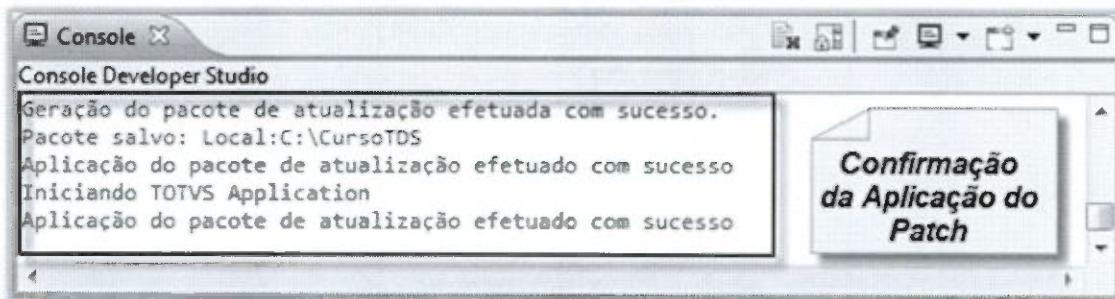


Janela do Windows Explorer(Tela-03)

2. Aguardar a Aplicação do Pacote de Atualização



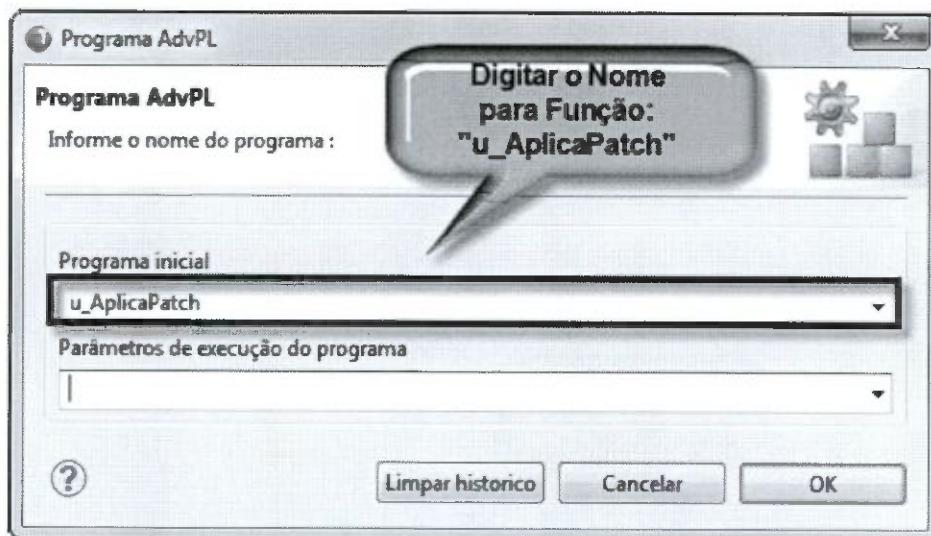
3. Mensagem da **Aba Console** com a confirmação da Aplicação do Patch.



4. Testando o Patch aplicado no Repositório de Objetos no Totvs Developer Studio.



5. Aguarde alguns instantes e lhe será solicitado o programa inicial



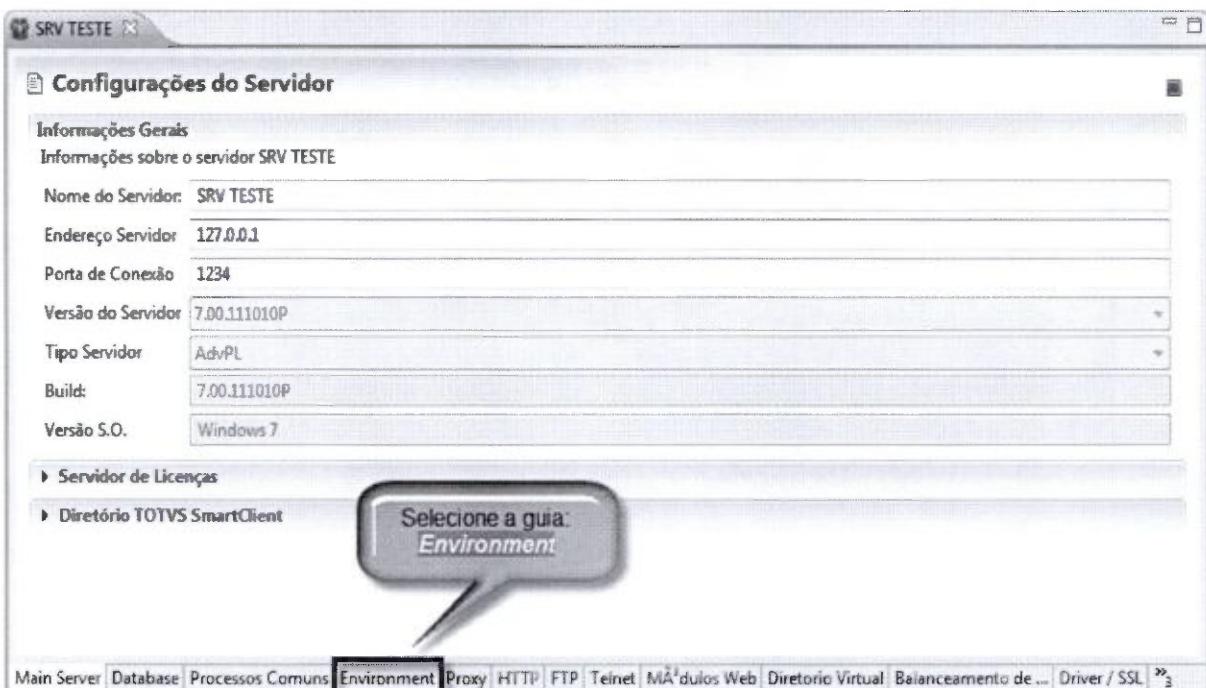
5.10.7. Log's dos Repositórios

Utilizado para visualizar todo o Histórico sobre o RPO em uso, poderemos através desta opção visualizar as Datas das Últimas Patchs aplicadas, o Conteúdo das Patchs e a Data da Build utilizada.



Servers Extension(Tela-01)

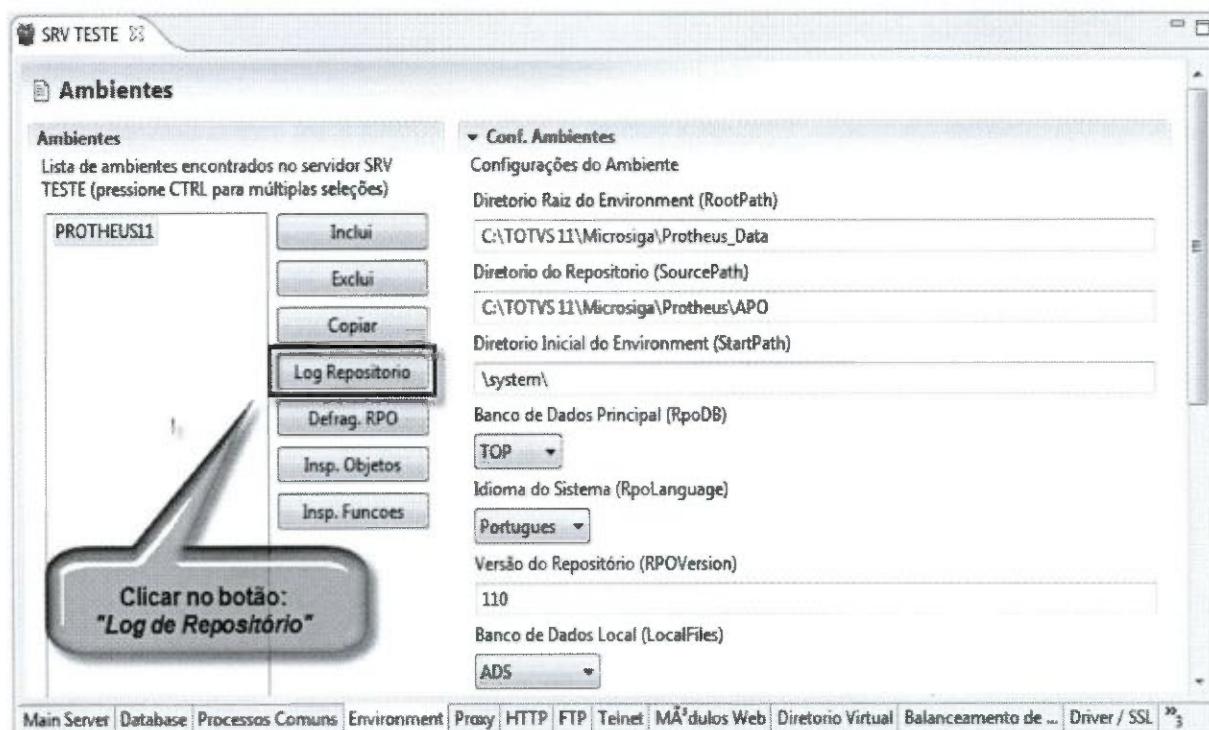
1. Clique Duplo com o Botão esquerdo, conforme figura (Tela-01), será apresentada a tela de configurações do Servidor (Main Server).



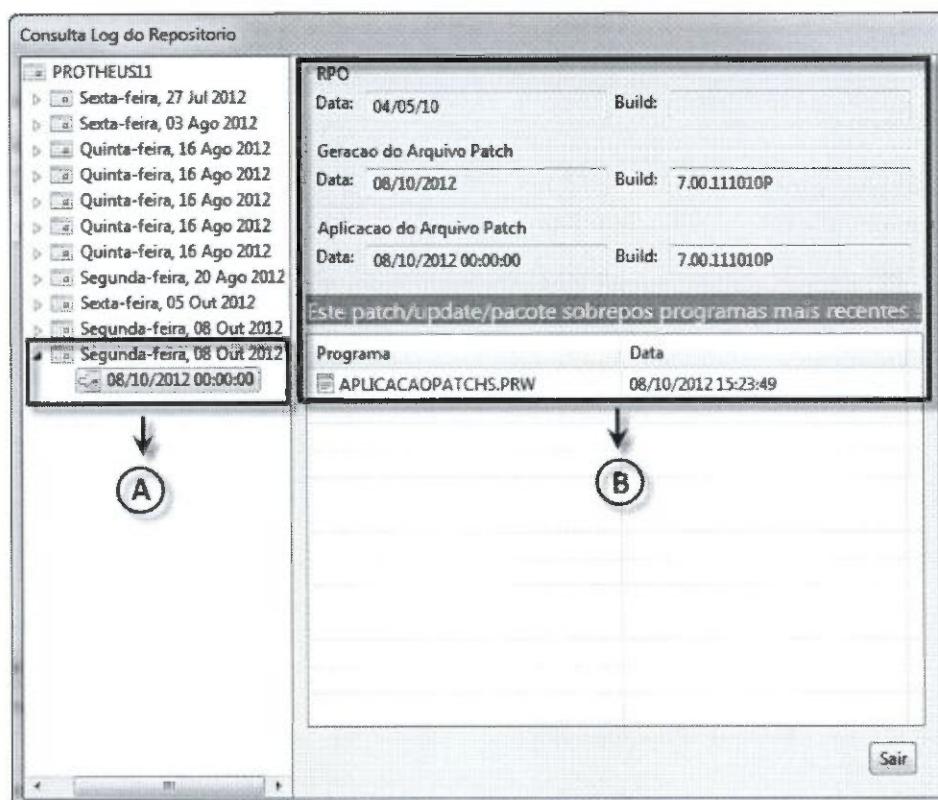
Configurações do Servidor "Main Server"(Tela -02)

2. Na Aba configurações do Servidor, clicar no botão “Log Repositório”, conforme figura abaixo.

Formação Programação ADVPL



3. Verifique que ao lado direito da tela será informada a "Data de Geração e Aplicação da Patch" (Tela-03-A), juntamente com o "Conteúdo" dela e a "Data da Build", conforme figura abaixo.



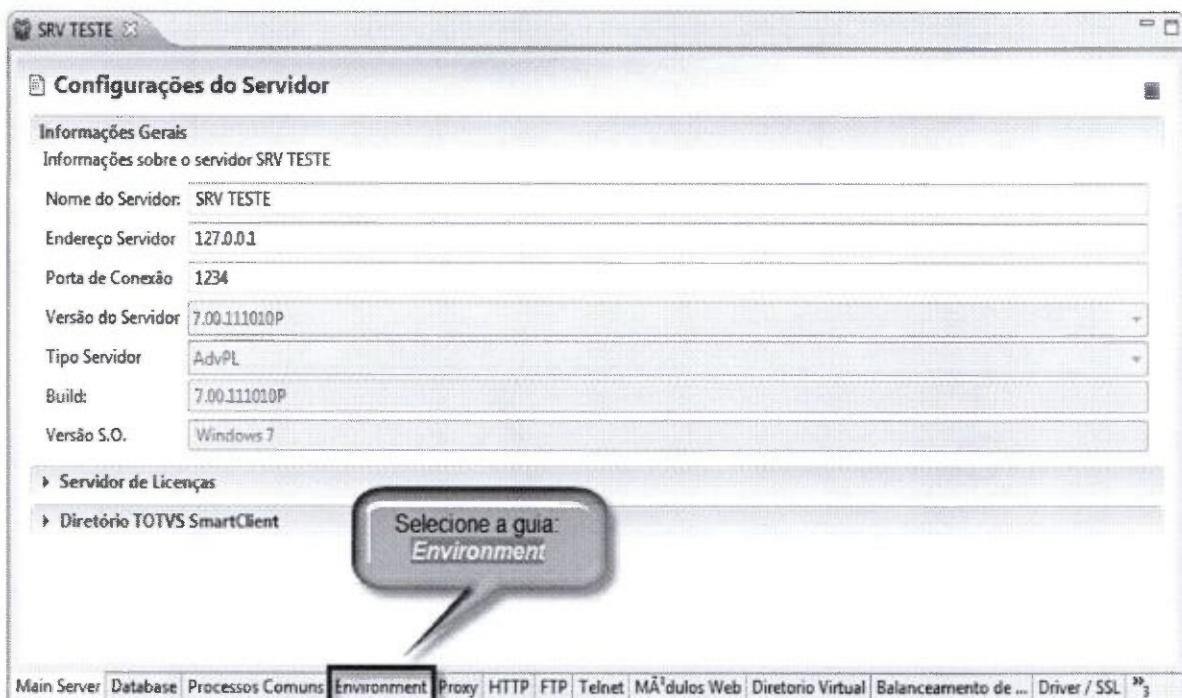
5.10.8. Desfragmentar Repositório

Para realizar a desfragmentação do repositório, é necessário que o TOTVS | Development Studio esteja configurado para compilar Por Velocidade de Compilação. Pois, desta maneira, o repositório (RPO) ficará fragmentado, devido ao fato das funções serem alocadas no repositório por Ordem de Compilação.



Servers Extension(Tela-01)

1. Clique Duplo com o Botão esquerdo, conforme figura (Tela-01), será apresentada a tela de configurações do Servidor (Main Server).

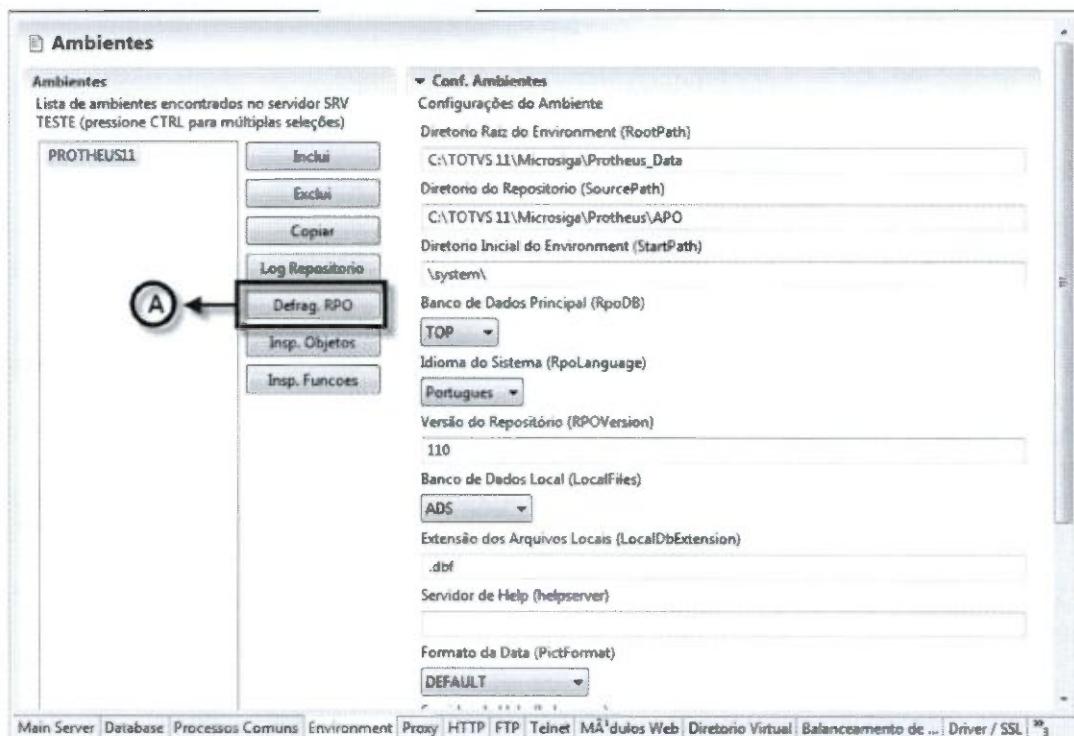


Formação Programação ADVPL



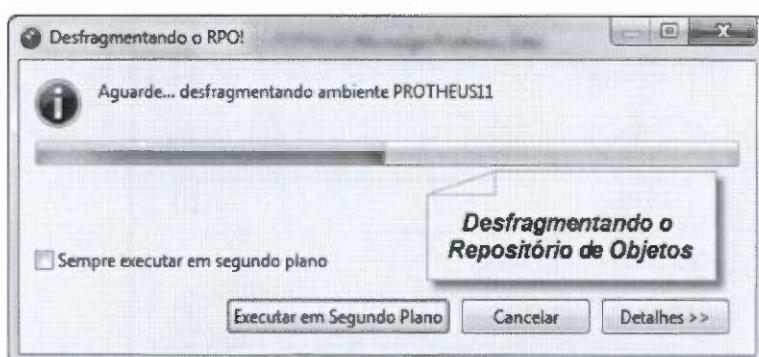
Configurações do Servidor "Main Server"(Tela -02)

2. Clique no botão "Defrag. RPO"(Tela-03-A)

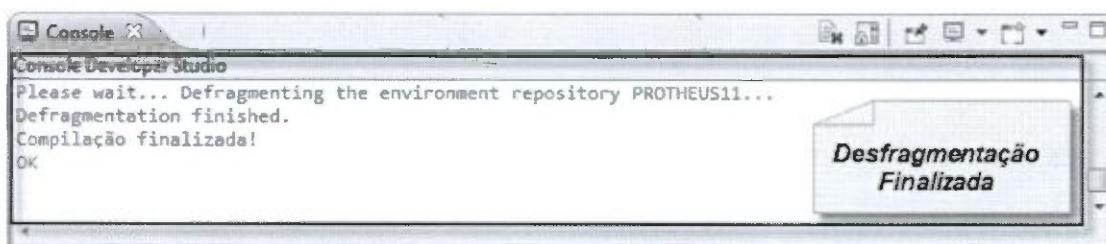


Configurações do Servidor "Environment"(Tela -03)

3. Será apresentado uma Mensagem de sobre o inicio do processo.



4. Na Aba Console é apresentado a mensagem de conclusão.



5.10.9. Compatibilizador - UPDDISTR

Um dicionário de dados diferencial é um dicionário parcial que é utilizado para atualizar o dicionário de dados (metadado) do Protheus em uso.

O dicionário de dados diferencial pode ser utilizado para diversos fins, a saber:

- Atualizar o dicionário de dados do Protheus para uma versão atualizada dentro de um mesmo release.
- Inserir o dicionário de um novo produto, módulo ou funcionalidade.
- Efetuar atualizações pontuais devido a um novo requisito legal.

Para aplicar o dicionário de dados diferencial, utilizamos uma ferramenta especial chamada UPDDISTR. O UPDDISTR atualiza o dicionário de dados do Protheus usando as mesmas regras e funcionalidades do atualizador de versão do Protheus.

De fato, o "core" (núcleo) do UPDDISTR e o do atualizador de versão são exatamente iguais, garantindo que não haja disparidade de regras entre as duas ferramentas. A maior diferença reside no fato do UPDDISTR poder utilizar um dicionário parcial, enquanto o atualizador de versão sempre exigir um dicionário completo.

Além do dicionário de dados, as alterações na base de dados decorrentes do novo dicionário (como por exemplo tamanho de campos) também são efetuadas, da mesma maneira que no atualizador de versão padrão.

O sistema é atualizado logo após a aplicação do pacote de atualizações (Patch) de Lib, contendo os fontes abaixo cuja data seja igual ou superior às datas de atualizado.

Para atualizar, efetuar a chamada do programa de atualização UPDDISTR, na tela inicial do Protheus, ou seja, digitar UPDDISTR como se fosse o nome de um módulo do Protheus.

Em "Programa Inicial", digite " UPDDISTR".



Clique no botão "OK" para confirmar.

Será apresentada uma janela com orientações sobre o processo de atualização.

Formação Programação ADVPL



Os arquivos devem estar atualizados na pasta "SYSTEMLOAD" atualizados:

DOWNLOAD HELPS DE CAMPOS/PERGUNTAS COMPLETO – Brasil

1. Linha Protheus
2. PROTHEUS VERSÃO 12. Versão
3. Categoria: HELPS DE CAMPOS/PERGUNTAS
4. Baixar a HELPS DE CAMPOS/PERGUNTAS COMPLETO – PAIS
5. ATUALIZAR:
6. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data/SystemLoad
 - hlpeng.txt
 - hlpvbra.txt
 - hppor.txt
 - hpspa.txt

DOWNLOAD HELPS DE CAMPOS/PERGUNTAS Diferencial – Brasil

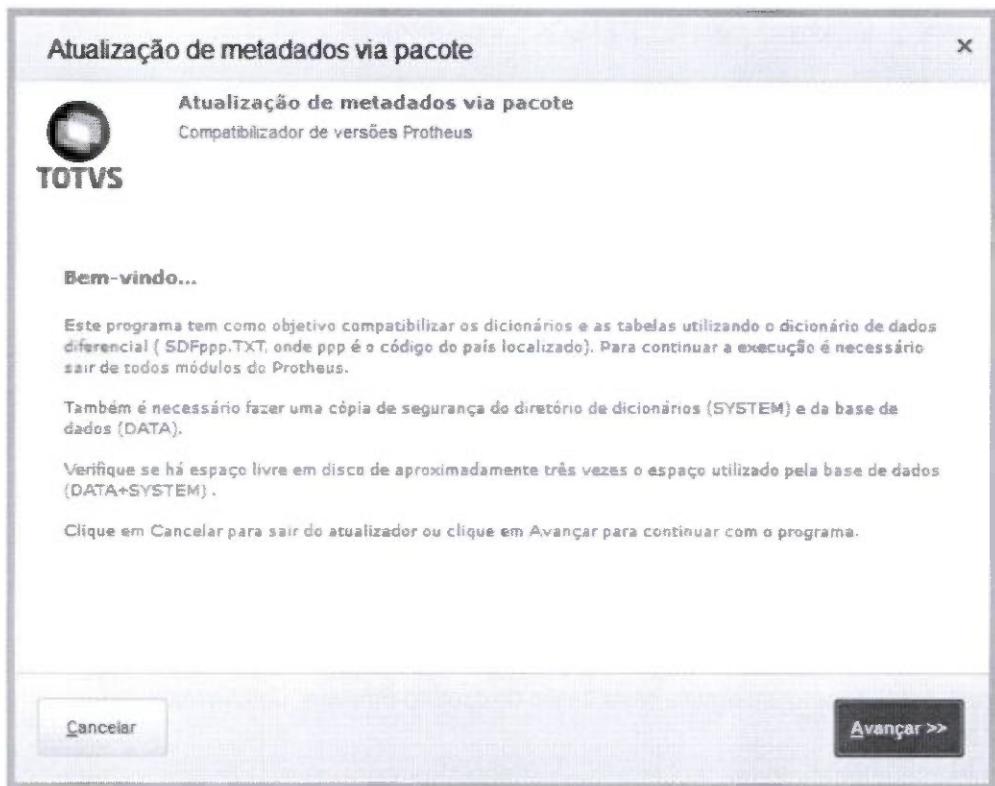
1. Linha Protheus
2. PROTHEUS VERSÃO 12. Versão
3. Categoria: HELPS DE CAMPOS/PERGUNTAS
4. Baixar o HELPS DE CAMPOS/PERGUNTAS DIFERENCIAL – PAIS
5. ATUALIZAR:
6. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data/SystemLoad
 - hpdfpor.txt
 - hpdfspa.txt
 - hpdfeng.txt

DOWNLOAD Dicionário de dados Completo – Brasil

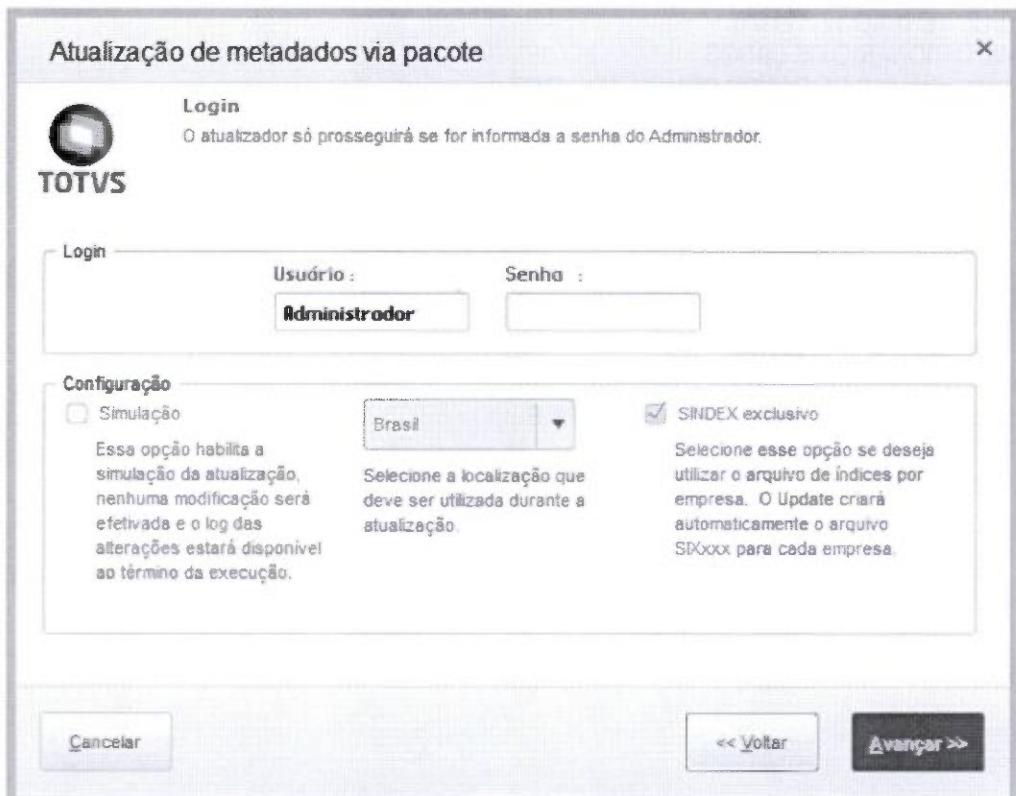
1. Linha Protheus
2. PROTHEUS VERSÃO 12. Versão
3. Categoria: DICIONÁRIO DE DADOS.
4. Baixar o DICIONÁRIO DE DADOS COMPLETO – PAIS
5. ATUALIZAR:
6. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data/SystemLoad
sxsbra.txt

DOWNLOAD Dicionário de dados Diferencial – Brasil

1. Linha Protheus
2. PROTHEUS VERSÃO 12. Versão
3. Categoria: DICIONÁRIO DE DADOS.
4. Baixar o DICIONÁRIO DE DADOS Diferencial – PAIS
5. ATUALIZAR:
6. Descompactar esse arquivo em alguma pasta dentro do diretório Protheus_Data/SystemLoad
sdfbra.txt



A tela seguinte adverte que apenas o Administrador pode processar a atualização de versão.



Formação Programação ADVPL



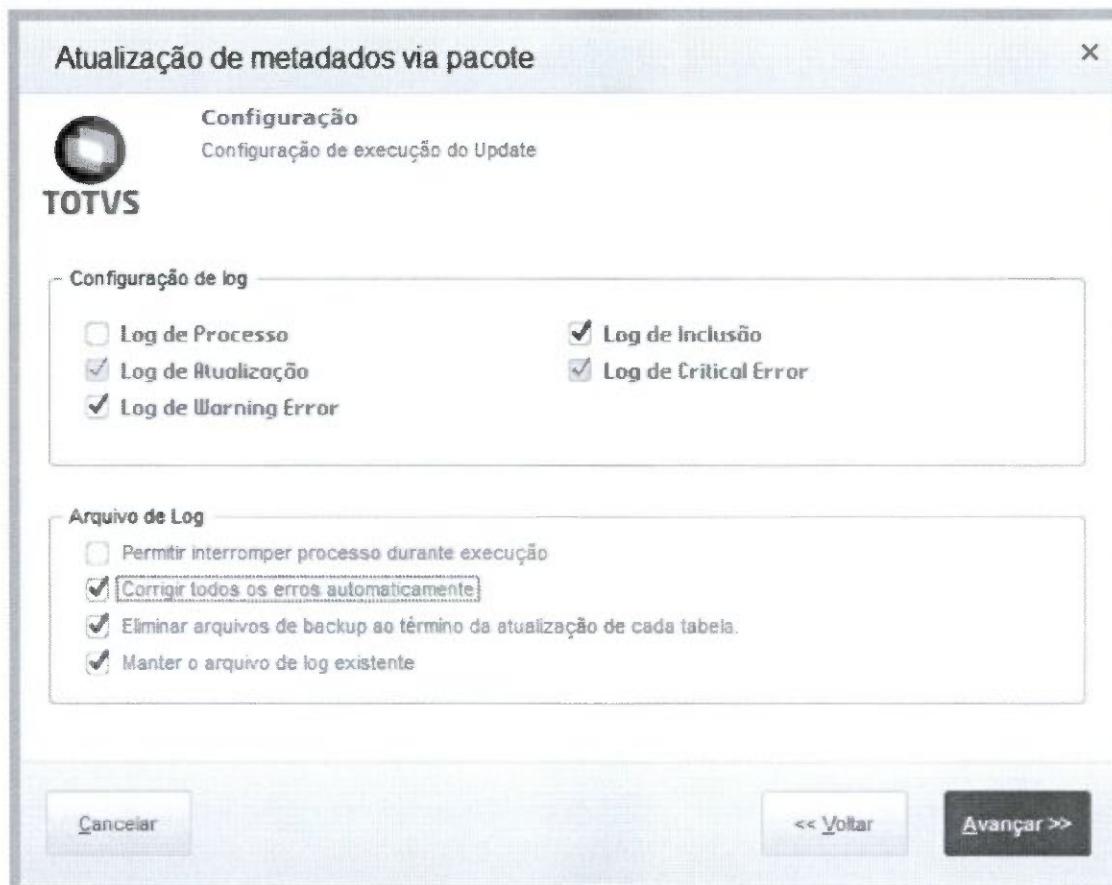
Para poder processar o UPDDISTR só usuário Administrador possui privilegio.

Se desejar apenas simular a atualização para verificar as alterações a serem processadas e eventuais problemas na base de dados, clique na caixa de verificação "Simulação".

Com esta opção ativada, não será gravada nenhuma alteração na base. O sistema gera um arquivo de log que pode ser impresso.

Se a opção "SINDEX exclusivo" for selecionada, o "UPDDISTR" criará automaticamente um dicionário de índice para cada empresa cadastrada. Se esta opção não for selecionada, será mantido o arquivo atual compartilhado entre as empresas.

Na tela Configuração de Log possui as opções:



- **Log de Processos:** O Log de Processos grava todas as operações que o Atualizador realiza, como abertura de arquivos, criação de índices e arquivos temporários.
- **Log de Atualização:** O Log de Atualização grava todas as alterações que o Atualizador fez na base, campo a campo, armazenando o valor anterior e o atual.
- **Log de Inclusão:** O Log de Inclusão grava todos os registros incluídos nos arquivos pelo Atualizador. Esta caixa é selecionada automaticamente.

- **Log Critical Error:** O Log Critical Error grava erros críticos que inviabilizam a continuidade do processo de atualização.

Durante a verificação da integridade, caso seja encontrado um erro crítico, o processo de atualização somente poderá prosseguir após a correção do mesmo.

- **Log Warning Error:** Este log grava erros que não impedem o processo de atualização e correções efetuadas através do Assistente do Atualizador. Somente serão indicados neste arquivo.

Os logs são gravados na pasta \PROTHEUS_DATA\SYSTEM\MPUPDLOG.LOG

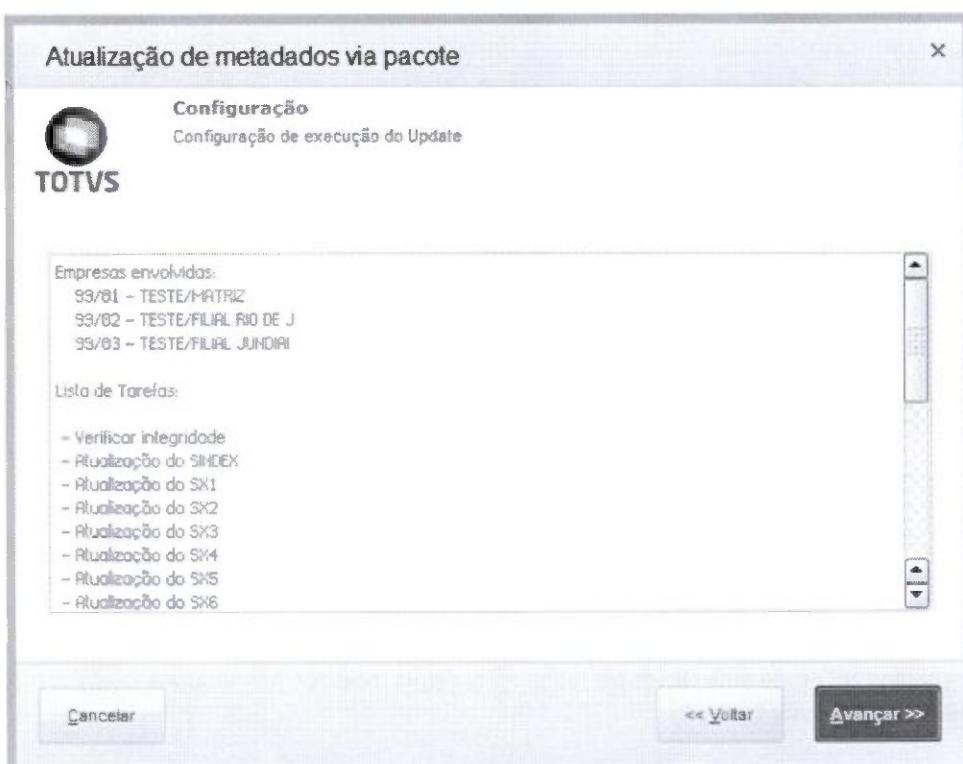
Se o usuário desejar saber de todas as alterações processadas, pode marcar todas as opções de Logs e também a caixa "Permite interromper processo durante a execução".

Neste caso, deve-se interromper ao final de cada processo, analisar os logs e executar a Atualização, novamente.

Deve-se optar por limpar o Arquivo de Log para evitar que este fique muito grande.

- **Corrigir todos os erros automaticamente:** Clique na caixa para que durante o processo de análise de integridade, o Atualizador corrija automaticamente alguns erros críticos.
- **Eliminar arquivos de backup:** Ao término da atualização para excluir automaticamente os arquivos de backup que o Atualizador cria durante o processo.
- **Mantener o arquivo de log existente** para manter o log ao reiniciar o processo de atualização de versão após uma pausa.

Será apresentada a janela relacionando as tarefas que serão executadas pelo Atualizador.



Formação Programação ADVPL

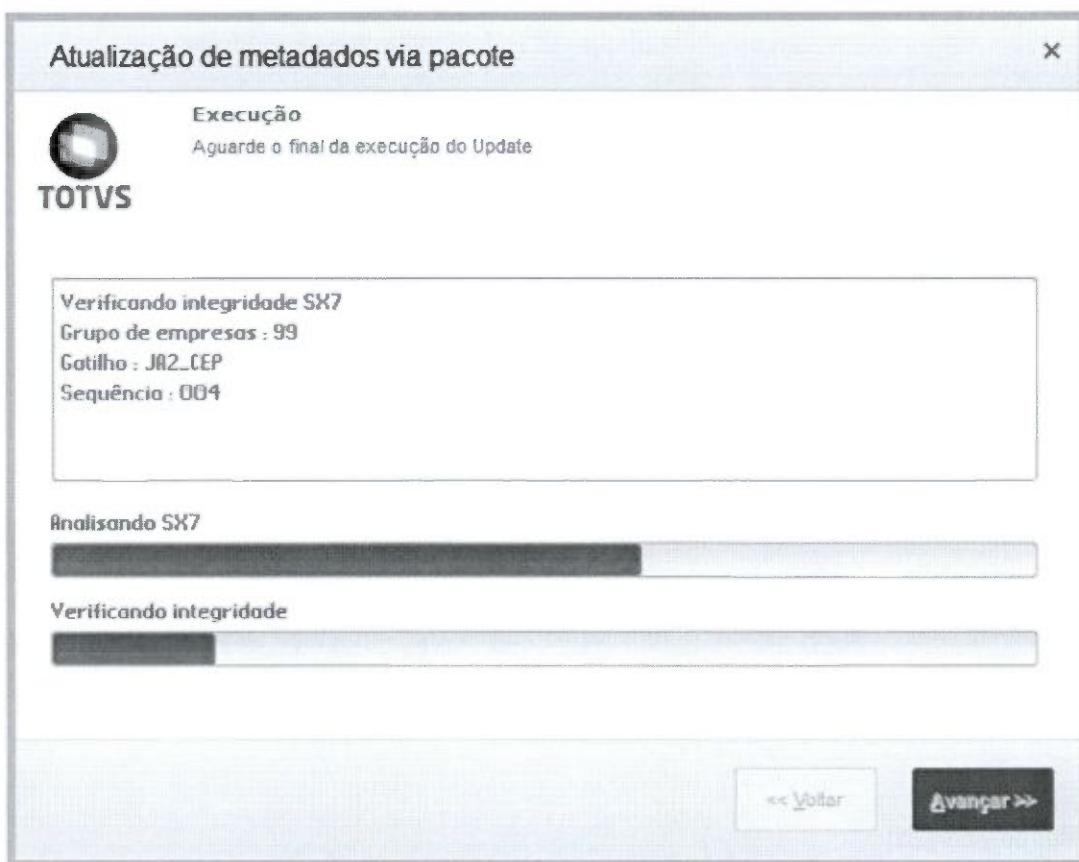


- Verificação da integridade das bases;
- Atualização dos arquivos customizadores SXs;
- Atualização das tabelas;
- Atualização dos arquivos de helps;
- Execução das funções compatibilizadores.

Importante

Ao iniciar o processo de conversão da base de dados, o Sistema gera o arquivo MPUPD.TSK na pasta \bin\appserver detalhando todo o processo de atualização realizado.

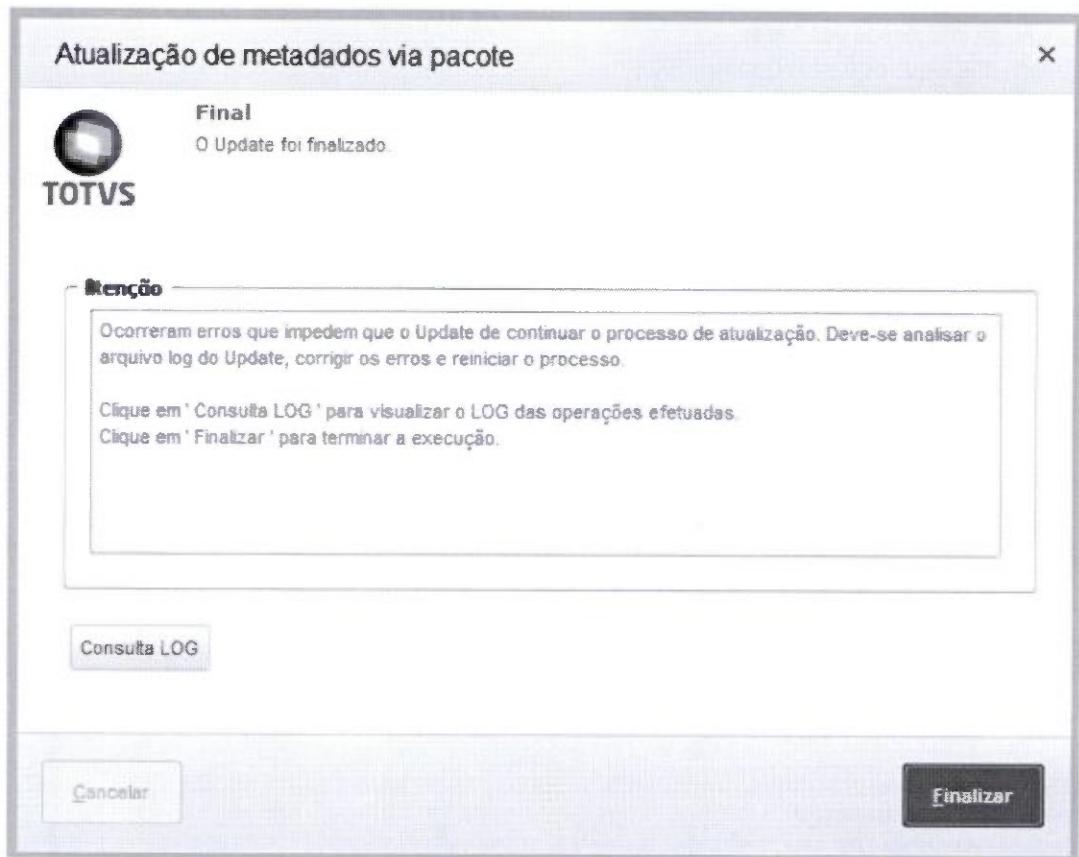
O Atualizador identifica cada processo iniciado e a sua evolução.



Se diagnosticado algum problema na base do cliente, este será apresentado com opções para a continuidade ou interrupção do processo.

Durante a verificação de integridade, se for diagnosticado algum "critical error" não previsto, o processo será interrompido e o administrador terá a possibilidade de analisar o arquivo de log, corrigir o erro e reiniciar a atualização.

Clique no botão "Consulta Log" para consultar o arquivo de log.



E verificar se ocorreu algum erro.

5.11. TOTVS Wizard - Assistente de Configuração

O TOTVS Wizard é um assistente para configuração do TOTVS Application Server e para instalação e configuração dos módulos Web, como: GE - Gestão Educacional, PP - Portal Protheus, DW - Datawarehouse, BSC - Balanced Scored Card, WPS - WebPrint/WebSpool, RH On-line - Terminal do Funcionário, WS - Web Services, GPR - Gestão de Pesquisas e Resultados e GAC - Gestão de Acervos.

Através do TOTVS Wizard, é possível configurar os seguintes tópicos:

- Ambientes;
- Servidor de Licenças;
- Módulos Web;
- Servidor Internet (HTTP/FTP);
- Hosts / URLs HTTP;
- Processos Comuns;
- Processos WEB / WEBEX;
- Conexão (Server/Remote);
- TOTVS DBAccess;
- Servidor CTREE;
- Balanceamento de Carga;
- Serviço do Windows NT/2000 (*);

- Geral.
- Protheus Search

O TOTVS Wizard é executado automaticamente ao final da instalação do Protheus 12, por meio do assistente de instalação do CD do Protheus. Porém, após a instalação do Protheus, também é possível executá-lo pela opção "Assistente de Configuração do Servidor", na pasta "Protheus 12\Ferramentas" no menu Iniciar do Windows, ou pelo aplicativo "TWizard.exe", disponível no diretório "\BIN\SmartClient".

É importante ressaltar que a maioria das configurações editadas por este Assistente apenas terão validade quando o servidor do Protheus for finalizado e reiniciado. Porém, determinadas configurações podem ser reconhecidas sem que o TOTVS Application Server seja reiniciado, e já passarem a fazer efeito, inclusive interferindo nos processos (Threads) em execução no servidor, o que pode acarretar em problemas no processamento.

Portanto, é recomendável que a utilização deste assistente seja realizada sem que existam processos em execução no servidor e sem que existam usuários ou estações remotas conectadas ao TOTVS Application Server.

Ao ser executado, o TOTVS Wizard identifica, no arquivo de configurações do TOTVS Application Server (totvsappserver.ini), as configurações atualmente definidas e apresenta a janela principal de trabalho do assistente.

Como acessar o Assistente de Configuração do Protheus 12:

- Selecione as seguintes opções "Iniciar" + "Programas" + "Protheus 12" + "Ferramentas" + "Assistente de Configuração do Servidor";
- O "Assistente" será carregado com sucesso.



5.11.1. Configurações de Ambientes

As seções de Ambiente, ou 'Environment', são criadas para identificar o comportamento e execução do Application Server para as conexões clientes. Quando o Protheus Remote se conecta ao Application Server, deve informar, entre outros parâmetros, o nome do Ambiente utilizado para que o Application Server prepare a execução daquele terminal.

É no Ambiente que informações como o idioma, a versão e o banco de dados utilizado são configuradas. Quando o Protheus Remote se conecta e informa o Ambiente que será utilizado, o TOTVS Application Server executa os procedimentos necessários: abre o repositório de objetos compilado para o idioma, banco de dados e versão indicados, e realiza a conexão com o banco de dados selecionado.

É possível, através da configuração de mais de um ambiente, utilizar um Server Protheus para executar simultaneamente mais de uma aplicação ADVPL, com ambientes completamente independentes, utilizando apenas uma instância do TOTVS Application Server.

É muito comum a criação de mais de um ambiente para o Protheus. Geralmente são criados os ambientes Teste, Produção e Desenvolvimento.

Clique na opção "Ambientes" e posicione no ambiente desejado;

Na barra de ferramentas clique no ícone "Editar Ambiente"; e verifique os "Parâmetros" do "Ambiente Padrão da Instalação", analisando os dados a seguir:

- Configuração de Ambiente para Testes

Duplique as "Pastas", informando os dados a seguir:

Nova Pasta	Descrição
C:\Protheus12\apo2	Repositório Independente
C:\Protheus12\dataTST	Base de dados Independente
C:\Protheus12\systemTST	Configurações Independentes

Retorne ao "Assistente de Configuração", para continuarmos a "Criação do Novo Ambiente";

Selecione novamente a opção "Ambiente" e na barra de ferramentas acima, clique no ícone "Novo Ambiente";

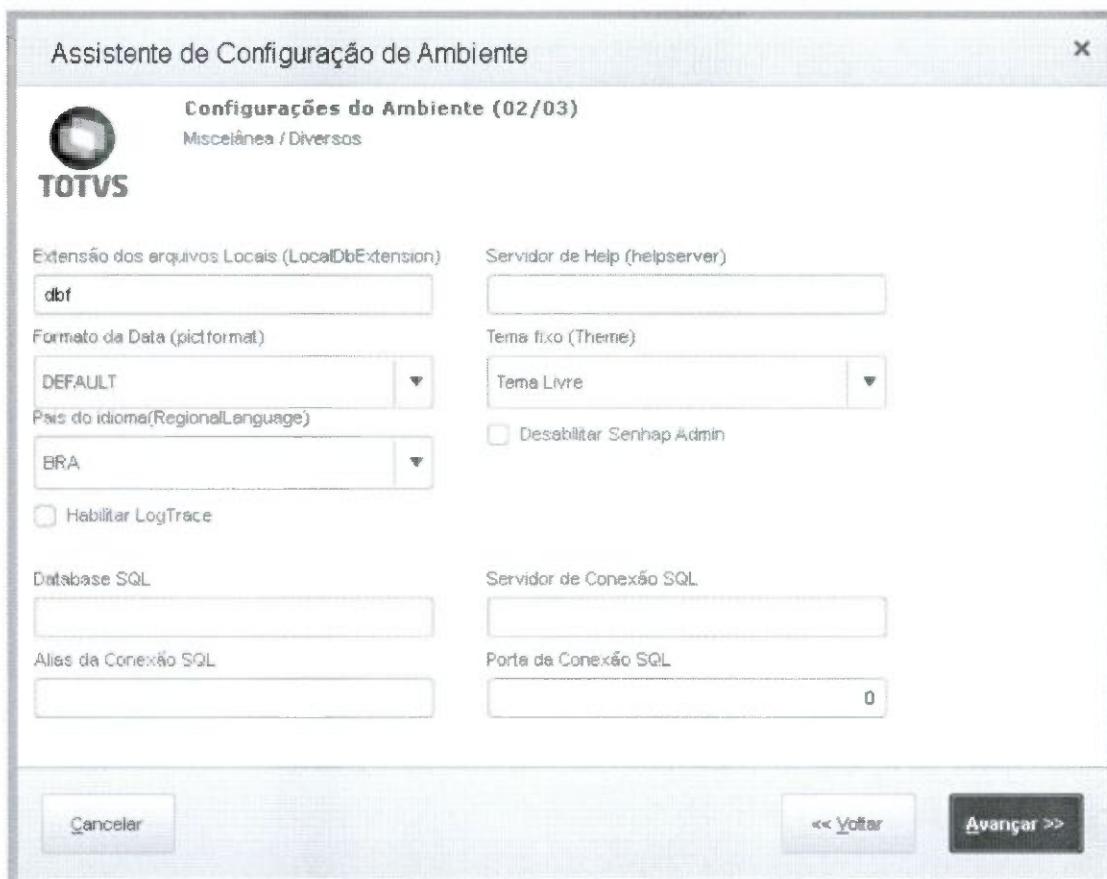
Na tela que se apresenta, configure o "Novo Ambiente", informando os dados a seguir:



- **Nome do Ambiente:** COMPILA
- **Diretório Raiz do environment:** Endereço da pasta Protheus_Data

- Banco de Dados Principal: SQL
- Versão do Repositório: SQL
- Diretório do Repositório: Endereço do RPO "APO"
- Diretório Inicial do Environment: Nome da pasta \system\
- Idioma do Sistema: Português
- Banco de Dados Local: ADS

Após preencher os dados selecionar o botão "Avançar".



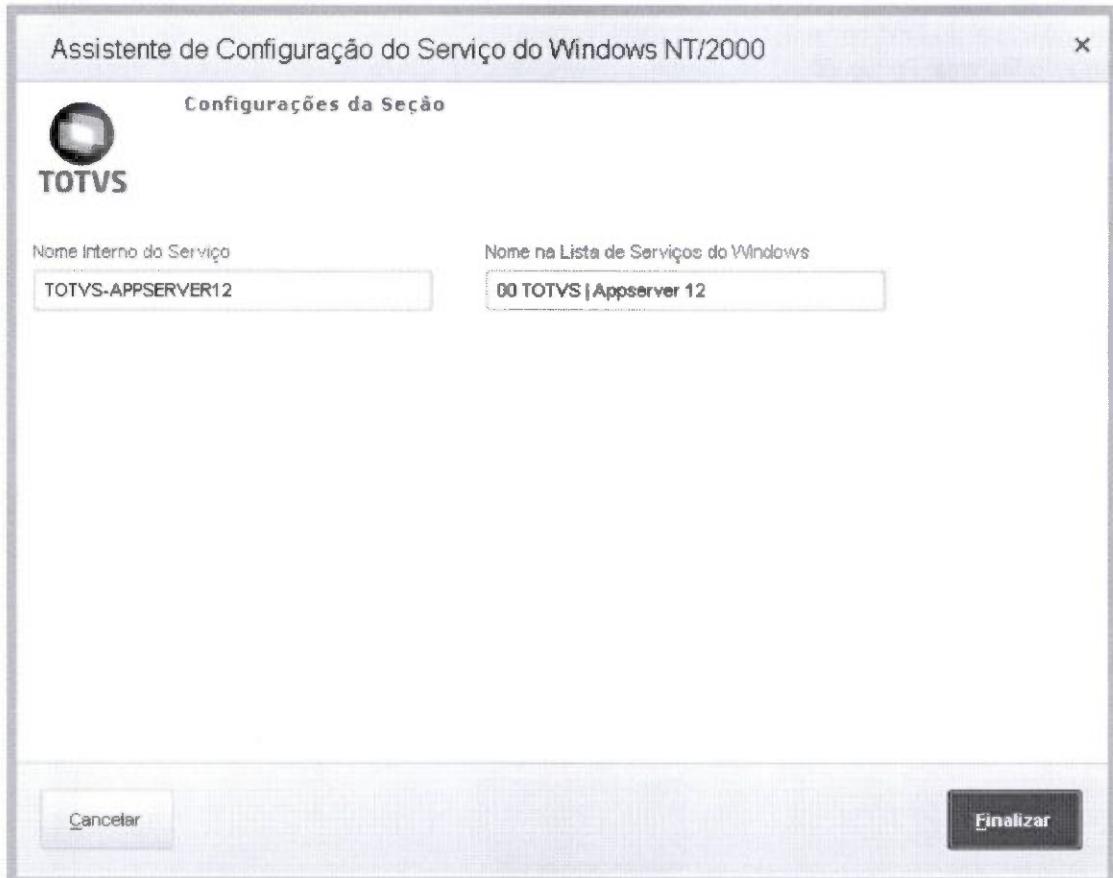
- Extensão dos arquivos Locais: Extensão dos arquivos customizadores
- Formato da Data: DEFAULT
- País do Idioma: BRA
- Habilitar LogTrace: Sim/Não

Clique no botão "Avançar";
Confira os dados e confirme clicando no botão "Finalizar";

5.11.2. Configurações de Serviços

Serviço é o processo utilizado, para que o Protheus, possa ser iniciado Automaticamente, quando o Servidor for ligado. Podemos Configurar os Serviços, manualmente utilizando o Assistente de Configuração. Selecione as seguintes opções "Iniciar" + "Programas" + "Protheus 12" + "Ferramentas" + "Assistente de Configuração do Servidor";

Abra a opção "Serviço do WindowsNT/2000" e posicione sobre "Service" e clique no ícone "Editar Configuração", confirme a tela a seguir;



- **Nome Interno Serviço:** Nome interno do serviço, não pode ter espaço
- **Nome na Lista de Serviços do Windows:** Nome do display do serviço que irá ficar na lista de serviços

Selecionar o botão "Finalizar"

5.11.3. Configuração do Balanceamento de Carga Load Balance

A Tecnologia do Protheus, permite que a execução do servidor, possa ser distribuída em mais de uma máquina ao mesmo tempo. Cada Servidor, fica responsável por um número limitado de requisições das estações, que se auto gerenciam.

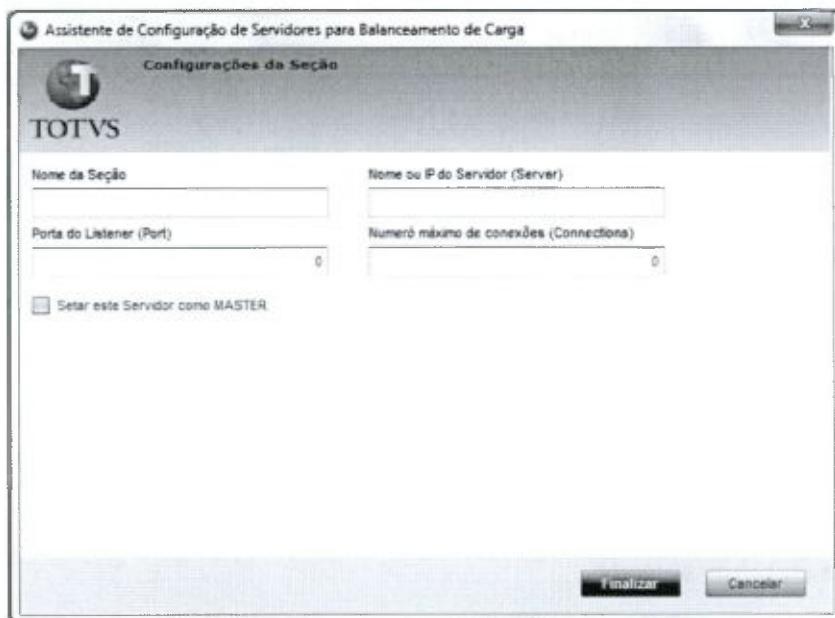
Este recurso é utilizado, quando a empresa possui processamento muito pesado e necessita diminuir o tempo de resposta das Estações com o Servidor.

Para configurar o servidor Master, selecione as seguintes opções "Iniciar" + "Programas" + "Protheus 12" + "Ferramentas" + "Assistente de Configuração do Servidor";

Clique na opção "Balanceamento de Carga" e no ícone "Novo Server para Balanceamento de Carga";



Preencha as "Configurações", informando os dados a seguir:



- Nome da Seção: Informa o nome do Slave
- Nome ou Ip do Servidor (Server): Endereço do servidor que está instalado o slave
- Porta do Lister(Port): Informa a porta de acesso da aplicação "AppServer"
- Número máximo de conexões (Connection): Quantidade de Threads simultâneas no mesmo Slaves

Após fazer o preenchimento selecionar o botão "Finalizar":

Os servidores Slave, requerem o tratamento da chave RootPath e para isso o diretório raiz do ambiente que está no servidor Master deve ser compartilhada com um único usuário com direitos suficientes para acessar, manipular, remover arquivos e pastas e que será utilizado por todos os demais servidores Slave.

Necessário ter o Binário com a mesma versão do servidor Master com a mesma versão de RPO

Fazer o mesmo procedimento de criar o serviço de balanceamento de carga realizado com servidor Master.

As configurações de um c-tree Server® para um ambiente com Load Balance e/ou um ambiente onde o RootPath é compartilhado por mais de um servidor ERP, deve ser utilizada uma configuração adicional em todos os ambientes, através da chave CtreeRootPath.

Ao utilizar a configuração CtreeRootPath, a mesma deve ser configurada em todos os ambientes dos servidores envolvidos no acesso aos arquivos da aplicação, inclusive o servidor de balanceamento (Balance). Caso exista acesso às tabelas (SXS ou SIGAPSS.SPF) realizado simultaneamente através de servidores configurados de modo diferente, por exemplo um com CtreeRootPath configurado (acessando através de drive:\pasta) e outro acessando sem o CtreeRootPath (acesso via \\servidor\pasta), o c-tree Server® vai fazer a rebuild da tabela e seus índices ao ser acessada a tabela por um dos servidores e quando o outro for acessar, a rebuild será realizada novamente, e vai falhar caso a tabela esteja em uso pelo outro servidor, podendo ainda apresentar ocorrência falsa de corromper o arquivo de senhas do ERP (sigapss.spf).

- CTREERootPath=E:\TOTVS\Protheus\Protheus_Data

```

1 [P12]
2 SourcePath=C:\TOTVS_12\Microsiga\protheus\apo\
3 RootPath=C:\TOTVS_12\Microsiga\protheus_data
4
5 CTREERootPath=C:\TOTVS_12\Microsiga\protheus_data
6
7 StartPath=\system\
8 RpoDb=SQL
9 RpoLanguage=Portuguese
10 RpoVersion=120
11 LocalFiles=CTREE
12 localdbextension=.DTC
13 TopMemoMega=1
14 DBDataBase=MSSQL

```

5.11.4. Teste de Balanceamento

Para fazer os testes do Slave iremos montar o cenário:

- Server_01 – Master (Gerenciador do Balanceamento), pois será ele que irá efetivamente, direcionar as "Requisições de Conexões", das "Estações do Protheus SmartClient";

No "Modelo" apresentado, balanceamos (2 Servidores), com (1 Conexão) cada, isto fará com que a primeira "Requisição de Conexão", entre no (Servidor 01), pois o mesmo, possui (1 Conexão Simultânea) apenas;

A segunda "Requisição de Conexão", será direcionada, para o (Servidor 02);

Quando iniciarmos os (Servidores – AppServer.exe), veremos que estes, tentarão se conectar pelo (Endereço IP), de cada um;

5.12. TOTVS Smart Client

O TOTVS Smart Client pode ser instalado fisicamente em uma estação, através de uma opção da tela inicial de instalação do Protheus 12.

Este recurso é útil para instalação do TOTVS Smart Client em computadores que não estão conectados à rede local, ou seja que necessitam que os executáveis estejam fisicamente na máquina. Por exemplo, em casos de filiais que acessam a matriz por uma linha discada através do protocolo TCP/IP.

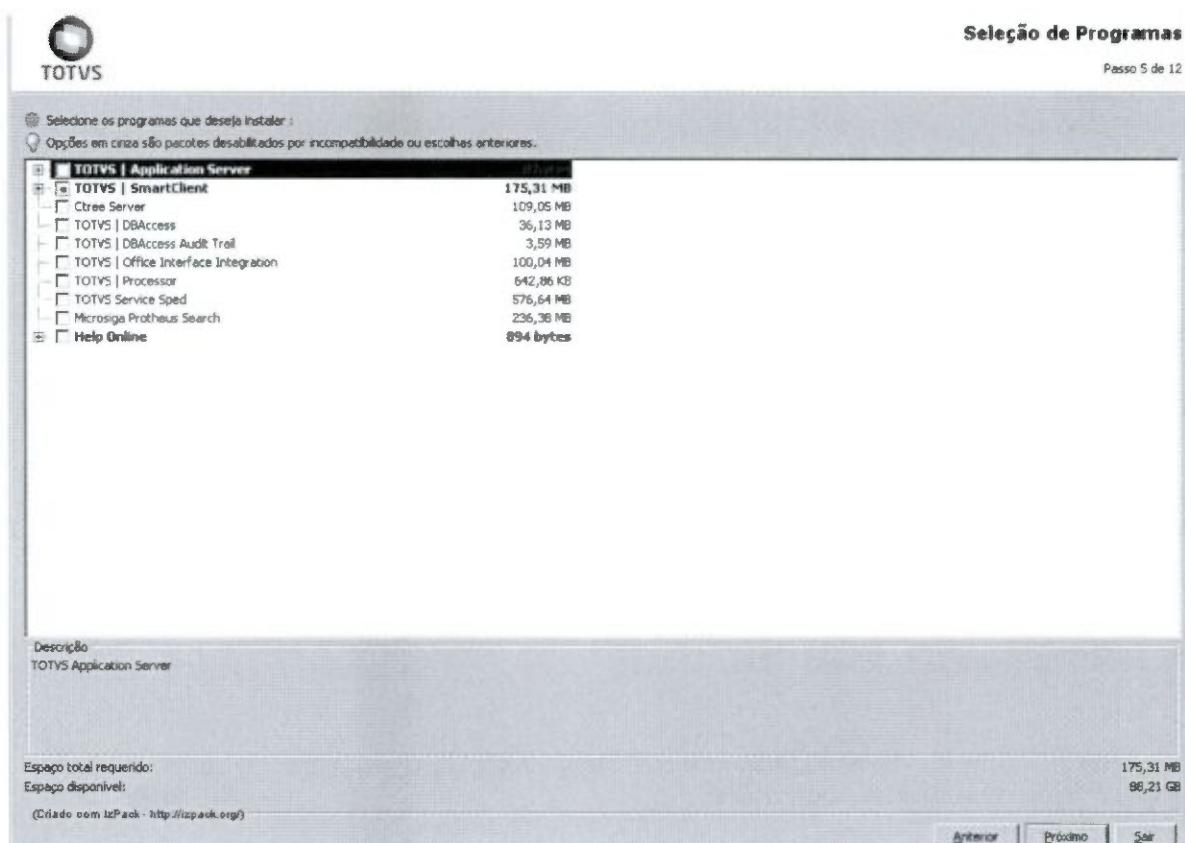
Devemos ressaltar que o procedimento de instalação do TOTVS Smart Client nas estações só é recomendado em casos específicos (utilização remota, alto tráfego de dados na rede, etc.). Preferencialmente, é indicado o uso centralizado do TOTVS Smart Client no Servidor, através da criação de atalho para acesso.

Para instalar o TOTVS Smart Client com AutoRun:

Coloque o CD-ROM no drive e aguarde a exibição da tela de abertura conforme a seguir.

São apresentados os idiomas disponíveis para apresentação da instalação do Protheus. Clique no idioma que deseja utilizar.

Clique na opção "TOTVS Smart Client".



Clique no botão "Próximo" para prosseguir.

Auto-Atualização do TOTVS Smart Client, essa opção facilita a Atualização das Estações dos usuários que possuam o TOTVS Smart Client instalado.

Dessa maneira, todas as Estações terão a sua Build atualizada automaticamente, assim que o usuário tentar conectar-se ao sistema.

5.12.1. Atualização do TOTVS Smart Client automática

Instale o "TOTVS Smart Client" na "Estação de Trabalho", seguindo as "Instruções Padrões" de instalação;

Edite o arquivo "SMARTCLIENT.INI", localizado na pasta:
"C:\TOTVS 12\MICROSIGA\Protheus12\bin\appserver".

Insira a nova seção conforme as linhas a seguir:

```
[UPDATE]
Enable=1
PathWindows="C:\TOTVS 12\MICROSIGA\Protheus12\bin\smartclient"
PathLinux=c:\protheus12\bin\remote_linux
```

Salve o "Arquivo" e encerre o "Editor de Textos";

Na sequência, tente conectar-se ao Smart Client";

Repare que, assim que o "TOTVS Smart Client" tentar conectar-se ao "Totvs Application Server", irá aparecer tela com régua de "Progressão".

Ao final, o TOTVS Smart Client estará atualizado.

5.12.2. SmartClient Activex

O acesso ao TOTVS Smart Client também pode ser realizado, através de um (Browser de Acesso à Internet).

Esse tipo de acesso, permite que o Protheus seja acessado em qualquer lugar, para isso basta que seja configurado o (Web Server) e que o usuário que irá utilizar o TOTVS Smart Client, tenha acesso à Internet.

Acesse a pasta ""C:\TOTVS 12\MICROSIGA\Protheus12\bin\APP SERVER\"" e edite o arquivo "TOTVSAPPSERVER.INI", acrescentando as seguintes instruções:

```
[HTTP]
Enable=1
Path=C:\Protheus12\Bin\smartclient_ActiveX
Port=90
```

Acesse a pasta ""C:\TOTVS 12\MICROSIGA\Protheus12\bin\SMARTCLIENT_ACTIVEX"" e edite o arquivo "SmartClient.htm", fazendo as seguintes modificações:

```
<param name="Server" value="Nome do Servidor">
<param name="TCPPort" value="1124">
<param name="Environments" value="Environment">
<param name="Language" value="1">
<param name="StartProgram" value="sigadv">
<param name="ReadOnly" value="OFF">
```

Acesse o "Browser de Internet" e digite no campo de "Endereço" os dados a seguir: <Http://Nome do Servidor/TotvsSmartClient.htm> ou executando o arquivo [TotvsSmartClient.htm](#).

Após uma breve espera, será solicitado o "Aceite do Certificado Digital":

Clique na opção "Ok", para continuar o processamento;

O sistema apresentará a "Tela de Parâmetros Iniciais", para acesso ao "SmartClient Activex", com as opções de "Programa Inicial e Ambiente", editados no "TotvsSmartClient.htm";

Informe uma "Senha Válida" e o "Ambiente de Acesso", desejado.

5.13. C-tree Server

O c-tree Server® é um servidor de banco de dados desenvolvido pela empresa Faircom.

O sistema ERP utiliza o c-tree Server® para manipulação dos dicionários de dados (SXs), arquivos de help, arquivo de senhas e profile de usuário. Para instalações com mais de 150 conexões simultâneas é necessário a aquisição da versão c-tree Server Enterprise que libera acesso acima desta quantidade e possui performance superior.

5.13.1. Instalação C-tree Server

O arquivo executável de instalação está disponível no próprio site de suporte da TOTVS.

- <https://suporte.totvs.com/download>

Fazer downloads da versão compatível com seu sistema operacional.

Após fazer o download executar o programa:

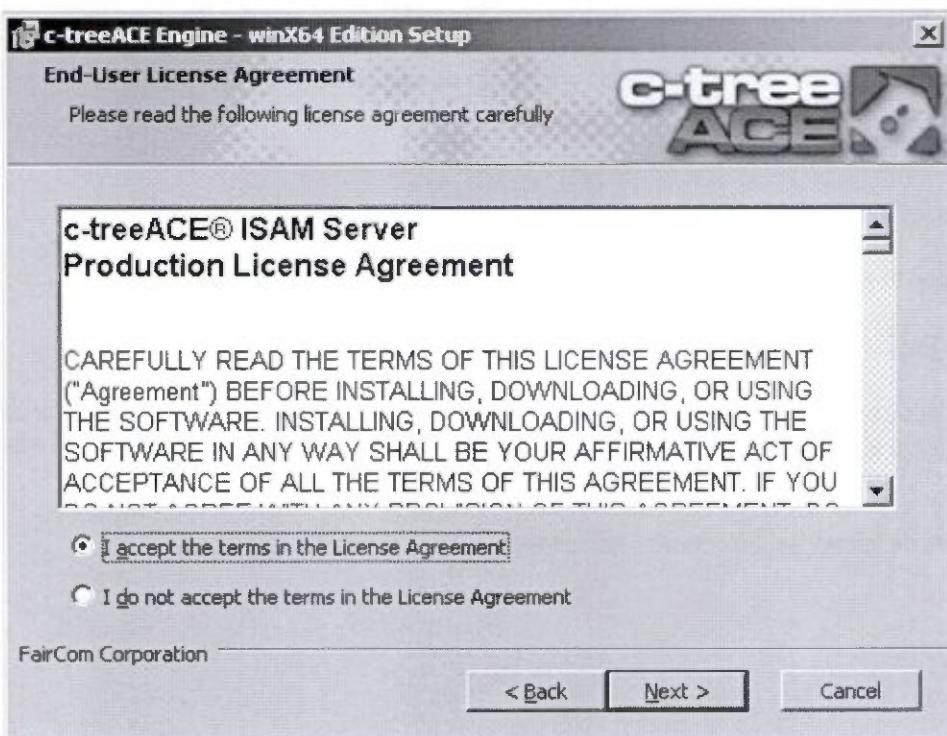
- [c-treeACE-Engine-ISAM.WinX64.130926.52178.Microsiga.msi](#)

Virá um arquivo zipado contendo o executável de instalação e um manual de instalação. Extrair o conteúdo em uma pasta. Execute o arquivo e siga as orientações avançando a cada etapa. No vídeo mostra passo a passo é bem simples só avançar mesmo.

Irá aparecer o Wizard de Instalação, Selecionar o botão Next.

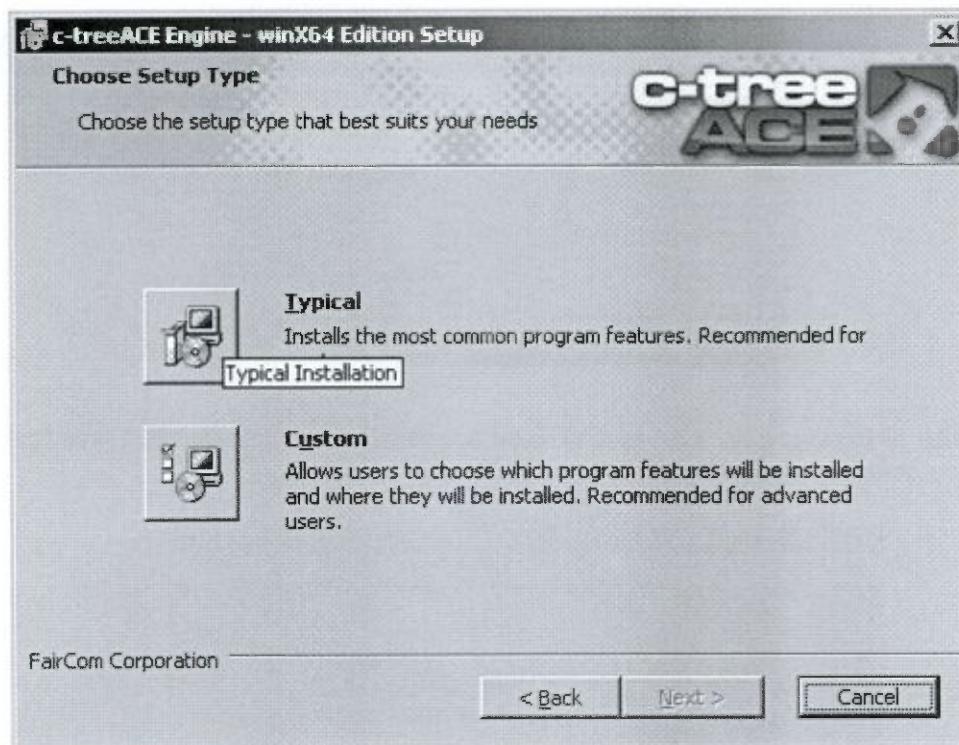


Selecionar que aceitar os termos de licenças.

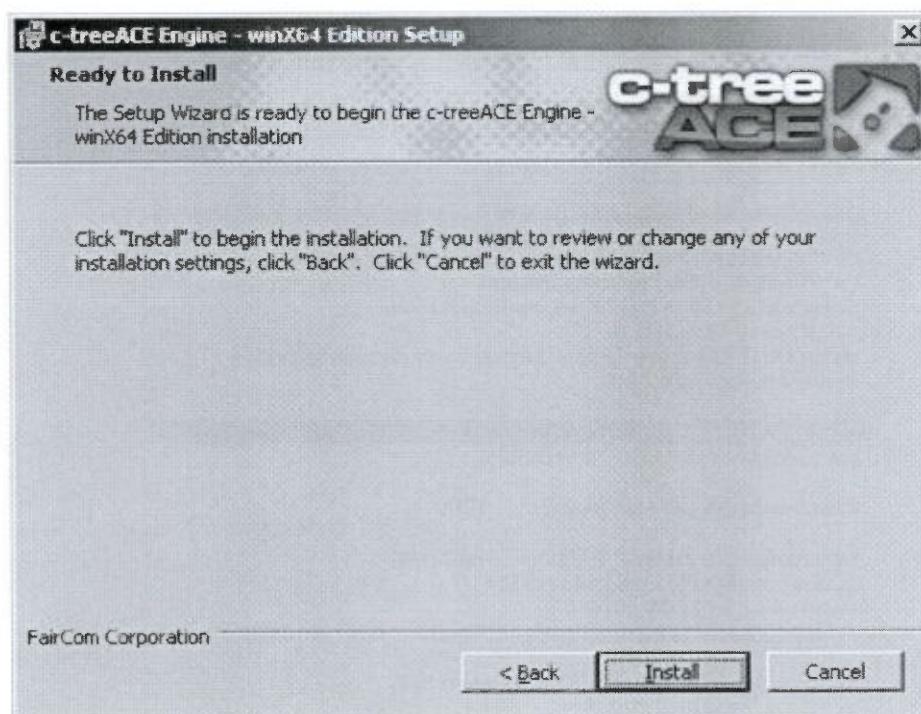


Selecionar a instalação "TYPICAL".

Formação Programação ADVPL



Selecionar o botão Install.



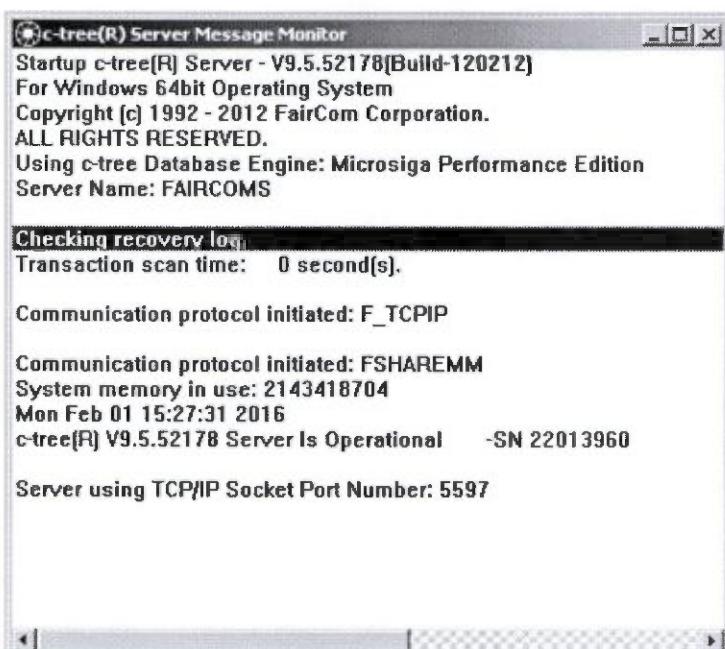
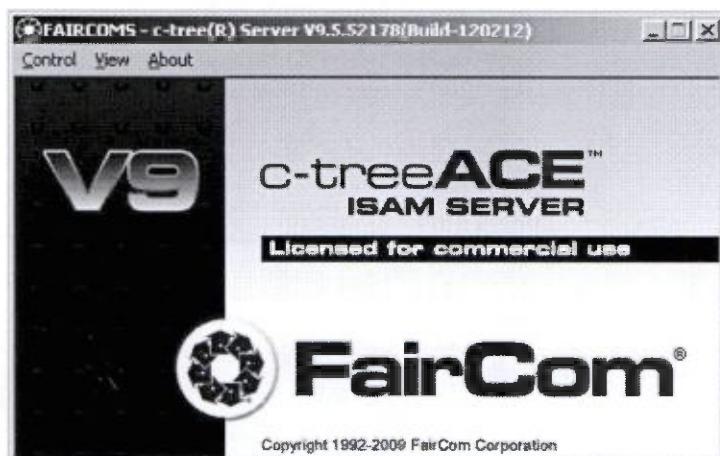
Após conclusão da instalação do c-tree Server é criado como um serviço do Windows, ele vem como manual. Recomendamos editar em propriedades do serviço e colocar como Automático.

Formação Programação ADVPL



Nome	Descrição	Estado	Tipo de Inicialização	Usuario Logon como
O avô... -	Iniciado	Automático	Serviço de rede	
O serviço...	Iniciado	Automático	Sistema local	
Ponto G...	Iniciado	Manual	Sistema local	
Método V...	Iniciado	Automático	Sistema local	
Regist... a...	Iniciado	Automático	Sistema local	
O serviço...	Iniciado	Automático	Serviço de rede	
Este serv...	Iniciado	Manual	Serviço de rede	
O serviço...	Iniciado	Automático	Serviço local	
Gerenci... -	Manual	Manual	Sistema local	
O wNCD...	Iniciado	Manual	Serviço local	
Gerenci... -	Iniciado	Manual	Sistema local	
O Serviço...	Iniciado	Automático	Sistema local	
O serviço...	Iniciado	Automático	Sistema local	
Este serv...	Iniciado	Automático	Serviço local	
O serviço...	Iniciado	Manual	Sistema local	
Gerenci... -	Iniciado	Automático	Sistema local	
Coordena... -	Manual	Manual	Serviço de rede	
Cópia de Sombra de Volume	Gerenci... -	Manual	Sistema local	
Dell KACE Agent	Manager ...	Iniciado	Automático	Sistema local
PowerShell Host	PowerShell Host	Brancos	Manual	Administrador

Ou podemos executar bin\ace\isam\lctsvr.exe, após executar selecionar o monitoramento do C-tree fica localizado próximo do relógio do Windows, nos ícones ocultos.

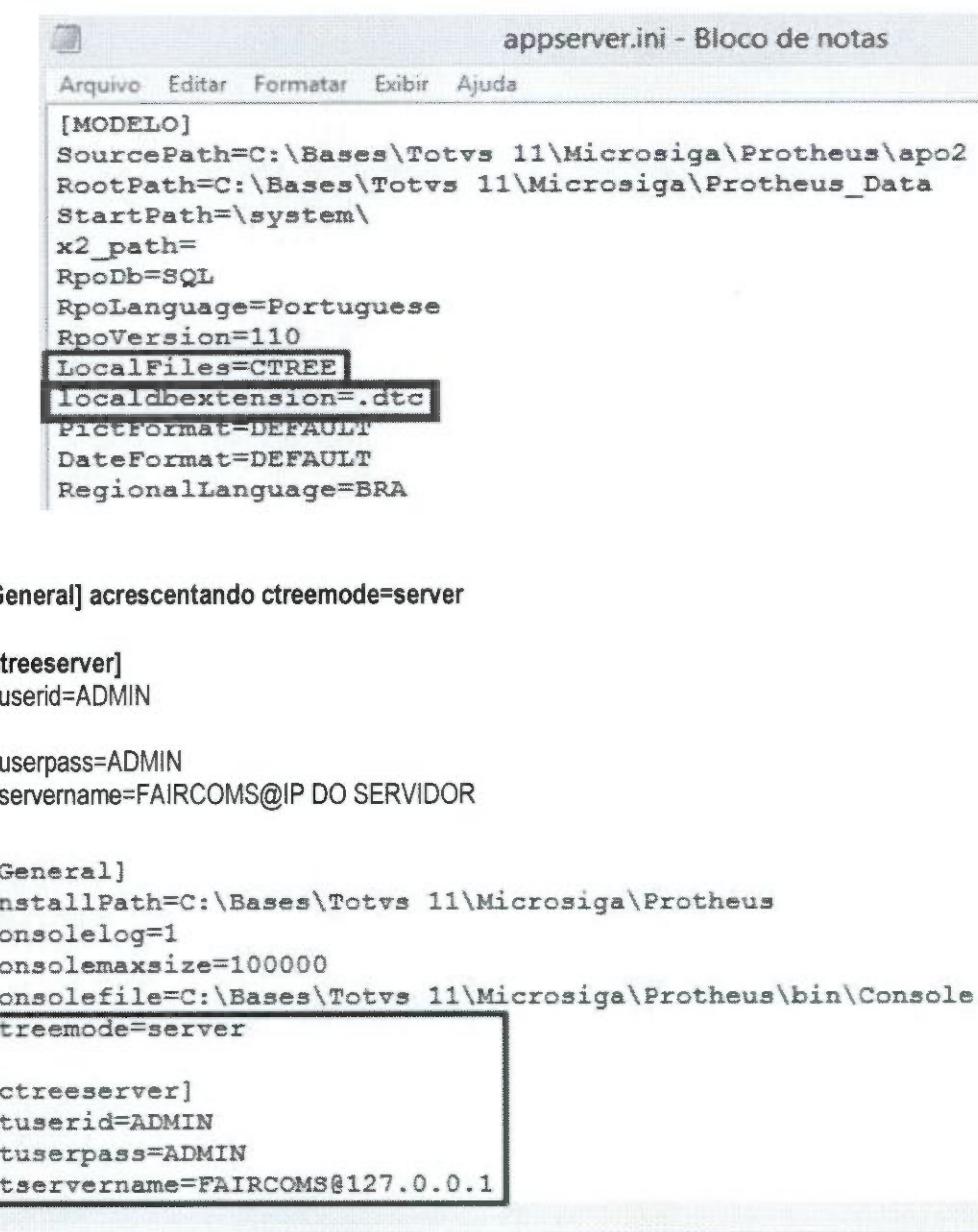


5.13.2. Configurações do C-tree Server

Para o C-tree Server funcionar necessário fazer algumas configurações. Localizar a pasta appserver e acessar o arquivo appserver.ini

Informar as configurações:

- LocalFiles=CTREE
- LocalDbExtension=.dtc



```
appserver.ini - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
[MODELO]
SourcePath=C:\Bases\Totvs 11\Microsiga\Protheus\apo2
RootPath=C:\Bases\Totvs 11\Microsiga\Protheus_Data
StartPath=\system\
x2_path=
RpoDb=SQL
RpoLanguage=Portuguese
RpoVersion=110
LocalFiles=CTREE
LocalDbExtension=.dtc
PictFormat=DEFAULT
DateFormat=DEFAULT
RegionalLanguage=BRA

[General]
InstallPath=C:\Bases\Totvs 11\Microsiga\Protheus
consolelog=1
consolemaxsize=100000
consolefile=C:\Bases\Totvs 11\Microsiga\Protheus\bin\Console.log
ctreemode=server

[ctreeserver]
ctuserid=ADMIN
ctuserpass=ADMIN
ctservername=FAIRCOMS@IP DO SERVIDOR
```

[General]

```
InstallPath=C:\Bases\Totvs 11\Microsiga\Protheus
consolelog=1
consolemaxsize=100000
consolefile=C:\Bases\Totvs 11\Microsiga\Protheus\bin\Console.log
ctreemode=server

[ctreeserver]
ctuserid=ADMIN
ctuserpass=ADMIN
ctservername=FAIRCOMS@127.0.0.1
```

Se sua estrutura não for Ctree antes de iniciar esse passo a passo você terá que converter seus SX's de codebase pra c-tree.

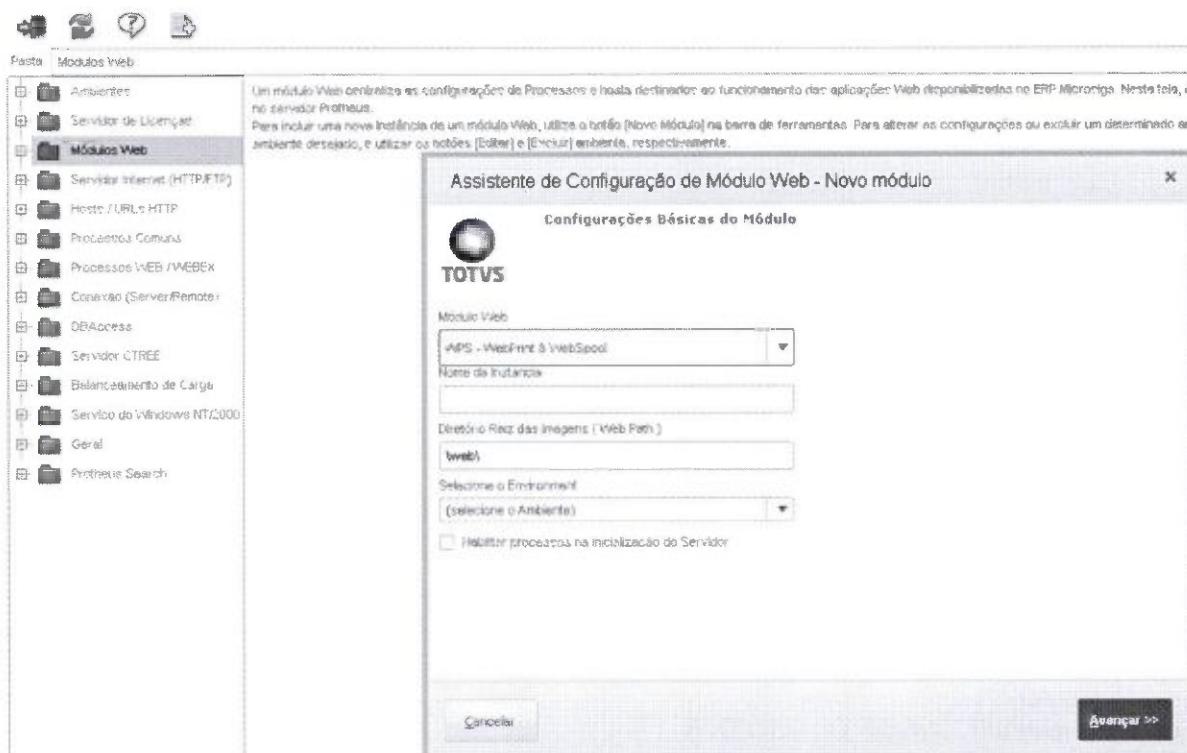
5.14. Configuração Portal

A opção de WebPrint, serve para que seja possível a impressão de Relatórios através do uso de um Browser de Internet.

Através dessa opção, será possível imprimir relatórios do Protheus, a partir de qualquer estação que possua o SmartClient Activex instalado e uma (Conexão de Internet) configurada.

Para configurar o WebPrint é necessário configura no Wizard um modulo Web.

Acessar o Wizard em modulo web selecionar o botão "Novo Modulo".



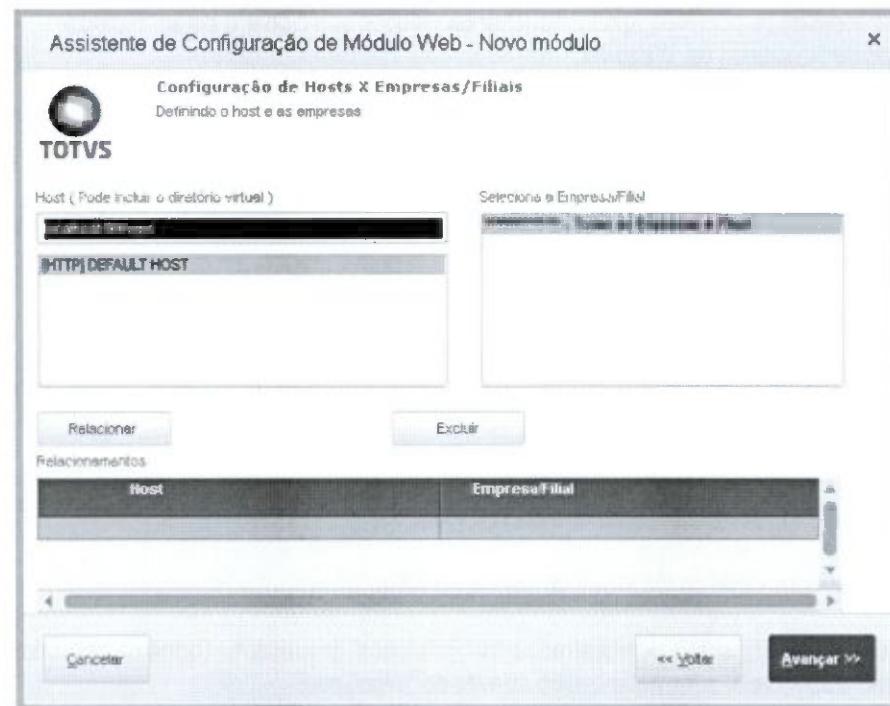
Preencher os campos:

- **Modulo Web:** Módulos Web disponível no Protheus selecionar a opção WPS – WebPrint e WebSpool
- **Nome da Instancia:** Um nome para idêntica o serviço na pasta Web\
- **Diretório raiz das imagens:** Diretório com os arquivos do portal disponível.
- **Selecionar o Environment:** Informar o ambiente para configurar o serviço.
- **Habilitar processos na inicialização do Servidor:** Serviço vai iniciar automaticamente

Selecionar o botão “Avançar”.

Tela para configurar os dados do portal.

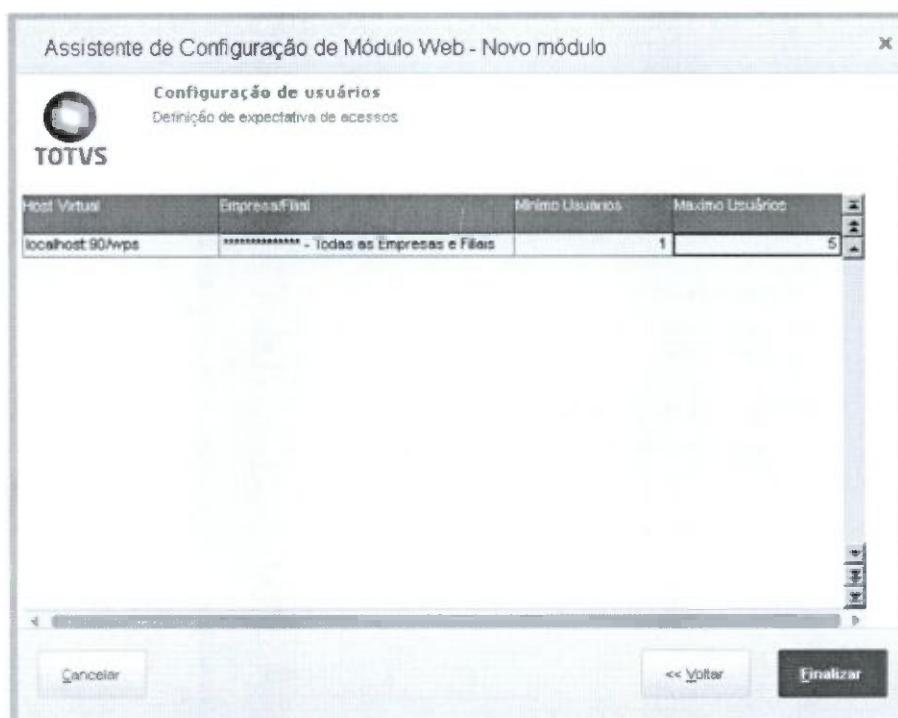
Formação Programação ADVPL



- **Host:** Endereço de acesso para o portal
- **Selecione a Empresa\Filial:** Qual empresa o serviço vai verificar os dados

Após preencher os campos selecionar o botão relacionar.

Ao fazer o relacionamento selecionar o botão “Avançar”



Informar a quantidade de acessos disponíveis, após preencher os campos selecionar o botão "Finalizar".
Como utilizar a impressão de Relatórios via Webprint:

1. Acesse o "Browser de Internet" e informe no campo de "Endereço" os dados a seguir:
"Http://Nome do Servidor/W_Aplogin.Apl";
2. Será apresentada uma "Tela de Acesso", na qual deverá ser digitado o "Nome do Usuário", sua respectiva "Senha" e "Empresa/Filial", que será utilizada para trabalho;
3. Confirme clicando na opção "Ok";
4. Pronto, todas as opções de relatórios que existem no "Protheus 12", estão disponíveis através do "Browser".

Como Visualizar Relatórios através do WebSpool:

1. Após enviar um "Relatório", para impressão através do "Webprint";
2. Acesse a opção "WebSpool", que se localiza logo abaixo no Menu;
3. Do lado direito do "Browser", serão demonstrados os "Relatórios" impressos. Todos os relatórios impressos "Via WebPrint", estarão disponíveis, para visualização através do "WebSpool".

6. Modulo Configurador

Veremos os principais arquivos de configuração do sistema, chamados de "Arquivos Customizadores", além de outros arquivos de uso interno da linha de Produto Microsiga Protheus.

Também analisaremos os diretórios específicos de cada arquivo dentro da estrutura do sistema.

Para acessar o configurador, é obrigatório escolher o programa inicial SIGACFG, somente os Administradores devem utilizá-lo.



6.1. Estrutura dos Diretórios

O Diretório, base da instalação é o \TOTVS 12\Microsiga, sendo definidos na instalação os subdiretórios, de acordo com a tabela a seguir:

PROTHEUS_DATA		Raiz do Sistema
\APO	Repositório de objetos (RPO).	
\BINISMARTCLIENT_ACTIVE	Destinado aos arquivos para acesso via Web por meio do recurso ACTIVEX.	
\BINISMARTCLIENT	Executáveis, bibliotecas e arquivos de configuração (.INI) do sistema.	
\BINIAPPSERVER	Executáveis, bibliotecas e arquivos de configuração (.INI) do sistema.	
\BINIAPPSERVERACE_9.99	Arquivos de configuração e bibliotecas para acesso aos arquivos SX's.	
\BINTOOLS	Onde são encontradas as ferramentas para manutenção do sistema.	
\CROVA	Destinado para a gravação dos lançamentos analíticos do ambiente Contábil.	
\CRYSTAL	Contém arquivos de bibliotecas e relatórios modelos do Crystal Report.	
\DATA	Contém o Banco de dados do Protheus (Codebase, CTREE ou ADS).	
\HANDHELD	Arquivos de biblioteca para integração com Palm-OS e Pocket PC.	
\INCLUDE	Contém as Bibliotecas (.CH) necessárias à execução e compilação do AP7.	
\MYPROJECTS\SAMPLES\ SOURCE	Fontes para exemplos de funções ADVPL.	
\SAMPLES\ DOCUMENTS	Arquivos modelos para integração com o pacote Microsoft Office.	
\SYSTEMLOAD	Arquivos de carga do Dicionário de Dados, Helps do Protheus e Indicadores Nativos, usados somente na instalação/migração do Protheus.	
\SPPOOL	Destinado para a gravação de relatórios gerados em disco.	
\SEMAFORO	Arquivos de semaforização de registros.	
\SYSTEM	Contém os arquivos de Customização, Empresa, Usuários, Fiscais, impressão e menus do Sistema.	
\ISISCOMEX	Contém arquivos específicos para uso dos ambientes de importação e exportação.	
\PROFILE	Armazena o perfil de cada usuário.	
\INCLUDE	Contém as Bibliotecas (.CH) necessárias à execução e compilação do AP7.	

6.2. Famílias de Arquivos

A TOTVS criou uma identificação para as tabelas e arquivos da Linha de Produto Microsiga Protheus, que consiste em codificar seus nomes seguindo um padrão pré-estabelecido, para que os usuários e analistas possam identificá-los com mais facilidade.

Os nomes das tabelas e arquivos são formados por um conjunto de três dígitos que os identificam. O quarto e quinto dígitos indicam o número da empresa e o sexto dígito é sempre zero, que é utilizado para identificação interna do sistema:

- 1^a posição – (S) de SIGA ou outra letra quando se tratar de um ambiente específico.
- 2^a posição – de (A a Z) ou (0 a 9), definindo a família do arquivo.
- 3^a posição – de (1 a Z), definindo a sequência dentro da família.

Como exemplo, vamos utilizar o Arquivo de Clientes – SA1010 no qual:

- “S” – significa que este arquivo pertence aos ambientes Genéricos;
- “A” – letra que representa a família à qual o Arquivo pertence;
- “1” – sequência do arquivo na família;
- “01” – caracteres que representam a numeração do grupo da empresa;
- “0” – dígito de uso exclusivo da TOTVS.

Para as customizações dos clientes, são reservadas as famílias SZ? e ZZ?, os usuários não devem utilizar as famílias reservadas para MICROSIGA, pois estas poderão ser sobrepostas em uma futura atualização de versão.

A Família SX fica armazenada no diretório “\SYSTEM”, e é formada pelos arquivos customizadores da Linha de Produto Microsiga Protheus. Referem-se a todos os ambientes, pois são de uso Genérico:

Tabela	Função
SIX	Índices dos Arquivos (SX2)
SX1	Manutenção de Perguntas de parametrização (movimentações, consultas e relatórios).
SX2	Mapeamento de Arquivos.
SX3	Dicionário de Dados.
SX4	Configuração de Agenda de Relatórios e Processos.
SX5	Tabelas genéricas do sistema.
SX6	Parâmetros.
SX7	Gatilhos de Campos (SX3).
SX9	Relacionamento entre Arquivos (SX2).
SXA	Pastas Cadastrais dos Arquivos (SX2).
SXB	Consultas-padrão.
SXD	Cadastro de Relatórios e Processos para Agendamento (SX4).
SXE	Controle de numeração (próximo número + 1).
SXF	Controle de numeração (Último Número Sequencial + 1).
SXG	Configuração padrão para grupo de campos.
SXH	Tabela de eventos do sistema.
SXI	Inscrição para acesso aos eventos do sistema.

Tabela	Função
SXK	Controle de Perguntas (SX1) por usuários.
SXM	Agendamento de Workflow.
SXO	Cadastro de Logs por Campo.
SXP	Histórico de Logs cadastrados no SXO.
SXQ	Cadastro de filtros inteligentes da mbrowse (contém as informações necessárias para a criação do filtro).
SXR	Cadastro de relacionamento entre programa x filtro (utilizada internamente pelo Protheus para verificar em quais programas os filtros poderão ser utilizados).
SXS	Cadastro de programas (utilizado na validação para mostrar/inibir os filtros na execução da mbrowse).
SXT	Tabela de usuários (contém as informações dos usuários que poderão utilizar os filtros da mbrowse).
SXU	Log gerado pela rotina de processamento (tNewProcess)
SXV	Mashups
SXOffice	Cadastro de relacionamento entre as entidades (tabelas) e as consultas TOII.

Também há arquivos que armazenam dados especiais, como script de planilhas, senhas, helps, menus, consultas etc. São eles:

SIGAHELP.HLP	Help de Campos.
SIGAMAT.EMP	Empresas.
SIGAMAT.IND	Índice do SIGAMAT.EMP
SIGAPSS.SPF	Senhas.

Temos ainda outros arquivos com extensões que também fazem parte da Linha de Produto Microsiga Protheus, os quais estão relacionados a seguir:

.#DB	Backup gerado pelo Configurador.
.DRV	Drivers de Impressoras.
.REM	Envio de Transmissão Bancária.
.RET	Recebimento de Transmissão Bancária.
.LOG	Arquivo TTS.
.SC999999.*	Arquivos temporários.
.Batch*.op	Arquivos temporários utilizados na geração de OP's.

As tabelas da linha de Produto Microsiga Protheus ficam armazenadas no banco de dados ou em uma pasta definida no dicionário de dados quando se tratar de base não relacional ou quando o formato de dados utilizado não for permitido pelo banco de dados.

6.3. Arquivos, Tabelas e Campos

A Linha de Produto Microsiga Protheus conta com um dicionário de dados dinâmico, este dicionário é responsável pela criação de todas as tabelas criadas no banco de dados utilizados pelo ERP. Além disso, há elementos do próprio sistema que podem ser configurados e criados.

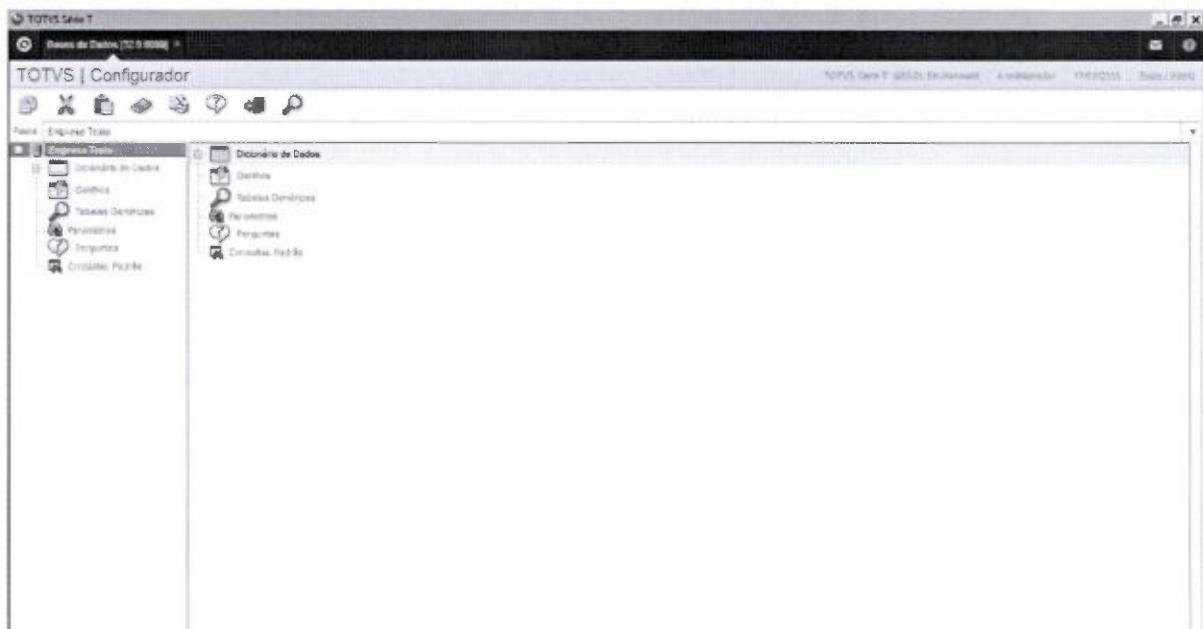
6.3.1. Perguntas (SX1)

Quando da necessidade de customizações próprias que envolvam o desenvolvimento de relatórios ou processos próprios, estes poderão ter parametrizações diferenciadas, de acordo com as necessidades específicas dos usuários ou da empresa. Para tanto, é possível através das perguntas, criar "filtros", que tenham por principal objetivo selecionar registros a serem impressos ou processados. Tais filtros, geralmente são: por código início/fim, por período inicial/final, por filial, etc.

Em síntese, cada processo ou filtro de um relatório, terá seu próprio grupo de perguntas, que serão "chamadas", todas às vezes que se processem esses relatórios ou processos.

Incluindo novas Perguntas:

Selecione as seguintes opções: "Base de Dados" + "Dicionário" + "Bases de Dados";
O sistema apresentará uma tela contendo todas as opções disponíveis para configuração.



Selecionar a opção "Perguntas" e clicar no botão "Incluir".



Clique em Incluir e preencha os campos conforme descrição a seguir:

- **Grupo:** Informe um nome para o conjunto de perguntas em cadastro.
- **Ordem:** Este campo é definido pelo Sistema, e registra a ordem de apresentação das perguntas na tela.
- **Pergunta:** Informe a pergunta desejada.
- **Pergunta Espanhol/Pergunta Inglês:** Informe a pergunta desejada traduzida para o espanhol e para o inglês, respectivamente.
- **Tipo:** Selecione o tipo do conteúdo de resposta da pergunta, se: 1 = Caracter, 2 = Numérico ou 3 = Data.
- **Tamanho:** Informe o tamanho do campo para a resposta da pergunta.
- **Decimal:** Informe o número de casas decimais da resposta, se houver
- **Formato:** Informe uma máscara para resposta da pergunta
- **Validação:** Este campo é possível fazer validações na resposta da pergunta
- **Help:** Informe o Help do campo para facilitar a identificação da pergunta
- **Objeto:** Selecione como a pergunta será apresentada na tela, se:
 - 1 = Edit - (Formato que permite editar o conteúdo expresso no campo).
 - 2 = Text - (Formato de texto, que não permite alteração).
 - 3 = Combo - (Formato que permite a opção de seleção de dados para o campo).
 - 4 = Range - (Permite definir intervalos de dados sequenciais).

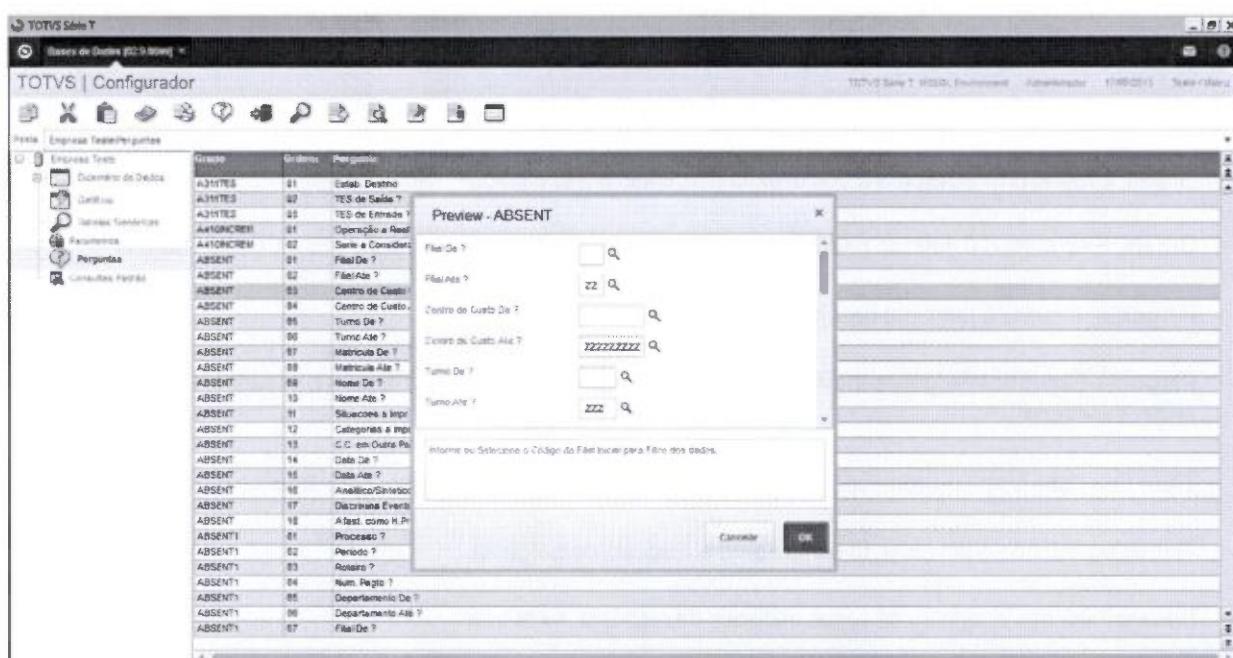
5 = File - (Permite selecionar um arquivo do servidor ou da estação local)

6 = Expression - (Permite definir uma expressão de filtro).

7 = Check - (Permite selecionar até cinco itens).

- **Consulta Padrão (Edit):** Informe o texto da consulta padrão F3
- **Conteúdo (Text):** Informe o conteúdo do campo, que não poderá ser alterado, desde que o campo Objeto tenha sido definido como 2 = Text.
- **Pré-seleção (Combo):** Informe os valores que definem a apresentação dos itens como opção de preenchimento na tela. Se informado 0, o primeiro item aparecerá só. Este campo somente poderá ser preenchido quando o campo Objeto for preenchido com 3 = Combo.
- **Item 1 a 5 (Combo):** Informe nestes campos os itens que serão apresentados na tela como opção de preenchimento da pergunta, quando o campo Objeto for preenchido com 3 = Combo. Pode haver até 5 itens para seleção, ou menos, desde que não pule a sequência de preenchimento.

Após a confirmação, clique em “Pesquisar” e digite o nome da pergunta criada recentemente. Posicionado nesse grupo de perguntas, clique no botão “Preview”, para verificar se as configurações estão de acordo com o desejado.



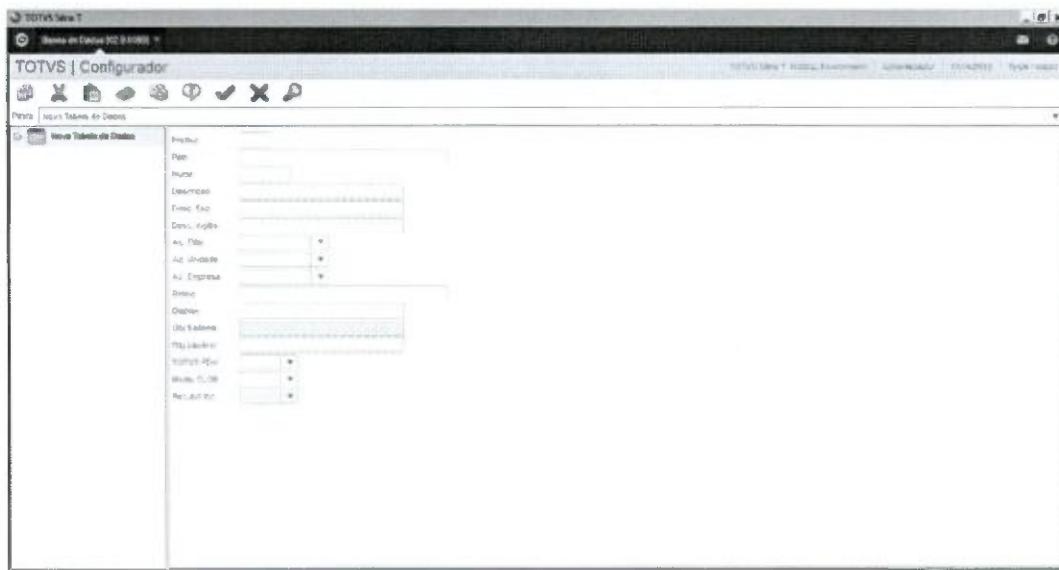
Importante

Os grupos de perguntas originais da Linha de Produto Microsiga Protheus não devem ser alterados senão em casos específicos, como exigência de um Boletim Técnico. Os nomes dos grupos criados em personalizações devem ser iniciados com “ZZ” sob o risco de serem sobreescritos numa atualização de versão.

6.3.2. Arquivos (Sx2)

A tabela SX2 do dicionário de dados define de uma forma padronizada as tabelas disponíveis dentro do Ambiente Microsiga Protheus, por meio desta tabela que temos o conhecimento das tabelas disponíveis, quais os alias associado a ela, caminho físico da tabela (quando CodeBase).

Como criar Novos Arquivos, utilizando Ambiente Configurador, selecione as seguintes opções: "Base de Dados" + "Dicionário" + "Arquivos" e clique no botão "Incluir".



Preenchimento dos campos obrigatórios conforme descrição a seguir:

- **Prefixo:** Informe um nome do arquivo "Tabela".
- **Path:** Endereço da estrutura do RootPath aonde deseja grava o arquivo, para quem trabalha com a estrutura Code-base.
- **Nome:** Nome do Arquivo
- **Descrição:** Descrição do Arquivo
- **Ac Filial:** Possui dois modos: Exclusivo e Compartilhado. Cada empresa possui arquivos próprios que podem ser dados comuns Ex: clientes, produtos, fornecedor, etc. Podendo haver o uso comum de qualquer um deles entre "Filial, Unidade de Negócio e Empresa". O campo "Filial" somente será gravado com brancos quando a Empresa, Unidade de Negócios e a Filial forem compartilhadas. Em caso de uso exclusivo, este campo recebe a identificação da Empresa, Unidade de Negócios e Filial conforme a configuração do Grupo de Empresas. Assim, o usuário tem acesso somente aos dados da sua Empresa, Unidade de Negócios e Filial.
- **Ac Unidade:** Possui dois modos: Exclusivo e Compartilhado
- **Ac Empresa:** Possui dois modos: Exclusivo e Compartilhado
- **Rotina:** Nome da rotina executada na abertura da tabela
- **Display:** Campos separados por + que serão apresentas em detalhes do browser.

- **Obj. Sistema:** Nome da fonte responsável pela manutenção e verificação de acesso a rotina. Definido pela Equipe da Microsiga Protheus Padrão.
 - **Obj. Usuário:** Nome da fonte responsável pela manutenção e verificação de acesso a rotina.
 - **TOTVS PDV:** Sim/Não
Campo permite que sejam carregados somente os arquivos utilizados pelo modulo "Novo PDV" e assim diminui o tempo de carga nas tabelas que não seriam usadas pelo modulo.
 - **Memo Clob:** Indica se a tabela utilizará o tipo de dado "CLOB" nos campos memo em bancos de dados SQL.O CLOB (Character large object) permite maior flexibilidade no uso do conteúdo em relação aos campos memo binários tradicionais. Esta configuração é válida apenas quando a tabela ainda não foi criada.
 - **Rec.aut.Inc**

Importante

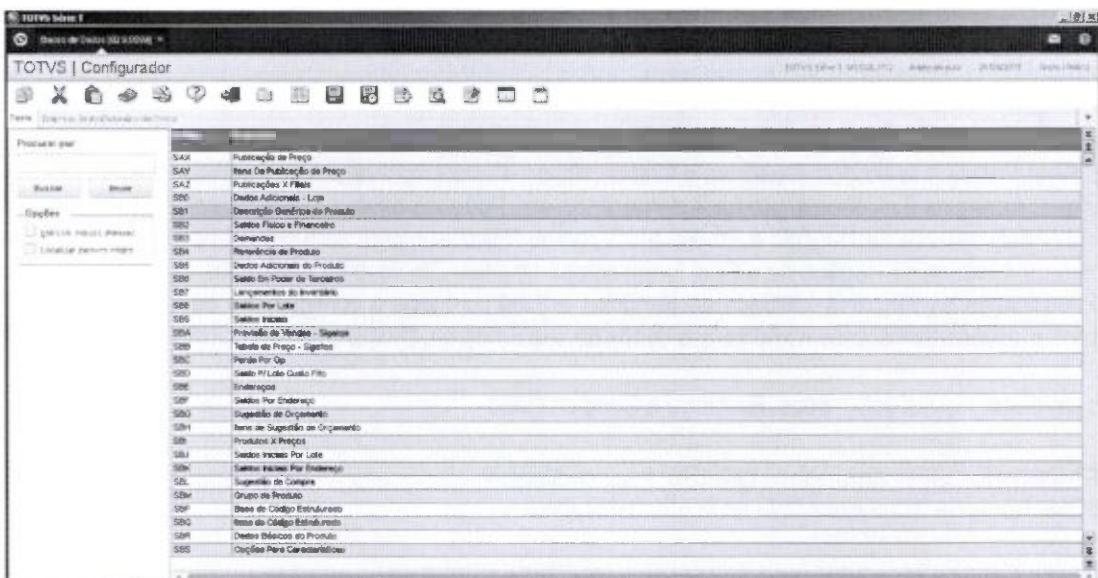
Os novos arquivos devem ser criados dentro das Famílias **SZ?ZZ?**, assim teremos a certeza de que durante uma atualização de versão, não ocorrerão problemas entre nossos arquivos e os da TOTVS.

6.3.3. Campos (SX3)

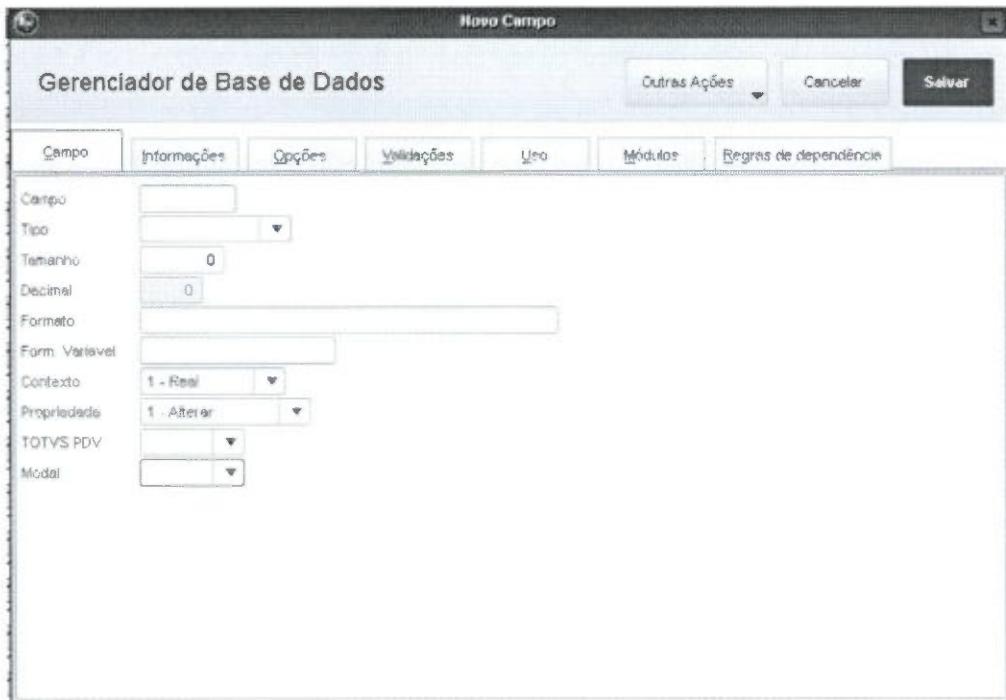
Os campos são vinculados aos arquivos utilizados no sistema. Na edição de arquivos SX3, é possível incluir novos campos, editá-los e alterá-los, excluir e ordenar os campos.

No menu principal, selecione as opções "Base de Dados" + "Dicionário" + "Arquivos".

O Sistema apresenta a janela Browser relacionando o dicionário de dados do Sistema, ou seja, todos os arquivos utilizados.



Selecione o arquivo desejado e selecione Editar selecionar "Campos" podendo Incluir ou Alterar.



O Sistema apresenta a tela subdividida em pastas.

Na pasta "Campo" e preencha os campos, conforme descrição a seguir:

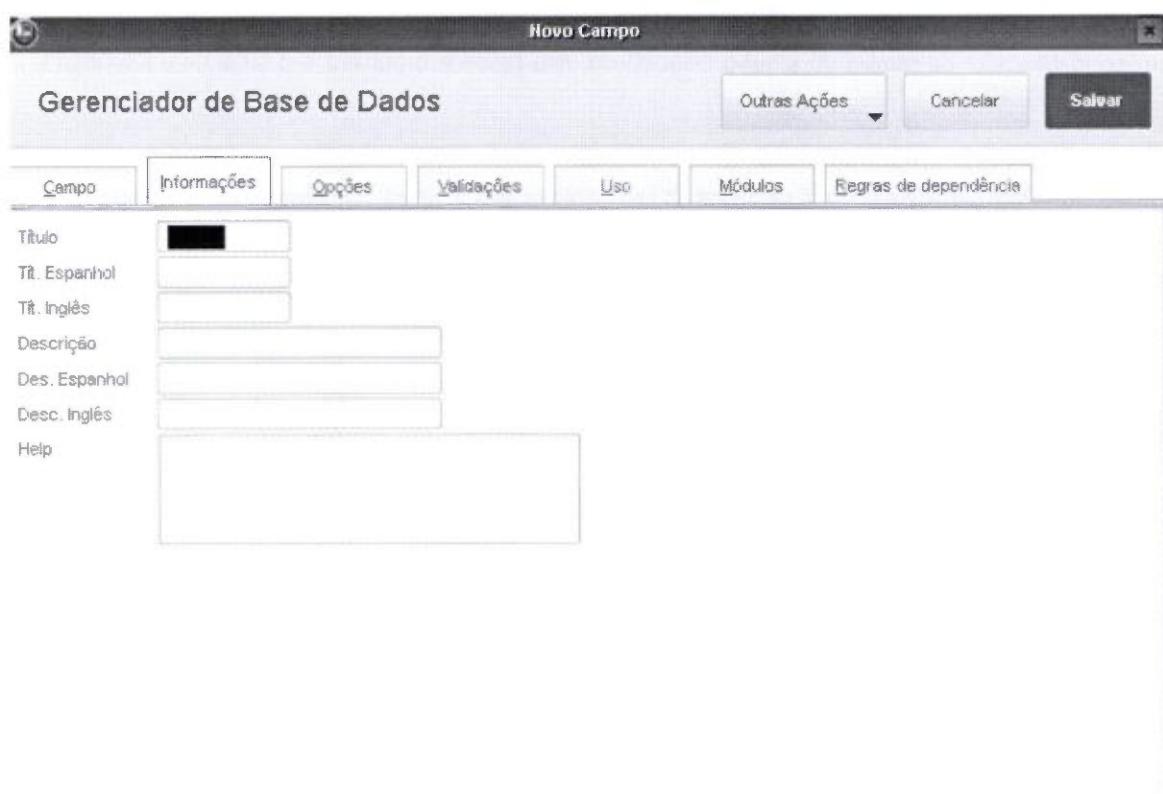
- **Campo:** Na inclusão, informar o nome do campo com até 10 caracteres, que deve conter obrigatoriamente o prefixo identificador da família do arquivo selecionado, seu respectivo número e o sinal de sublinhado (_). Exemplo: A1_OBS.
- **Tipo:** Informar o tipo do dado a ser inserido, que pode ser:
 - C - Para permitir a entrada de dados do tipo Caracter, isto é, letras, números e símbolos esp
 - N - Para permitir a entrada de dados do tipo Numérico, isto é: números e sinais de positivo (+), negativo (-) e separadores decimais.
 - L - Para permitir a entrada de dados do tipo Lógico, isto é: formato verdadeiro/falso ou sim/não
 - D - Para permitir a entrada de dados do tipo Data, ou seja, DD/MM/AA.
 - M - Para permitir a entrada de dados extensos do tipo Caracter.
- **Tamanho:** Informar a quantidade máxima de dígitos que o campo pode conter.

Tipo	Tamanho
CARACTER	Máximo 512 dígitos
NUMÉRICO	Máximo 18 dígitos
LÓGICO	1 dígito
DATA	8 dígitos
MEMO	O tamanho informado será o espaço reservado para apresentação, porém não há limite para o cadastramento.

- **Decimal:** Digite a quantidade de casas decimais para campos do tipo numérico (de 0 a 9). O número de casas decimais, incluindo o ponto decimal, é deduzido do tamanho do campo informado.
- **Formato:** Informar uma máscara para edição do campo, ou seja, os tipos de dados que serão aceitos e a sua formatação.
- **Form. Variável:** O campo X3_PICTVAR permite a edição de pictures variáveis durante a entrada de dados. Este campo irá suportar qualquer função ou formato. Obrigatoriamente o campo deverá ser do tipo Caracter, porém o campo enquanto editado será do tipo Numérico. A função RdMake executada deverá retornar a picture desejado.
- **Contexto:** Esta opção permite configurar se os campos criados serão reais ou virtuais. Um campo definido como virtual não faz parte da estrutura real do arquivo, permitindo ao usuário a visualização de informações obtidas através de fórmulas, gatilhos e EXECBLOCK nas telas de entrada de dados do Sistema.
- **Propriedade:** Esta opção permite definir o tipo de atualização do campo nas telas de inclusão e alteração de dados: se alteração ou somente visualização.
- **Totvs PDV:** Sim/Não Campo permite que sejam carregados somente os campos utilizados pelo modulo "Novo PDV" e assim diminui o tempo de carga nas tabelas que não seriam usadas pelo modulo.
- **Modal:** Sim/Não

Indica se o campo será apresentado em janelas de estilo "modal".

Após preencher os cadastros selecionar a pasta "Informações".



Formação Programação ADVPL



Para preencher a pasta "Informações":

- **Título/ Tít. Espanhol / Tít. Inglês:** Informe o título do campo a ser apresentado na tela de entrada de dados, com até 12 caracteres.
- **Descrição/ Desc. Espanhol / Desc. Inglês:** Informe uma explicação resumida sobre o conteúdo do campo, com até 25 caracteres.
- **Help:** Informe a descrição detalhada sobre a funcionalidade do campo, sendo está acessada no módulo quando pressionado [F1] sobre o campo.

Após preencher os cadastros selecionar a pasta "Opções".

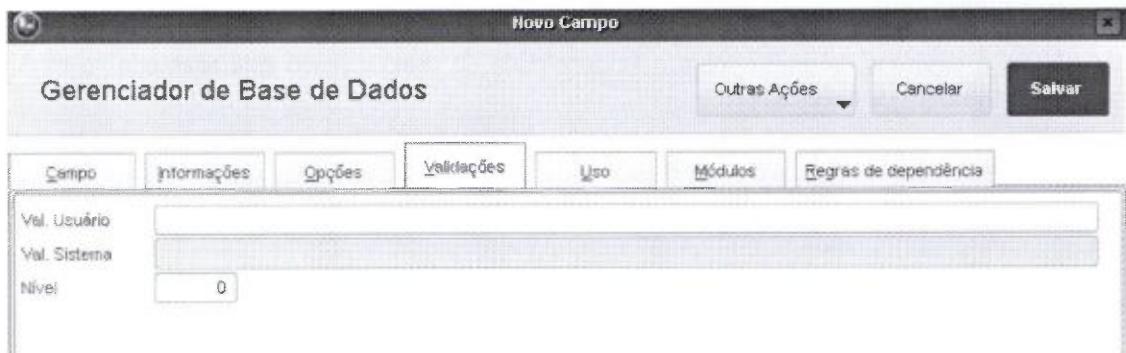


Na pasta "Opção" e preencha os campos, conforme descrição a seguir:

- **Lista de Opção/Lista Espanhol/Lista Inglês:** Quando o campo for acessado no ambiente, será aberta uma lista de opções permitindo ao usuário selecionar uma delas, agilizando o trabalho de digitação da seguinte forma: <opção>=<descrição>;<opção>=<descrição>.
- **Inicializador Padrão:** Informar o conteúdo inicial padrão para o campo. Este dado será sugerido na inclusão de um novo registro.
- **Inic. Browse:** O campo X3_INIBRW irá suportar a apresentação de um campo virtual com qualquer conteúdo durante a função mbrowse. Assim, este campo deverá possuir um comando (execblock, por exemplo), que devolve um campo ou expressão de qualquer outro lugar.
- **Modo Edição:** O campo X3_WHEN permite a edição de um campo apenas quando determinada condição for verdadeira. A regra poderá ser uma condição, função ou execblock. Note que o retorno obrigatoriamente deverá ser do tipo Lógico (True ou False).
- **Corretor:** Habilita o uso do corretor ortográfico.

- P Search:** Habilita o uso do campo para ferramenta Protheus Search.
- Consulta Padrão [F3]:** Neste campo deve ser informado o nome da consulta padrão ou número da tabela a ser chamada sempre que a tecla [F3] for acionada.

Após preencher a opções, clique na pasta "Validações" para cadastrar as validações do campo;



Preencha os campos conforme descrição a seguir:

- Val. Usuário/Val. Sistema:** Permite utilizar funções, isto é, programas especiais do Microsiga Protheus® e da linguagem AdvPL que fazem consistências dos dados digitados. As funções são identificadas pelo sinal de parênteses () à frente de seu nome. O último parâmetro tem como default o campo que está sendo digitado.
- Nível:** Informar um número para estabelecer o nível de acesso deste campo em relação ao sistema como um todo. Este nível, comparado ao nível definido para um determinado usuário quando da designação de sua senha, faz com que a rotina de controle de senhas permita ou não o acesso do usuário a esta informação.

Após preencher a opções, clique na pasta "Uso" para definir o uso do campo;



Preencha os campos conforme descrição a seguir:

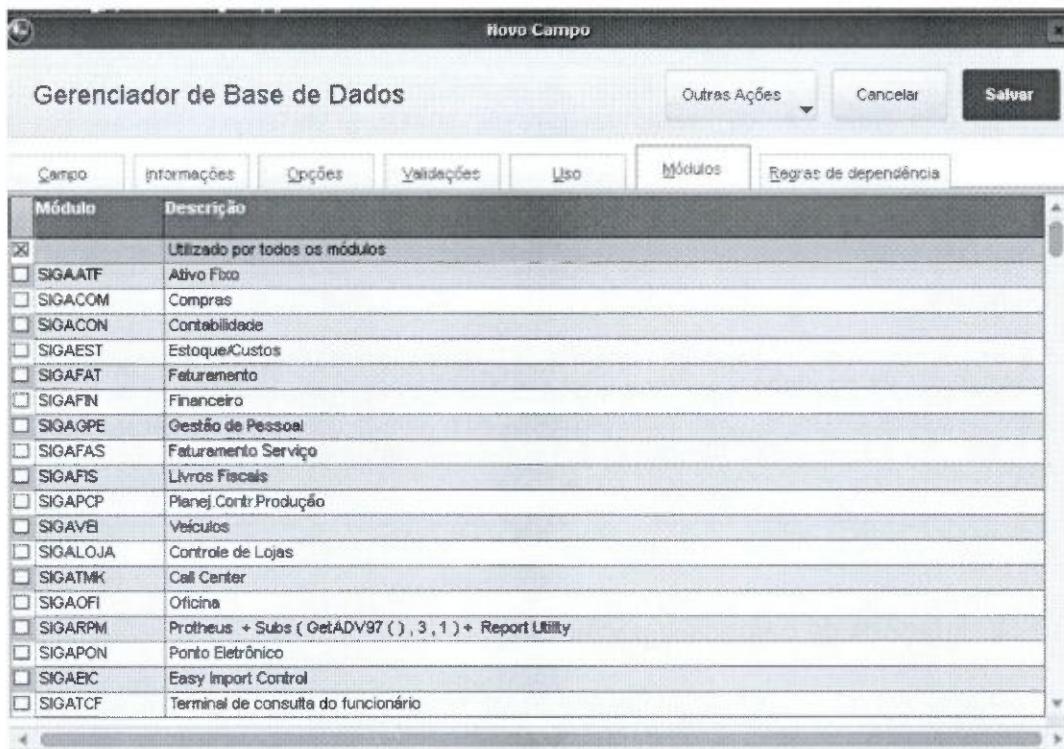
- Obrigatório:** Esta opção permite configurar se os campos criados terão seu preenchimento obrigatório ou não.
- Usado:** Nesta opção é possível definir quais campos deste arquivo devem ser utilizados.

Formação Programação ADVPL



- **Browse:** Define se para determinados ambientes, o campo selecionado será apresentado no browser.

Após preencher as opções de uso do campo, clique na pasta "Módulos" para definir o uso do campo nos módulos desejados;



Caso desejar a limitação do campo por módulos, selecionar os módulos desejados. Após preencher as opções de uso do campo, clique na pasta "Regra de dependência" para definir a ordem dos campos.



A regra de dependência entre campos é utilizada por rotinas escritas em MVC, para se forçar o preenchimento em ordem de campos, fazendo que um campo dependa do outro para liberar sua edição.

- **Tipo 1 Pré-Validação:** Adiciona uma relação de dependência entre campos do formulário, impedindo a atribuição de valor caso os campos de dependência não tenham valor atribuído. Por exemplo, o preenchimento do campo Código da Loja só pode ser preenchido após o preenchimento do campo Código do Cliente.
- **Tipo 2 Pós-Validação:** Adiciona uma relação de dependência entre a referência de origem e destino, provocando uma reavaliação do destino em caso de atualização da origem. Por exemplo, após o preenchimento do campo Código da Loja a validação é reavaliada caso o Código do Cliente seja alterado.
- **Tipo 3 Pré e Pós-Validação:** São os tipos 1 e 2 simultaneamente.

Importante

Os campos criados em tabelas padrão da Linha de Produto Microsiga Protheus devem utilizar a nomenclatura XX_ZZ?????, onde XX é o alias da tabela e ????? as possibilidades de nome entre caracteres e números, para evitar problemas com campos que venham a ser incluídos pela TOTVS em atualizações. Estrutura do SX3 suporta até 349 campos por arquivo

6.3.3.1 Validações de campos e perguntas

As funções de validação têm como característica fundamental um retorno do tipo lógico, ou seja, um conteúdo.T. – Verdadeiro ou.F. – Falso.

Com base nesta premissa, a utilização de validações, no Dicionário de Dados (SX3) ou nas Perguntas de Processos e Relatórios (SX1), deverá focar sempre na utilização de funções ou expressões que resultem em um retorno lógico.

Através do módulo Configurador é possível alterar as propriedades de um campo ou de uma pergunta, de forma a incluir regras de validação para as seguintes situações:

- SX3 – Validação de usuário (X3_VLDUSER).
- SX1 – Validação da pergunta (X1_VALID).

Dentre as funções que a linguagem ADVPL, em conjunto com os recursos desenvolvidos pela aplicação ERP, para validação de campos e perguntas serão detalhadas:

EXISTCHAV()

Sintaxe	ExistChav(cAlias, cConteudo, nÍndice)
Descrição	Retorna .T. ou .F. se o conteúdo especificado existe no alias especificado. Caso exista será exibido um help de Sistema com um aviso informando da ocorrência. Função utilizada normalmente para verificar se um determinado código de cadastro já existe na tabela na qual a informação será inserida, como por exemplo, o CNPJ no cadastro de clientes ou fornecedores.

EXISTCPO()

Sintaxe	<code>ExistCpo(cAlias, cConteudo, nIndice)</code>
Descrição	Retorna .T. ou .F. se o conteúdo especificado não existe no alias especificado. Caso não exista será exibido um help de Sistema com um aviso informando da ocorrência. Função utilizada normalmente para verificar se a informação digitada em um campo, a qual depende de outra tabela, realmente existe nesta outra tabela, como por exemplo, o código de um cliente em um pedido de venda.

NAOVAZIO()

Sintaxe	<code>NaoVazio()</code>
Descrição	Retorna .T. ou .F. se o conteúdo do campo posicionado no momento não está vazio.

NEGATIVO ()

Sintaxe	<code>Negativo ()</code>
Descrição	Retorna .T. ou .F. se o conteúdo digitado para o campo é negativo.

6.3.3.2 Pictures de formação disponíveis

Com base na documentação disponível no TDN, a linguagem ADVPL e a aplicação ERP Protheus admitem as seguintes pictures:

Dicionário de Dados (SX3) e GET

Conteúdo		Funcionalidade
A	Permite apenas caracteres alfabéticos.	
C	Exibe CR depois de números positivos.	
E	Exibe numérico com o ponto e vírgula invertidos (formato Europeu).	
R	Insere caracteres diferentes dos caracteres de template na exibição, mas não os insere na variável do GET.	
S<n>	Permite rolamento horizontal do texto dentro do GET, <n> é um número inteiro que identifica o tamanho da região.	
X	Exibe DB depois de números negativos.	
Z	Exibe zeros como brancos.	
(Exibe números negativos entre parênteses com os espaços em branco iniciais.	
)	Exibe números negativos entre parênteses sem os espaços em branco iniciais.	
I	Converte caracteres alfabéticos para maiúsculo.	

Conteúdo	Funcionalidade
A	Permite apenas caracteres alfabéticos.
C	Exibe CR depois de números positivos.
E	Exibe numérico com o ponto e vírgula invertidos (formato Europeu).
R	Insere caracteres diferentes dos caracteres de template na exibição, mas não os insere na variável do GET.
S<n>	Permite rolamento horizontal do texto dentro do GET, <n> é um número inteiro que identifica o tamanho da região.
X	Exibe DB depois de números negativos.
Z	Exibe zeros como brancos.

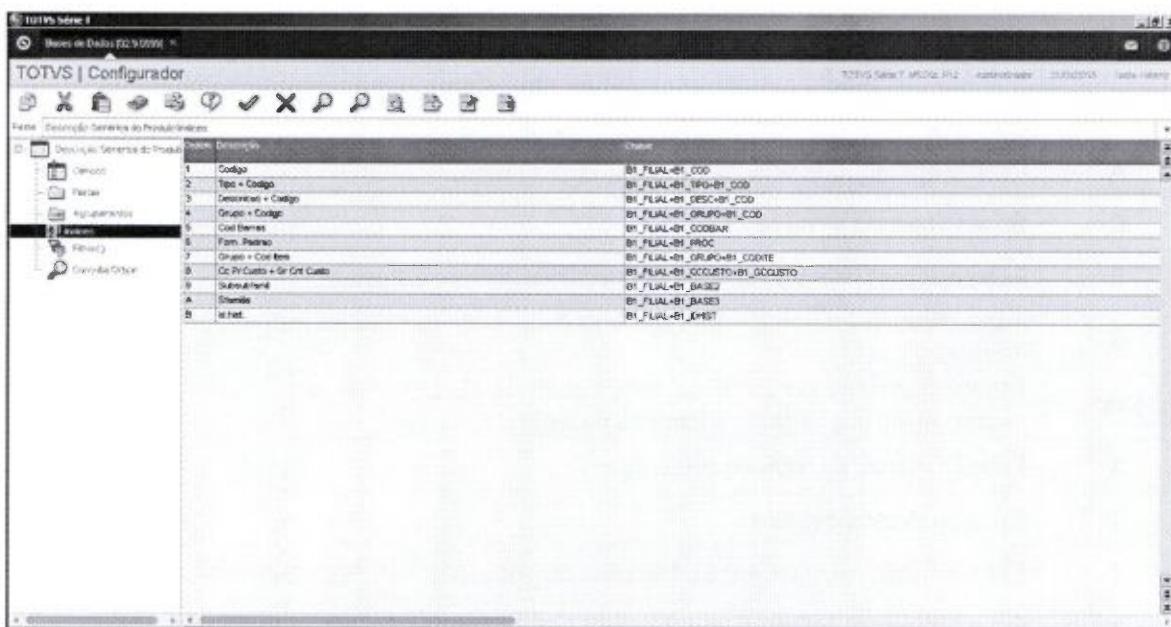
6.4. Índices (SIX)

Os índices da tabela são a organização de campos que definem a ordem de apresentação e pesquisa dos dados do Sistema.

No menu principal, selecione as opções "Base de Dados" + "Dicionário" + "Arquivos";

O Sistema apresenta a janela browser relacionando o dicionário de dados do Sistema, ou seja, todos os arquivos utilizados. Posicione o cursor sobre o arquivo desejado e clique em "Editar" seleciona a opção "Índice" no menu.

O Sistema apresenta na tela os índices vinculados ao arquivo selecionado podendo Incluir, Alterar e Excluir os índices.



Para incluir índices, clique em "Incluir", abrirá uma pequena janela para informar os dados dos índices.

Formação Programação ADVPL



Novo Índice

Gerenciador de...

Outras Ações Cancelar Salvar

Chave:

Nickname:

Descrição:

Desc Espanhol:

Desc Ingles:

Mostra pesq.

Preencha os campos conforme descrição a seguir:

- **Chave:** Informe o campo chave para a criação do índice. No botão ações relacionadas possui a opção "Campos" disponível para consultar os campos do arquivo selecionado.
- **NickName:** Campo responsável para apelidar o índice, sendo boas práticas na criação de novos índices customizados em arquivos padrões.
- **Descrição/Descrição Espanhol/Descrição Ingês:** Informa a descrição do campo selecionado nas três línguas: português, espanhol e inglês.
- **Mostra pesq:** Campo responsável por disponibilizar o índice na consulta do Browser.

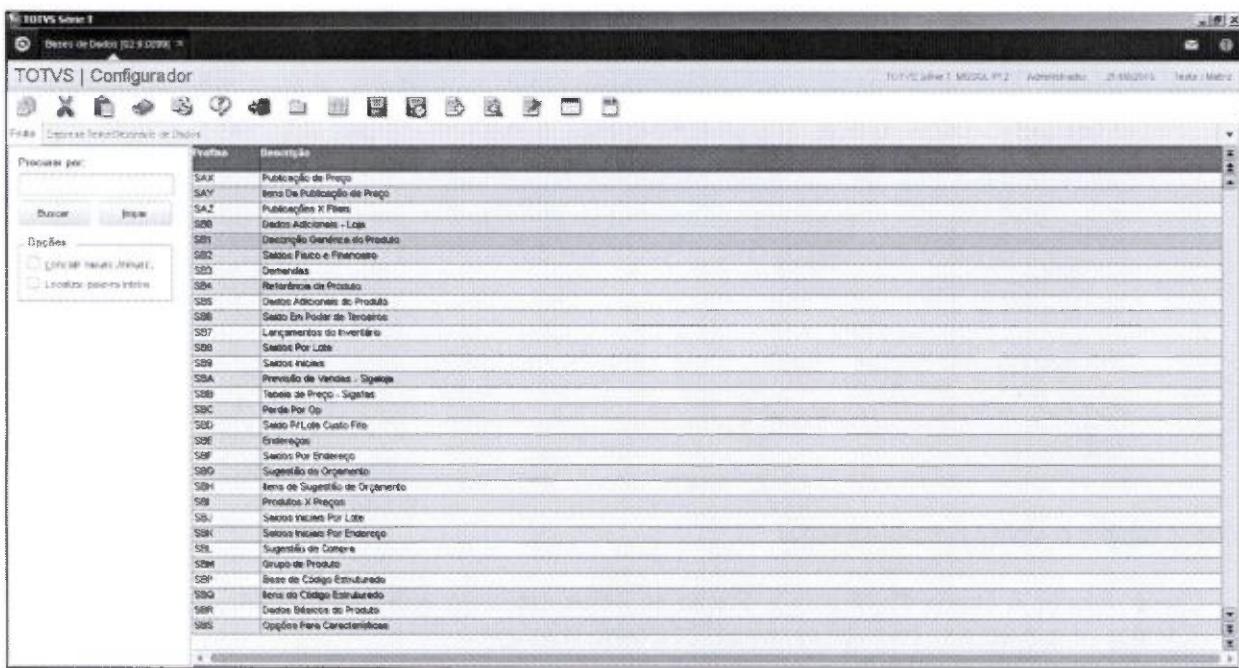
Importante

Campos virtuais não podem ser utilizados como índice, devido à presença de um campo Filial, as chaves devem sempre ser no formato Caracter, assim campos DATA e NUMERO devem utilizar as funções de conversão STOD() e STRZERO() respectivamente.

6.4.1. Atualização dos dicionários de dados

Toda alteração feita no dicionário de dados em suas estruturas será necessário salvar para as alterações serem aplicada em suas respectivas tabelas.

Para aplicar as alterações selecionar o botão "Atualizar Base de Dados".



Após "Atualizar Base de Dados", irá aparecer toda estrutura alterada no dicionário de dados.



As alterações que interferem na estrutura da tabela somente podem ser gravadas com o processamento em modo exclusivo (inclusão de campos, alteração de título, exclusão de campos, etc).

Selecionar o botão avançar, irá parecer o log das operações realizada, selecionar "Finalizar".

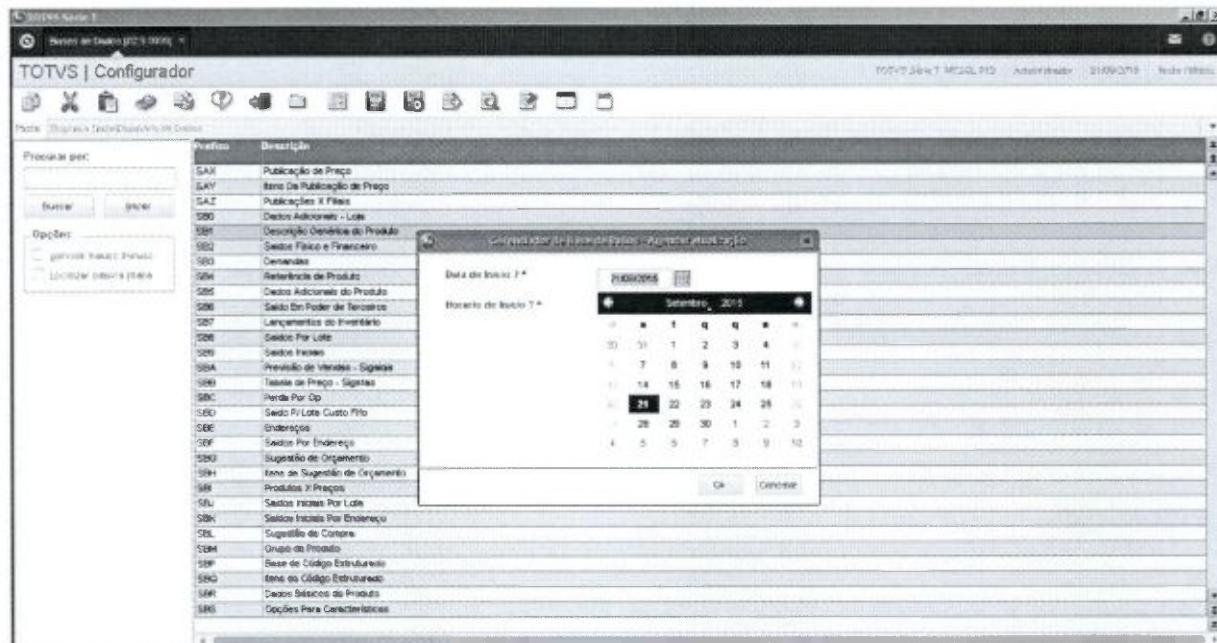
Importante

Quando a opção Base de Dados for acessada e existirem pendências de gravação de alterações, o sistema exibirá uma mensagem de advertência, permitindo que o Administrador restaure as alterações que não puderam ser gravadas.

6.4.2. Atualização agendada dos dicionários de dados

Com o objetivo de facilitar o processo de atualização dos dicionários de dados, o administrador poderá programar o horário da atualização automática,

Selecione as seguintes opções: "Base de Dados" + "Dicionário" + "Arquivos" + "Agendar".



Ressalta-se que o processo de atualização deve ser utilizado em modo exclusivo, ou seja, nenhum usuário poderá utilizar o sistema durante a realização do processo.

No momento agendado se o sistema estiver sendo utilizado, as atualizações automáticas não serão efetuadas e outra data deverá ser agendada. Quando o usuário estiver com o sistema aberto, porém sem processos em uso por inatividade ativar o *time out*, o processo se realizará normalmente.

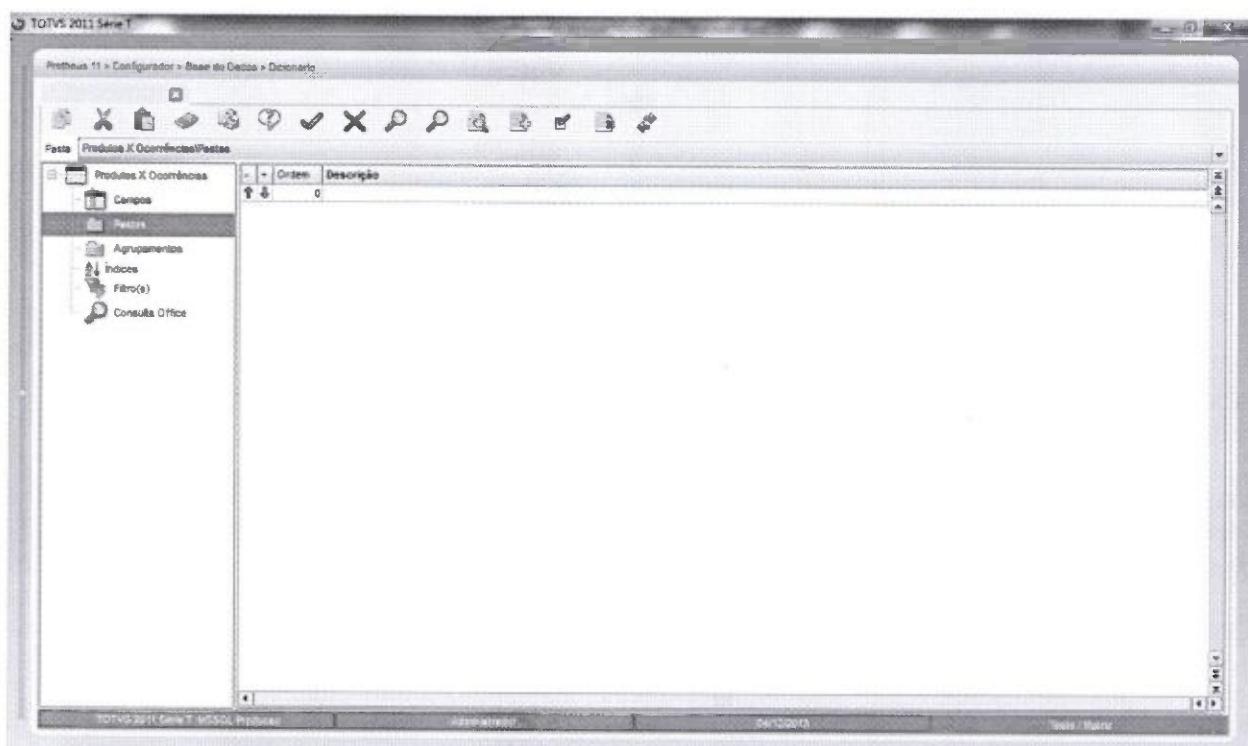
Após preencher os dados, o sistema irá fazer o agendamento das atualizações efetuadas no dicionário de dados.



6.4.3. Pasta (SXA)

No menu principal, selecione as opções "Base de Dados" + "Dicionário" + "Arquivos", o sistema apresenta a janela browser relacionando o dicionário de dados do Sistema, listando todos os arquivos utilizados. Posicione o cursor sobre o arquivo desejado e clique em "Editar" seleciona a opção "Pasta".

Na edição de arquivos, é possível incluir novas pastas, editá-las e alterá-las; excluir, ordenar e mover campos.



Para incluir novas pastas, clique em "Incluir".

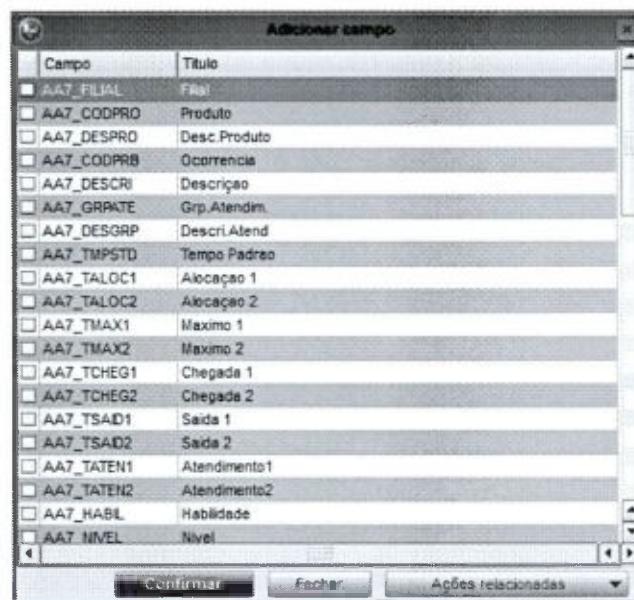
O Sistema apresenta a tela de inclusão dos dados da pasta:

Formação Programação ADVPL



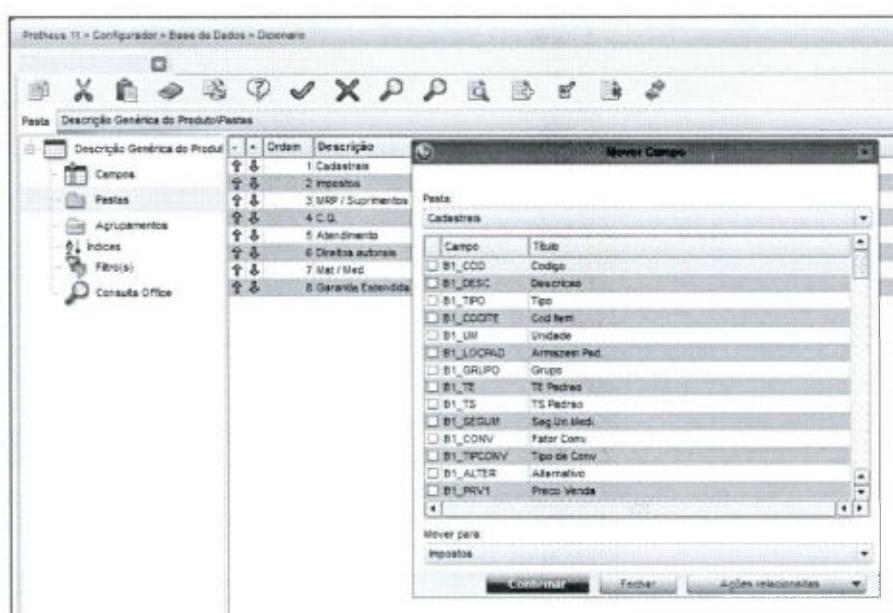
- **Descrição / Desc. Espan. / Desc. Inglês:** Informe a descrição da pasta criada.

Após preencher os dados, clique na pasta "Campos" e em seguida clique em "Adicionar". O Sistema apresenta a tela com a relação dos campos do arquivo.



Selecione os campos desejados para constarem na pasta em cadastro, marcando-os com um duplo clique do mouse, até que seja apresentado um "x" (somente campos que não estejam vinculados a nenhuma pasta). Confirme a definição dos campos e confirme a criação da pasta. É possível alterar a estrutura mover os campos de suas respectivas pastas (origem) para outras pastas (destino):

Selecionando o botão mover campos:



Selecione a pasta origem na área "Pasta"

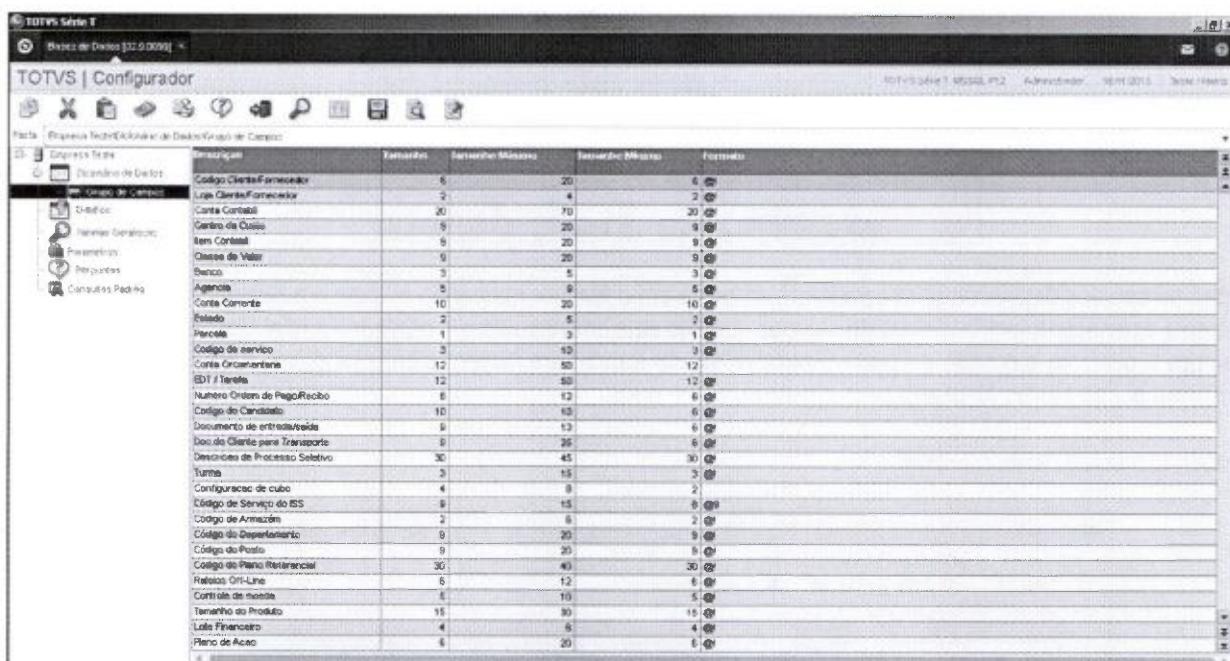
Os campos pertencentes a esta pasta será relacionada na tela, marque os campos desejados destacados com um "X" e selecione a pasta destino na área "Mover Para", após fazer as alterações confirme.

6.4.4. Grupos de Campos (SXG)

A Linha de Produto Microsiga Protheus contém em seu dicionário várias tabelas que são compartilhadas entre diversos ambientes, tais tabelas contêm campos cujos tipos e tamanhos devem ser coerentes entre ambientes e rotinas. Quando houver a necessidade de alterar o tamanho de um desses campos, o procedimento comum seria bastante trabalhoso e demandaria um tempo inviável dependendo do tamanho da base. Para simplificar esse processo, a Linha de Produto Microsiga Protheus tem o conceito de grupos de campos, onde é possível alterar os campos de diversas tabelas relacionadas de uma única vez.

Para alterar o grupo de campos:

1. No menu **Base de dados**, selecione **Dicionário e Base de dados**;
2. Na opção **Dicionário** à esquerda, clique em (+) para expandir a opção **Grupo de campos**;



3. Escolha o grupo que deseja alterar, altere o tamanho e confirme.
4. Clique no botão **Atualizar** para que o configurador atualize todo o dicionário e também as tabelas do banco de dados.

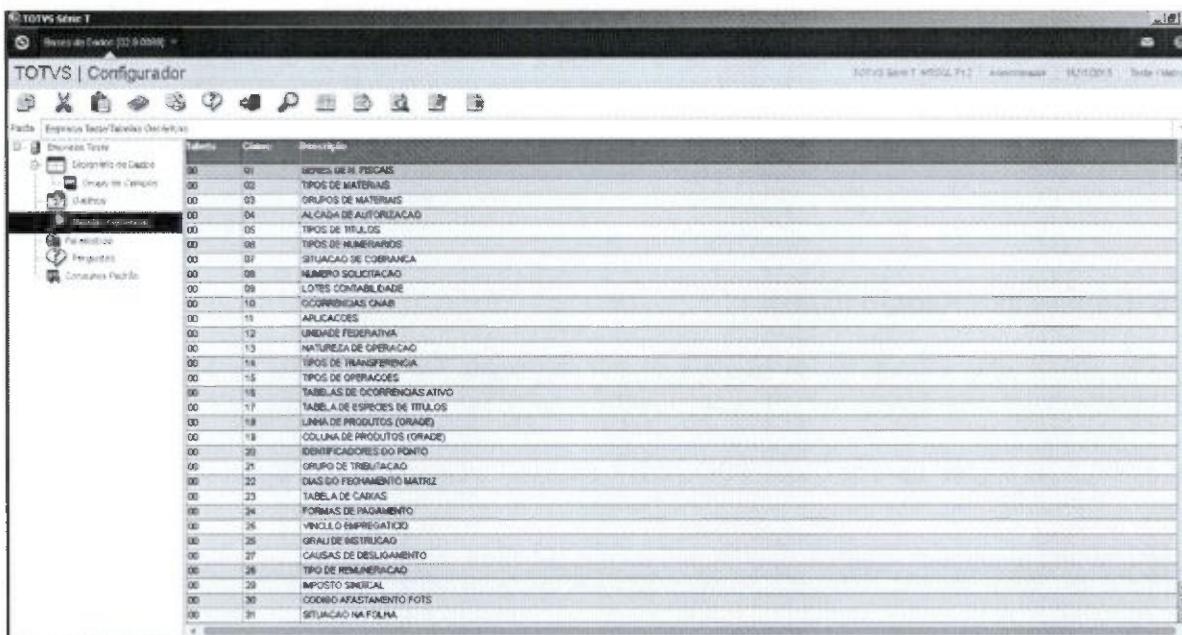
Importante

As opções de grupos de campos envolvem apenas campos caracteres. A alteração de campos numéricos podem envolver rotinas de cálculos que exigem que campos relacionados tenham o mesmo tamanho. Caso tenha necessidade de alterar casas decimais, por exemplo, abra um chamado na TOTVS para verificar essa possibilidade e quais as tabelas envolvidas.

6.4.5. Tabelas Genéricas (SX5)

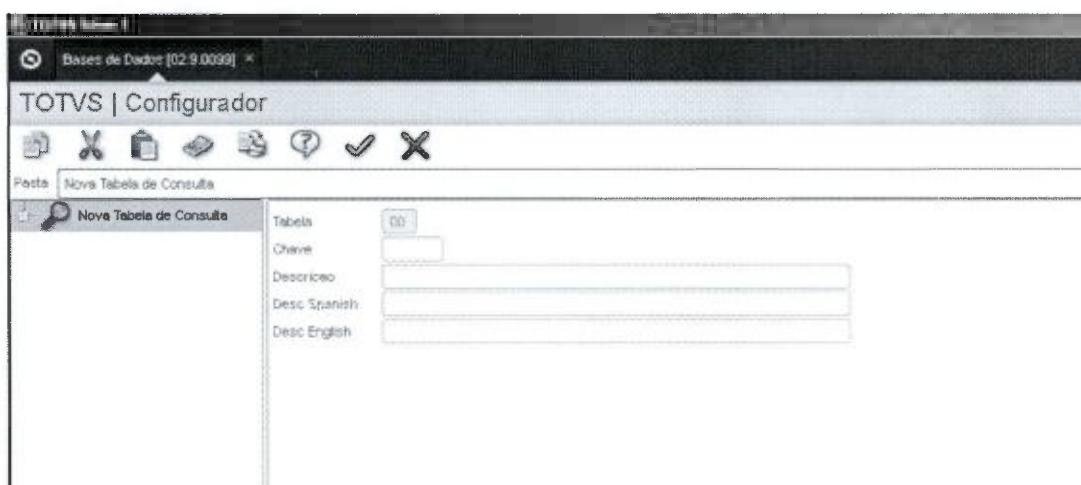
Há situações em que a criação de uma nova tabela com índices e consumo de um Alias não é a solução mais inteligente. Quando os dados forem do tipo Código-Descrição e as informações forem estáticas a Linha de Produto Microsiga Protheus conta com uma tabela destinada especificamente pra isso.

No menu principal, selecione as opções **Ambiente + Cadastros + Tabelas**. O Sistema apresenta uma janela relacionando todas as tabelas utilizadas pelo Sistema.



Clique em Incluir para cadastrar uma nova tabela. O Sistema apresenta a tela para cadastramento da tabela e dos respectivos itens;

Com a pasta Nova Tabela de Consulta selecionada, preencha os campos correspondentes ao cadastro, conforme descrição a seguir:



Preencha os campos conforme descrição a seguir:

- Tabela:** Este campo é preenchido automaticamente pelo Sistema com "00".
- Chave:** Informe o nome ou valor que identificará a tabela. Utilize 2 (até 06) caracteres.
- Descrição:** Informe a descrição da tabela em cadastro, identificando seu conteúdo.

Clique na pasta **Itens** para cadastrar os itens da tabela.

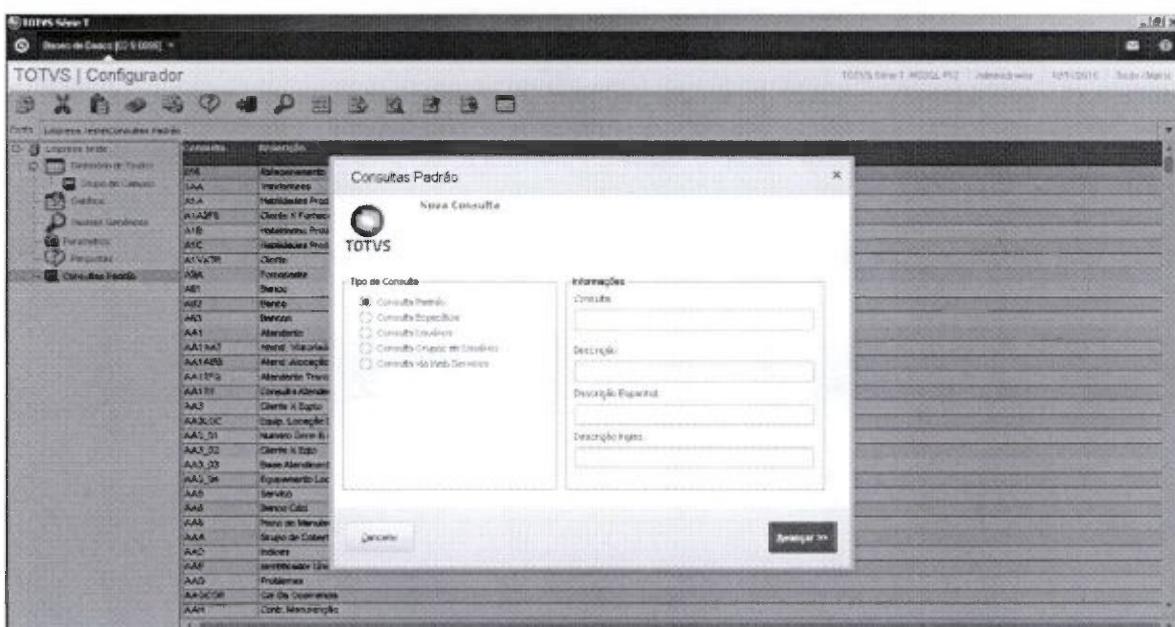
Importante

Utilize a faixa de segurança "Z*", para não correr o risco de a tabela ser sobreescrita em atualizações. A tabela recém-criada deve ser associada a um dos campos do dicionário de dados para que seja utilizada. Para tanto, é preciso informar a chave na opção Consulta padrão do campo associado.

6.4.6. Consultas - Padrão (SXB)

Essa opção permite, durante a edição de um campo, consultar uma tabela associada a este campo, pesquisar está utilizando índices e visualizando apenas os campos de interesse. A opção escolhida pode trazer um ou mais retornos.

Selecione as seguintes opções: Base de Dados + Dicionário + Base de Dados + Consulta Padrão, selecionar o botão "Incluir".



Preencha os campos conforme descrição a seguir:

- Consulta:** Nome da consulta
- Descrição / Descrição Espanhol / Descrição Inglês:** Descrição da Consulta

Formação Programação ADVPL

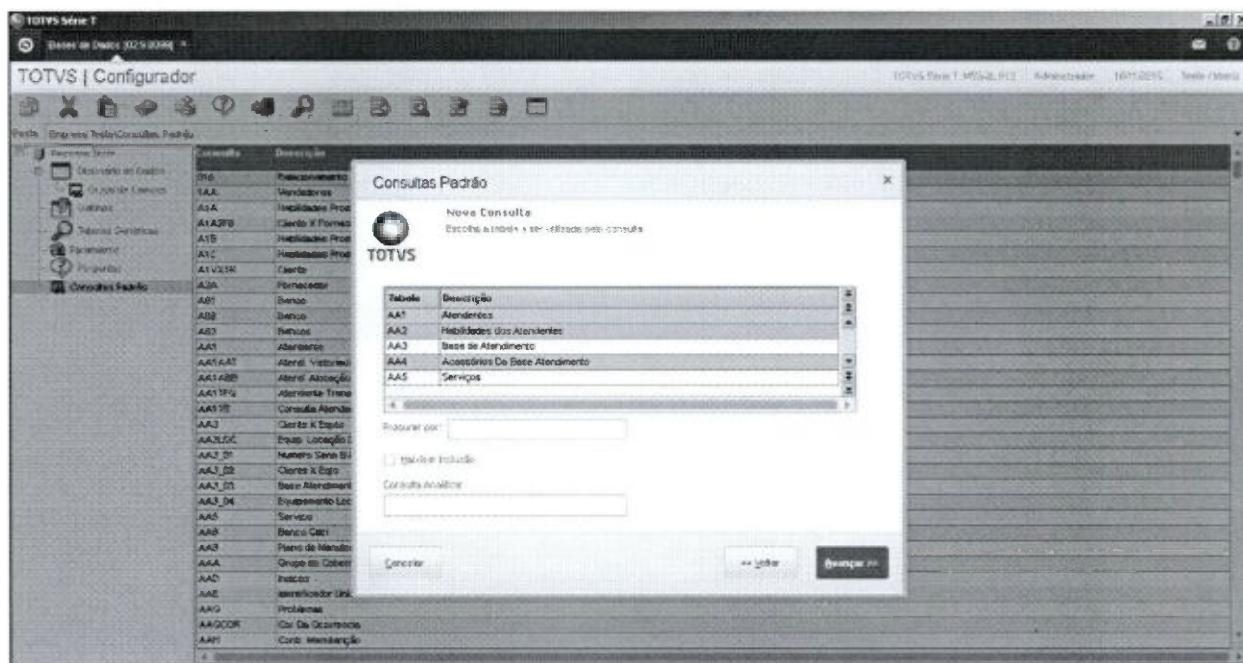


As consultas padrão podem ser classificados nos seguintes tipos:

- **Consulta Padrão:** Estrutura de criação com arquivos
- **Consulta Específica:** Estrutura de criação com função ADVPL.
- **Consulta Usuários:** A consulta usuários permite que o campo de uma tabela realize consultas nos usuários cadastrados o retorno seja vinculado a usuários definidos.
- **Consulta a Grupo de usuário:** A consulta a Grupo de usuários permite que o campo de uma tabela realize consultas nos Grupos cadastrados os retornos sejam vinculados a Grupo definidos.
- **Consulta via Web Service:** O sistema, também, disponibiliza consulta padrão para acessar tabelas que estejam em outro servidor e ambiente **Microsiga Protheus**.

Selecionar a opção "Consulta Padrão" preencher as informações e "Avançar".

O Sistema apresenta a tela para seleção do Arquivo a ser utilizada na consulta.



Preencha os campos conforme descrição a seguir:

- **Procurar por:** Arquivo que vai aparecer os dados para consulta.
- **Habilitar Inclusão:** Habilita a opção de incluir novos registros pela consulta.
- **Consulta analítica:** Se desejar utilizar um programa próprio (*User function*) para uma consulta padrão, informe o seu nome na caixa Consulta Analítica.

Importante

Utilize preferencialmente uma nomenclatura Z????? para personalizações. A utilização da opção "Permitir inclusão" não consome licenças no uso da consulta padrão. Para que a consulta seja utilizada ela deve ser especificada num campo que vai utilizar essa associação.

Pesquise pela tabela origem dos dados que pretende consultar, após o preenchimento selecionar o botão "Avançar". O Sistema apresenta nova tela, onde devem ser informados os Índices utilizados pela consulta, e suas respectivas colunas:



Na opção dos Índices informados os dados referentes aos índices, clique em “Adicionar Índice”.

O Sistema apresenta a tela para montagem de filtros e retornos.



Formação Programação ADVPL



Preencha os campos conforme descrição a seguir:

- **Ordem:** Os índices do Arquivo corrente.

Para cada índice que vamos exibir, devemos definir os campos que serão mostrados para ele. Na opção das colunas informados os dados referentes a estrutura dos dados na Consulta Padrão, clique em “Adicionar Colunas”.

Para facilitar o preenchimento selecionar o botão “Consultar Campos”.

Os preenchimentos são individuais selecionar um campo por vez, após os preenchimentos selecionar o botão “Avançar”. O Sistema apresenta a tela para montagem de filtros e retornos.



Preencha os campos conforme descrição a seguir:

- **Filtro:** No registro de filtro deverá conter uma expressão ADVPL que retorna um valor lógico
- **Retorno:** Preenchidos os campos de retorno, quais serão os valores retornados pela consulta, quando o usuário selecionar um registro.

Selecionar o botão “Adicionar Retorno”.



O Sistema apresenta uma pequena janela para informar a expressão desejada. Na no campo expressão selecionar o botão "Consultar campos" nessa opção irá trazer todos os campos da tabela do arquivo corrente.

Selecione o campo para expressão e clique em Ok os preenchimentos são individuais preencher um por vez.

Preenchidos os campos, clique em Finalizar para concluir a inclusão da consulta padrão.

Posicionado na consulta criada clique no botão "Preview", localizado na barra de ferramentas, será apresentada a tela com a Consulta criada, para verificação da interface com o usuário.

Importante

Vale lembrar que uma consulta irá preenchendo os campos subsequentes ao campo que estamos numa digitação, depende de quantos campos a nossa consulta retorna. Portanto se nossa consulta retorna o código do cliente e sua loja, a tela que estamos usando tem que estar preparada para receber o código e a loja do cliente em sequência.

6.4.7. Parâmetros (Sx6)

Um parâmetro é uma variável que atua como elemento-chave na execução de um determinado processamento. De acordo com o seu conteúdo, será possível modificar vários processos e, consequentemente obter diferentes resultados. Para cadastrar novos "Parâmetros", selecione as seguintes opções: "Ambiente" + "Cadastros" + "Parâmetros".

Formação Programação ADVPL



Id	Nome	Descrição	
FM_GCTCOT	Variável para coleção		
MC_FLCTD4	Código do provedor para SUS		
MV_10902	Define os meses em que haverá referência do IR ao informar em UF que não terão consideração para	Isqueles por conta no relatório de Arrecadação	
MV_13891UF	Informar em UF que não terão consideração para	geração do CNPJ referente as operações do Convênio	13806 na apuração de ICMS
MV_13892	Define inicialização de 1 percento do título gerado	Ex: A -> para sequência arte	1 - para sequência numerada
MV_13893AT	Campo ou dado a ser gravado na metade do título	Quando o mesmo for gerado automaticamente pelo módulo de Faturamento.	dado de faturamento.
MV_13893PF	Campo ou dado a ser gravado no preâmbulo do título	Quando o mesmo for gerado automaticamente pelo módulo de COMPRA.	dado de COMPRA.
MV_13894AT	Campo ou dado a ser gravado na metade do título	Quando o mesmo for gerado automaticamente pelo módulo de Compras.	
MV_13894PF	Campo ou dado a ser gravado no preâmbulo do título		
MV_20904C	Não Forçar o Refresh na Proximidade para que os operações utilizadas sejam mais otimizadas		
MV_20904CNA	Informação do preço médio das saídas dos fornecedores associados. DF_BEDIAS		
MV_20905P	Campo ou dado a ser gravado no preâmbulo do título	Quando o mesmo for gerado automaticamente pelo módulo de Atualização de contas (E_MBTM0)	
MV_2CLFOR	1 - indica que existe o cadastro de cliente e fornecedor com o mesmo CNPJ para o SNC0	atendendo o código do participante no site gerado	
MV_24703M	Indica se deve ser gerado o registro de inventário	R04-Situação Em / De posse de Terceiros	Se: I, ou F.
MV_26020TE	Campo da tabela SB que contém a data de saída do veículo	criado para o processar de Rmt. ICMS ST MG	
MV_26998S	Campo da tabela SF4 (TBS) que define se é CFOPa que indicam saídas por venda, rodovia ou estrada	operação deve ser considerada para gerar o recréamento para o cálculo do restituto de MG.	
MV_26999D	CFOPa que indicam saídas por venda, rodovia ou estrada	do Tipo BRS/TE3 independentemente se existe saldo do Registro de Apuração de ISS	
MV_26999ES	Indica se deve ser apresentado os registros de saída de veículos / SP	deve ser gravado no registro Código de Município.	
MV_21404H	Indica se o MATR914, rubro de impressão de identificação e tempo do cadastro do cliente que	deve ser gravado no registro Código de Município	
MV_24020U	Identifica o campo do cadastro do fornecedor que	deve ser gravado no registro Código de Município	
MV_24020F	Identifica o campo do cadastro do fornecedor que	de Vendas / SP	
MV_271A0R	Arquivo de texto de saída do Registro de ISSQN		
MV_37704C	Arquivo de texto de encerramento do registro de bases SB1 (Descrição gênerica de produtos) que		
MV_A010FAC	Este parâmetro tem como objetivo informar campos de bases SA2 (Catálogo de fornecedores) que não são considerados no pedido.		
MV_A020FAC	Este parâmetro tem como objetivo informar campos de bases SA1 (Catálogo de clientes) que não são considerados no pedido.		
MV_A030FAC	Este parâmetro tem como objetivo informar campos de bases SF4 (Tipos de entrada e saída) que não são considerados no pedido.		
MV_A050FAC	Este parâmetro tem como objetivo informar campos de bases SPM (TBS Histórico) que não são considerados no pedido.		
MV_A175VLD	Ativa e altera o dia de data de validade do Lote e/ou Substitui no nome de Detentor do CGI		
MV_A11COMUN	Identifica o código do município do cliente.		
MV_A11DEF	Informa campo na tabela SA1 que contém o código do município do cliente, conforme a tabela da SEFA-PA.		

Clique no botão "Incluir", e informe os dados do parâmetro:



Importante

Antes de alterar algum Parâmetro, se campo X6_FILIAL estiver sem conteúdo, tal mudança influenciará em toda a Empresa e não apenas naquela Filial. Lembre-se sempre antes de alterar algum Parâmetro e se o campo X6_FILIAL estiver sem conteúdo, que tal mudança irá influenciar em toda a Empresa e não apenas naquela Filial. Este parâmetro que estamos criando, fará diferença apenas para a empresa SX6??0 e a filial "01".

6.4.8. Gatilhos (SX7)

O Gatilho permite estabelecer uma atualização dinâmica dos dados através da definição de critérios, isto é, a partir da digitação de um dado é possível atualizar automaticamente outros campos. Para criar Gatilhos, Selecione as seguintes opções: "Base de dados" + "Dicionário" + "Gatilhos", irá apresentar alguns gatilhos já cadastrados utilizados pelo próprio Sistema, selecionar o botão incluir.



Preencha os campos conforme descrição a seguir:

- **Campo:** Nome do campo que ao ser alterado inicia o processo de atualização.
- **Sequência:** Número sequencial gerado pelo sistema para cada gatilho.
- **Cnt. Domínio:** Nome do campo que deverá ser atualizado automaticamente.
- **Tipo:**
 - P - Primário para atualizações visuais e externas do mesmo arquivo.
 - E - Estrangeiro para atualizações de dados em outros arquivos.
 - X - Posicionamento para posicionar o arquivo mencionado no Alias sem efetuar nenhuma atualização. Utilizado para casos em que o usuário deseja estabelecer um relacionamento entre os arquivos.
- **Regra:** Expressão em linguagem AdvPL a ser transportada para o contra-domínio.
- **Posiciona:** Selecione a opção "Sim" para movimentar o ponteiro em outro arquivo com base na expressão definida em PROCURA. (SEEK), ou "Não", caso contrário.
- **Alias:** Aliás do arquivo, as três letras iniciais do arquivo cujo ponteiro deve ser movimentado.
- **Ordem:** Número da chave de índice a ser utilizada para movimentação do ponteiro.
- **Chave:** Expressão em linguagem que determina o posicionamento do ponteiro (SEEK).

- **Condição:** Informe a condição, Execblocks, etc., que irão determinar quando o gatilho deve ser executado.

Importante

Caso esteja trabalhando com o tipo "P" e fizer referências a campos do próprio arquivo, estes devem ser precedidos da notação "M->", que representa a variável de memória. Exemplo: M->C6_QTDVEN.

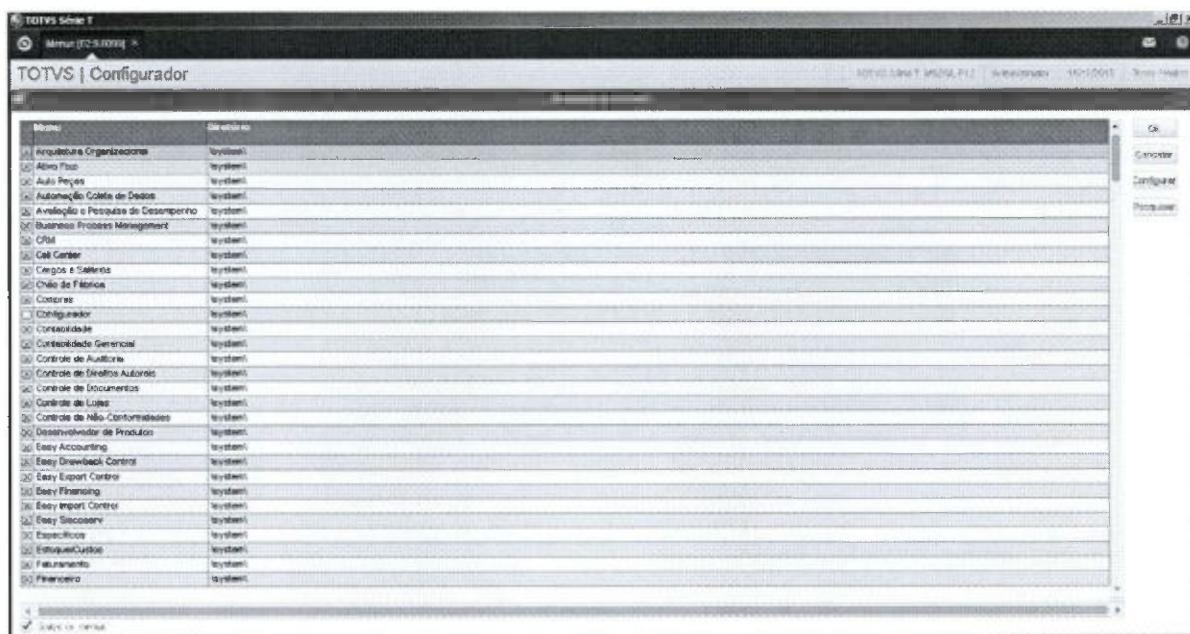
Exemplo: Ao criar o campo Saldo (B1_SALDO) no Arquivo de Produtos, pode-se definir um gatilho para atualizar automaticamente este campo, quando houver a inclusão de uma previsão de venda para o produto (C4_PRODUTO). A regra deve determinar que o Saldo do Produto seja a soma de sua quantidade atual e a quantidade da Previsão de Venda.

6.5. Menus e Senhas

Aprenderemos a configurar novos menus a partir de arquivos padrões já existentes. Também iremos verificar como devem ser criados os usuários, configurar seus acessos e restrições, grupos e as relações entre eles. Os Menus do sistema podem ter suas opções reformuladas de maneira que cada usuário possa ter um menu de acesso próprio, de acordo com o tipo de trabalho por ele desenvolvido. Neste programa, deve ser definido um menu para cada usuário e na configuração de senhas, associa-se o menu à senha do usuário que determinará quais ambientes, movimentações, empresas e filiais ele poderá acessar.

Para criar Menus para os Usuários:

1. Selecione as seguintes opções: Ambiente + Cadastros + Menus;



São selecionados todos os menus, desmarque a opção **Todos os Menus** e selecione os menus para que seja usado como base, ou nenhum para criar um menu limpo, selecione o botão "OK".

Formação Programação ADVPL

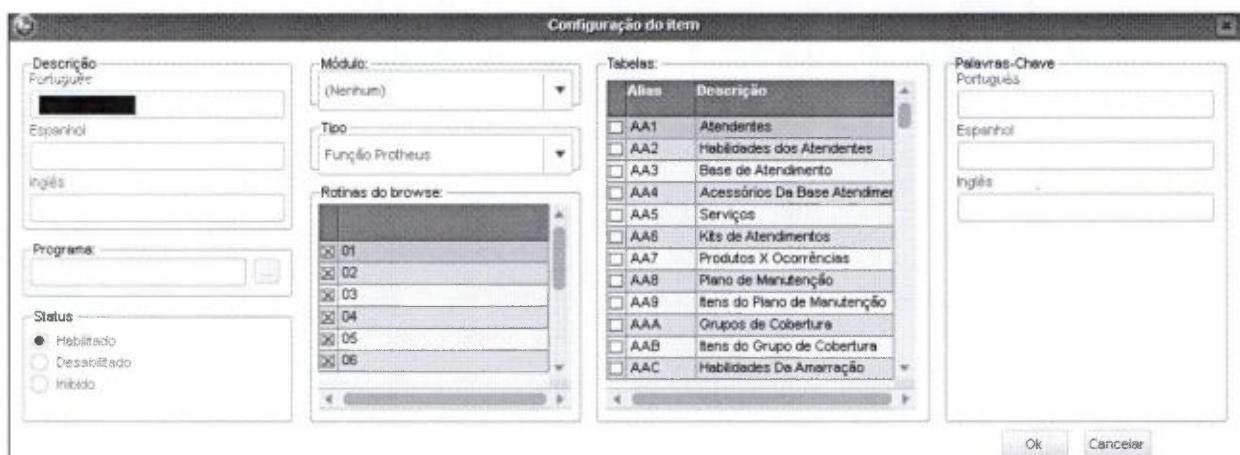


Clique em Adicionar para copiar todas as opções do menu padrão ao novo menu.



Opções dos Botões conforme descrição a seguir:

- **Remover:** Para remover funções ou pasta da estrutura do menu;
- **Novo Grupo:** Posicione o cursor na pasta principal que deve receber a nova estrutura;
- **Novo Item:** Para incluir novas chamadas de rotinas no menu, posicione o cursor na pasta do grupo que deverá receber a opção;



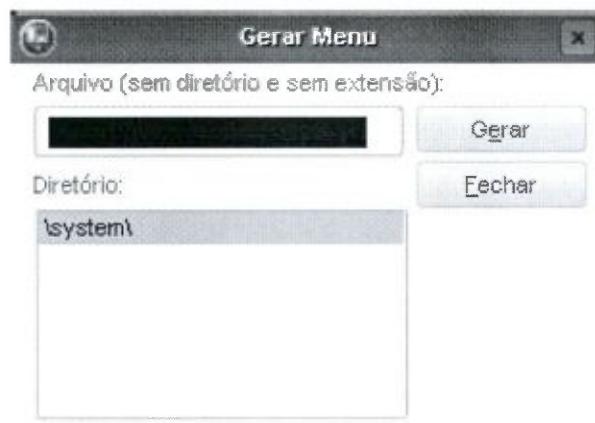
Preencha os campos conforme descrição a seguir:

- **Descrição:** Nome da nova opção no menu.
- **Programa:** Nome do programa que corresponde a essa opção. Esse programa deve constar no executável ou estar precedido por "#" caso seja em linguagem AdvPL.

Formação Programação ADVPL



- **Status:**
Habilitado: exibe a opção no menu e permite sua seleção.
Desabilitado: desativa a opção posicionada e não a exibe no menu.
Inibido: apresenta a opção no menu, porém não permite que ela seja selecionada.
- **Modulo:** Selecione o módulo cujo tratamento deve ser aplicado à opção.
- **Tipo:** Indica o tipo da nova opção, que pode ser: Função Protheus, Relatório SIGARPM, Função de Usuário, Função Template, Relatório Crystal ou Cons. Genérica Relacional.
- **Rotinas do Browse:** Define o acesso às opções dos menus. O Sistema apresenta 10 opções equivalentes às funções de cada programa na sequência em que aparecem. Exemplo: em um programa de atualização, são possíveis as opções 1-Pesquisar, 2-Visualizar, 3-Incluir, 4-Alterar e 5-Excluir. Clique sobre o número correspondente à opção que deve ter seu acesso desabilitado. Em seguida ela fica desmarcada. Após a configuração, quando o usuário acessar o programa correspondente e selecionar esta opção é exibido uma mensagem sobre a impossibilidade de acesso.
- **Tabelas:** Selecionar as tabelas que está relacionada no programa.
- **Palavras - Chave:** Define mais uma opção de nome para facilitar no localizar do Modulo; Selecionar o botão OK para confirmar a criação do novo acesso.
- **Detalhes:** Para alterar a opção deseja selecionar o botão detalhes é possível desativar ou ocultar uma opção do menu sem remover.
- **Mover p/Cima ou Mover p/Baixo:** Para alterar a ordem de exibição.
- **Gerar:** Para poder salvar as alterações ou fazer a criação de novos menus é necessário selecionar a opção "Gerar".



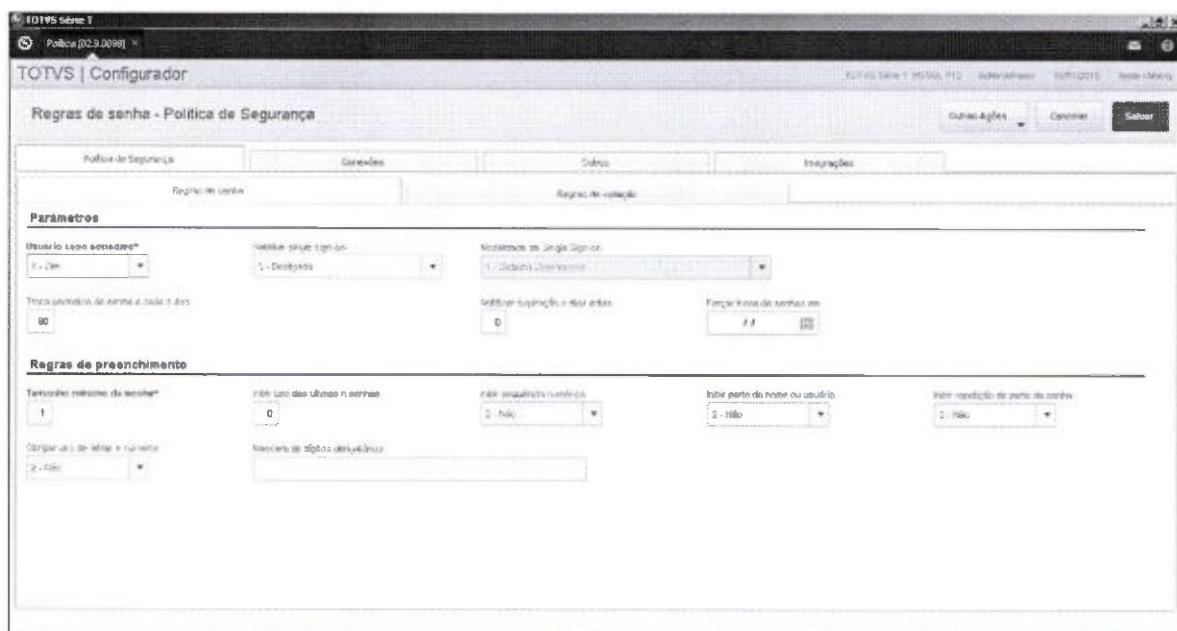
Preencha os campos conforme descrição a seguir:

- **Arquivo:** Informar o nome do menu, para criar um menu novo informar um nome que desejar, para alterar as opções de um menu existente informar o nome do menu

Verifique que na tela onde se encontram todos os Menus, o “Nome do Menu Criado.xnu” acaba de ser criado como última opção, pronto para ser utilizado.

6.6. Política de Segurança

A **Política de Segurança** define as regras de acesso do sistema. Essas regras devem ser definidas antes da criação de usuários e senhas. Para ativar a Política de segurança, selecione as seguintes opções: “Usuário” + “Senhas” + “Política”, irá aparecer às regras pré-definidas, o sistema apresenta a tela subdividida em pastas.



Na pasta “Política de Segurança / Regras de Senhas” preencher os campos, conforme as descrições a seguir:

Parâmetros:

- **Usuário Case Sensitive:** 1-SIM / 2-NÃO
Informe se a pesquisa pelo código do usuário (LogOn) irá considerar maiúsculas e minúsculas
- **Habilitar Single Sign-On:** 1-Obrigatório / 2-Opcional / 3-Desligado
Quando a opção do Single Sign-On está habilitada o sistema vincula o usuário logado no sistema de acordo com a “Modalidade de Single Sign-On”
- **Modalidade de Single Sign-On:** 1 - Sistema Operacional / 2 - Fluiq Identity / 3- Active Directory
- **Troca periódica da senha a cada N dias:** Informe a periodicidade estabelecida na política de segurança em que o usuário deve trocar sua senha.
- **Notificar expiração N antes:** Informe a quantidade de dias que antecedem a data de troca de senha em que o usuário deve ser notificado.
- **Forçar troca da senha em:** Informe a data em que a senha dos usuários deverá ser trocada.

Formação Programação ADVPL



Regras de preenchimento:

- **Tamanho mínimo da senha:** Informe o número de dígitos mínimos definidos pela política de segurança na definição da senha do usuário.
- **Inibir uso das últimas N Senhas:** Informe a quantidade das últimas senhas definidas na política de segurança que deverão ser armazenadas pelo sistema e não poderão ser reutilizadas pelo usuário em caso de troca.
- **Inibir sequência numérica:** 1=Sim / 2=Não
Informe se a política de segurança deve restringir a utilização de sequências numéricas
- **Inibir parte do nome ou usuário:** 1=Sim / 2=Não
Informe se a política de segurança deve restringir o uso de parte do nome ou código de acesso (login).
- **Inibir repetição de parte da senha:** 1=Sim / 2=Não
Informe se a política de segurança irá restringir o uso de parte da última senha cadastrada.
- **Obrigar uso de letras e números:** 1=Sim / 2=Não
Informe se a política de segurança obriga o uso de letras e números na senha.
- **Máscara de dígitos obrigatórios:** Informe a quantidade de vezes que um caracter deve ser utilizado na senha, desenvolvendo uma máscara de como deve ser preenchida a senha do usuário
- **Na pasta "Regras de violação" e preencha os campos, conforme descrição a seguir:**

Regras para Bloqueio:

- **Bloquear Usuário após N Tentativas:** Informe a quantidade de tentativas de autenticação, definidas na política de segurança, que se excedidas irão bloquear o acesso ao sistema.

- **Intervalo entre tentativas (Minutos):** Informe o intervalo em minutos, definidos na política de segurança, que o sistema deve aguardar para considerar como uma tentativa de acesso.
- **Desbloquear Usuário após n minutos:** Informe o intervalo de tempo, em minutos, que o sistema deve considerar para desbloquear um usuário, automaticamente, após ter seu acesso bloqueado por erro de autenticação.

Parâmetros

- **Notificar Administrador sobre Bloqueio:** 1=Sim / 2=Não
Informe se o Administrador do sistema deve ser notificado sobre o bloqueio automático de usuários no sistema por erros de autenticação.
- **Notificar sobre tentativa de violação:** 1=Sim / 2=Não
Informe se o Administrador deve ser notificado quando um usuário tiver um erro de autenticação.
- **Permitir envio de senha por e-mail:** 1=Sim / 2=Não
É necessário que o usuário tenha um e-mail cadastrado no seu cadastro.
Com a configuração feita, o usuário digita o login e clique em "Esqueceu sua senha?", então aparece uma mensagem de confirmação, e uma nova senha é enviada ao e-mail do usuário.



Seja bem-vindo,

Identifique-se abaixo para usar a linha de produtos
Microsiga Protheus série T.

Usuário

Senha

[Esqueceu sua senha?](#)

Entrar

Múltiplos Terminais

- **Inibir acesso a mais de um terminal:** 1=Sim / 2=Não
Informe se a política de segurança permite ou não o acesso de um mesmo usuário a mais de um terminal do sistema.

- **Bloquear usuário ao acessar outro terminal:** 1=Sim / 2=Não

Informe se a política de segurança do sistema deve bloquear um usuário caso ele tente acessar mais de um terminal ao mesmo tempo.

Na pasta "Conexão" possui as configurações da conta de Proxy e e-mail para notificações das políticas excedidas; Exemplos: Notificar Administrador sobre Bloqueio, Notificar sobre tentativa de violação e Permitir envio de senha por e-mail.

Na pasta "Outros \ Genéricos" preencher os campos, conforme descrição a seguir:



Genérico

- **Habilitar consulta genérica "F3" ?:** 1=Sim / 2=Não

Informe se a tecla F3 estará disponível para uso pelos usuários do sistema.

- **Habilitar Log de erros por e-mail:** 1=Sim / 2=Não

Informe se o Administrador deve ser comunicado se houver algumas não conformidades nos processos do sistema.

- **Habilitar Arquivo diferencial:** 1=Sim / 2=Não

Campo foi criado para dar possibilidade de ligar ou desligar a opção de geração de arquivo diferencial do dicionário padrão.

Data base do sistema

- **Configurar dias de database:** 1=Sim / 2=Não

Informe se o usuário poderá alterar a data base do sistema.

- **Retroceder:** Informe a quantidade de dias em que o usuário poderá retroceder a database do sistema.

- **Avançar:** Informe a quantidade de dias em que o usuário poderá avançar a database do sistema.

Totvs Up

- **Usuário TotvsUp:** Informe o código do usuário para acesso ao TOTVS UP (Portal de relacionamento).
- **Senha TotvsUp:** Informe a senha do usuário para acesso ao TOTVS UP (Portal de relacionamento).

Vínculo funcional

- **Marque as opções do vínculo a serem bloqueadas:** Informe os critérios de restrição de acesso vinculados à folha de pagamento. Esta opção somente pode ser habilitada após a implantação da folha de pagamento (SIGAGPE).

Regra de auditoria

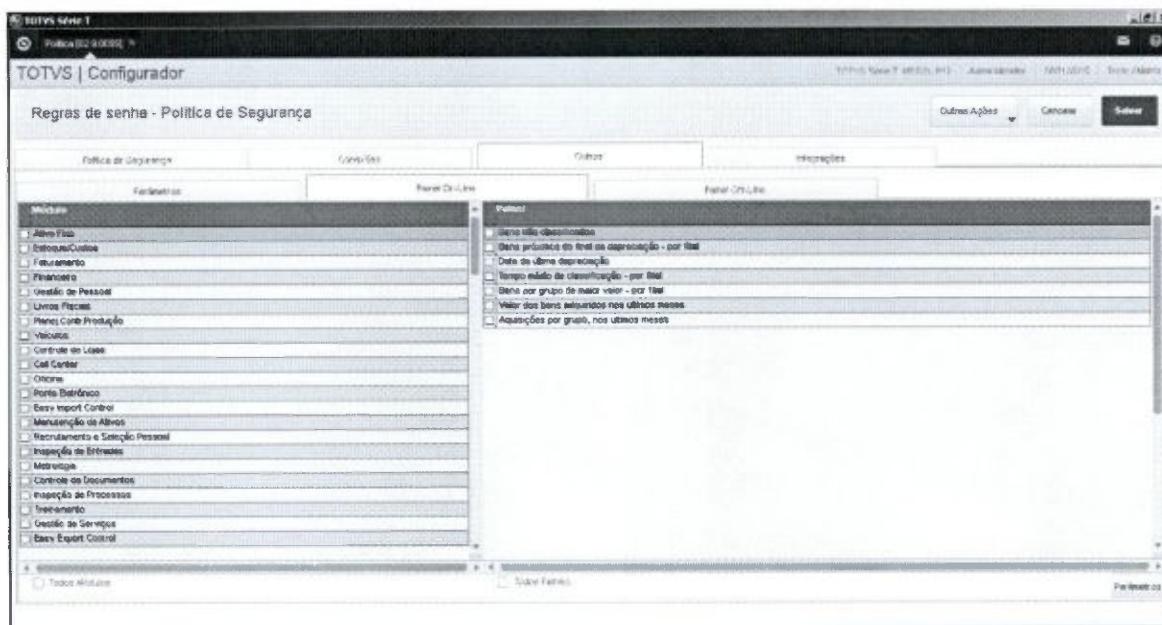
- **Auditar atualizações no dicionário de dados:** 1=Sim / 2=Não
Informe se o sistema deve auditar as atualizações no dicionário de dados.
- **Auditar atualizações no cadastro de usuário:** 1=Sim / 2=Não
Informe se o sistema deve auditar atualizações no cadastro de usuário.
- **Auditar autenticação/acesso:** 1=Sim / 2=Não
Informe se o sistema deve auditar as autenticações de acesso.

Recursos do sistema

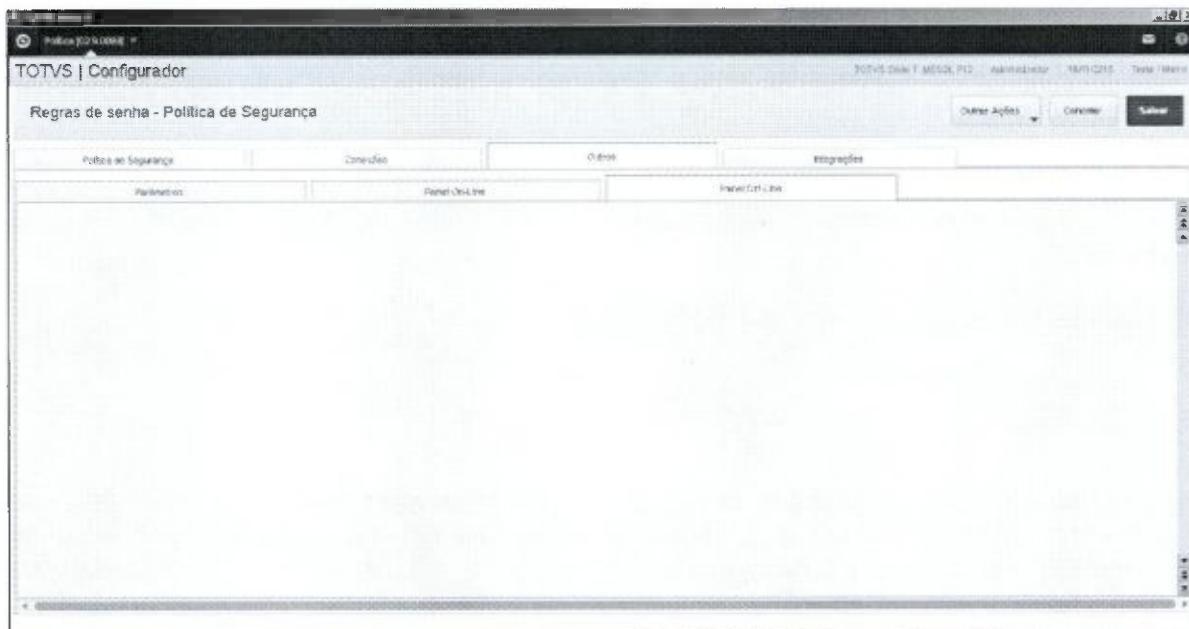
- **Menu Funcional:** Informe se o usuário pode acessar o menu funcional
- **Painel Online:** Informe se o usuário pode acessar o Painel Online disponível
- **Browser de Internet:** Informe se o usuário pode acessar o Browser de Internet
- **Detalhes do Browser:** Informe se o usuário pode acessar os Detalhes do Browser
- **Painel Transparente:** Informe se o usuário pode acessar o Painel Transparente

Na pasta "Outros \ Painel On-Line" lista o painel disponível no Protheus.

Formação Programação ADVPL



Na pasta "Outros \ Painel Off-Line" lista os painéis disponíveis no Protheus.



Na pasta "Integrações" lista todas as configurações da integração do Protheus com Fluig com protocolo OAuth, a autenticação e autorização do SmartClient HTML ao Fluig.



- “**Integração SAML**” - lista todas as configurações para integração com o FluiG Identity será feita pelo protocolo SAML 2.0 – (Security Assertion Markup Language)
- “**Client oAuth**” - É possível utilizar oAuth como autenticação no APP de Gestão, tanto para a utilização das api REST, como para acesso às rotinas via HTML.
- “**SCIM**” - O padrão SCIM foi criado para simplificar o gerenciamento do usuário na nuvem através da definição de um esquema de representação usuários e grupos e uma API REST para todas as operações de Cloud necessárias

Após definir as políticas de segurança e clicar no botão “Confirmar”

6.6.1. Usuários

O Protheus trabalha com o conceito de senhas de usuário, ou seja, a senha será validada pelo (Nome + Senha do Usuário), e não apenas pela sua senha. A opção Senhas de Usuário deve ser utilizada para restringir o acesso às movimentações do sistema, direcionar a gravação dos relatórios em disco e configurar drivers de impressão específico para um usuário/grupo.

O Protheus permite reunir usuários em grupos, para facilitar a atribuição de direitos e restrições. Neste caso, o acesso será sempre verificado de acordo com o maior nível de direito, ou seja, se um grupo não tiver acesso a um determinado módulo, mas o usuário sim, prevalecerá o acesso do usuário. Caso o usuário não tenha acesso a um módulo, mas seu grupo possua, prevalecerá o direito de acesso do grupo e o mesmo terá o acesso permitido. Para cadastrar usuários, selecione as seguintes opções: “Usuário” + “Senhas” + “Senhas de Usuário”;

Posicione o cursor sobre a opção “Usuários” e clique no botão “Incluir”;

Irá aparecer o cadastro dos usuários dividido em pasta, na pasta “Usuário” preencha os campos, conforme descrição a seguir:

Formação Programação ADVPL



Dados do Usuário

- Usuário:** Informe o código do usuário que será utilizado no (LogOn) processo de autenticação de acesso.
- Nome Completo:** Informe o nome do usuário
- Senha:** Informe a senha do usuário conforme a política de segurança definida
- Confirme a senha:** Confirme a senha digitada.
- Usuário bloqueado:** 1-Sim / 2-Não

Informe a situação do usuário

- Data do Bloqueio (Validade):** Informe a data em que o sistema deverá bloquear o acesso do usuário.
- E-mail:** Informe o e-mail do usuário
- Departamento:** Informe o departamento do usuário
- Cargo:** Informe o cargo do usuário

Parâmetros

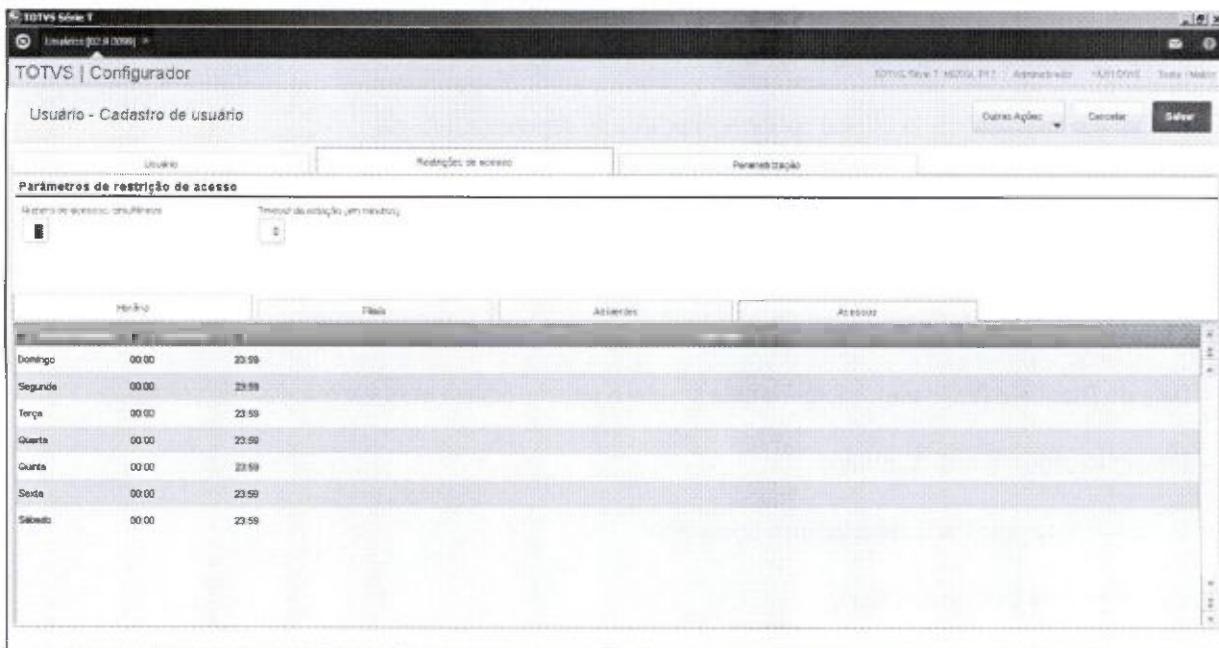
- Troca periódica da senha a cada N dias:** Informe a periodicidade estabelecida na política de segurança em que o usuário deve trocar sua senha.
- Forçar troca de senha no prox. Logon:** Forçar a troca de senha do usuário

- Número de dígitos no ano:** Informe o número de dígitos para o ano que o sistema irá utilizar na apresentação dos dados do tipo data
- Regras de acesso por grupo:** 1:Priorizar/2:Desconsiderar/3:Somar
Informe a regra de relacionamento do usuário com o grupo de usuários que ele possui.
- Exibir utilização de papel de Trabalho, quando disponível:** Informa se deseja visualizar a notificação do papal de parede.

Senhap

- Número de série do Senhap:** Informe o número de série do token de segurança (SenhaP)
- Usuário / Pastas**
- Superior:** Informar o superior do Usuário, opção disponível para trabalhar com política de segurando por Supervisor.
- Grupo:** Informe grupo que o usuário está relacionado.
- Papel de Trabalho:** Informe papel de trabalho que o usuário está relacionado.

Pasta "Restrição de acesso" preencha os campos, conforme descrição a seguir:



Parâmetros de restrição de acessos

- Número de acessos simultâneos:** Informe a quantidade de acessos simultâneos que o usuário pode realizar.
- TimeOut da estação (em minutos):** Indica o tempo máximo que a conexão pode ficar inativa

Formação Programação ADVPL



Pastas Horário

- **Horário:** Informar os horários que o usuário poderá acessar o sistema
- **Filiais:** Informar as empresas e filiais disponíveis para acesso do usuário
- **Ambiente:** Informar os módulos disponíveis para o usuário
- **Acessos:** Informar os acessos do usuário

Na pasta "Parametrização" preencha os campos, conforme descrição a seguir:

Database

- **Configurar dias de troca de database:** 1=Sim / 2=Não
Informe se o usuário poderá alterar a data base do sistema.
- **Retroceder:** Informe a quantidade de dias em que o usuário poderá retroceder a database do sistema.
- **Avançar:** Informe a quantidade de dias em que o usuário poderá avançar a database do sistema.

Telefonia

- **Habilitar Listener de ligação:** Informar se o usuário poderá ativar a escuta telefonia
- **Ramal:** Informar o ramal do usuário

Restrição de acesso

- **Nível global de campos (Leitura):** Informe o nível de acesso global aos campos do dicionário de dados, restrição de visualização de campos no dicionário de dados.

Recursos do sistema

- **Menu Funcional:** Informe se o usuário pode acessar o menu funcional
- **Painel Online:** Informe se o usuário pode acessar o Painel Online disponível

- Browser de Internet:** Informe se o usuário pode acessar o Browser de Internet
- Detalhes do Browser:** Informe se o usuário pode acessar os Detalhes do Browser
- Painel Transparente:** Informe se o usuário pode acessar o Painel Transparente
- Refresh Browser:** Informe se o usuário possui Refresh Browser automático

Pasta Impressão:

Impressão	Log de operações	Vínculo funcional	Painel OnLine	Painel OffLine	Indicadores Native	Acessibilidade
Configuração do diretório						
Diretório de impressão padrão: ISPOOL <input type="checkbox"/> Habilitar acesso a outros diretórios						
Configuração de impressão						
Tipo de impressão padrão: 1 - Em disco	Formato de impressão padrão: 1 - Retrato	Ambiente de impressão padrão: 1 - Servidor				
Configuração da impressora						
Nome da impressora padrão: Caminho da impressora padrão:						

Configuração do diretório

- Diretório de impressora Padrão:** Informe o diretório padrão de impressão.
- Habilitar acesso a outros diretórios:** Informe se o usuário poderá escolher o destino da impressão

Configuração de impressão:

- Tipo de impressão padrão:** 1=Em disco \ 2=Via Windows \ 3=Direto na porta
Informe o tipo de impressão padrão
- Formato de impressão padrão:** 1=Retrato\2=Paisagem
Informe o formato de impressão padrão
- Ambiente de impressão padrão:** 1=Servidor\2=Cliente
Informe o ambiente de impressão padrão

Pastas Log de Operação

Impressão	Log de operações	Vínculo funcional	Painel Online	Painel Offline	Indicadores Native	Acessibilidade
Processo*	Inclusão de registro*	Alteração do registro*	Relatórios*	Transações*		
1 - Padrão	1 - Padrão	1 - Padrão	1 - Padrão	1 - Padrão		
E-mail*	Resposta de parâmetros*					
2 - Padrão	2 - Padrão					

- Processo:** 1=Sim / 2=Não / 3=Padrão
- Inclusão de Registro:** 1=Sim / 2=Não / 3=Padrão
- Alteração de Registro:** 1=Sim / 2=Não / 3=Padrão
- Relatórios:** 1=Sim / 2=Não / 3=Padrão
- Transações:** 1=Sim / 2=Não / 3=Padrão
- E-mail:** 1=Sim / 2=Não / 3=Padrão
- Resposta de parâmetro:** 1=Sim / 2=Não / 3=Padrão

Formação Programação ADVPL



Esta configuração trabalha em conjunto com o parâmetro **MV_LOGSIGA** (Onde o padrão da aplicação é determinado). A utilização de qualquer um dos LOGS pode acarretar em perda de performance, especialmente nos Logs de Inclusão e alteração de registros, devendo ser utilizado apenas em análises específicas.

Pasta Vínculo funcional

Impressão	Log de operações	Vínculo funcional	Painel Online	Painel Off-line	Indicadores Métricos	Acessibilidade
Grupo de empresas	Filial do sistema		Nome do grupo de empresas			
Filial de acesso			Endereço Social			Matrícula
Nome do funcionário						

- **Grupo de Empresas:** Informe o código do grupo de empresas que o usuário poderá acessar
- **Filial do Sistema:** Informe o código da filial que o usuário poderá acessar
- **Matrícula:** Informe o código da matrícula do funcionário

Pasta Painel Online

Impressão	Log de operações	Vínculo funcional	Painel Online	Painel Off-line	Indicadores Métricos	Acessibilidade

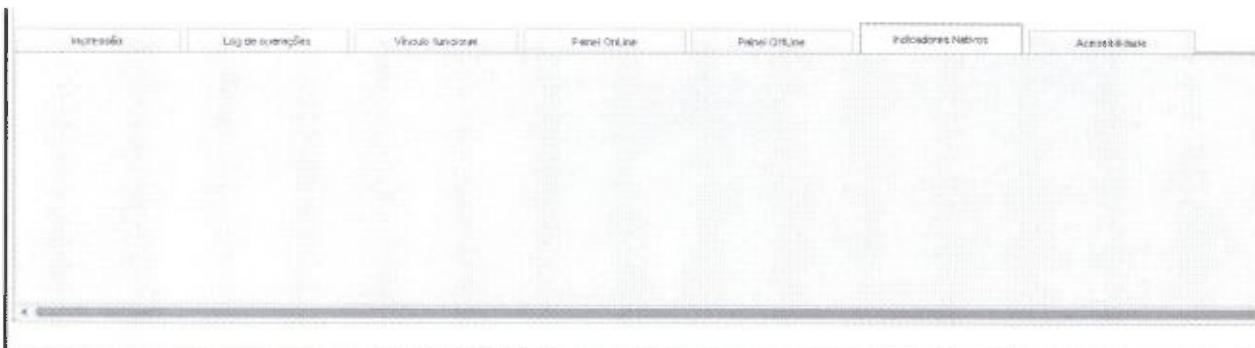
Lista todos os painéis disponíveis no sistema, os painéis disponíveis são liberados na política de acesso, número máximo de painéis visíveis simultaneamente são 8

Pasta Painel Off-line

Impressão	Log de operações	Vínculo funcional	Painel Online	Painel Off-line	Indicadores Métricos	Acessibilidade

Lista o painel disponível no sistema, os painéis disponíveis são liberados na política de acesso, para a utilização dos painéis de gestão off-line, é necessária estar (armazenado na ferramenta DW)

Pasta Indicadores nativos



Lista todos os Indicadores Nativos disponíveis no sistema. Os indicadores de nativos são para desempenho, normalmente, são obtidos por meio da implementação de ferramentas de Business Intelligence com o objetivo de auxiliar o processo de tomada de decisões.

Pasta Acessibilidade

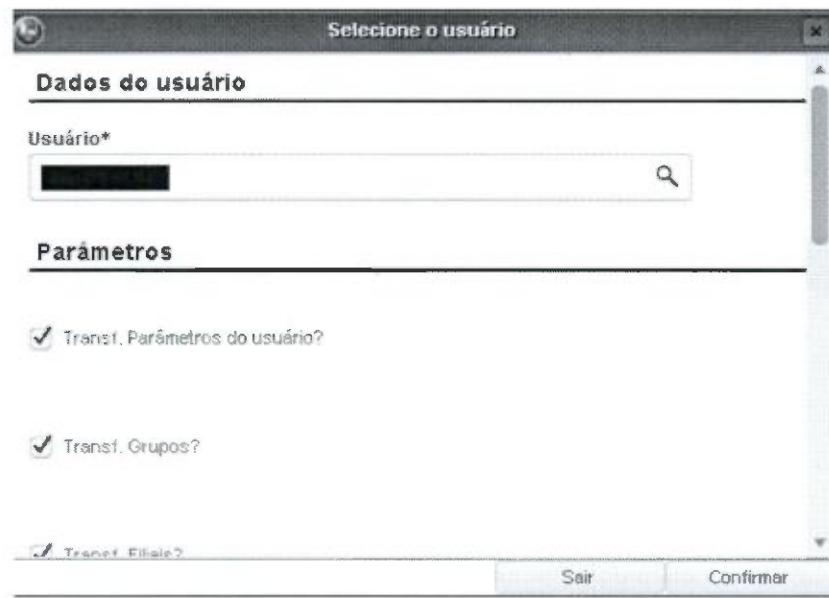


- Opção de Legenda:** 1=Legenda Colorida/2=Legenda numérica
Informe o tipo de legenda que o usuário pode visualizar no Browser

Após fazer todas as configurações desejadas selecionar o botão "Confirmar"

6.6.1.1 Transferência de Direitos do Usuário

No Browser Cadastro de Usuários selecionar o botão "**Ações Relacionadas**", a opção transferência de direitos é um facilitador utilizado para fazer manutenção no usuário, transferindo os direitos e/ou privilégio de outro usuário.



Preencha os campos conforme descrição a seguir:

- **Usuário:** Informa o usuário que deseja receber os parâmetros do usuário selecionado no Browser do cadastro. Após selecionar o usuário e informar os parâmetros que deseja transferir, clicar no botão "Confirmar", irá aparecer uma tela notificando se deseja fazer as alterações, clicar no botão "SIM".

6.6.1.2 Cópia de Perfil do Usuário

No Browser Cadastro de Usuários selecionar o botão "**Ações Relacionadas**", a opção Copia, essa opção copia todas as configurações do usuário selecionado, evitando fazer um cadastro do início.



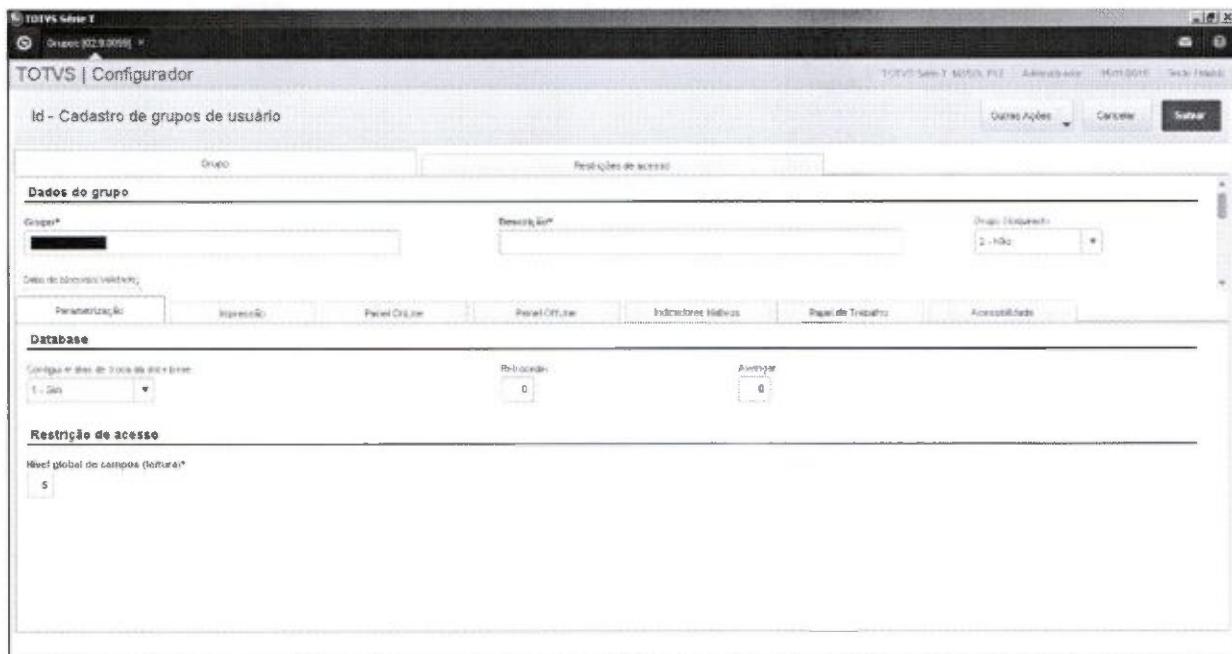
Após copiar o perfil do alterar as configurações desejadas

Essa opção está disponível no grupo de usuários.

6.6.1.3 Grupos de Usuários

A criação de Grupos de Usuários tem a finalidade de facilitar a administração de um determinado número de usuários que possuam as mesmas características dentro do Sistema, ou seja, que possuam o mesmo perfil de acesso.

Para cadastrar Grupos de Usuários, selecione as seguintes opções: "Usuário" + "Senhas" + "Grupos" e clique no botão "Incluir".



Preencha os campos conforme descrição a seguir:

- **Grupo:** Informe o código do grupo de usuário
- **Descrição:** Informe a descrição do grupo de usuário
- **Grupo Bloqueado:** 1-Sim / 2-Não - Informe se o grupo está ativo.
- **Data de Bloqueio(Validade):** Informe a data em que o grupo deixará de ser utilizado.

As configurações dos campos e parâmetros do cadastro do grupo são iguais do cadastro de usuário.

Após fazer o cadastro dos grupos acessar o castro de usuário guia grupo e relacionar o grupo e alterar o campo “Regra de Acesso por Grupo”

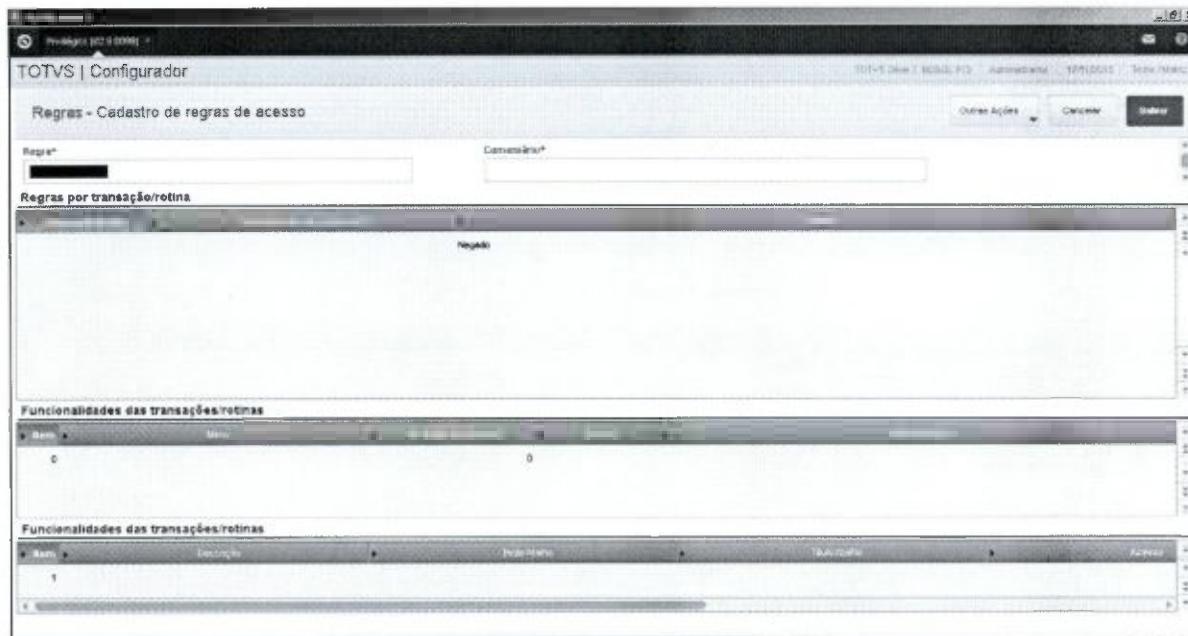
6.7. Privilégios

A Microsiga Protheus permite definir privilégios de acesso para os “Usuários” e “Grupos de Usuários do Sistema”, para facilitar a atribuição de direitos e simplificar seu processo de manutenção e administração.

Devem ser cadastradas informações que considerem as restrições de acesso e a segregação de funções que cada Usuário ou Grupo de Usuário possui. É possível ter a quantidade de privilégios necessários, sendo que os acessos são somados e reconhecidos no processo de autenticação do Usuário.

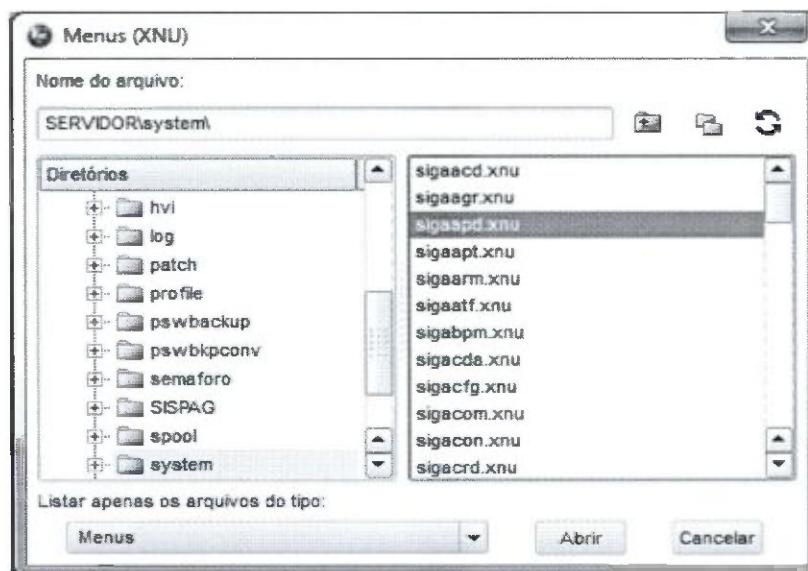
Para cadastrar os “Privilégios”, selecione as seguintes opções: “Usuário” + “Senhas” + “Privilegio” e clique no botão “Incluir”.

Formação Programação ADVPL



Para carregar as informações do "Modulo" selecionar o botão "Ações Relacionadas" clicar na opção Menu, irá aparecer um Wizard.

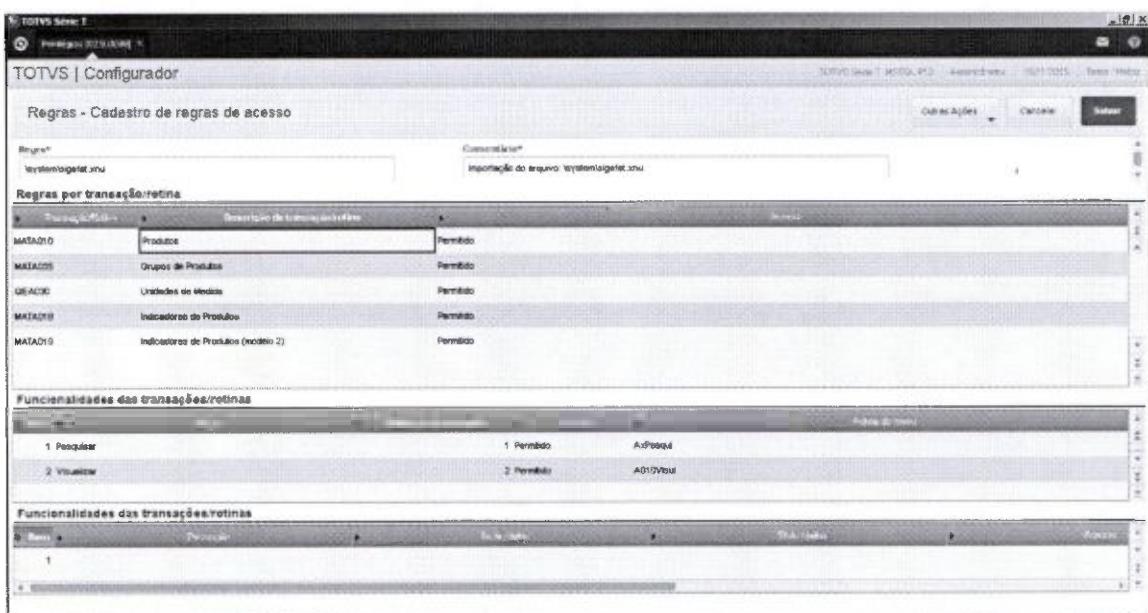
1. Selecionar o botão "Avançar"
2. Selecionar o botão localizar, informar o endereço do Menu que irá criar os privilégios



Preencha os campos conforme descrição a seguir:

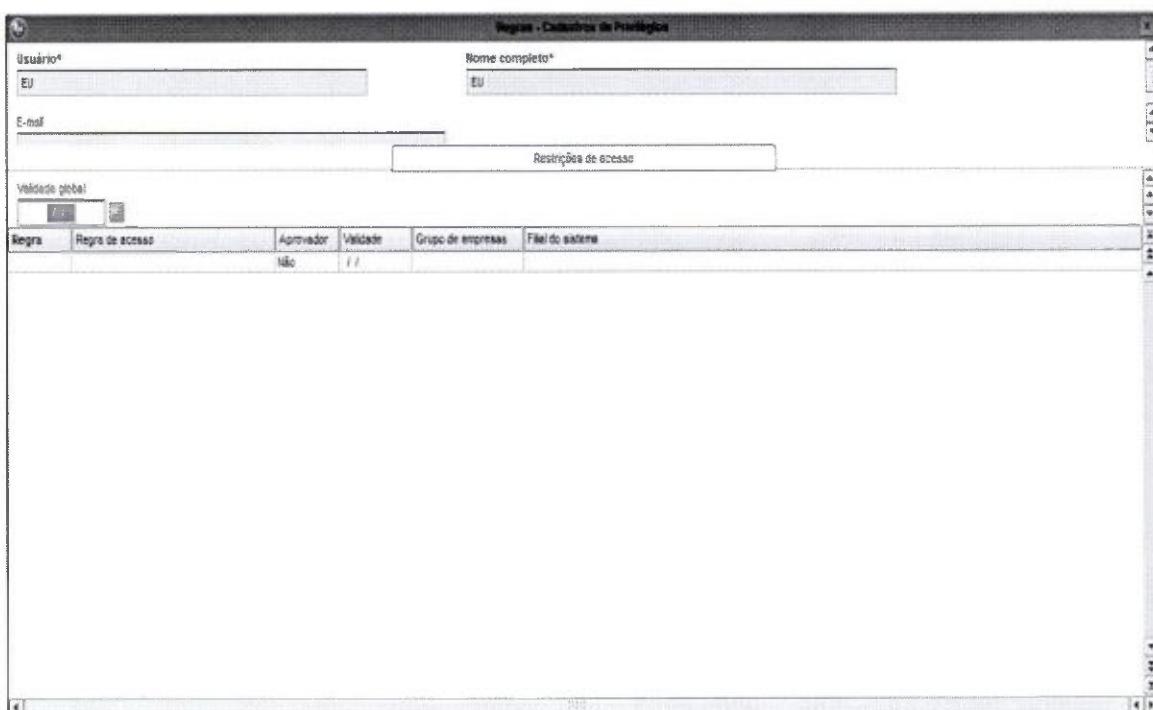
- **Regra:** Informe o código do privilégio.
- **Comentário:** Informe a descrição do privilégio
- **Regras por Transação/Rotina:** Informe as rotinas em que o usuário poderá acessar

- **Funcionalidade das Transações/Rotinas:** Informe os acessos que o usuário possui acesso dentro da estrutura do browser do Protheus
- **Transação/Rotina:** Informe as teclas de atalhos usuário possui acesso



Após definir os acessos selecionar o botão "Salvar".

Para relacionar os privilégio nos usuários ou grupos de usuários, acessar o cadastro de usuário ou grupo de usuário, selecionar o cadastro desejado, para fazer o relacionamento clicar no botão "Ações Relacionadas" opção "PRIVILEGIOS".



Preencha os campos conforme descrição a seguir:

- **Validade Global:** Informe a data de expiração do vínculo do privilégio com o usuário ou grupo.
- **Regra:** Informe o código do privilégio criado
- **Grupo da Empresa:** Informe o código do grupo de empresas em que o usuário tem este privilégio. Se não informado, ele terá em todos os grupos.
- **Filial do Sistema:** Informe a filial do sistema em que o usuário tem este privilégio. Se não informado, ele terá em todas as filiais.

Após vincular os privilegios ao Usuário ou Grupo de usuário clicar no botão "Confirmar".

6.7.1. Config. Perguntas

Define uma resposta padrão para perguntas apresentadas durante a configuração de alguns processamentos, os usuários não conseguem alterar a resposta padrão.

Para cadastrar os "Config. Perguntas" selecionar as seguintes opções: "Usuário" + "Senhas" + "Config. Perguntas" e clique no botão "Incluir".



Preencha os campos conforme descrição a seguir:

- **Grupo:** Informe o grupo de perguntas correspondente à rotina envolvida. Utilize a tecla [F3] para apresentar os grupos de perguntas
- **Usuário:** Informe o usuário ou grupo de usuários que deve utilizar a configuração. Utilize a tecla [F3] para apresentar os usuários ou grupos de usuários.
- **Perguntas:** Utilize a tecla [DEL] para desmarcar a linha selecionada e poder alterar os conteúdos das respostas
- **Conteúdo:** Informar o valor desejado.

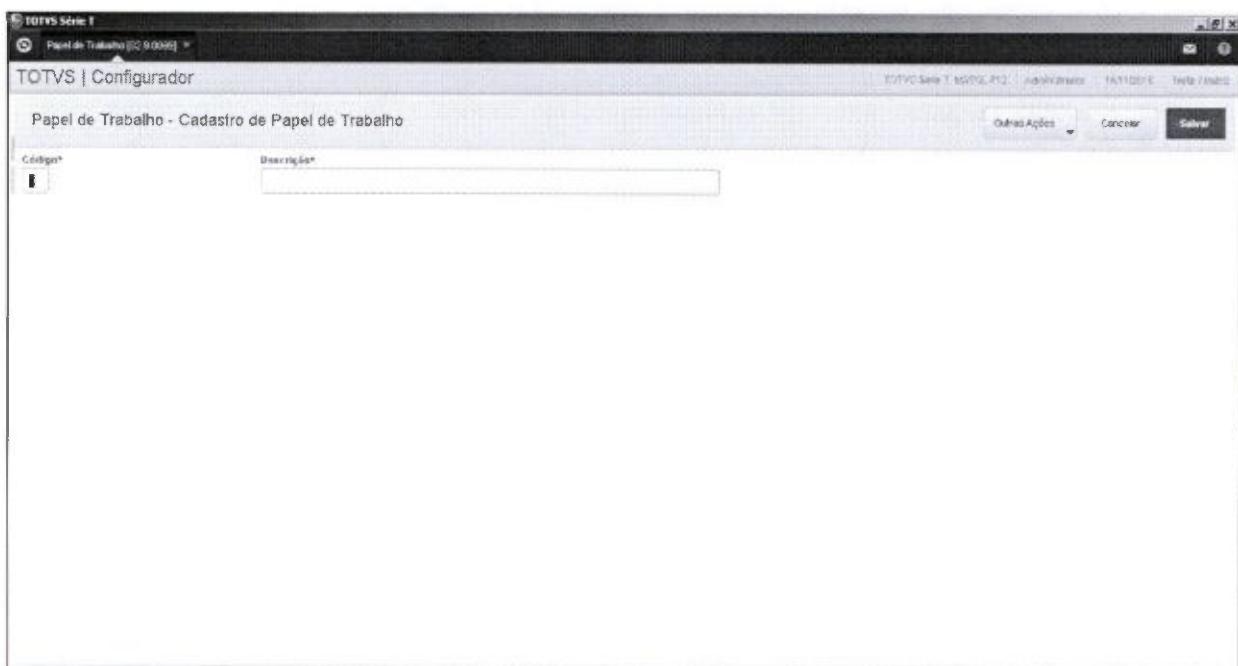
- **Formula:** Informar o valor da formula, é possível compor uma expressão em sintaxe AdvPL para preenchimento da pergunta

Após preencher as informações desejadas clicar no botão confirmar.

6.8. Papel de Trabalho

Papel de trabalho foi desenvolvido para personalizar o browser, possibilitando alterar ordem dos botões, alterar as fontes e cores deixando a estrutura como desejada.

Para acessar papel de trabalho opções: "Usuário" + "Papel de Trabalho" + "Papel de Trabalho" e clique no botão "Incluir".



Preencha os campos conforme descrição a seguir:

- **Código:** Informe o identificador do papel de trabalho
- **Descrição:** Informe a descrição do papel de trabalho

Após preencher os campos selecionar o botão confirmar.

Foi criado o código identificador do Papel de trabalho, após fazer o cadastro iremos acessar "Usuário" + "Papel de Trabalho" + "Configura Papel de Trabalho" e clique no botão "Incluir".

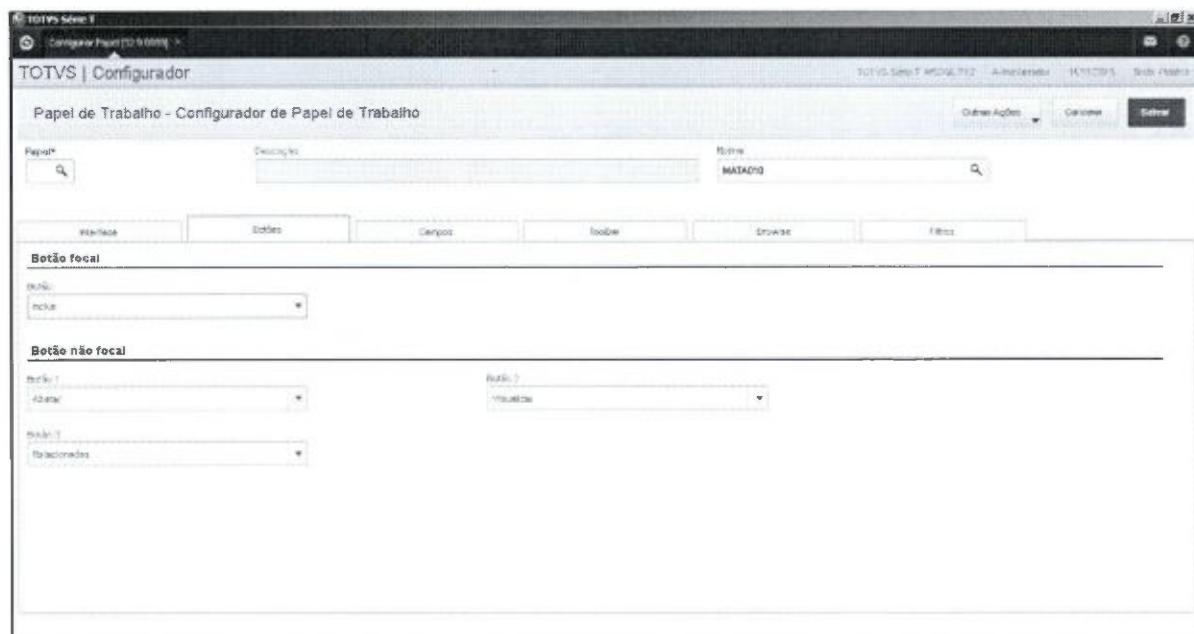
Formação Programação ADVPL



Preencha os campos conforme descrição a seguir:

- **Papel:** Informe o código do papel de trabalho
- **Rotina:** Informe a rotina para realizar a configuração do papel de trabalho. Caso não seja informada a rotina, a configuração será utilizada por todas as rotinas.

Pasta Botões



- **Botão focal:** Informe à ação que será utilizada pelo botão focal.
- **Botão não focal:** Possui 3 opções de ações de botões, as opções de função não informada ira aparecer no browser na opção Ações Relacionadas.

Pasta Browser



Altura da Linha

- **Altura da linha:** Altura da linha que será utilizada no Browse

Restrição de Funcionalidades

- **Habilita Filtrar?**
- **Habilita Localizar?**
- **Habilita Configurar?**
- **Habilita Pesquisa?**
- **Habilita Imprimir?**
- **Habilita Detalhes?**

São as opções que ficam disponíveis na parte superior do Browser, caso desejar desativar alguma das funcionalidades desmarcarem as opções desejadas.

Configuração de Fonte

- **Fonte:** Informe fonte que será utilizada no Browse
- **Tamanho da Fonte:** Informe o tamanho da fonte que será utilizada no Browse
- **Suprimido**
- **Negrito**
- **Itálico**

Configuração de Cores

- **Legenda:** Cor/Número: Informe o tipo de legenda que será utilizada no Browse.
- **Cor Alternada:** Cor alternada do browser

Filtro do Browser

- **Alias:** Informe a tabela que será utilizada no filtro do Browser
- **Montar Filtro:** Opção para montar a estrutura do filtro
- **Limpar Filtro:** Opção para limpar o filtro
- **Filtro:** Visualização da estrutura do filtro montado
- **Executa Filtro:** Informar se o filtro vai ser executado
- **Exibe o filtro na entrada do Browser:** Informar se o filtro vai aparecer na entrada do Browser

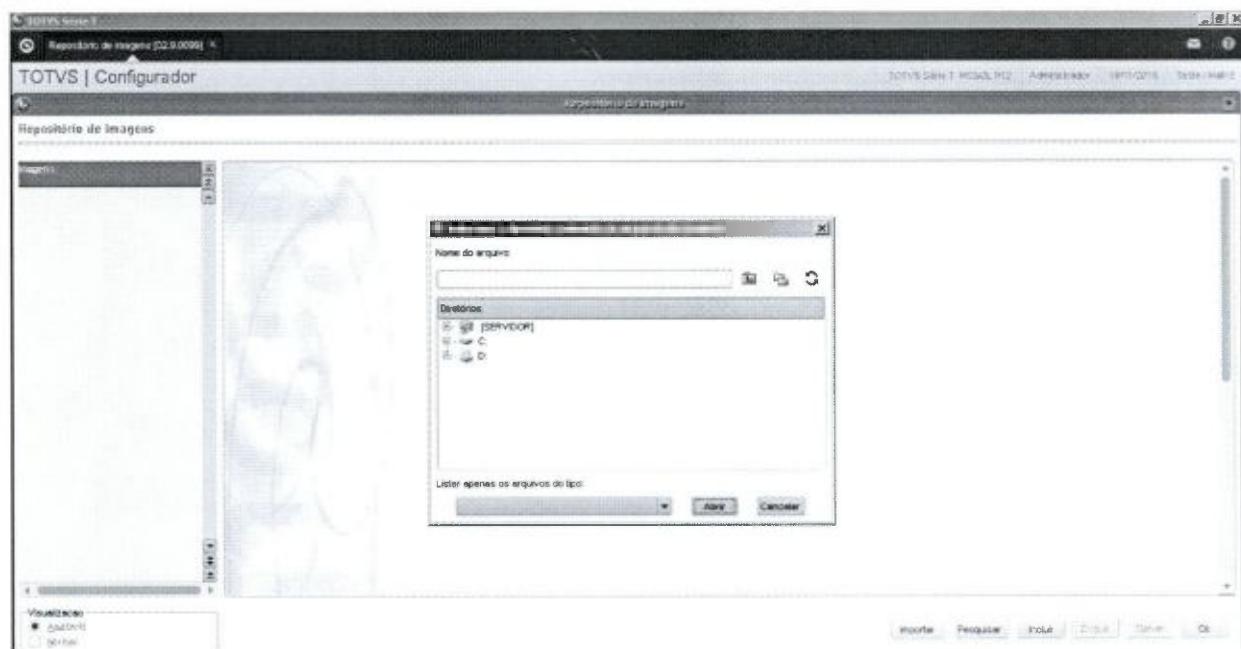
Após configurar o papel de trabalho e necessário relacionar o papel no usuário e/ou grupo para poder validar as configurações, selecione as seguintes opções: "Usuário -> Senhas -> Usuário" na pasta "Papel de Trabalho" informar o papel de trabalho e selecionar a opção **Prioriza: SIM**.

6.9. Repositório de Imagens

Algumas rotinas do Sistema permitem a associação de imagens. Para que elas possam ser utilizadas, é necessário cadastrá-las nesse repositório para manipulação das informações gráficas.

Para cadastrar Imagens no Repositório de Imagens, selecione as seguintes opções: "Ambiente" + "Cadastros" + "Reposit. Imagens"

Clique em **Incluir** para adicionar um arquivo de imagem *bitmap*. Informe a localização e seleção do arquivo Selecionar o endereço das imagens:



Para ajustar a imagem clique em **Normal/Ajustado** para ampliar ou restaurar a visualização da imagem.

Após selecionar as imagens que serão salvas no repositório clicar no botão "Confirmar".

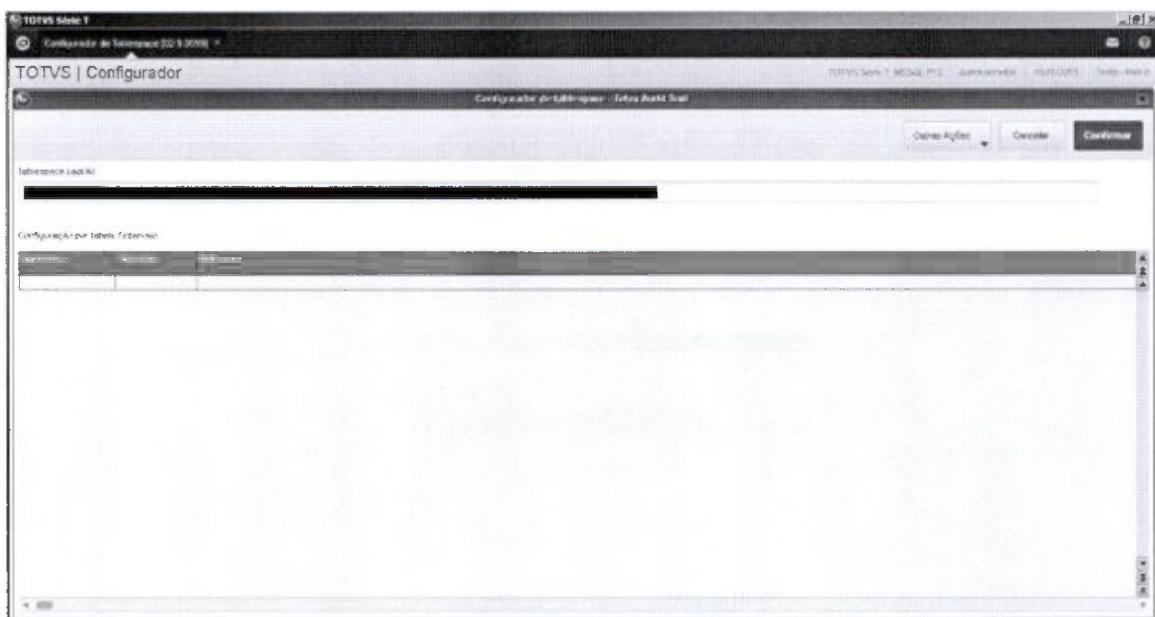
Importante

O repositório de imagens é o arquivo SIGAADV.BMD que fica na pasta \SYSTEM\, as imagens são copiadas do local de origem para esse arquivo. Arquivos BMP não são compactados quando incluídos no repositório, dê preferência a arquivos JPG. O repositório também é único para todos os grupos de empresas cadastrados, pode ser mais inteligente utilizar o recurso base de conhecimento.

6.10. Embedded Audit Trail

O administrador do Embedded Audit Trail pode especificar tablespaces diferentes da tablespace padrão do sistema Protheus para alocar os dados gerados pelo Audit Trail. Se esta configuração não for feita, os dados gerados pelo Audit Trail serão alocados na tablespace padrão utilizada pelo Protheus.

O configurador de tablespace permite que seja definida uma tablespace padrão do Audit Trail e também tablespaces específicas por intervalo de tabelas. O intervalo de tabelas é uma configuração onde é informada uma tabela inicial e uma tabela final. A tablespace definida para o intervalo de tabelas tem prioridade sobre a tablespace padrão do Audit Trail e sobre a tablespace padrão do Protheus.



Abaixo listamos a ordem de prioridade na escolha da tablespace:

- Intervalo de tabelas
- Padrão do Audit Trail
- Padrão do Protheus

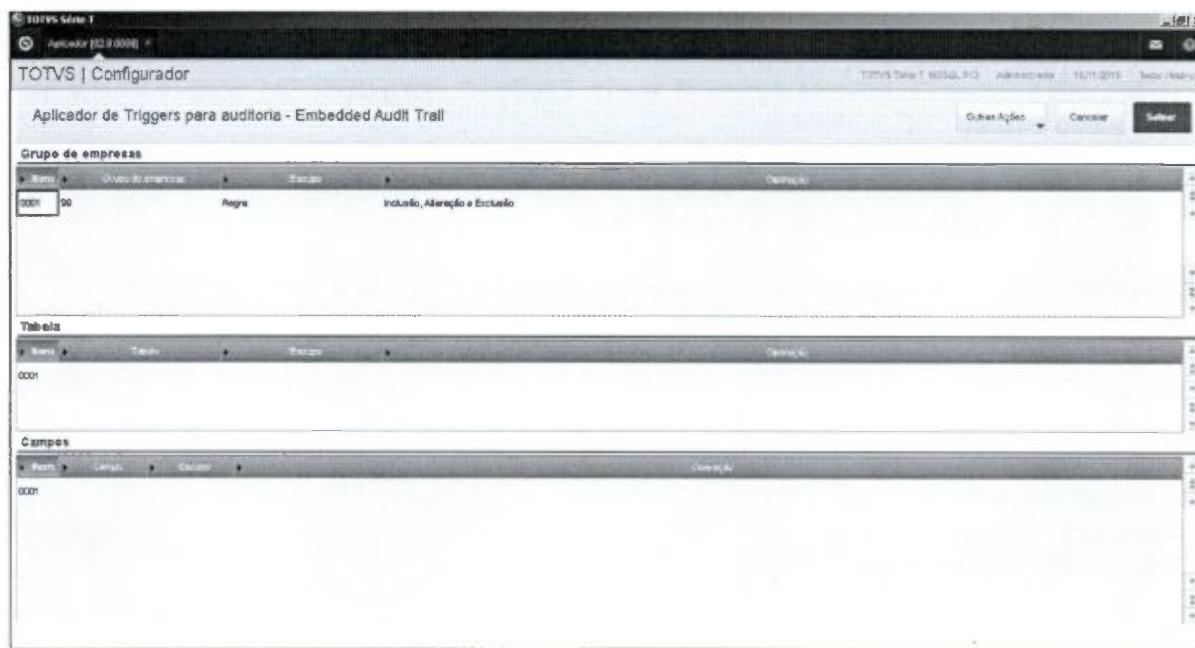
As configurações do TableSpace não são obrigatórios, podemos fazer as configurações pelo Aplicador

Para configurar o Aplicador acessar, Ambiente>Embedded Audit trail> Aplicador.

Formação Programação ADVPL



Para que o recurso seja ativado, deve-se escolher a abrangência de seu funcionamento sobre a instalação do sistema Protheus.



Por abrangência entendem-se as entidades em que o Audit Trail auditará:

- **Grupo de empresas**
- **Tabelas**
- **Campos**

Essa configuração é efetuada pela rotina do Aplicador. Através de um mecanismo de exceção e regra, o administrador poderá facilmente escolher as entidades que deseja auditar, ou ao contrário, as entidades que não deseja editar. Essa característica facilita o cadastramento por diminuir a quantidade de regras que é necessário informar.

Existem três níveis de configuração:

- **Nível superior:** nesse nível são informados os grupos de empresas
- **Nível intermediário:** nesse nível são informadas as tabelas.
- **Nível inferior:** nesse nível são informados os campos.

Cada nível de configuração possui, além da entidade informada, o escopo e a operação de aplicação. O escopo de aplicação possui os tipos básicos REGRA e EXCEÇÃO. Quando um item possui o escopo "regra", a auditoria será aplicada a ele.

O escopo de um nível não pode ser igual ao escopo do nível acima ou abaixo, quando na mesma área de abrangência.

Exemplo: foi definido um item no nível superior com o escopo REGRA para o grupo de empresas 01. No nível intermediário, vinculado ao grupo 01, foi definido o item com escopo REGRA para a tabela SA1. Trata-se de uma incoerência, pois ao definirmos que vamos auditar o grupo de empresas 01, significa que todas as tabelas e campos estão incluídos. Logo, não é preciso nem correto informar o nível intermediário.

Por outro lado, se for desejado auditar todas as tabelas, exceto a tabela de prefixo SA1, deve-se incluir um item do nível intermediário para SA1 com o escopo EXCEÇÃO.

Em relação ao nível inferior, o mesmo ocorre. O item (campo) apontado no nível inferior deve possuir escopo diferente do informado no nível intermediário, caso este campo pertença à tabela informada neste nível. Se for usado o exemplo acima, onde foi usado o escopo REGRA para o grupo 01, escopo EXCEÇÃO para a tabela SA1, pode-se informar um item REGRA para o campo A1_COD, por exemplo.

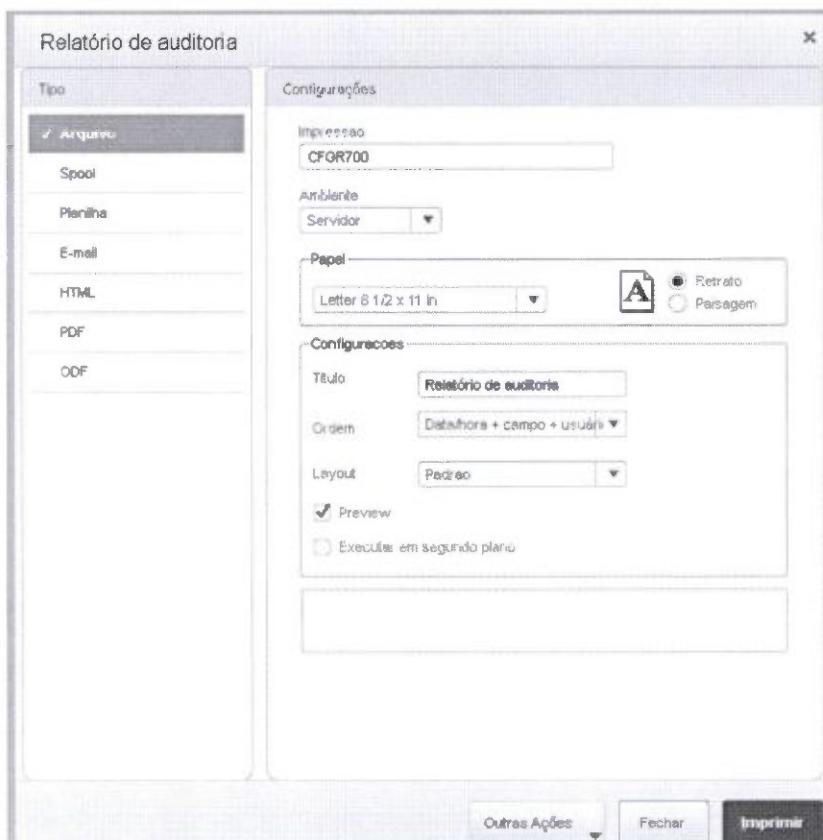
Além do escopo, deve-se informar qual a operação de banco de dados que se deseja auditar. As operações básicas de banco de dados são três: INCLUSÃO, ALTERAÇÃO e EXCLUSÃO.

No campo operação, existem sete opções que cobrem todas as combinações entre as três operações. São elas:

- Inclusão
- Alteração
- Exclusão
- Inclusão e Alteração
- Inclusão e Exclusão
- Alteração e Exclusão
- Inclusão, Alteração e Exclusão

As operações acima devem ser informadas apenas quando o escopo for REGRA. Quando o escopo for diferente de REGRA, não faz sentido informar uma operação, pois nada será auditado. Nesse caso deverá ser informada a operação "0=Não se aplica".

O relatório emite a listagem dos dados já armazenados pelo Embedded Audit Trail.



7. Desenvolvimento

Para iniciar o desenvolvimento de novas customizações é necessária do Totvs Developer Studio, ferramenta que interage diretamente com RPO, adicionando e removendo fontes customizado, para iniciar o desenvolvimento abrir o IDE

7.1. Perspectiva TOTVS Developer

Você pode selecionar uma perspectiva utilizando as opções disponíveis através dos botões e ícones de seleção, presentes na barra de ferramentas. Conforme figura abaixo:



1. Selecione a Opção TOTVS Developer:

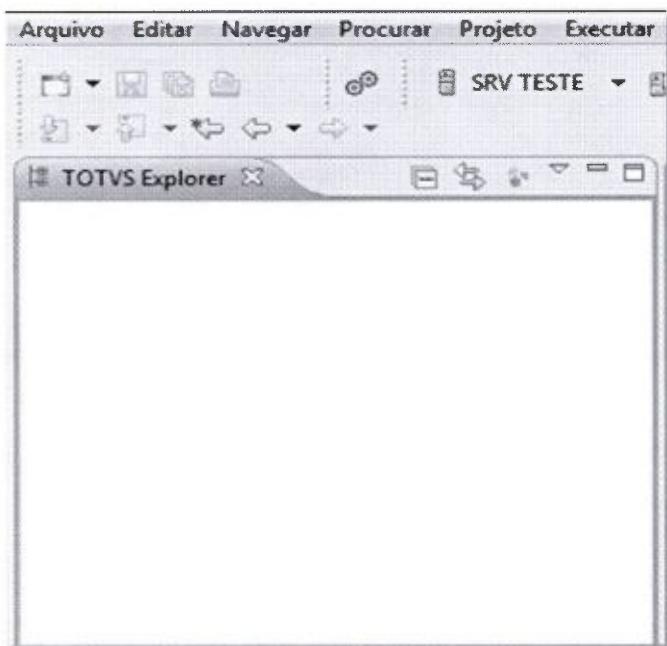


Na figura abaixo mostra as Perspectiva do TOTVS Developer e Servers configurada no Ambiente de Desenvolvimento.

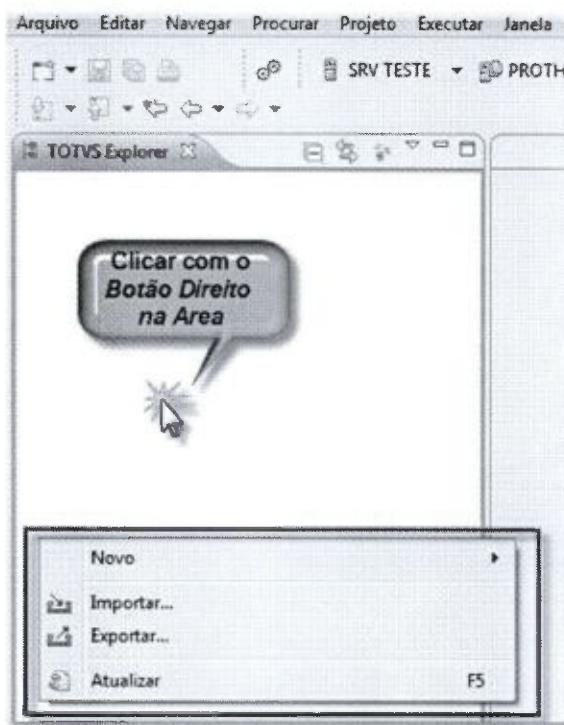


7.2. Iniciando um Projeto

A organização inicial de um trabalho, é o projeto. Em cada área de trabalho (workspace) você pode colocar quantos projetos desejar e faz a administração através da visão "TOTVS Explorer".



1. Logo abaixo do título da aba e da barra de ferramentas, ação o menu de contexto e nas opções apresentadas ação o menu Novo | Outras.... ou Importar. Será apresentado o assistente de novos elementos ou assistente de importação (conforme a opção utilizada).



Nota: Dependendo do licenciamento e versão do "TDS", algumas opções podem estar indisponíveis ou não existirem.

Utilizando a árvore de navegação, localize "TOTVS Wizard", expanda-o e obterá as opções (Tela 16 destacado).

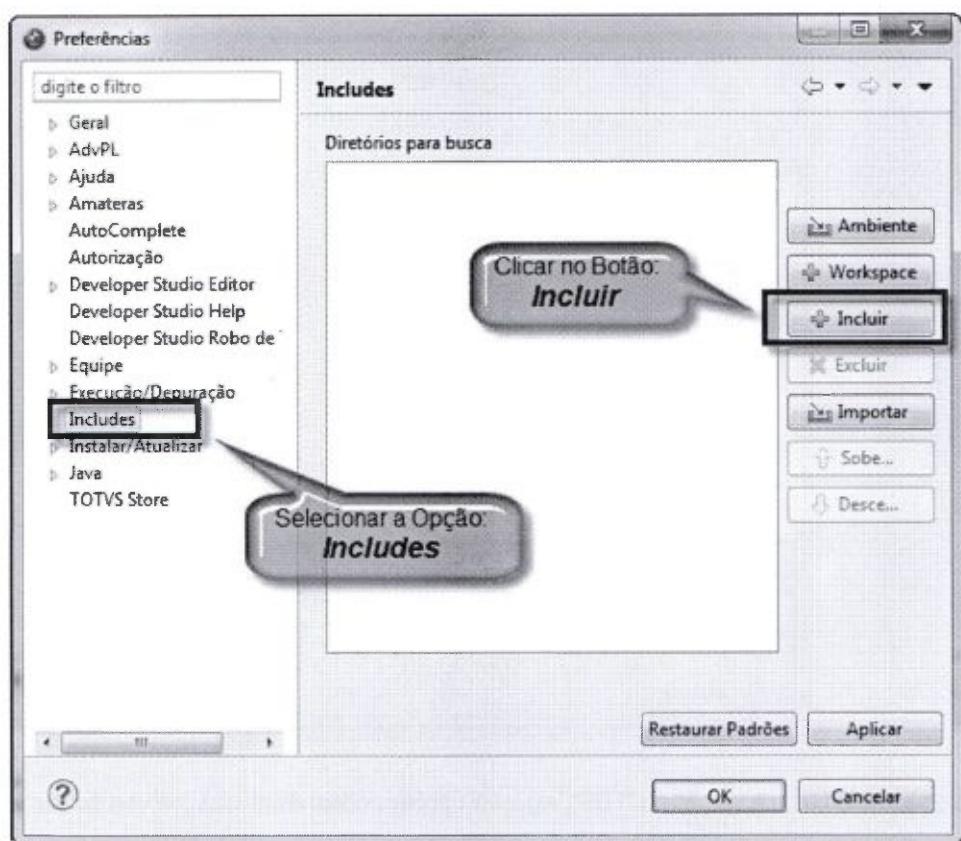
- **Importar projeto Adv/PL** – A partir de um arquivo de projetos (prj) lhe é preparado um projeto "TOTVS".
- **Novo Projeto TOTVS** – É inicializado um projeto "TOTVS" vazio, contendo uma estrutura básica.

NOTA : O arquivo de projeto (*.prj) é utilizado somente no momento da importação. Qualquer modificação posterior neste arquivo, não será refletido no projeto importado para o "TDS" e vice-versa.
Selecione a opção que melhor lhe atende e prossiga com a execução do assistente.

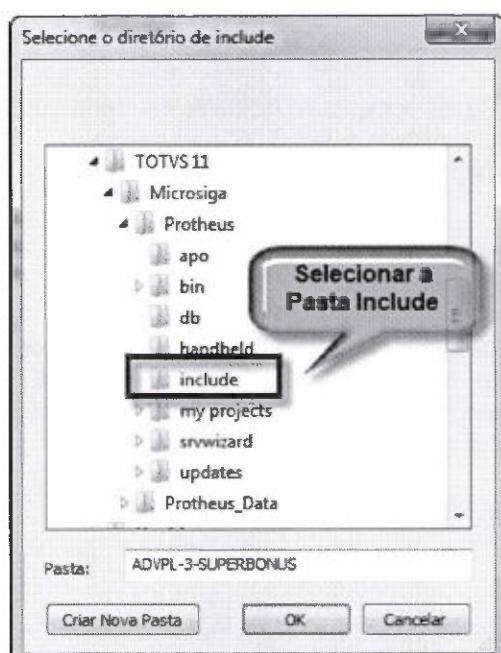
7.2.1. Configuração de Includes

Após a definição de todas as Preferências que serão utilizadas, deveremos então configurar o Ambiente de Trabalho que será utilizado pelo compilador, ou seja, qual o caminho do "Diretório de Includes", informe o "Caminho das Pastas", onde se encontram os "Arquivos de Cabeçalhos de Programas (*.CH)", essa configuração pode apontada para um único Projeto ou no Modo Global para todos os Projetos.

1. Acessando o menu **Janelas | Preferências...**



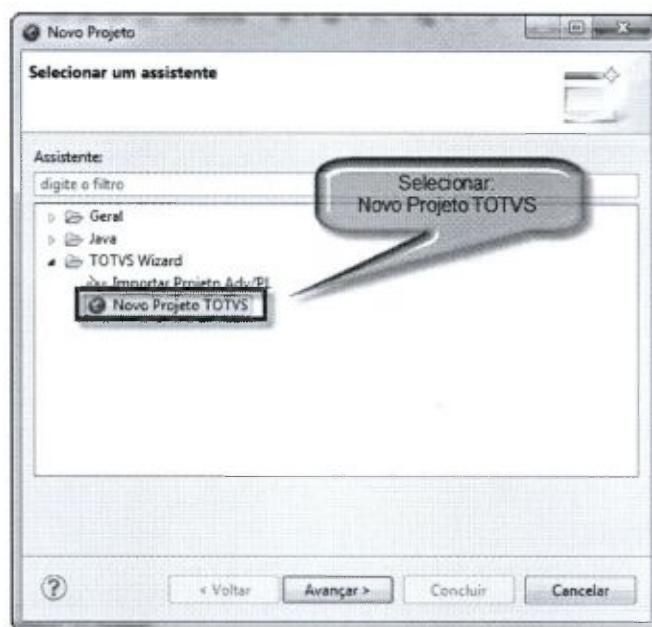
2. Selecionar o Diretório que contém a Pasta Include:



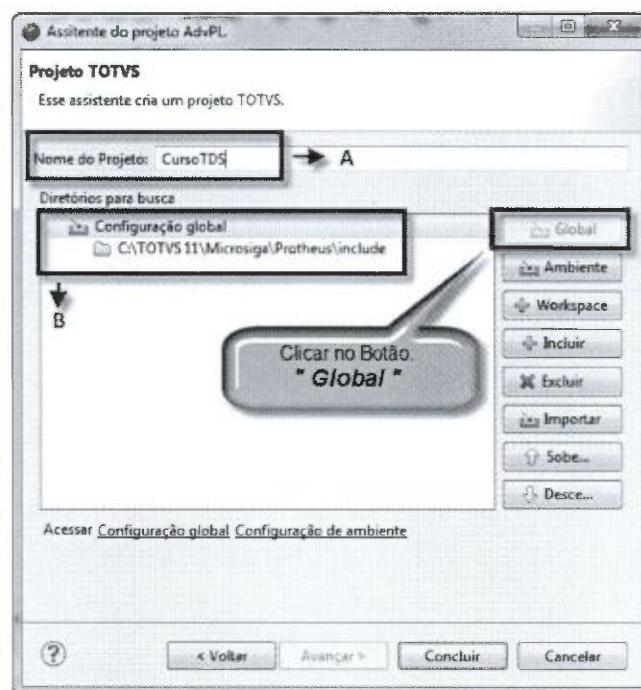
7.2.2. Novo Projeto TOTVS

Tem por finalidade, ajudar o desenvolvedor à administrar de maneira prática e organizada os programas envolvidos em uma determinada Customização. Como por exemplo, um módulo que será customizado poderá ser representado por um Projeto, onde cada Tópico abordado pode ser classificado em uma Pasta diferente dentro do mesmo.

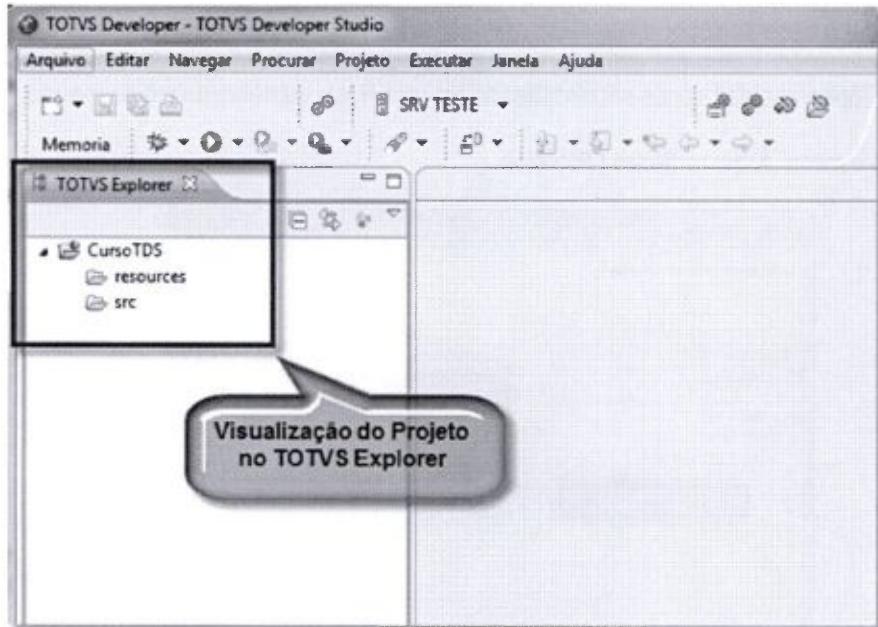
1. Acessando o menu **Arquivo |Novo | Projetos...**



2. No Assistente de Projeto, conforme digitar o nome para o projeto “Formação ADVPL” e clicar no botão “Global” para inclusão das Includes, conforme figura.



3. Visualização do Projeto criado no TOTVS Explorer.



7.2.3. Novo Arquivo Fonte

A opção Novo Arquivo, irá possibilitar a edição de novos programas, dentro do TOTVS Developer Studio.

1. Acessando o menu Arquivo |Novo | Nova Função ADVPL ...



Formação Programação ADVPL



Neste exemplo, criamos o arquivo-fonte "OlaTDS.prw", na pasta "resources" do projeto corrente. O uso do editor de fontes é bastante semelhante a outros que você deve estar acostumado, portanto saia digitando o seu código.

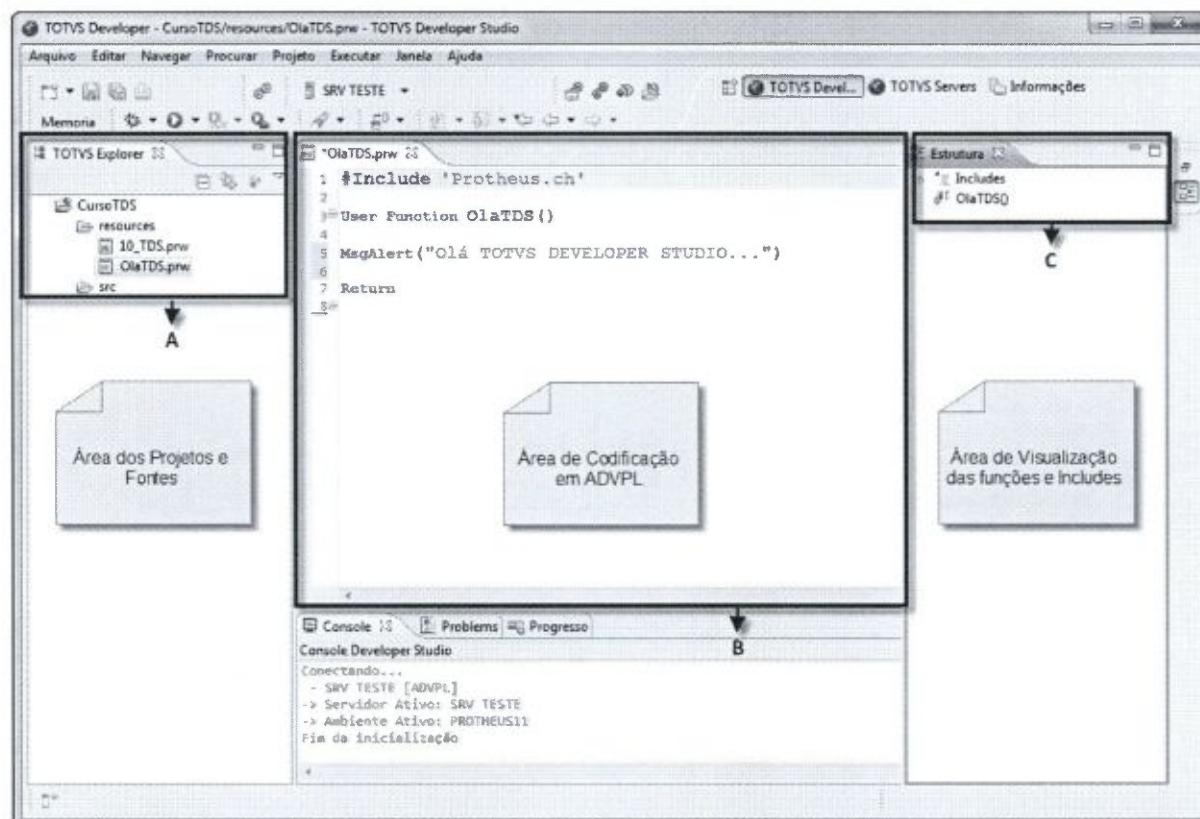
Exemplo do Código:

User Function OlaTDS()

MsgAlert ("Olá TOTVS Developer Studio 2012...")

Return

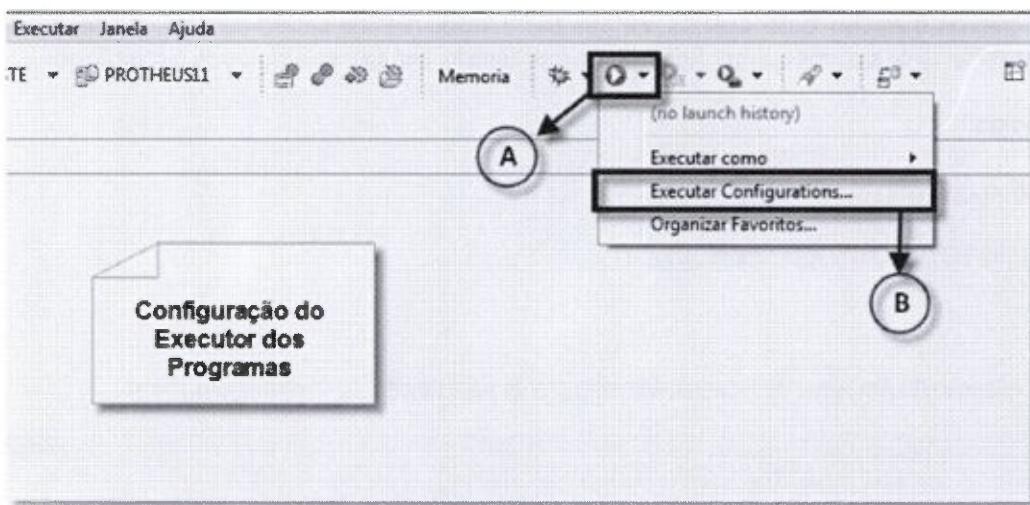
- Visualização do Ambiente de Desenvolvimento pronto para edição, conforme figura abaixo:



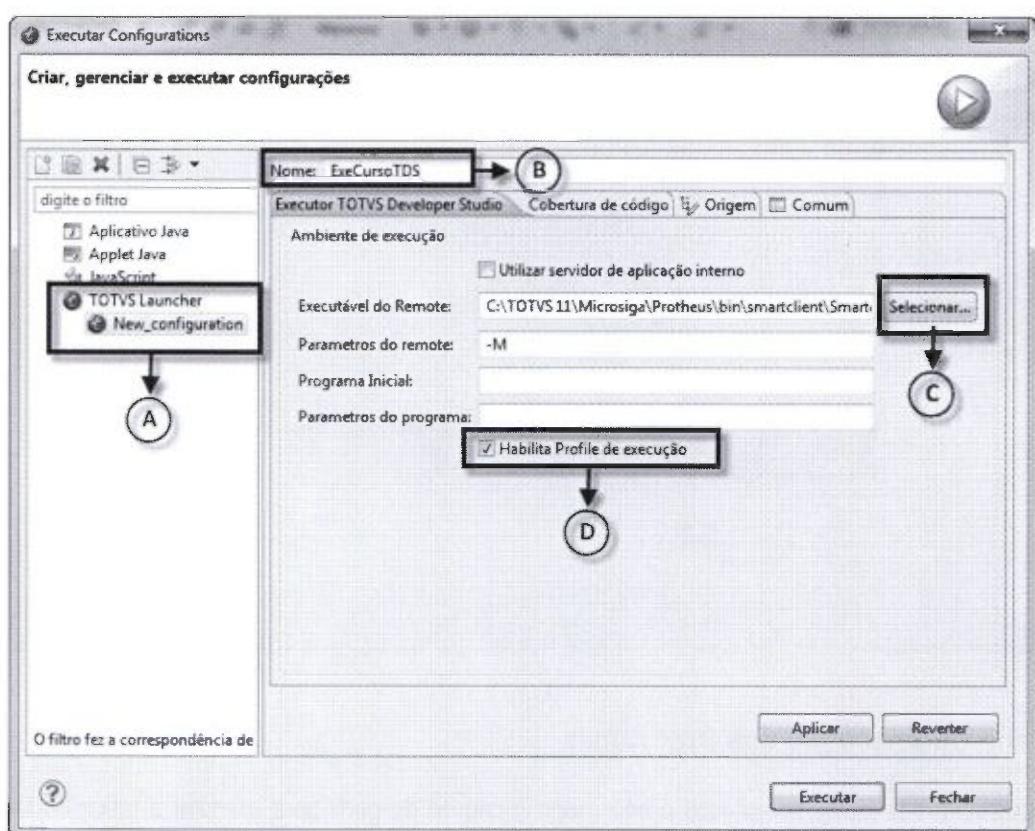
7.2.4. Perfil de Execução dos Programas

Para iniciar a execução dos fontes compilados é necessário a criação do perfil para executar e testar os arquivos, através de uma configuração de executor (laucher), podemos ativá-la quando necessário.

- Acesse a configuração de executores(laucher) para configuração, conforme figura abaixo:



- Clique no "Botão Executar", selecione a opção " Executar Configurations..." .



- Selecione o menu "**TOTVS Launcher|New Configuration**",
- **Executável do Remote:** Informar o diretório do SmartClient
- **Parâmetros do Remote:** Informar os parâmetros que podemos colocar no atalho do SmartClient Exemplo: -M -Q
- **Programa Inicial:** Podemos informar em qual modelo queremos acessar Exemplo: SigaMDI

- **Parâmetros do Programa:** Podemos passar algum conteúdo para a fonte ADVPL, fonte tem que está preparado para receber esses valores.
- **Habilita Profile de execução:** Quando estiver depurando irá trazer todo Profile de execução das rotinas.
ESSA OPÇÃO DEIXA APLICAÇÃO LENTA

7.2.5. Depuração de Arquivo Fonte (Debug)

A configuração do executor (laucher) com depuração é exatamente o mesmo utilizado na execução, portanto, para configurá-la utilize as instruções contidas em "Compilação de arquivos e projetos".

NOTA: Se você já efetuou a configuração do executor, já possuirá a mesma configuração para a depuração.

Para demonstrar o uso do depurador, usaremos o programa abaixo:

```

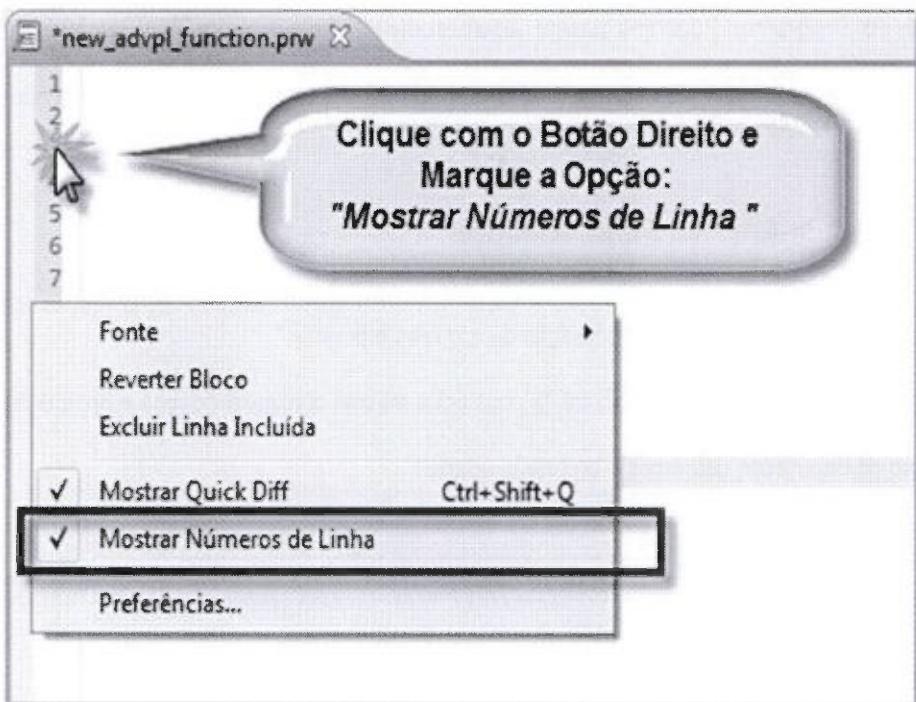
20_Debug.prw 13
1 #Include 'Protheus.ch'
2
3 // Projeto: Curso Totvs Developer Studio
4 // Módulo : Treinamento Presencial
5 // Fonte : 20_Debug.prw
6
7 // Data | Autor CT-TOTVS | Descrição
8 // -----+-----+-----+
9 // 25/09/12 | Debug/Depuração de Códigos
10 // -----+-----+
11
12 User Function 20_Debug()
13 Local nNum1 := 10
14 Local nNum2 := 3
15
16 Local cMsg := "1. Ponto de Parada"
17 Static cMsg1 := "2. Ponto de Parada"
18 Private cMsg2 := "3. Ponto de Parada"
19 Private aDias := ("Debug no TDS",date(),"Curso Totvs Developer Studio")
20
21 // Teste do primeiro ponto de parada
22 If(nNum1 > 5).And. (nNum2 <> 3)
23   MsgBox(cMsg)
24 ElseIf (nNum1 > 5).And. (nNum2 == 3)
25   MsgBox(cMsg)
26 Else
27   MsgBox(cMsg)
28 EndIf
29
30 // Fim da Comunicação da rotina

```



Arquivo Fonte para Depuração.

1. Acessando o menu **Arquivo |Abrir Arquivo ...** Selecionar o diretório: selecione o arquivo "**EXEMPLO.prw**".
2. Com o editor ativo (a aba fica destacada), acione **Ctrl+F10** e ative a numeração de linhas.



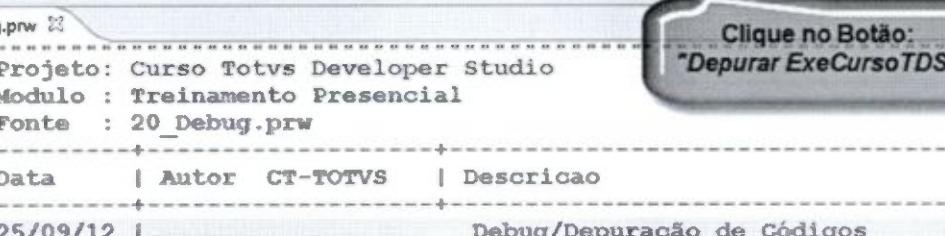
3. Efetue um duplo-click sobre o número "23" (linha 23). Repare que na barra cinza do lado esquerdo, aparece uma marca azul. Esta marca indica um ponto de parada (breakpoint). Quando iniciarmos a depuração, o depurador para a execução nos pontos de parada e aguarda instruções do desenvolvedor. Repita o processo para as Linhas 32, 42 . Conforme figura abaixo.

```

21
22 // Teste do primeiro ponto de parada
23 If(nNum1 > 5).And. (nNum2 <> 3)
24
25 // Teste do Segundo ponto de parada
26 Do Case
27
28 // Teste do Terceiro ponto de parada
29 For nNum=1 To 5
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52 Return
53

```

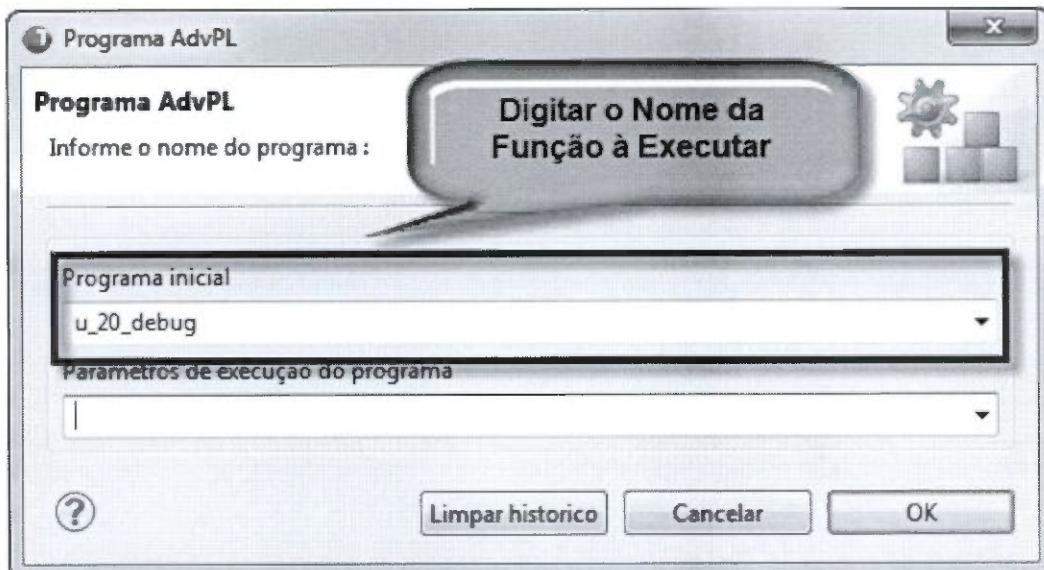
4. Inicie o depurador, acessando o botão de menu suspenso "Depurar" e selecione o executor.



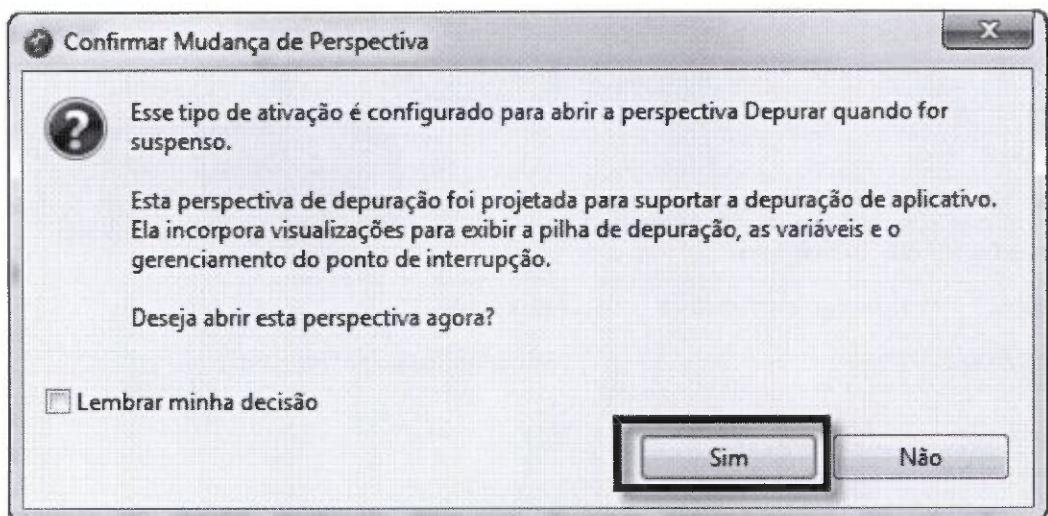
The screenshot shows the Totvs Developer Studio environment. The title bar displays "SRV TESTE" and "PROTHEUS11". The main area is a code editor with the file "20_Debug.prw" open. The code contains comments defining a user function named "20_Debug()". A tooltip box is overlaid on the interface, pointing to a button in the toolbar. The tooltip text reads: "Clique no Botão: 'Depurar ExecursoTDS'".

```
20_Debug.prw
1 // Projeto: Curso Totvs Developer Studio
2 // Modulo : Treinamento Presencial
3 // Fonte  : 20_Debug.prw
4 // 
5 // Data    | Autor   CT-TOTVS | Descrição
6 //          +-----+
7 //          +-----+
8 // 25/09/12 |           Debug/Depuração de Códigos
9 //          +-----+
10 //          +-----+
11 //          +-----+
12
13 User Function 20_Debug()
14 Local   nNum1 := 10
15 Local   nNum2 := 3
16
17 Local   cMsg  := "1. Ponto de Parada"
18 Static  cMsg1 := "2. Ponto de Parada"
19 Private cMsg2 := "3. Ponto de Parada"
20 Private aDias := ("Debug no TDS",date(),"Curso Totvs Developer Studio")
```

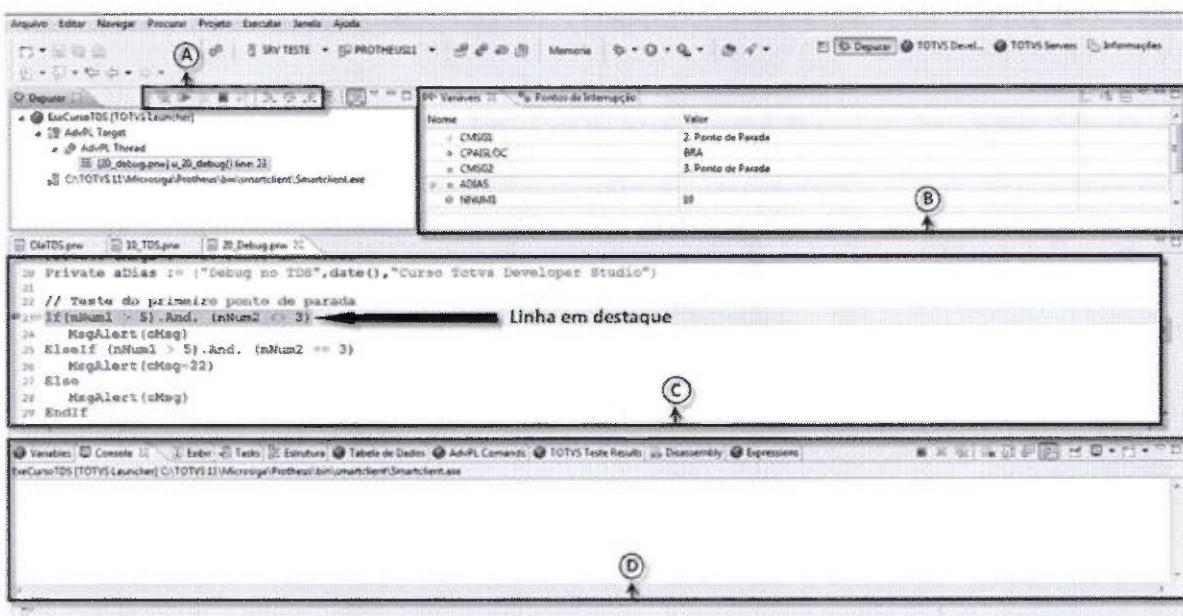
5. Aguarde alguns instantes e lhe será solicitado o programa inicial.



6. Clique no botão "OK". Devido ao ponto de parada definido (Tela 60), o depurador será selecionado. A perspectiva atual, TOTVS Developer, não é a mais indicada para depuração. Por isso, você será convidado a trocá-la para a perspectiva TOTVS Debugger.

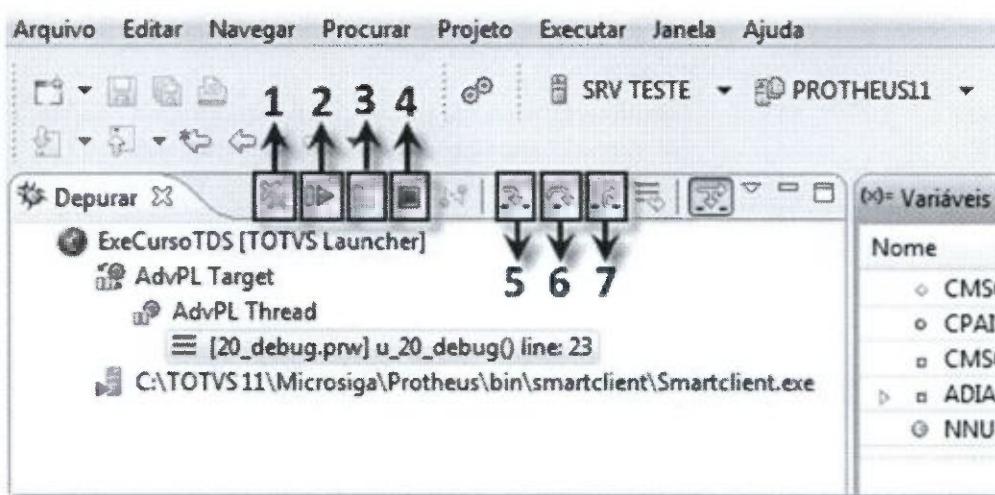


7. Aceite a troca e a perspectiva de depuração lhe será apresentada (Tela-04).



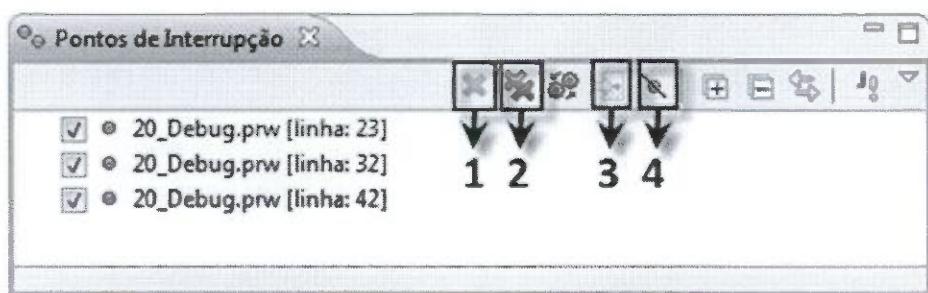
Tela principal do ambiente de depuração, fica alguns comandos de depuração como os Breakpoints(Pontos de Parada), na (Tela-C) área em destaque que apresenta a linha em debug, na figura (Tela-D) apresenta as abas informativas de apoio ao depurador.

Visão "debug" (Tela -06) – apresenta a pilha de execução, indicando o "caminho" efetuado até chegar ao ponto de parada. Também temos a barra de ferramentas desta visão, com as funcionalidades:



1. Remove histórico de execução já finalizadas;
2. Continua a execução até encontrar novo ponto de parado ou o término do programa;
3. Pausa a execução;
4. Encerra a execução;
5. Executa um passo, se for chamada de método ou função, para na primeira instrução desta ou a tecla "F5" para percorrer uma Linha;
6. Executa um passo, se for chamada de método ou função, execute-a sem parar dentro desta ou "F6" para pular de Linha;
7. Executa até finalizar (retomar) do método ou função ou a tecla "F8" para executar.

Visão breakpoints (Tela - 07) - apresenta a lista de pontos de paradas definidos. Também temos a barra de ferramentas desta visão, com as funcionalidades:



1. Remove os pontos de parada selecionados;
2. Remove todos;
3. Abre e posiciona-se no arquivo-fonte associado;
4. Ignora todos os pontos de parada.

Tabela de dados - apresenta as tabelas abertas durante o processamento

Objeto	Tipo	Conteúdo
SM0	F	File:c:\totvs 11\microsiga\protheus_data\system\sigamat.emp; Filter : ; C
XX8	F	File:c:\totvs 11\microsiga\protheus_data\system\x08.dbf; Filter : ; C
XX9	F	File:c:\totvs 11\microsiga\protheus_data\system\x09.dbf; Filter : ; C
PROFALIAS	F	File:c:\totvs 11\microsiga\protheus_data\profile\profile.usr; Filter : ; C
SIX	F	File:c:\totvs 11\microsiga\protheus_data\system\sia990.dbf; Filter : ; C
SX2	F	File:c:\totvs 11\microsiga\protheus_data\system\sx2990.dbf; Filter : ; C
SX3	F	File:c:\totvs 11\microsiga\protheus_data\system\sx3990.dbf; Filter : ; C
SX6	F	File:c:\totvs 11\microsiga\protheus_data\system\sx6990.dbf; Filter : ; C

Adv/PL Commands – permiti ao desenvolvedor executar comandos Adv/PL em tempo de execução com resposta imediata conforme figura.

```
nNum2
3
(nNum1 > 5).And. (nNum2 <> 3)
.F.
```

Variables – permiti verificar os valores das variáveis declaradas.

Nome	Valor
CMMSG1	2. Ponto de Parada
CPAISLOC	BRA
LWSISPORTAL	.F.
OAPP	O
HELP	.F.
USIGAADV	.F.

Legenda dos Escopos das Variáveis

Legendas dos Escopos das Variáveis.

- Variável Local
- Variável Static
- Variável Private
- Variável Public

Expressions – Informa o conteúdo de matriz, variáveis e objetos.

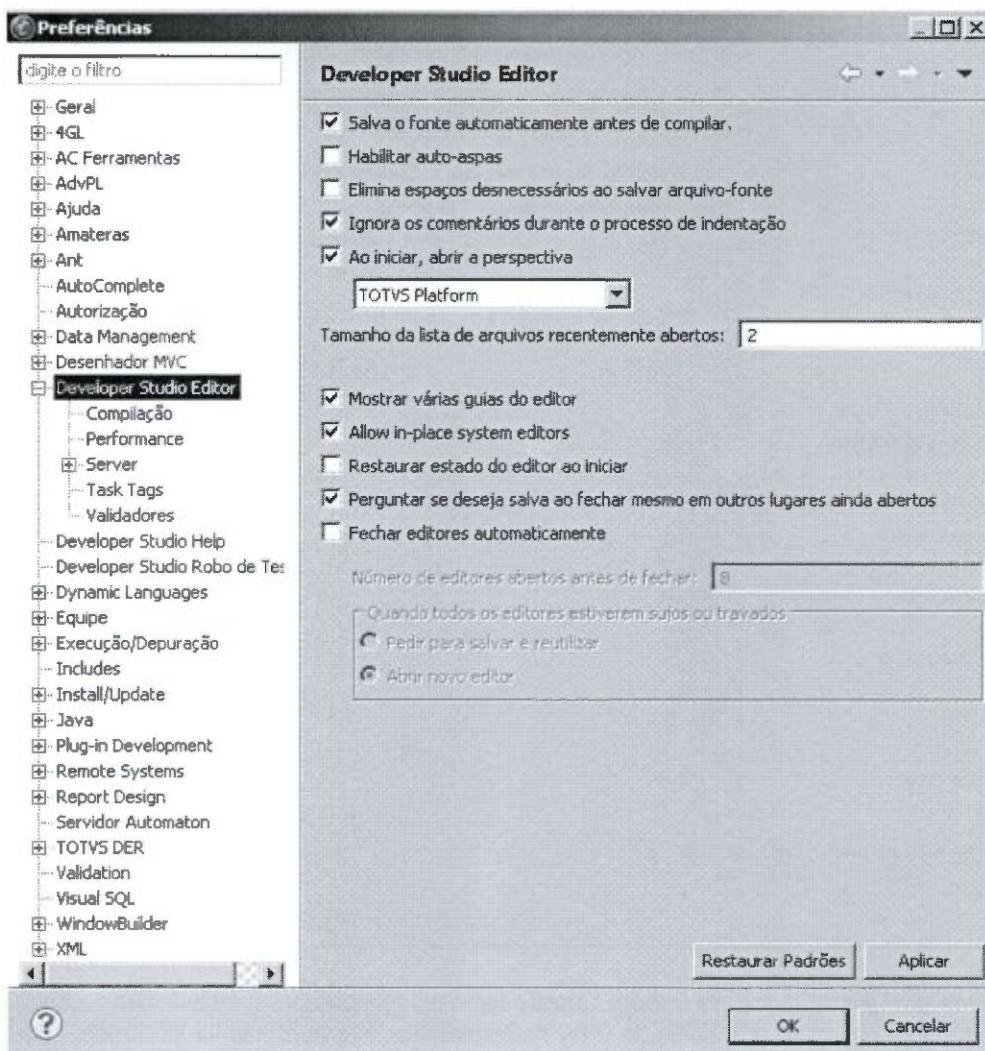
Formação Programação ADVPL



Nome	Valor
XIV "ADIAS"	
● ADIAS[1]	Debug no TDS
● ADIAS[2]	04/10/2012
● ADIAS[3]	Curso Totvs Developer Studio
+ Add new expression	

7.2.6. Preferências de configurações

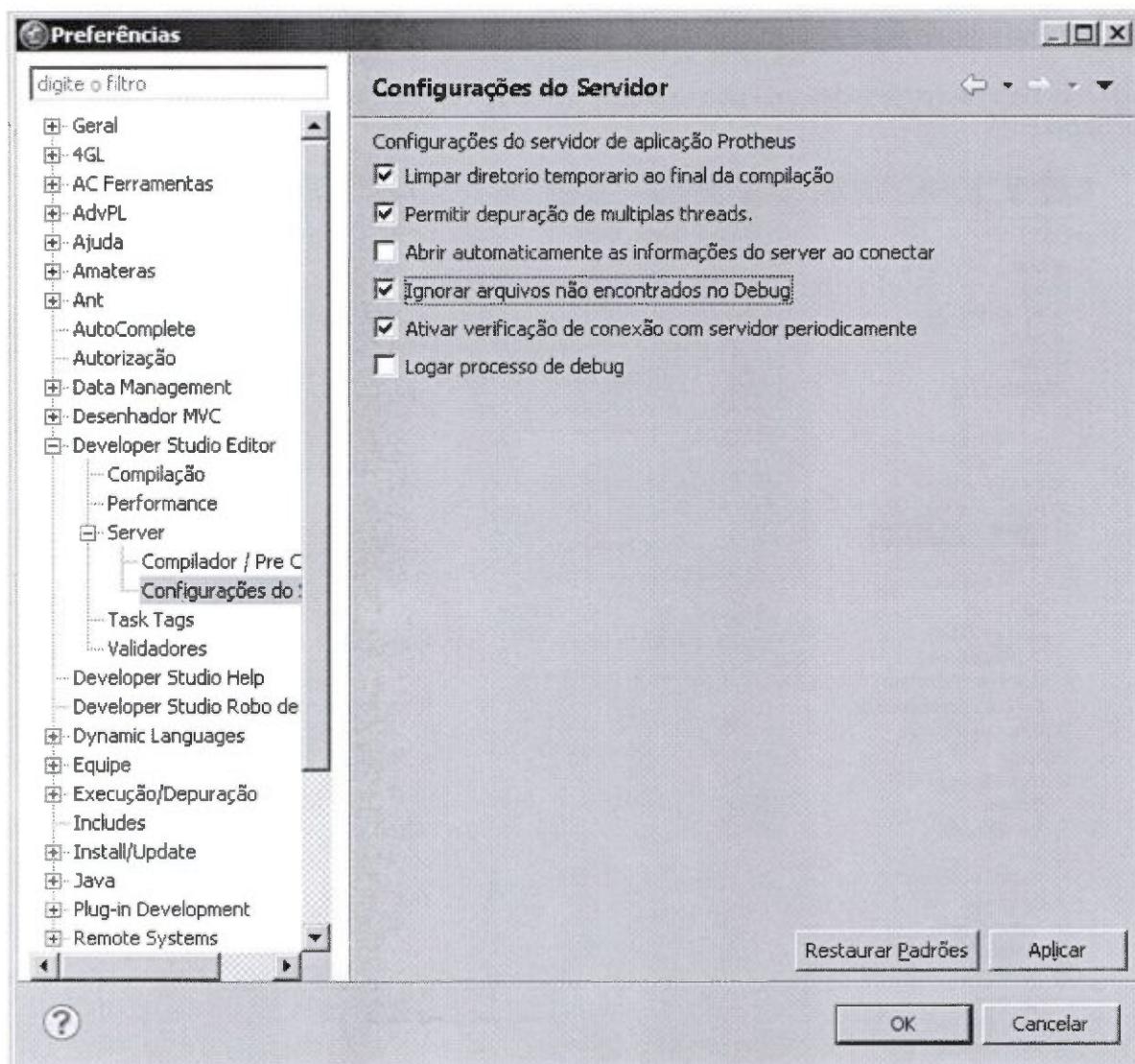
Ao executar o debug possui preferências que ajuda no desenvolvimento na barra de ferramentas Janela > Preferencia > Developer Studio Editor.



Opções disponíveis:

- Salvar o fonte automaticamente antes de compilar;
- Habilitar auto-aspas: Quando preencher uma aspa fecha automaticamente
- Eliminar espaços desnecessários ao salvar arquivo-fonte
- Ignorar os comentários durante o processo de endentação: Quando usamos o identador automático ignora os comentários
- Ao iniciar, Abri a perspectiva: Ao abrir o TDS podes informar qual perspectiva irá receber o foco.
- Tamanho da lista de arquivos recentes aberto.

Dentro da estrutura da raiz, possui alguma opção muito importante para o desenvolvimento em Server > Configurações do servidor.



- Ignorar arquivos não encontrados no Debug: Quando estamos debugando um fonte e não possui o PRW da função chamada, abre uma guia com nome do PRW notificando que o arquivo não foi localizado, para não ficar aparecendo essas notificações selecionar essa opção.

- **Permitir depuração de multiplas threads:** Quando estamos debugando WebService ou ADVPL Web deixar essa opção habilitada. Não é aconselhado deixar essa opção habilitada para depuração normal, trava o RPO deixando ninguém acessar o Protheus.

7.2.7. Criação de um Programa

Um programa de computador nada mais é do que um grupo de comandos logicamente dispostos com o objetivo de executar determinada tarefa. Esses comandos são gravados em um arquivo texto que é transformado em uma linguagem executável por um computador através de um processo chamado compilação. A compilação substitui os comandos de alto nível (que os humanos comprehendem) por instruções de baixo nível (comprendida pelo sistema operacional em execução no computador). No caso do AdvPI, não é o sistema operacional de um computador que irá executar o código compilado, mas sim o AP6 Server.

Dentro de um programa, os comandos e funções utilizados devem seguir regras de sintaxe da linguagem utilizada, pois caso contrário o programa será interrompido por erros. Os erros podem ser de compilação ou de execução.

- **Erros de compilação** são aqueles encontrados na sintaxe que não permitem que o arquivo de código do programa seja compilado. Podem ser comandos especificados de forma errônea, utilização inválida de operadores, etc.
- **Erros de execução** são aqueles que acontecem depois da compilação, quando o programa está sendo executado. Podem ocorrer por inúmeras razões, mas geralmente se referem a funções não existentes, ou variáveis não criadas ou inicializadas, etc.

7.2.8. Linhas de Programa

As linhas existentes dentro de um arquivo texto de código de programa podem ser linhas de comando, linhas de comentário ou linhas mistas.

7.2.8.1 Linhas de Comando

Linhas de comando possuem os comandos ou instruções que serão executadas. Por exemplo:

```
Local nCnt
Local nSoma := 0

For nCnt := 1 To 10
  nSoma += nCnt
Next nCnt

Return( NIL )
```

7.2.8.2 Linhas de Comentário

Linhas de comentário possuem um texto qualquer, mas não são executadas. Servem apenas para documentação e para tornar mais fácil o entendimento do programa. Existem três formas de se comentar linhas de texto. A primeira delas é utilizar o sinal de * (asterisco) no começo da linha:

```
// Programa para cálculo do total
```

// Autor: Microsiga Software S.A.
 // Data: 2 de outubro de 2001

Outra forma de documentar textos é utilizar as barras transversais juntamente com o asterisco, podendo-se comentar todo um bloco de texto sem precisar comentar linha a linha:

```
/*
Programa para cálculo do total
Autor: Microsiga Software S.A.
Data: 2 de outubro de 2001
*/
```

Todo o texto encontrado entre a abertura (indicada pelos caracteres /*) e o fechamento (indicada pelos caracteres */) é considerado como comentário.

7.2.8.3 Linhas Mistas

O AdvPI também permite que existam linhas de comando com comentário. Isto é possível incluindo-se as duas barras transversais (//) ao final da linha de comando e adicionando-se o texto do comentário:

```
Local nCnt
Local nSoma := 0 // Inicializa a variável com zero para a soma
For nCnt := 1 To 10
  nSoma += nCnt
Next nCnt
```

7.2.8.4 Tamanho da Linha

Assim como a linha física, delimitada pela quantidade de caracteres que pode ser digitado no editor de textos utilizado, existe uma linha considerada linha lógica. A linha lógica, é aquela considerada para a compilação como uma única linha de comando.

A princípio, cada linha digitada no arquivo texto é diferenciada após o pressionamento da tecla <Enter>. Ou seja, a linha lógica, é a linha física no arquivo. Porém algumas vezes, por limitação física do editor de texto ou por estética, pode-se "quebrar" a linha lógica em mais de uma linha física no arquivo texto. Isto é efetuado utilizando-se o sinal de ponto-e-vírgula (;).

```
If !Empty(cNome) .And. !Empty(cEnd) .And. ; <enter>
  !Empty(cTel) .And. !Empty(cFax) .And. ; <enter>
  !Empty(cEmail)
```

GravaDados(cNome,cEnd,cTel,cFax,cEmail)

Endif

Neste exemplo existe uma linha de comando para a checagem das variáveis utilizadas. Como a linha torna-se muito grande, pode-se dividi-la em mais de uma linha física utilizando o sinal de ponto-e-vírgula. Se um sinal de ponto-e-vírgula for esquecido nas duas primeiras linhas, durante a execução do programa ocorrerá um erro, pois a segunda linha física será considerada como uma segunda linha de comando na compilação. E durante a execução esta linha não terá sentido.

7.2.8.5 Estrutura de um Programa

Apesar de não ser uma linguagem de padrões rígidos com relação à estrutura do programa, é importante identificar algumas de suas partes. Considere o programa de exemplo abaixo:

```
/*=====
| Programa: Cálculo do Fatorial
| Autor : Microsiga Software S.A.
| Data  : 02 de outubro de 2001
=====
*/
Local nCnt
Local nResultado := 1 // Resultado do fatorial
Local nFator   := 5 // Número para o cálculo

// Cálculo do fatorial
For nCnt := nFator To 1 Step -1
nResultado *= nCnt
Next nCnt

// Exibe o resultado na tela, através da função alert
Alert("O factorial de " + cValToChar(nFator) +
      " é " + cValToChar(nResultado))

// Termina o programa
Return
```

Pode-se classificar um programa em AdvPI em quatro partes básicas:

- Área de Identificação
- Área de Ajustes Iniciais
- Corpo do Programa
- Área de Encerramento

7.2.8.6A Área de Identificação

Esta é uma área que não é obrigatória e é dedicada a documentação do programa. Quando existente, contém apenas comentários explicando a sua finalidade, data de criação, autor, etc, e aparece no começo do programa, antes de qualquer linha de comando.

O formato para esta área não é definido. Pode-se colocar qualquer tipo de informação desejada e escolher a formatação apropriada.

Opcionalmente pode-se incluir definições de constantes utilizadas no programa ou inclusão de arquivos de cabeçalho nesta área.

7.2.8.7A Área de Ajustes Iniciais

Nesta área geralmente se fazem os ajustes iniciais, importantes para o correto funcionamento do programa. Entre os ajustes se encontram declarações de variáveis, inicializações, abertura de arquivos, etc. Apesar do AdvPI não ser uma linguagem rígida e as variáveis poderem ser declaradas em qualquer lugar do programa, é aconselhável fazê-lo nesta área visando tornar o código mais legível e facilitar a identificação de variáveis não utilizadas.

7.2.8.8 O Corpo do Programa

É nesta área que se encontram as linhas de código do programa. É onde se realiza a tarefa necessária através da organização lógica destas linhas de comando.

Espera-se que as linhas de comando estejam organizadas de tal modo que no final desta área o resultado esperado seja obtido, seja ele armazenado em um arquivo ou em variáveis de memória, pronto para ser exibido ao usuário através de um relatório ou na tela.

7.2.8.9A Área de Encerramento

É nesta área onde as finalizações são efetuadas. É onde os arquivos abertos são fechados, e o resultado da execução do programa é utilizado. Pode-se exibir o resultado armazenado em uma variável ou em um arquivo ou simplesmente finalizar, caso a tarefa já tenha sido toda completada no corpo do programa. É nesta área que se encontra o encerramento do programa. Todo programa em AdvPI deve sempre terminar com a palavra chave return.

7.2.8.10 Tipos de Dados

O AdvPI não é uma linguagem de tipos rígidos (strongly typed), o que significa que variáveis de memória podem receber diferentes tipos de dados durante a execução do programa. Variáveis podem também conter objetos, mas os tipos primários da linguagem são:

7.2.8.11 Numérico

O AdvPI não diferencia valores inteiros de valores com ponto flutuante, portanto pode-se criar variáveis numéricas com qualquer valor dentro do intervalo permitido. Os seguintes elementos são do tipo de dado numérico:

2
43.53
0.5
0.00001
1000000

Uma variável do tipo de dado numérico pode conter um número de dezoito dígitos incluindo o ponto flutuante, no intervalo de 2.2250738585072014 E-308 até 1.7976931348623158 E+308.

7.2.8.12 Lógico

Valores lógicos em AdvPI são identificados através de .T. ou .Y. para verdadeiro e .F. ou .N. para falso (independentemente se os caracteres estiverem em maiúsculo ou minúsculo).

7.2.8.13 Caracter

Strings ou cadeias de caracteres são identificadas em AdvPI por blocos de texto entre aspas duplas ("") ou aspas simples (''):

"Olá mundo!"
'Esta é uma string'
"Esta é 'outra' string"

Uma variável do tipo caracter pode conter strings com no máximo 1 Mb, ou seja, 1048576 caracteres.

7.2.8.14 Data

O AdvPI tem um tipo de dados específico para datas. Internamente as variáveis deste tipo de dado são armazenadas como um número correspondente a data. Variáveis do tipo de dados Data não podem ser declaradas diretamente, e sim através da utilização de funções específicas como por exemplo ctod que converte uma string para data.

7.2.8.15 Matriz (Array)

Matrizes são um tipo de dado especial. É a disposição de outros elementos em colunas e linhas. O AdvPI suporta matrizes uni ou multidimensionais. Os elementos de uma matriz são acessados através de índices numéricos iniciados em 1, identificando a linha e coluna para quantas dimensões existirem.

Uma matriz pode conter no máximo 100000 elementos, independentemente do número de dimensões.

Matrizes devem ser utilizadas com cautela, pois se forem muito grandes podem exaurir a memória do servidor.

7.2.8.16 Bloco de Código

O bloco de código é um tipo de dado especial. É utilizado para armazenar instruções escritas em AdvPI que poderão ser executadas posteriormente.

Criação e Atribuição de Variáveis

Variáveis de memória são um dos recursos mais importantes de uma linguagem. São áreas de memória criadas para armazenar informações utilizadas por um programa para a execução de tarefas. Por exemplo, quando o usuário digita uma informação qualquer, como o nome de um produto, em uma tela de um programa esta informação é armazenada em uma variável de memória para posteriormente ser gravada ou impressa.

A partir do momento que uma variável é criada, não é necessário mais se referenciar ao seu conteúdo, e sim ao seu nome. O nome de uma variável é um identificador único que segue duas regras:

- **Máximo de 10 caracteres:** O AdvPI não impede a criação de uma variável de memória cujo nome contenha mais de 10 caracteres, porém apenas os 10 primeiros serão considerados para a localização do conteúdo

armazenado. Portanto se forem criadas duas variáveis cujos 10 primeiros caracteres forem iguais, como nTotalGeralAnual e nTotalGeralMensal, as referências a qualquer uma delas no programa resultarão o mesmo.

- **LIMITAÇÃO DE CARACTERES NO NOME:** Os nomes das variáveis devem sempre começar por uma letra ou o caractere de sublinhado (_). No restante, pode conter letras, números e o caractere de sublinhado. Qualquer outro caractere, incluindo espaços em branco, não são permitidos.

Exemplo de declarações erradas:

- **1CÓDIGO** (Inicia por um número)
- **M CARGO** (contém um espaço em branco)
- **LOCAL** (palavra reservada do AdvPI)

O AdvPI não é uma linguagem de tipos rígidos para variáveis, ou seja, não é necessário informar o tipo de dados que determinada variável irá conter no momento de sua declaração, e o seu valor pode mudar durante a execução do programa. Também não há necessidade de declarar variáveis em uma seção específica do seu código fonte, embora seja aconselhável declarar todas as variáveis necessárias no começo, tornando a manutenção mais fácil e evitando a declaração de variáveis desnecessárias.

Para declarar uma variável deve-se utilizar um identificador de escopo, seguido de uma lista de variáveis separadas por vírgula (,). Um identificador de escopo é uma palavra chave que indica a que contexto do programa a variável declarada pertence.

7.2.8.17 O Contexto de Variáveis dentro de um Programa

As variáveis declaradas em um programa ou função, são visíveis de acordo com o escopo onde são definidas. Como também do escopo depende o tempo de existência das variáveis. A definição do escopo de uma variável é efetuada no momento de sua declaração.

- Local
- Static
- Private
- Public

7.2.8.18 Variáveis de escopo local

Variáveis de escopo local são pertencentes apenas ao escopo da função onde foram declaradas e devem ser explicitamente declaradas com o identificador LOCAL, como no exemplo:

```
Function Pai()
Local nVar := 10, aMatriz := {0,1,2,3}
<comandos>
Filha()
<mais comandos>
Return()
```

Neste exemplo, a variável nVar foi declarada como local e atribuída com o valor 10, quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Quando a execução da função Pai terminar, a variável nVar é destruída.

Qualquer variável, com o mesmo nome no programa que chamou a função Pai, não é afetada. Variáveis de escopo local são criadas automaticamente, cada vez que a função onde forem declaradas for ativada. Elas continuam a existir e mantêm seu valor até o fim da ativação da função (ou seja, até que a função retorne o controle para o código que a executou). Se uma função é chamada recursivamente (por exemplo, chama a si mesma), cada chamada em recursão cria um novo conjunto de variáveis locais.

7.2.8.19 Variáveis de escopo static

Variáveis de escopo static funcionam basicamente como as variáveis de escopo local, mas mantêm seu valor através da execução e devem ser declaradas explicitamente no código, com o identificador STATIC.

O escopo das variáveis static depende de onde são declaradas. Se forem declaradas dentro do corpo de uma função ou procedimento, seu escopo será limitado àquela rotina. Se forem declaradas fora do corpo de qualquer rotina, seu escopo afeta a todas as funções declaradas no fonte.

Neste exemplo, a variável nVar é declarada como static e inicializada com o valor 10:

```
Function Pai()
  Static nVar := 10
  <comandos>
  Filha()
  <mais comandos>
  Return()
```

Quando a função Filha é executada, nVar ainda existe mas não pode ser acessada. Diferente de variáveis declaradas como LOCAL ou PRIVATE, nVar continua a existir e mantém seu valor atual quando a execução da função Pai termina. Entretanto, somente pode ser acessada por execuções subsequentes da função Pai.

7.2.8.20 Variáveis de escopo Private

A declaração é opcional para variáveis privadas. Mas podem ser declaradas explicitamente com o identificador PRIVATE.

Adicionalmente, a atribuição de valor a uma variável não criada anteriormente, de forma automática cria-se a variável como privada. Uma vez criada, uma variável privada continua a existir e mantém seu valor até que o programa ou função onde foi criada termine (ou seja, até que a função onde foi feita retorno para o código que a executou). Neste momento, é automaticamente destruída. É possível criar uma nova variável privada com o mesmo nome de uma variável já existente. Entretanto, a nova (duplicada) variável pode apenas ser criada em um nível de ativação inferior ao nível onde a variável foi declarada pela primeira vez (ou seja, apenas em uma função chamada pela função onde a variável já havia sido criada).

A nova variável privada esconderá qualquer outra variável privada ou pública (veja a documentação sobre variáveis públicas) com o mesmo nome enquanto existir. Uma vez criada, uma variável privada é visível em todo o programa, enquanto não for destruída automaticamente, quando a rotina que a criou terminar ou uma outra variável privada com o mesmo nome for criada em uma sub-função chamada (neste caso, a variável existente torna-se inacessível até que a nova variável privada seja destruída).

Em termos mais simples, uma variável privada é visível dentro da função de criação e todas as funções chamadas por esta, a menos que uma função chamada crie sua própria variável privada com o mesmo nome.

Por exemplo:

```
Function Pai()
Private nVar := 10
<comandos>
Filha()
<mais comandos>
Return()
```

Neste exemplo, a variável nVar é criada com escopo private e inicializada com o valor quando a função Filha é executada, nVar ainda existe e, diferente de uma variável de escopo local, pode ser acessada pela função Filha. Quando a função Pai terminar, nVar será destruída e qualquer declaração de nVar anterior se tornará acessível novamente.

7.2.8.21 Variáveis de escopo public

Podem-se criar variáveis de escopo public dinamicamente, no código com o identificador PUBLIC. As variáveis deste escopo continuam a existir e mantêm seu valor até o fim da execução da thread (conexão).

É possível criar uma variável de escopo private com o mesmo nome de uma variável de escopo public existente, entretanto, não é permitido criar uma variável de escopo public com o mesmo nome de uma variável de escopo private existente.

Uma vez criada, uma variável de escopo public é visível em todo o programa em que foi declarada, até que seja escondida por uma variável de escopo private, criada com o mesmo nome. A nova variável de escopo private criada esconde a variável de escopo public existente, e esta se tornará inacessível até que a nova variável private seja destruída.

Por exemplo:

```
Function Pai()
Public nVar := 10
<comandos>
Filha()
<mais comandos>
Return()
```

Neste exemplo, nVar é criada como public e inicializada com o valor 10. Quando a função Filha é executada, nVar ainda existe e pode ser acessada. Diferente de variáveis locais ou privates, nVar ainda existe após o término da execução da função Pai. Diferentemente dos outros identificadores de escopo, quando uma variável é declarada como pública sem ser inicializada, o valor assumido é falso (.F.) e não nulo (nil).

7.2.8.22 Atribuição de variáveis

Uma vez que um valor lhe seja atribuído, o tipo de dado de uma variável é igual ao tipo de dado do valor atribuído. Ou seja, uma variável passa a ser numérica se um número lhe é atribuído, passa a ser caractere se uma string de texto lhe for atribuída etc.

Tipo	Descrição	Indicador
Numeric	Utilizado para valores numéricos inteiros ou decimais, positivos ou negativos.	N
char / character	Utilizado para valores do tipo caracter	C
Date	Utilizado para valores tipo data	D
block / codeblock	Armazena um bloco de código para macro execução	B
Logical	Armazena valores lógicos, verdadeiro (.T.) ou falso (.F.)	L

Exemplos:

```

Local cCaracter: = " " // ou ''
Local nNumerico: = 0
Local dData: = Date()
Local lLogica: = .T.
Local aMatriz: = {}
Local xVar: = NIL

```

Exemplo de declaração tipada:

```

Local cCaracter as character
Local nNumerico as numeric
Local dData as date
Local lLogica as logical
Local aMatriz as array

```

Uma vez que um valor lhe seja atribuído, o tipo de dado de uma variável é igual ao tipo de dado do valor atribuído. Ou seja, uma variável passa a ser numérica se um número lhe é atribuído, passa a ser caractere se uma string de texto lhe for atribuída, etc. Porém mesmo que uma variável seja de determinado tipo de dado, pode-se mudar o tipo da variável atribuindo outro tipo a ela:

```

01 Local xVariavel // Declara a variável inicialmente com valor nulo
02
03 xVariavel := "Agora a variável é caracter..."
04 Alert("Valor do Texto: " + xVariavel)
05
06 xVariavel := 22 // Agora a variável é numérica
07 Alert(cValToChar(xVariavel))
08
09 xVariavel := .T. // Agora a variável é lógica
10 If xVariavel
11 Alert("A variável tem valor verdadeiro...")

```

```

12 Else
13 Alert("A variável tem valor falso...")
14 Endif
15
16 xVariavel := Date() // Agora a variável é data
17 Alert("Hoje é: " + DtoC(xVariavel))
18
19 xVariavel := nil // Nulo novamente
20 Alert("Valor nulo: " + xVariavel)
21
22 Return

```

No programa de exemplo anterior, a variável xVariavel é utilizada para armazenar diversos tipos de dados. A letra "x" em minúsculo no começo do nome é utilizada para indicar uma variável que pode conter diversos tipos de dados, segundo a Notação Húngara (consulte documentação específica para detalhes). Este programa troca os valores da variável e exibe seu conteúdo para o usuário através da função alert. Essa função recebe um parâmetro que deve ser do tipo string de caractere, por isso dependendo do tipo de dado da variável xVariavel é necessário fazer uma conversão antes.

Apesar dessa flexibilidade de utilização de variáveis, deve-se tomar cuidados na passagem de parâmetros para funções ou comandos, e na concatenação (ou soma) de valores. Note a linha 20 do programa de exemplo. Quando esta linha é executada, a variável xVariavel contém o valor nulo. A tentativa de soma de tipos de dados diferentes gera erro de execução do programa. Nesta linha do exemplo, ocorrerá um erro com a mensagem "type mismatch on +".

Excetuando-se o caso do valor nulo, para os demais deve-se sempre utilizar funções de conversão quando necessita-se concatenar tipos de dados diferentes (por exemplo, nas linhas 07 e 17. Note também que quando uma variável é do tipo de dado lógico, ela pode ser utilizada diretamente para checagem (linha 10):

```

If xVariavel
é o mesmo que
If xVariavel = .T.

```

7.2.9. Operadores da linguagem ADVPL

7.2.9.1 Operadores Matemáticos

Os operadores utilizados em ADVPL para cálculos matemáticos são:

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
** ou ^	Exponenciação
%	Módulo (Resto da Divisão)

7.2.9.2 Operadores de String

Os operadores utilizados em ADVPL para tratamento de caracteres são:

- + Concatenação de strings (união).
- Concatenação de strings com eliminação dos brancos finais das strings intermediárias.
- \$ Comparação de Substrings (contido em).

7.2.9.3 Operadores Relacionais

Os operadores utilizados em ADVPL para operações e avaliações relacionais são:

- < Comparação Menor
- > Comparação Maior
- = Comparação Igual
- == Comparação Exatamente Igual (para caracteres)
- <= Comparação Menor ou Igual
- >= Comparação Maior ou Igual
- <> ou # ou != Comparação Diferente

7.2.9.4 Operadores Lógicos

Os operadores utilizados em ADVPL para operações e avaliações lógicas são:

- | | |
|------------|------------|
| .And. | E lógico |
| .Or. | OU lógico |
| .Not. ou ! | NÃO lógico |

7.2.9.5 Operadores de Atribuição

Os operadores utilizados em ADVPL para atribuição de valores a variáveis de memória são:

- | | |
|-----------|---|
| := | Atribuição Simples |
| += | Adição e Atribuição em Linha |
| -= | Subtração e Atribuição em Linha |
| *= | Multiplicação e Atribuição em Linha |
| /= | Divisão e Atribuição em Linha |
| **= ou ^= | Exponenciação e Atribuição em Linha |
| %= | Módulo (resto da divisão) e Atribuição em Linha |

7.2.9.6 Atribuição Composta

Os operadores de atribuição composta são uma facilidade da linguagem ADVPL para expressões de cálculo e atribuição. Com eles pode-se economizar digitação:

- | | | |
|-----------|---------|------------|
| += | X += Y | X = X + Y |
| -= | X -= Y | X = X - Y |
| *= | X *= Y | X = X * Y |
| /= | X /= Y | X = X / Y |
| **= ou ^= | X **= Y | X = X ** Y |
| %= | X %= Y | X = X % Y |

7.2.10. Ordem de Precedência dos Operadores

Dependendo do tipo de operador, existe uma ordem de precedência para a avaliação dos operandos. Em princípio, todas as operações, com os operadores, são realizadas da esquerda para a direita se eles tiverem o mesmo nível de prioridade.

A ordem de precedência, ou nível de prioridade de execução, dos operadores em ADVPL é:

- Operadores de Incremento/Decremento pré-fixado
- Operadores de String
- Operadores Matemáticos
- Operadores Relacionais
- Operadores Lógicos
- Operadores de Atribuição
- Operadores de Incremento/Decremento pós-fixado

Em expressões complexas com diferentes tipos de operadores, a avaliação seguirá essa sequencia. Caso exista mais de um operador do mesmo tipo (ou seja, de mesmo nível), a avaliação se dá da esquerda para direita.

7.2.11. Operação de Macro Substituição

O operador de macro substituição, simbolizado pelo "e" comercial (&), é utilizado para a avaliação de expressões em tempo de execução. Funciona como se uma expressão armazenada fosse compilada em tempo de execução, antes de ser de fato executada.

Considere o exemplo:

```
01 X := 10  
02 Y := "X + 1"  
03 B := &Y // O conteúdo de B será 11
```

A variável X é atribuída com o valor 10, enquanto a variável Y é atribuída com a string de caracteres contendo "X + 1".

A terceira linha utiliza o operador de macro. Esta linha faz com que o número 11 seja atribuído à variável B. Pode-se perceber que esse é o valor resultante da expressão em formato de caractere contida na variável Y.

Utilizando-se uma técnica matemática elementar, a substituição, temos que na segunda linha, Y é definido como "X + 1", então pode-se substituir Y na terceira linha:

```
B := &"X + 1"
```

O operador de macro cancela as aspas: 03 B := X + 1

Pode-se perceber que o operador de macro remove as aspas, o que deixa um pedaço de código para ser executado.

Deve-se ter em mente que tudo isso acontece em tempo de execução, o que torna tudo muito dinâmico. Uma utilização interessante é criar um tipo de calculadora, ou avaliador de fórmulas, que determina o resultado de algo que o usuário digita.

O operador de macro tem uma limitação: variáveis referenciadas dentro da string de caracteres (X nos exemplos anteriores) não podem ser locais.

7.2.12. Conversões entre tipos de variáveis

As funções mais utilizadas nas operações entre conversão entre tipos de variáveis são:

CTOD()

Sintaxe	CTOD(cData)
Descrição	Realiza a conversão de uma informação do tipo caractere no formato "DD/MM/AAAA", para uma variável do tipo data.

DTOC()

Sintaxe	DTOC(dData)
Descrição	Realiza a conversão de uma informação do tipo data para em caractere, sendo o resultado no formato "DD/MM/AAAA".

STOD()

Sintaxe	STOD(cData)
Descrição	Realiza a conversão de uma informação do tipo caractere, com conteúdo no formato "AAAAMMDD", em data.

CVALTOCHAR()

Sintaxe	CVALTOCHAR(nValor)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, sem a adição de espaços à informação.

STR()

Sintaxe	STR(nValor)
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando espaços à direita.

STRZERO()

Sintaxe	<code>STRZERO(nValor, nTamanho)</code>
Descrição	Realiza a conversão de uma informação do tipo numérico em uma string, adicionando zeros à esquerda do número convertido, de forma que a string gerada tenha o tamanho especificado no parâmetro.

VAL()

Sintaxe	<code>VAL(cValor)</code>
Descrição	Realiza a conversão de uma informação do tipo caractere em numérica.

StrToKarr()

Sintaxe	<code>StrToKarr (cValue,cToken)</code>
Descrição	Retorna um array, de acordo com os dados passados como parâmetro à função. Desta forma, a função receberá uma string e uma sequência de um ou mais caracteres, que representa um separador, onde cada porção da string entre separadores será um elemento do array retornado.

Transform ()

Sintaxe	<code>Transform (cDado, [cSayPicture])</code>
Descrição	Converte qualquer valor para uma string formatada. Para isso, formata valores, do tipo caractere, data, lógico e numérico, conforme uma string de máscara especificada que inclui uma combinação de strings de template e funções de picture.

7.2.13. Manipulação de strings

As funções mais utilizadas nas operações de manipulação do conteúdo de strings são:

ALLTRIM()

Sintaxe	<code>ALLTRIM(cString)</code>
Descrição	Retorna uma string sem os espaços à direita e à esquerda, referente ao conteúdo informado como parâmetro. <code>ALLTRIM()</code> <code>RTRIM ("right trim")</code> <code>LTRIM ("left trim")</code> .

ASC()

Sintaxe	<code>ASC(cCaractere)</code>
Descrição	Converte uma informação caractere em seu valor, de acordo com a tabela ASCII.

AT()

Sintaxe	AT(cCaractere, cString)
Descrição	Retorna a primeira posição de um caractere ou string, dentro de outra string especificada.

CHR()

Sintaxe	CHR(nASCII)
Descrição	Converte um valor número referente a uma informação da tabela ASCII, no caractere que esta informação representa.

LEN()

Sintaxe	LEN(cString)
Descrição	Retorna o tamanho da string especificada no parâmetro.

RAT()

Sintaxe	RAT(cCaractere, cString)
Descrição	Retorna a última posição de um caractere ou string, dentro de outra string especificada.

STUFF()

Sintaxe	STUFF(cString, nPosInicial, nExcluir, cAdicao)
Descrição	Permite substituir um conteúdo caractere em uma string já existente, especificando a posição inicial para esta adição e o número de caracteres que serão substituídos.

SUBSTR()

Sintaxe	SUBSTR(cString, nPosInicial, nCaracteres)
Descrição	Retorna parte do conteúdo de uma string especificada, de acordo com a posição inicial deste conteúdo na string e a quantidade de caracteres que deverá ser retornada a partir daquele ponto (inclusive).

UPPER()

Sintaxe	UPPER(cString)
Descrição	Retorna uma string com todos os caracteres maiúsculos, tendo como base a string passada como parâmetro.

LOWER()

Sintaxe	LOWER(cString)
Descrição	Retorna uma string com todos os caracteres minúsculos, tendo como base a string passada como parâmetro.

StrTran ()

Sintaxe	StrTran (cString, cSearch)
Descrição	Pesquisa e substitui um conjunto de caracteres de uma string.

Space ()

Sintaxe	Space (nQuantidade)
Descrição	Retorna uma string com uma quantidade especificada de espaços.

Space ()

Sintaxe	Space (nQuantidade)
Descrição	Retorna uma string com uma quantidade especificada de espaços.

Replicate()

Sintaxe	Replicate(cString, nCoun)
Descrição	Gera uma string repetida a partir de outra.

7.2.14. Verificação de tipos de variáveis**TYPE ()**

Sintaxe	TYPE("cVariável")
Descrição	Determina o tipo do conteúdo de uma variável, a qual não foi definida na função em execução.

VALTYPE()

Sintaxe	VALTYPE(cVariável)
Descrição	Determina o tipo do conteúdo de uma variável, a qual foi definida na função em execução.

7.3. Estruturas Básicas de Programação

O ADVPL suporta várias estruturas de controle que permitem mudar a sequência de fluxo de execução de um programa. Estas estruturas permitem a execução de código baseado em condições lógicas e a repetição da execução de pedaços de código em qualquer número de vezes.

Em ADVPL, todas as estruturas de controle podem ser "alinhadas" dentro de todas as demais estruturas, contanto que estejam aninhadas propriamente. Estruturas de controle têm um identificador de início e um de fim, e qualquer estrutura aninhada deve se encontrar entre estes identificadores.

Também existem estruturas de controle para determinar que elementos, comandos, etc. em um programa serão compilados. Estas são as diretivas do pré-processador

#ifdef...#endif e #ifndef...#endif. Consulte a documentação sobre o pré-processador para maiores detalhes.

As estruturas de controle em ADVPL estão divididas em:

- Estruturas de repetição.
- Estruturas de decisão.

7.3.1. Repetição de Comandos

Estruturas de repetição são desejadas para executar uma seção de código mais de uma vez. Por exemplo, imaginando-se a existência de uma função para imprimir um relatório, pode-se desejar imprimi-lo quatro vezes. Claro, pode-se simplesmente chamar a função de impressão quatro vezes em sequência, mas isto se tornaria pouco profissional e não resolveria o problema se o número de relatórios fosse variável.

Em AdvPI existem dois comandos para a repetição de seções de código:

- FOR..NEXT ;
- WHILE...ENDDO;

7.3.1.10 Comando FOR...NEXT

A estrutura de controle FOR...NEXT, ou simplesmente o loop FOR, repete uma seção de código em um número determinado de vezes.

Sintaxe:

```
FOR Variável := nValorInicial TO nValorFinal [STEP nIncremento]  
Comandos...  
[EXIT]  
[LOOP]  
NEXT
```

Variável

Especifica uma variável ou um elemento de uma matriz para atuar como um contador. A variável ou o elemento da matriz não precisa ter sido declarado antes da execução do comando FOR...NEXT. Se a variável não existir, será criada como uma variável privada.

nValorInicial TO nValorFinal	nValorInicial é o valor inicial para o contador; nValorFinal é o valor final para o contador. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
STEP nIncremento	nIncremento é a quantidade que será incrementada ou decrementada no contador após cada execução da seção de comandos. Se o valor de nIncremento for negativo, o contador será decrementado. Se a cláusula STEP for omitida, o contador será incrementado em 1. Pode-se utilizar valores numéricos literais, variáveis ou expressões, contanto que o resultado seja do tipo de dado numérico.
Comandos	Especifica um ou mais instruções de comando AdvPl que serão executadas.
EXIT	Transfere o controle de dentro do comando FOR...NEXT para o comando imediatamente seguinte ao NEXT, ou seja, finaliza a repetição da seção de comandos imediatamente. Pode-se colocar o comando EXIT em qualquer lugar entre o FOR e o NEXT.
LOOP	Retorna o controle diretamente para a cláusula FOR sem executar o restante dos comandos entre o LOOP e o NEXT. O contador é incrementado ou decrementado normalmente, como se o NEXT tivesse sido alcançado. Pode-se colocar o comando LOOP em qualquer lugar entre o FOR e o NEXT.

Exemplo:

```

Local nCnt
Local nSomaPar := 0
For nCnt := 0 To 100 Step 2
nSomaPar += nCnt
Next
Alert( "A soma dos 100 primeiros números pares é: " +
cValToChar(nSomaPar) )
Return
  
```

7.3.1.20 Comando WHILE...ENDDO

A estrutura de controle WHILE...ENDDO, ou simplesmente o loop WHILE, repete uma seção de código enquanto uma determinada expressão resultar em verdadeiro (.T.).

Sintaxe:

```

WHILE lExpressao
  Comandos...
  [EXIT]
  [LOOP]
ENDDO
  
```

lExpressao	Especifica uma expressão lógica cujo valor determina quando os comandos entre o WHILE e o ENDDO são executados. Enquanto o resultado de lExpressao for avaliado como verdadeiro (.T.), o conjunto de comandos são executados.
-------------------	---

Comandos	Especifica um ou mais instruções de comando AdvPI que serão executadas enquanto IExpressao for avaliado como verdadeiro (.T.).
EXIT	Transfere o controle de dentro do comando WHILE...ENDDO para o comando imediatamente seguinte ao ENDDO, ou seja, finaliza a repetição da seção de comandos imediatamente. Pode-se colocar o comando EXIT em qualquer lugar entre o WHILE e o ENDO.
LOOP	Retoma o controle diretamente para a cláusula WHILE sem executar o restante dos comandos entre o LOOP e o ENDDO. A expressão em IExpressao é reavaliada para a decisão se os comandos continuarão sendo executados. Especifica um ou mais instruções de comando AdvPI que serão executadas.

Os comandos entre o WHILE e o ENDDO são executados enquanto o resultado da avaliação da expressão em IExpressao permanecer verdadeiro (.T.). Cada palavra chave WHILE deve ter uma palavra chave ENDDO correspondente.

Exemplo:

```
Local nNumber := nAux := 350
nAux := Int(nAux / 2)
While nAux > 0
  nSomaPar += nCnt
  Next
  Alert( "A soma dos 100 primeiros números pares é: " + cValToChar(nSomaPar) )
Return()
```

7.3.2. Desviando a Execução

Estruturas de desvio são designadas para executar uma seção de código se determinada condição lógica resultar em verdadeiro (.T.). Em AdvPI existem dois comandos para execução de seções de código de acordo com avaliações lógicas. O comando IF...ENDIF e o comando DO CASE...ENDCASE.

7.3.2.10 Comando IF...ENDIF

Executa um conjunto de comandos baseado no valor de uma expressão lógica.

Sintaxe

```
IF IExpressao
  [Comandos...]
  [ELSE
    [Comandos...]
  ENDIF]
```

IExpressao	Especifica uma expressão lógica que é avaliada. Se IExpressao resultar em verdadeiro (.T.), qualquer comando seguinte ao IF e antecedente ao ELSE ou ENDIF (o que ocorrer primeiro) será executado. Se IExpressao resultar em falso (.F.) e a cláusula ELSE for definida, qualquer comando após essa cláusula e anterior ao ENDIF será executada. Se a cláusula ELSE não for definida, todos os comandos entre o IF e o ENDIF são ignorados. Neste caso, a execução do programa continua com o primeiro comando seguinte ao ENDIF.
Comandos	Conjunto de comandos AdvPl que serão executados dependendo da avaliação da expressão lógica em IExpressao.

Pode-se aninhar um bloco de comando IF...ENDIF dentro de outro bloco de comando IF...ENDIF. Porém, para a avaliação de mais de uma expressão lógica, deve-se utilizar o comando DO CASE...ENDCASE.

Exemplo:

```

Local dVencto := CTOD("31/12/16")

If Date() > dVencto
  MsgStop("Vencimento ultrapassado!","Atenção")
Else
  MsgStop("Dentro do prazo","Atenção")
Endif

Return()
  
```

7.3.2.2CASE...ENDCASE

Executa o primeiro conjunto de comandos cuja expressão condicional resulta em verdadeiro (.T.).

Sintaxe:

```

DO CASE
  CASE IExpressao1
    Comandos

  CASE IExpressao2
    Comandos

  CASE IExpressaoN
    Comandos/
    
  OTHERWISE
    Comandos
  END CASE
  
```

CASE
IF Expressao1
Comandos...

Quando a primeira expressão CASE resultante em verdadeiro (.T.) for encontrada, o conjunto de comandos seguinte é executado. A execução do conjunto de comandos continua até que a próxima cláusula CASE, OTHERWISE ou ENDCASE seja encontrada. Ao terminar de executar esse conjunto de comandos, a execução continua com o primeiro comando seguinte ao ENDCASE.

Se uma expressão CASE resultar em falso (.F.), o conjunto de comandos seguinte a esta até a próxima cláusula é ignorado.

Apenas um conjunto de comandos é executado. Estes são os primeiros comandos cuja expressão CASE é avaliada como verdadeiro (.T.). Após a execução, qualquer outra expressão CASE posterior é ignorada (mesmo que sua avaliação resultasse em verdadeiro).

OTHERWISE
Comandos

Se todas as expressões CASE forem avaliadas como falso (.F.), a cláusula OTHERWISE determina se um conjunto adicional de comandos deve ser executado. Se essa cláusula for incluída, os comandos seguintes serão executados e então o programa continuará com o primeiro comando seguinte ao ENDCASE. Se a cláusula OTHERWISE for omitida, a execução continuará normalmente após a cláusula ENDCASE.

O Comando DO CASE...ENDCASE é utilizado no lugar do comando IF...ENDIF quando um número maior do que uma expressão deve ser avaliada, substituindo a necessidade de mais de um comando IF...ENDIF aninhados.

Exemplo:

```
Local nMes := Month(Date())
Local cPeriodo := ""

DO CASE
CASE nMes <= 3
  cPeriodo := "Primeiro Trimestre"
CASE nMes >= 4 .And. nMes <= 6
  cPeriodo := "Segundo Trimestre"
CASE nMes >= 7 .And. nMes <= 9
  cPeriodo := "Terceiro Trimestre"
OTHERWISE
  cPeriodo := "Quarto Trimestre"
ENDCASE

Return()
```

7.4. ARRAYS E BLOCOS DE CÓDIGO

7.4.1. Arrays

Arrays ou matrizes são coleções de valores, semelhantes a uma lista. Uma matriz pode ser criada através de diferentes maneiras.

Cada item em um array é referenciado pela indicação de sua posição numérica na lista, iniciando pelo número 1.

O exemplo a seguir declara uma variável, atribui um array de três elementos a ela, e então exibe um dos elementos e o tamanho do array:

```
Local aLetras // Declaração da Variável
```

```
aLetras := {"A","B","C"}      // Atribuição do array a variável
MsgInfo(aLetras[2])          // Exibe o segundo elemento do array
MsgInfo(cValToChar(Len(aLetras))) // Exibe o tamanho do array
```

O ADVPL permite a manipulação de arrays facilmente. Enquanto que em outras linguagens como C ou Pascal é necessário alocar memória para cada elemento de um array (o que tornaria a utilização de "ponteiros" necessária), o ADVPL se encarrega de gerenciar a memória e torna simples adicionar elementos a um array, utilizando a função AADD():

```
AADD(aLetras,"D") // Adiciona o quarto elemento ao final do array.
```

```
MsgInfo(aLetras[4]) // Exibe o quarto elemento.
MsgInfo(aLetras[5]) // Erro! Não há um quinto elemento no array.
```

7.4.1.1 Arrays como Estruturas

Uma característica interessante do ADVPL é que um array pode conter qualquer tipo de dado: números, datas, lógicos, caracteres, objetos, etc., e ao mesmo tempo. Em outras palavras, os elementos de um array não precisam ser necessariamente do mesmo tipo de dado, em contraste com outras linguagens como C e Pascal.

```
aFunct1 := {"Pedro",32,.T.}
```

Este array contém uma string, um número e um valor lógico. Em outras linguagens como C ou Pascal, este "pacote" de informações pode ser chamado como um "struct" (estrutura em C, por exemplo) ou um "record" (registro em Pascal, por exemplo). Como se fosse na verdade um registro de um banco de dados, um pacote de informações construído com diversos campos. Cada campo tendo um pedaço diferente de dado.

Suponha que no exemplo anterior, o array aFunct1 contenha informações sobre o nome de uma pessoa, sua idade e sua situação matrimonial. Os seguintes #defines podem ser criados para indicar cada posição dos valores dentro de um array:

```
#define FUNCT_NOME 1
#define FUNCT_IDADE 2
#define FUNCT_CASADO 3
```

E considere mais alguns arrays para representar mais pessoas:

```
aFunct2 := {"Maria" , 22, .T.}
aFunct3 := {"Antônio", 42, .F.}
```

Os nomes podem ser impressos assim:

```
Alert(aFunct1[FUNCT_NOME])
Alert(aFunct2[FUNCT_NOME])
Alert(aFunct3[FUNCT_NOME])
```

Agora, ao invés de trabalhar com variáveis individuais, pode-se agrupá-las em um outro array, do mesmo modo que muitos registros são agrupados em uma tabela de banco de dados:

aFuncs := {aFunct1, aFunct2, aFunct3} Que é equivalente a isso:

```
aFuncs := { {"Pedro" , 32, .T.}, ;  
           {"Maria" , 22, .T.}, ;  
           {"Antônio" , 42, .F.} }
```

aFuncs é um array com 3 linhas por 3 colunas. Uma vez que as variáveis separadas foram combinadas em um array, os nomes podem ser exibidos assim:

Local nCount

```
For nCount := 1 To Len(aFuncs)  
  aFuncs[nCount][FUNC_NOME] // pode ser realizado também desta forma:  
Next nCount
```

A variável nCount seleciona que funcionário (ou que linha) é de interesse. Então a constante FUNC_NOME seleciona a primeira coluna daquela linha.

7.4.1.2 Cuidados com Arrays

Arrays são listas de elementos, portanto memória é necessária para armazenar estas informações. Como estes arrays podem ser multidimensionais, a memória necessária será a multiplicação do número de itens em cada dimensão do array, considerando-se o tamanho do conteúdo de cada elemento contido nesta. Portanto o tamanho de um array pode variar muito.

A facilidade da utilização de arrays, mesmo que para armazenar informações em pacotes como descrito anteriormente, não é compensada pela utilização em memória quando o número de itens em um array for muito grande. Quando o número de elementos for muito grande deve-se procurar outras soluções, como a utilização de um arquivo de banco de dados temporário.

7.4.1.3 Inicializando arrays

Algumas vezes o tamanho da matriz é conhecido previamente. Outras vezes o tamanho do array somente será conhecido em tempo de execução.

Se o tamanho do array é conhecido no momento que o programa é escrito, há diversas maneiras de implementar o código:

```
01 Local nCnt  
02 Local aX[10]  
03  
04 Local aY := Array(10)  
05 Local aZ := {0,0,0,0,0,0,0,0,0,0}  
06  
07 For nCnt := 1 To 10  
08   aX[nCnt] := nCnt * nCnt  
09 Next nCnt  
10  
11 Return()
```

Este código preenche o array com uma tabela de quadrados. Os valores serão 1, 4, 9, 16 ... 81, 100. Note que a linha 08 se refere à variável aX, mas poderia também trabalhar com aY ou aZ.

O objetivo deste exemplo é demonstrar três modos de criar um array de tamanho conhecido, no momento da criação do código.

Na linha 02 o array é criado usando `aX[10]`. Isto indica ao ADVPL para alocar espaço para 10 elementos no array. Os colchetes [] são utilizados para indicar o tamanho necessário.

Na linha 04 é utilizada a função `array` com o parâmetro 10 para criar o array, e o retorno desta função é atribuído à variável `aY`. Na linha 04 é efetuado o que se chama "desenhar a imagem do array". Como se pode notar, existem dez 0's na lista encerrada entre chaves {}. Claramente, este método não é o utilizado para criar uma matriz de 1000 elementos.

O terceiro método difere dos anteriores porque inicializa a matriz com os valores definitivos. Nos dois primeiros métodos, cada posição da matriz contém um valor nulo (Nil) e deve ser inicializado posteriormente.

A linha 08 demonstra como um valor pode ser atribuído para uma posição existente em uma matriz, especificando o índice entre colchetes.

7.4.1.4 Se o tamanho do array não é conhecido

Se o tamanho do array não é conhecido até o momento da execução do programa, há algumas maneiras de criar um array e adicionar elementos a ele. O exemplo a seguir ilustra a idéia da criação de um array vazio (sem nenhum elemento), e adição de elementos dinamicamente.

```
01 Local nCnt
02 Local aX[0]
03 Local aY := Array(0)
04 Local aZ := {}
05
06 For nCnt := 1 To nSize
07   AADD(aX, nCnt*nCnt)
08 Next nCnt
09
10 Return()
```

A linha 02 utiliza os colchetes para criar um array vazio. Apesar de não ter nenhum elemento, seu tipo de dado é array.

Na linha 03 a chamada da função `array` cria uma matriz sem nenhum elemento.

Na linha 04 está declarada a representação de um array vazio em ADVPL. Mais uma vez, estão sendo utilizadas as chaves para indicar que o tipo de dados da variável é array. Note que {} é um array vazio (tem o tamanho 0), enquanto {Nil} é um array com um único elemento nulo (tem tamanho 1).

Porque cada uma destes arrays não contém elementos, a linha 07 utiliza a função `AADD()` para adicionar elementos sucessivamente até o tamanho necessário (especificado por exemplo na variável `nSize`).

7.4.1.5 Funções de manipulação de arrays

A linguagem ADVPL possui diversas funções que auxiliam na manipulação de arrays, dentre as quais podemos citar as mais utilizadas:

ARRAY()

Sintaxe	ARRAY(nLinhas, nColunas)
Descrição	A função Array() é utilizada na definição de variáveis de tipo array, como uma opção a sintaxe, utilizando chaves ("{}").

AADD()

Sintaxe	AADD(aArray, xItem)
Descrição	A função AADD() permite a inserção de um item em um array já existente, sendo que este item pode ser um elemento simples, um objeto ou outro array.

ACLONE()

Sintaxe	ACLONE(aArray)
Descrição	A função ACLONE() realiza a cópia dos elementos de um array para outro array integralmente.

ADEL()

Sintaxe	ADEL(aArray, nPosição)
Descrição	A função ADEL() permite a exclusão de um elemento do array. Ao efetuar a exclusão de um elemento, todos os demais são reorganizados de forma que a última posição do array passará a ser nula.

ASIZE()

Sintaxe	ASIZE(aArray, nTamanho)
Descrição	A função ASIZE permite a redefinição da estrutura de um array pré-existente, adicionando ou removendo itens do mesmo.

ASORT()

Sintaxe	ASORT(aArray, nInício, nItens, bOrdem)
Descrição	A função ASORT() permite que os itens de um array sejam ordenados, a partir de um critério pré-estabelecido.

ASCAN()

Sintaxe	ASCAN(aArray, bSeek)
Descrição	A função ASCAN() permite que seja identificada a posição do array que contém uma determinada informação, através da análise de uma expressão descrita em um bloco de código.

AINS()

Sintaxe	<code>AINS(aArray, nPosicao)</code>
Descrição	A função AINS() permite a inserção de um elemento, no array especificado, em qualquer ponto da estrutura do mesmo, diferindo desta forma da função AADD(), a qual sempre insere um novo elemento ao final da estrutura já existente.

7.4.1.6 Cópia de arrays

Conforme comentado anteriormente, um array é uma área na memória, o qual possui uma estrutura que permite as informações serem armazenadas e organizadas das mais diversas formas.

Com base nesse conceito, o array pode ser considerado apenas como um "mapa" ou um "guia" de como as informações estão organizadas e de como elas podem ser armazenadas ou consultadas. Para se copiar um array deve-se levar este conceito em consideração, pois caso contrário o resultado esperado não será obtido na execução da "cópia".

Para "copiar" o conteúdo de uma variável, utiliza-se o operador de atribuição "`:=`", conforme abaixo:

```
nPessoas := 10
nAlunos := nPessoas
```

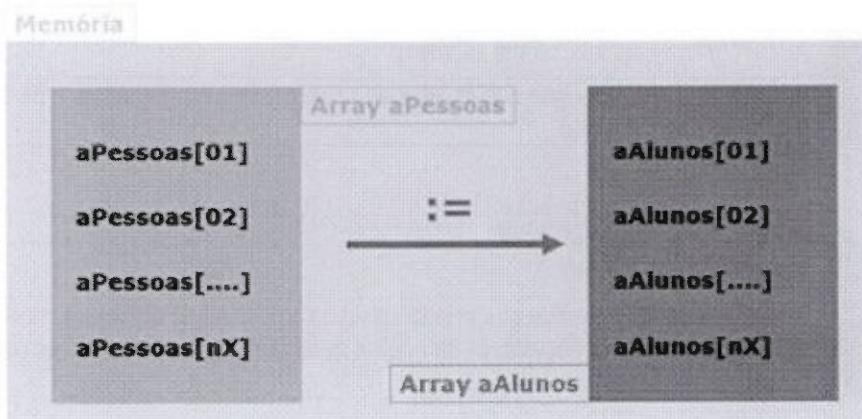
Ao executar a atribuição de `nAlunos` com o conteúdo de `nPessoas`, o conteúdo de `nPessoas` é atribuído a variável `nAlunos`, causando o efeito de cópia do conteúdo de uma variável para outra.

Isto porquê o comando de atribuição copia o conteúdo da área de memória, representada pelo nome "`nPessoas`" para a área de memória representada pelo nome "`nAlunos`". Mas ao utilizar o operador de atribuição "`:=`", da mesma forma que utilizado em variáveis simples, para se copiar um array o efeito é diferente:

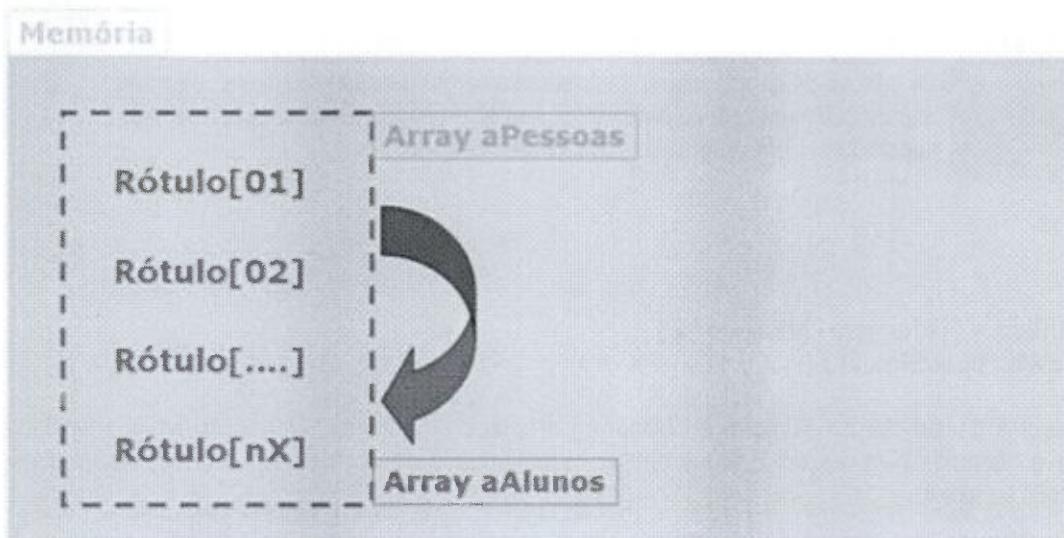
```
aPessoas := {"Ricardo", "Cristiane", "André", "Camila"}
```

```
aAlunos := aPessoas
```

A variável `aPessoas` representa uma área de memória que contém a estrutura de um array ("mapa"), não as informações do array, pois cada informação está em sua própria área de memória.



Desta forma, ao atribuir o conteúdo representado pela variável `aPessoas`, a variável `aAlunos` não está se "copiando" as informações e sim o "mapa" das áreas de memória, em que as informações estão realmente armazenadas.



Como foi copiado o "mapa" e não as informações, qualquer ação utilizando o rótulo `aAlunos` afetará as informações do rótulo `aPessoas`. Com isso ao invés de se obter dois arrays distintos, tem-se o mesmo array com duas formas de acesso (rótulos) diferentes.

Por esta razão deve ser utilizado o comando `ACLONE()`, quando se deseja obter um array com a mesma estrutura e informações que compõe outro array já existente

7.4.2. Blocos de Código

Blocos de código são um conceito existente há muito tempo em linguagens xBase. Não como algo que apareceu da noite para o dia, e sim uma evolução progressiva utilizando a combinação de muitos conceitos da linguagem para a sua implementação.

A linguagem ADVPL possui diversas funções que auxiliam na manipulação de blocos de código, dentre as quais podemos citar as mais utilizadas:

7.4.2.1 EVAL

A função `EVAL()` é utilizada para avaliação direta de um bloco de código, utilizando as informações disponíveis no mesmo de sua execução. Esta função permite a definição e passagem de diversos parâmetros que serão considerados na interpretação do bloco de código.

Sintaxe	<code>EVAL(<bBlock> [, <xVal> [...]])</code>
Descrição	<p>Esta função avalia o bloco de código expresso como < BBLOCK > e retorna o seu valor avaliado. Se os seus são múltiplas expressões dentro do bloco de código , a última expressão será valor desta função .</p> <p>Se o bloco de código requer parâmetros a serem passados para ele , eles são especificados na lista de parâmetros < xVal > e seguintes. Cada parâmetro é separado por uma vírgula dentro da lista de expressão.</p>

Exemplo 01:

```
Local bCodBloc := { | nNumero | nNumero * 2 }
MsgAlert( EVAL( bCodBloc, 2 ) )
```

No exemplo possui a declaração da estrutura do bloco: { | nNumero | nNumero * 2 }, para o bloco funcionar é necessário fazer a chamada pela função Eval() e passar os parâmetros de acordo com o bloco, para ivocada a estrutura retornando o valor 4.

Exemplo 02:

```
Local bBloco := { |x,y| If(x>y,"Maior","Menor") }
MsgAlert(EVal(bBloco, 2, 4))
```

No exemplo possui a declaração da estrutura do bloco: { |x,y| If(x>y,"Maior","Menor") }, para o bloco funcionar é necessário fazer a chamada pela função Eval() e passar os parâmetros foi passado dois parâmetro 2 que será o X e 4 que será Y, retornando o conteúdo Menor.

Exemplo 03

```
Local nInt := 10
```

```
bBloco := { |N| x:= 10, y:= x*N, z:= y/(x*N) }
nValor := EVAL(bBloco, nInt)
```

// O retorno será dado pela avaliação da última ação da lista de expressões, no caso "z".
// Cada uma das variáveis definidas, em uma das ações da lista de expressões, fica
// disponível para a próxima ação.

// Desta forma temos:
// N → recebe nInt como parâmetro (10)
// X → tem atribuído o valor 10 (10)
// Y → resultado da multiplicação de X por N (100)
// Z → resultado da divisão de Y pela multiplicação de X por N (100 / 100) → 1

7.4.2.2 DBEVAL

A função DBEval() permite que todos os registros de uma determinada tabela sejam analisados, e para cada registro será executado o bloco de código definido.

bBloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada registro do alias ativo.
bFor	Condição para continuação da análise dos registros, com o efeito de uma estrutura For ... Next.
bWhile	Condição para continuação da análise dos registros, com o efeito de uma estrutura While ... End.

Exemplo 01:

```

dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()

While !Eof() .And. X5_FILIAL == xFilial("SX5") .And.; X5_TABELA <= mv_par02
  nCnt++
  dbSkip()
EndDo

```

O mesmo pode ser reescrito com o uso da função DBEVAL():

```
dbEval({||x|nCnt++ },,{||X5_FILIAL==xFilial("SX5") .And. X5_TABELA <= mv_par02 })
```

Exemplo 02:

```

dbSelectArea("SX5")
dbSetOrder(1)
dbGotop()

While ! Eof() .And. X5_TABELA == cTabela
  AADD(aTabela,{X5_CHAVE, Capital(X5_DESCRI)})
  dbSkip()
EndDo

```

O mesmo pode ser reescrito com o uso da função DBEVAL():

```
dbEval({||aAdd(aTabela,{X5_CHAVE,Capital(X5_DESCRI)})},,{||X5_TABELA==cTabela})
```

7.4.2.3 AEVAL

A função AEVAL() permite que todos os elementos de um determinada array sejam analisados e para cada elemento será executado o bloco de código definido.

Sintaxe: AEVAL(aArray, bBloco, nInício, nFim)

aArray	Array que será avaliado na execução da função.
bBloco	Bloco de código principal, contendo as expressões que serão avaliadas para cada elemento do array informado.
nInicio	Elemento inicial do array, a partir do qual serão avaliados os blocos de código.
nFim	Elemento final do array, até o qual serão avaliados os blocos de código.

Exemplo 01:

```

AADD(aCampos,"A1_FILIAL")
AADD(aCampos,"A1_COD")
SX3->(dbSetOrder(2))

For nX:=1 To Len(aCampos)
  If SX3->(dbSeek(aCampos[nX]))
    aAdd(aTitulos,AllTrim(SX3->X3_TITULO))
  EndIf

  Next nX

```

O mesmo pode ser reescrito com o uso da função AEVAL():

```

aEval(aCampos,{|x|SX3->(dbSeek(x)),IIF(Found(),AAdd(aTitulos,;
  AllTrim(SX3->X3_TITULO)))})

```

7.5. Funções

A maior parte das rotinas que queremos escrever em programas são compostas de um conjunto de comandos, rotinas estas que se repetem ao longo de todo o desenvolvimento. Uma função nada mais é do que um conjunto de comandos que para ser utilizada basta chamá-la pelo seu nome.

7.5.1. Tipos e escopos de funções

Em ADVPL podem ser utilizados os seguintes tipos de funções:

- **Function()**
- **User Function()**
- **Static Function()**
- **Main Function()**

Funções ADVPL convencionais, restritas ao desenvolvimento da área de Inteligência Protheus da Microsiga.

O interpretador ADVPL distingue nomes de funções do tipo Function() com até dez caracteres. A partir do décimo caracter, apesar do compilador não indicar quaisquer tipos de erros, o interpretador ignorará os demais caracteres.

7.5.2. User Function()

As "User Defined Functions" ou funções definidas pelos usuários, são tipos especiais de funções implementados pelo ADVPL, para garantir que desenvolvimentos específicos não realizados pela Inteligência Protheus da Totvs sobreponham as funções padrões desenvolvidas para o ERP.

O interpretador ADVPL considera que o nome de uma User Function é composto pelo nome definido para a função, precedido dos caracteres "U_". Desta forma a User Function XMAT100 será tratada pelo interpretador como "U_XMAT100"

Funções do tipo User Function são acessíveis por quaisquer outras funções em uso pela aplicação, desde que em sua chamada sejam utilizados os caracteres "U_", em conjunto com o nome da função.

7.5.3. Static Function()

Funções ADVPL tradicionais, cuja visibilidade está restrita às funções descritas no mesmo arquivo de código fonte no qual estão definidas.

Exemplo:

```
//Fonte FINA010.PRW
Function FINA010()
CriaSx1("FIN010")
Return

Static Function CRIASX1()
//Fonte FINA020.PRW Function FINA020()
CriaSx1("FIN020")
Return
Static Function CRIASX1()
```

No exemplo acima, existem duas funções denominadas CRIASX1(), definidas em arquivos de código fonte distintos: FINA010.PRW e FINA020.PRW.

A função FINA010() terá visibilidade apenas da função CRIASX1(), definida no arquivo de código fonte FINA010.PRW, sendo que o mesmo ocorre com a função FINA020().

Este recurso permite isolar funções de uso exclusivo de um arquivo de código fonte, evitando a sobreposição ou duplicação de funções na aplicação.

Neste contexto as Static Functions() são utilizadas para:

- Padronizar o nome de uma determinada função, que possui a mesma finalidade, mas que sua implementação pode variar de acordo com a necessidade da função principal / aplicação.
- Redefinir uma função padrão da aplicação, adequando-a as necessidades específicas de uma função principal / aplicação.
- Proteger funções de uso específico de um arquivo de código fonte / função principal.

7.5.4. Main Function()

Main Function() é outro tipo de função especial do ADVPL incorporado, para permitir tratamentos diferenciados na aplicação ERP.

Uma Main Function() tem a característica de poder ser executada através da tela inicial de parâmetros do client do ERP (Microsiga Protheus SmartClient), da mesma forma que uma User Function, com a diferença que as Main Functions somente podem ser desenvolvidas com o uso da autorização de compilação, tornando sua utilização restrita a Inteligência Protheus da Microsiga.

Na aplicação ERP é comum o uso das Main Functions(), nas seguintes situações:

- Definição dos módulos da aplicação ERP: Main Function Sigaadv()
- Definição de atualizações e updates: AP710TOMP120()
- Atualizações específicas de módulos da aplicação ERP: UpdateATF()

7.5.5. Passagem de parâmetros entre funções

Como mencionado anteriormente os parâmetros das funções descritas, utilizando a linguagem ADVPL são posicionais, ou seja, na sua passagem não importa o nome da variável e sim a sua posição dentro da lista de parâmetros. Complementando esta definição, podem ser utilizadas duas formas distintas de passagens de parâmetros para funções descritas na linguagem ADVPL:

- Passagem de parâmetros por conteúdo.
- Passagem de parâmetros por referência.

7.5.6. Passagem de parâmetros por conteúdo

A passagem de parâmetros por conteúdo é a forma convencional de definição dos parâmetros recebidos pela função chamada, na qual a função recebe os conteúdos passados pela função chamadora, na ordem com os quais são informados.

User Function CalcFator(nFator)

```

Local nCnt
Local nResultado := 0

For nCnt := nFator To 1 Step -1
  nResultado *= nCnt
Next nCnt

Alert("O fatorial de " + cValToChar(nFator) + ; " é " + cValToChar(nResultado))

Return()
  
```

Avaliando a função CalcFator() descrita anteriormente, podemos verificar que a mesma recebe como parâmetro para sua execução a variável nFator.

Com base nesta função, podemos descrever duas forma de passagem de parâmetros por conteúdo:

- Passagem de conteúdos diretos.
- Passagem de variáveis como conteúdos.

Exemplo 01 – Passagem de conteúdos diretos

```
User Function DirFator()  
Local nResultado := 0  
nResultado := CalcFator(5)
```

A passagem de conteúdos diretos implica na definição explícita do valor do parâmetro, na execução da chamada da função. Neste caso foi informado o conteúdo 5 (numérico) como conteúdo para o primeiro parâmetro da função CalcFator. Como a linguagem ADVPL trata os parâmetros de forma posicional, o conteúdo 5 será atribuído diretamente à variável, definida como primeiro parâmetro da função chamada, no nosso caso, nFator.

Por ser uma atribuição de parâmetros por conteúdo, o interpretador da linguagem basicamente executa uma operação de atribuição normal, ou seja, nFator := 5

Exemplo 02 – Passagem de variáveis como conteúdos

```
User Function DirFator()  
Local nResultado := 0  
Local nFatorUser := 0  
nFatorUser := GetFator() // Função ilustrativa na qual o usuário informa o fator a ser utilizado.  
nResultado := CalcFator(nFatorUser)
```

A passagem de conteúdos como variáveis implica na utilização de variáveis de apoio para executar a chamada de uma função. Neste caso foi informada a variável nFatorUser, a qual será definida pelo usuário através da função ilustrativa GetFator(). O uso de variáveis de apoio flexibiliza a chamada de outras funções, pois elas serão parametrizadas de acordo com as necessidades daquele processamento específico, no qual se encontra a função chamadora.

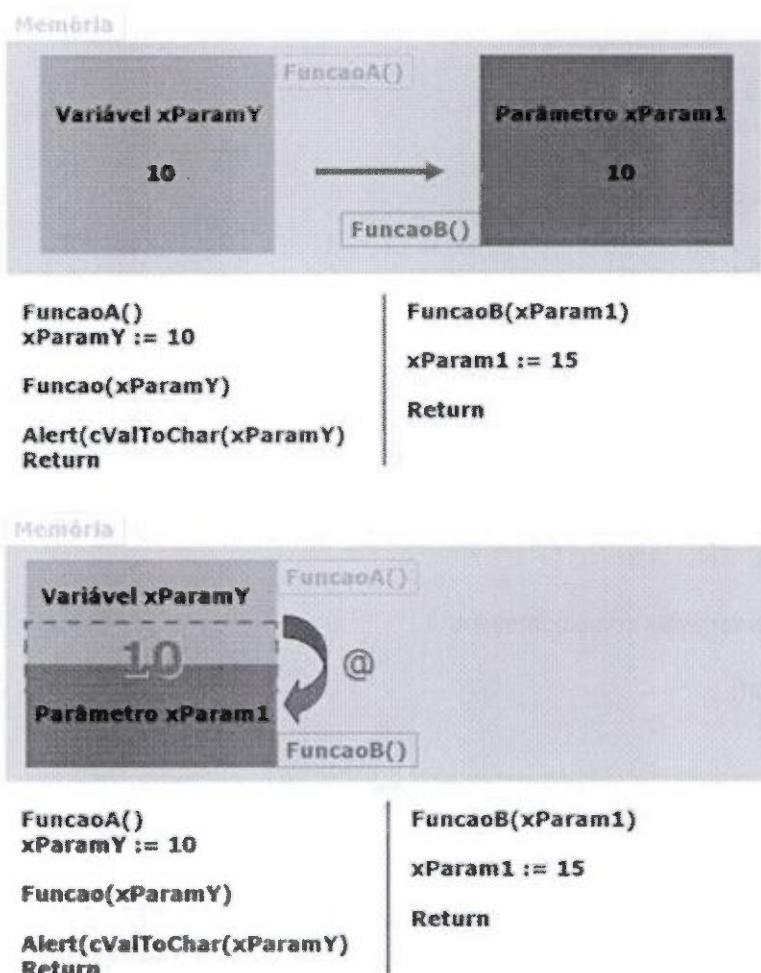
Como a linguagem ADVPL trata os parâmetros de forma posicional, o conteúdo da variável nFatorUser será atribuído diretamente à variável definida como primeiro parâmetro da função chamada, no nosso caso nFator.

Por ser uma atribuição de parâmetros por conteúdo, o interpretador da linguagem basicamente executa uma operação de atribuição normal, ou seja, nFator := nFatorUser.

7.5.6.1 Passagem de parâmetros por referência

A passagem de parâmetros por referência é uma técnica muito comum nas linguagens de programação, a qual permite que variáveis de escopo LOCAL tenham seu conteúdo manipulado por funções específicas, mantendo o controle destas variáveis restrito à função que as definiu e as funções desejadas pela aplicação.

A passagem de parâmetros por referência utiliza o conceito de que uma variável é uma área de memória e, portanto, passar um parâmetro por referência nada mais é do que, ao invés de passar o conteúdo para a função chamada, passar qual a área de memória utilizada pela variável passada



Desta forma, a função chamada tem acesso não apenas ao conteúdo, mas à variável em si, pois a área de memória é a variável, e qualquer alteração nesta será visível à função chamadora quando tiver o retorno desta função.

7.6. Diretivas De Compilação

O compilador ADVPL possui uma funcionalidade denominada pré-processador, o qual nada mais é do que um programa que examina o programa fonte escrito em ADVPL e executa certas modificações nele, baseadas nas Diretivas de Compilação.

As diretivas de compilação são comandos que não são compilados, sendo dirigidos ao pré-processador, o qual é executado pelo compilador antes da execução do processo de compilação propriamente dito.

Portanto, o pré-processador modifica o programa fonte, entregando para o compilador um programa modificado de acordo com as diretivas de compilação, estas são iniciadas pelo caractere "#".

As diretivas podem ser colocadas em qualquer parte do programa, sendo que as implementadas pela linguagem ADVPL são:

- #INCLUDE
- #DEFINE

- #IFDEF
- #IFNDEF
- #ELSE
- #ENDIF
- #COMMAND

Importante

As diretivas de compilação também são conhecidas como UDC – User Defined

DESENVOLVIMENTO DE PEQUENAS CUSTOMIZAÇÕES

As diretivas #IFDEF, #IFNDEF, #ELSE e #ENDIF permitem ao desenvolvedor criar fontes flexíveis e sensíveis a determinadas configurações da aplicação ERP.

7.7. Acesso e manipulação na bases de dados

Como a linguagem ADVPL é utilizada no desenvolvimento de aplicação para o sistema ERP Protheus, ela deve possuir recursos que permitam o acesso e a manipulação de informações, independentemente da base de dados para o qual o Sistema foi configurado.

Desta forma a linguagem possui dois grupos de funções distintos para atuar com os bancos de dados:

- Funções de manipulação de dados genéricos;
- Funções de manipulação de dados específicas para ambientes TOTVSDBAccess.

7.7.1. Funções de manipulação de dados genéricos

As funções de manipulação de dados, ditos como genéricos, permitem que uma aplicação ADVPL seja escrita da mesma forma, independente se a base de dados configurada para o sistema ERP for do tipo ISAM ou padrão SQL.

Muitas destas funções foram inicialmente herdadas da linguagem CLIPPER, e mediante novas implementações da área de Tecnologia da foram melhoradas e adequadas às necessidades do ERP. Por esta razão é possível encontrar em documentações da linguagem CLIPPER informações sobre funções de manipulação de dados utilizados na ferramenta ERP.

Dentre as melhorias implementadas pela área de Tecnologia da Microsiga, podemos mencionar o desenvolvimento de novas funções como, por exemplo, a função MsSeek() - versão da Microsiga para a função DbSeek(), e a integração entre a sintaxe ADVPL convencional e a ferramenta de acesso a bancos de dados no padrão SQL – TOTVSDBAccess.

A integração, entre a aplicação ERP e a ferramenta TOTVSDBAccess, permite que as funções de acesso e manipulação de dados escritos em ADVPL sejam interpretados e convertidos para uma sintaxe compatível com o padrão SQL ANSI e desta forma aplicados aos SGDBs (Sistemas Gerenciadores de Bancos de Dados) com sua sintaxe nativa.

7.7.2. Estrutura dos registros (informações)

Nas bases de dados padrão ISAM, cada registro possui um identificador nativo ou ID sequencial e ascendente que funciona como o endereço base daquela informação.

Este ID, mas conhecido como RECNO ou RECNOMBRE é gerado no momento de inclusão do registro na tabela e somente será alterado caso a estrutura dos dados desta tabela sofra alguma manutenção. Dentre as manutenções que uma tabela de dados ISAM pode sofrer, é possível citar a utilização do comando PACK, o qual apagará fisicamente os registros deletados da tabela, forçando uma renumeração dos identificadores de todos os registros. Esta situação também torna necessária a recriação de todos os índices vinculados àquela tabela.

Isto ocorre nas bases de dados ISAM devido ao conceito de exclusão lógica de registros que as mesmas possuem. Já os bancos de dados padrão SQL nativamente utilizam apenas o conceito de exclusão física de registros, o que para outras aplicações seria transparente, mas não é o caso do ERP Protheus.

Para manter a compatibilidade das aplicações desenvolvidas para bases de dados padrão ISAM, a área de Tecnologia e Banco de Dados da Microsiga implementou, nos bancos de dados padrão SQL, o conceito de exclusão lógica de registros existente nas bases de dados ISAM, por meio da criação de campos de controle específicos: R_E_C_N_O_, D_E_L_E_T_ e R_E_C_D_E_L.

Estes campos permitem que a aplicação ERP gerencie as informações do banco de dados, da mesma forma que as informações em bases de dados ISAM.

Com isso o campo R_E_C_N_O_ será um identificador único do registro dentro da tabela, funcionando como o ID ou RECNOMBRE de uma tabela ISAM, mas utilizando um recurso adicional disponível nos bancos de dados relacionais conhecido com Chave Primária.

Para a aplicação ERP Protheus o campo de controle R_E_C_N_O_ é definido em todas as tabelas como sendo sua chave primária, o que transfere o controle de sua numeração sequencial ao banco de dados.

O campo D_E_L_E_T_ é tratado internamente pela aplicação ERP como um "flag" ou marca de exclusão. Desta forma, os registros que estiverem com este campo marcado serão considerados como excluídos logicamente. A execução do comando PACK, em uma tabela de um banco de dados padrão SQL, visa excluir fisicamente os registros com o campo D_E_L_E_T_ marcado, mas não causará o efeito de renumeração de RECNO (no caso R_E_C_N_O_) que ocorre na tabela de bases de dados ISAM.

7.7.2.1 Funções de acesso e manipulação de dados

As funções de acesso e manipulação de dados, descritas neste tópico, são as classificadas anteriormente como funções genéricas da linguagem ADVPL, permitindo que as mesmas sejam utilizadas independentemente da base de dados para a qual a aplicação ERP está configurada.

As funções de acesso e manipulação de dados definem basicamente:

- Tabela que está sendo tratada;
- Campos que deverão ser lidos ou atualizados;
- Método de acesso direto às informações (registros e campos).

Formação Programação ADVPL



Dentre as funções ADVPL disponíveis para acesso e manipulação de informações, este material detalhará as seguintes opções:

DBLOCK()

Sintaxe	DBLOCK(xIdentificador)
Descrição	Função de base de dados, que efetua o lock (travamento) do registro identificado pelo parâmetro xIdentificador. Este parâmetro pode ser o Recno() para tabelas em formato ISAM, ou a chave primária para bancos de dados relacionais. Se o parâmetro xIdentificador não for especificado, todos os locks da área de trabalho serão liberados, e o registro posicionado será travado e adicionado em uma lista de registros bloqueados.

DBCLOSEAREA()

Sintaxe	DbCloseArea()
Descrição	Permite que um alias presente na conexão seja fechado, o que viabiliza novamente seu uso em outro operação. Este comando tem efeito apenas no alias ativo na conexão, sendo necessária sua utilização em conjunto com o comando DbSelectArea().

DBCOMMIT()

Sintaxe	DBCOMMIT()
Descrição	Efetua todas as atualizações pendentes na área de trabalho ativa.

DBCOMMITALL()

Sintaxe	DBCOMMITALL()
Descrição	Efetua todas as atualizações pendentes em todas as áreas de trabalho, em uso pela thread (conexão) ativa.

DbGoto()

Sintaxe	DbGoto(nRecno)
Descrição	Move o cursor da área de trabalho ativa para o record number (recno) especificado, realizando um posicionamento direto, sem a necessidade de uma busca (seek) prévia.

DBGOTOP()

Sintaxe	DbGoTop()
Descrição	Move o cursor da área de trabalho ativa para o primeiro registro lógico.

Formação Programação ADVPL



DBGOBUTTON()

Sintaxe	<code>DbGoButton()</code>
Descrição	Move o cursor da área de trabalho ativa para o último registro lógico.

DBRLOCKLIST()

Sintaxe	<code>DBRLOCKLIST()</code>
Descrição	Retorna um array contendo o record number (recno) de todos os registros, travados da área de trabalho ativa.

DBSEEK() E MSSEEK()

Sintaxe	<code>DbSeek(cChave, !SoftSeek, !Last)</code>
Descrição	<p>DbSeek: Permite posicionar o cursor da área de trabalho ativo no registro com as informações especificadas na chave de busca, fornecendo um retorno lógico indicando se o posicionamento foi efetuado com sucesso, ou seja, se a informação especificada na chave de busca foi localizada na área de trabalho.</p> <p>MsSeek(): Função desenvolvida pela área de Tecnologia da Microsiga, a qual possui as mesmas funcionalidades básicas da função DbSeek(), com a vantagem de não necessitar acessar novamente a base de dados para localizar uma informação já utilizada pela thread (conexão) ativa.</p>

DBSKIP()

Sintaxe	<code>DbSkip(nRegistros)</code>
Descrição	Move o cursor do registro posicionado para o próximo (ou anterior, dependendo do parâmetro), em função da ordem ativa para a área de trabalho.

DBSELECTAREA()

Sintaxe	<code>DbSelectArea(nArea cArea)</code>
Descrição	Define a área de trabalho especificada como sendo a área ativa. Todas as operações subsequentes que fizerem referência a uma área de trabalho para utilização, a menos que a área desejada seja informada explicitamente.

DBSETFILTER()

Sintaxe	<code>DbSetFilter(bCondicao, cCondicao)</code>
Descrição	Define um filtro para a área de trabalho ativa, o qual pode ser descrito na forma de um bloco de código ou através de uma expressão simples.

DBSETORDER()

Sintaxe	<code>DbSetOrder(nOrdem)</code>
----------------	---------------------------------

Descrição	Define qual índice será utilizado pela área de trabalho ativa, ou seja, pela área previamente selecionada através do comando DbSelectArea(). As ordens disponíveis no Ambiente Protheus são aquelas definidas no SINdex /SIX, ou as ordens disponibilizadas por meio de índices temporários.
------------------	--

DBORDERNICKNAME()

Sintaxe	DbOrderNickName(NickName)
Descrição	Define qual índice criado pelo usuário será utilizado. O usuário pode incluir os seus próprios índices e no momento da inclusão deve criar o NICKNAME para o mesmo.

DBUNLOCK()

Sintaxe	DBUNLOCK()
Descrição	Mesma funcionalidade da função UNLOCK(), só que recomendada para ambientes de rede nos quais os arquivos são compartilhados. Libera o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

DBUNLOCKALL()

Sintaxe	DBUNLOCKALL()
Descrição	Libera o travamento de todos os registros de todas as áreas de trabalho disponíveis na thread (conexão) ativa.

DBUSEAREA()

Sintaxe	DbUseArea(lNovo, cDriver, cArquivo, cAlias, lComparilhado, lSoLeitura)
Descrição	Define um arquivo de base de dados como uma área de trabalho disponível na aplicação.

MSUNLOCK()

Sintaxe	MsUnLock()
Descrição	Libera o travamento (lock) do registro posicionado, confirmando as atualizações efetuadas neste registro.

RECLOCK()

Sintaxe	RecLock(cAlias, lInclui)
Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa, permitindo a inclusão ou alteração das informações do mesmo.

RLOCK()

Sintaxe	RLOCK() à lSucesso
----------------	--------------------

Descrição	Efetua o travamento do registro posicionado na área de trabalho ativa.
------------------	--

SELECT()

Sintaxe	Select(cArea)
Descrição	Determina o número de referência de um determinado alias em um ambiente de trabalho. Caso o alias especificado não esteja em uso no Ambiente, será retornado o valor 0 (zero).

SOFTLOCK()

Sintaxe	SoftLock(cAlias)
Descrição	<p>Permite a reserva do registro posicionado na área de trabalho ativa de forma que outras operações, com exceção da atual, não possam atualizar este registro. Difere da função RecLock() pois não gera uma obrigação de atualização, e pode ser sucedido por ele.</p> <p>Na aplicação ERP Protheus, o SoftLock() é utilizado nos browses, antes da confirmação da operação de alteração e exclusão, pois neste momento a mesma ainda não foi efetivada, mas outras conexões não podem acessar aquele registro pois o mesmo está em manutenção, o que implementa da integridade da informação.</p>

UNLOCK()

Sintaxe	UNLOCK()
Descrição	Liberá o travamento do registro posicionado na área de trabalho ativa e confirma as atualizações efetuadas naquele registro.

7.8. Diferenciação entre variáveis e nomes de campos

Muitas vezes uma variável pode ter o mesmo nome que um campo de um arquivo ou de uma tabela aberta no momento. Neste caso, o ADVPL privilegiará o campo, de forma que uma referência a um nome, que identifique tanto uma variável como um campo, resultará no conteúdo do campo.

Para especificar qual deve ser o elemento referenciado, deve-se utilizar o operador de identificação de apelido (->) e um dos dois identificadores de referência, MEMVAR ou FIELD.

cRes := MEMVAR->NOME

Esta linha de comando identifica que o valor atribuído à variável cRes deve ser o valor da variável de memória chamada NOME. cRes := FIELD->NOME. Neste caso, o valor atribuído à variável cRes será o valor do campo NOME, existente no arquivo ou tabela aberto na área atual.

O identificador FIELD pode ser substituído pelo apelido de um arquivo ou tabela abertos, para evitar a necessidade de selecionar a área antes de acessar o conteúdo de terminado campo.

cRes := CLIENTES->NOME

As tabelas de dados, utilizadas pela aplicação ERP, recebem automaticamente do Sistema o apelido ou ALIAS, especificado para as mesmas no arquivo de sistema SX2. Assim, se o campo NOME pertence a uma tabela da aplicação PROTHEUS, o mesmo poderá ser referenciado com a indicação do ALIAS pré-definido desta tabela.

```
cRes := SA1->NOME
```

7.9. Controle de numeração sequencial

Alguns campos de numeração do Protheus são fornecidos pelo Sistema em ordem ascendente. É o caso, por exemplo, do número do pedido de venda e outros que servem como identificador das informações das tabelas. É preciso ter um controle do fornecimento desses números, em especial quando vários usuários estão trabalhando simultaneamente.

7.9.1. Semáforos

Para definir o conceito do que é um semáforo de numeração deve-se avaliar a seguinte sequencia de eventos no sistema:

- Ao ser fornecido um número, ele permanece reservado até a conclusão da operação que o solicitou;
- Se esta operação for confirmada, o número não sera disponibilizado, mas se a operação for cancelada, o número voltará a ser disponível mesmo que naquele momento números maiores já tenham sido oferecidos e utilizados.

Com isso, mesmo que tenhamos vários processos solicitando numerações sequenciais para uma mesma tabela, como por exemplo inclusões simultâneas de pedidos de vendas, teremos para cada pedido um número exclusivos e sem o intervalos e numerações não utilizadas.

7.9.2. Funções de controle de semáforos e numeração sequencial

A linguagem ADVPL permite a utilização das seguintes funções para o controle das numerações sequenciais utilizadas nas tabelas da aplicação ERP:

GETSXENUM()

Sintaxe	GETSXENUM(cAlias, cCampo, cAliasSXE, nOrdem)
Descrição	Obtém o número sequencia do alias especificado no parâmetro, através da referência aos arquivos de sistema SXE/SXF ou ao servidor de numeração, quando esta configuração está habilitada no Ambiente Protheus.

CONFIRMSXE()

Sintaxe	CONFIRMSXE(lVerifica)
Descrição	Confirma o número alocado através do último comando GETSXENUM().

ROLLBACKSXE()

Sintaxe	ROLLBACKSXE()
Descrição	Descarta o número fornecido pelo último comando GETSXENUM(), retornando a numeração disponível para outras conexões.

7.10. Customização de parâmetros – Configurador

Os parâmetros de sistema utilizados pela aplicação ERP e definidos através do módulo configurador possuem as seguintes características fundamentais:

Tipo do parâmetro: de forma similar a uma variável, um parâmetro terá um tipo de conteúdo pré-definido em seu cadastro.

Esta informação é utilizada pelas funções da aplicação ERP, na interpretação do conteúdo do parâmetro e retorno desta informação à rotina que o consulto. **Interpretação do conteúdo do parâmetro:** Diversos parâmetros do Sistema têm seu conteúdo macro executado durante a execução de uma rotina do ERP. Estes parâmetros macro executáveis tem como única característica em comum seu tipo: caractere, mas não existe nenhum identificador explícito que permite a fácil visualização de quais parâmetros possuem um retorno simples e de quais parâmetros terão seu conteúdo macro executado para determinar o retorno "real".

A única forma eficaz de avaliar como um parâmetro é tratado (simples retorno ou macro execução) é através do código fonte da rotina, no qual deverá ser avaliado como é tratado o retorno de uma destas funções:

- GETMV()
- SUPERGETMV()
- GETNEWPAR()

Um retorno macro executado é determinado através do uso do operador "&" ou de uma das funções de execução de blocos de código, em conjunto com uma das funções citadas anteriormente

GETMV()

Sintaxe	GETMV(cParametro)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do sistema informando a ocorrência.

GETNEWPAR()

Sintaxe	GETNEWPAR(cParametro, cPadrao, cFilial)
Descrição	Retorna o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do Sistema informando a ocorrência. Difere do SuperGetMV() pois considera que o parâmetro pode não existir na versão atual do Sistema, e por consequência não será exibida a mensagem de help.

PUTMV()

Sintaxe	<code>PUTMV(cParametro, cConteudo)</code>
Descrição	Atualiza o conteúdo do parâmetro especificado no arquivo SX6, de acordo com as parametrizações informadas.

SUPERGETMV()

Sintaxe	<code>SUPERGETMV(cParametro , lHelp , cPadrao , cFilial)</code>
Descrição	Retoma o conteúdo do parâmetro especificado no arquivo SX6, considerando a filial parametrizada na conexão. Caso o parâmetro não exista, será exibido um help do Sistema informando a ocorrência. Difere do GetMv() pois os parâmetros consultados são adicionados em uma área de memória, que permite que em uma nova consulta não seja necessário acessar e pesquisar o parâmetro na base de dados.

7.10.1.1 Cuidados na utilização de um parâmetro

Um parâmetro de sistema tem a finalidade de propiciar um retorno válido a um conteúdo previamente definido na configuração do módulo para uma rotina, processo ou quaisquer outros tipos de funcionalidades disponíveis na aplicação.

Apesar de haver parâmetros que permitem a configuração de expressões, e por consequência a utilização de funções para definir o retorno que será obtido com a consulta deste parâmetro, é expressamente proibido o uso de funções em parâmetros para manipular informações da base de dados do Sistema.

Caso haja a necessidade de ser implementado um tratamento adicional a um processo padrão do Sistema, o mesmo deverá utilizar o recurso de ponto de entrada.

A razão desta restrição é simples:

- As rotinas da aplicação ERP não protegem a consulta de conteúdos de parâmetros, quanto a gravações realizadas dentro ou fora de uma transação.
- Desta forma, qualquer alteração na base realizada por uma rotina, configurada em um parâmetro, pode ocasionar a perda da integridade das informações do Sistema.

7.11. Pontos de Entrada – Conceitos

Um ponto de entrada é uma User Function desenvolvida com a finalidade de interagir com uma rotina padrão da aplicação ERP.

A User Function deverá ter um nome pré-estabelecido no desenvolvimento da rotina padrão do ERP, e de acordo com esta pré-disposição e o momento no qual o ponto de entrada é executado durante um processamento, ele poderá:

- Complementar uma validação realizada pela aplicação;
- Complementar as atualizações realizadas pelo processamento em tabelas padrões do ERP;

- Implementar a atualização de tabelas específicas durante o processamento de uma rotina padrão do ERP;
- Executar uma ação sem processos de atualizações, mas que necessite utilizar as informações atuais do Ambiente, durante o processamento da rotina padrão para determinar as características do processo;
- Substituir um processamento padrão do Sistema por uma regra específica do cliente, no qual o mesmo será implementado.

7.11.1.1 Premissas e Regras

Um ponto de entrada não deve ser utilizado para outras finalidades senão para as quais o mesmo foi pré-definido, sob pena de causar a perda da integridade das informações da base de dados ou provocar eventos de erro durante a execução da rotina padrão.

Um ponto de entrada deve ser transparente para o processo padrão, de forma que todas as tabelas, acessadas pelo ponto de entrada e que sejam utilizadas pela rotina padrão, deverão ter sua situação imediatamente anterior à execução do ponto restaurado ao término do mesmo, e para isto recomenda-se o uso das funções GETAREA() e RESTAREA().

- GETAREA()

Função utilizada para proteger o ambiente ativo no momento de algum processamento específico. Para salvar uma outra área de trabalho (alias) que não o ativo, a função GetArea() deve ser executada dentro do alias: ALIAS->(GetArea()).

Retorno: Array contendo {Alias(),IndexOrd(),Recno()}

- RESTAREA()

Função utilizada para devolver a situação do ambiente salva, através do comando GETAREA(). Deve-se observar que a última área restaurada é a área que ficará ativa para a aplicação.

Sintaxe: RESTAREA(aArea)

Retorno: aArea Array contendo: {cAlias, nOrdem, nRecno}, normalmente gerado pelo uso da função GetArea().

Como um ponto de entrada não é executado da forma tradicional, ou seja, ele não é chamado como uma função, ele não recebe parâmetros. A aplicação ERP disponibiliza uma variável de Sistema denominada PARAMIXB, a qual recebe os parâmetros da função chamadora e os disponibiliza para serem utilizados pela rotina customizada.

A variável PARAMIXB não possui um padrão de definição nos códigos fontes da aplicação ERP, desta forma seu tipo pode variar de um conteúdo simples (caractere, numérico, lógico e etc.) a um tipo complexo como um array ou um objeto. Assim, é necessário sempre avaliar a documentação sobre o ponto, bem como proteger a função customizada de tipos de PARAMIXB não tratados por ela.

7.12. Interfaces Visual

A linguagem ADVPL possui duas formas distintas para definição de interfaces visuais no Ambiente ERP: sintaxe convencional, nos padrões da linguagem CLIPPER e a sintaxe orientada a objetos.

Além das diferentes sintaxes disponíveis para definição das interfaces visuais, o ERP Protheus possui funcionalidades pré-definidas, as quais já contêm todos os tratamentos necessários a atender as necessidades básicas de acesso e manutenção das informações do Sistema.

Neste tópico serão abordadas as sintaxes convencionais para definição das interfaces visuais da linguagem ADVPL e as interfaces de manutenção, disponíveis no Ambiente ERP Protheus.

7.12.1. Sintaxe e componentes das interfaces visuais

A sintaxe convencional para definição de componentes visuais da linguagem ADVPL depende diretamente, em include especificado no cabeçalho do fonte. Os dois includes, disponíveis para o Ambiente ADVPL Protheus, são:

- **RWMAKE.CH:** Permite a utilização da sintaxe CLIPPER na definição dos componentes visuais.
- **TOTVS.CH:** Permite a utilização da sintaxe ADVPL convencional, a qual é um aprimoramento da sintaxe CLIPPER, com a inclusão de novos atributos para os componentes visuais disponibilizados no ERP Protheus.

Para ilustrar a diferença na utilização destes dois includes, segue abaixo as diferentes definições para os componentes Dialog e MsDialog:

Exemplo 01 – Include Rwmake.ch

```
#include "rwmake.ch"

@ 0,0 TO 400,600 DIALOG oDig TITLE "Janela em sintaxe Clipper" ACTIVATE DIALOG oDig CENTERED
```

Exemplo 02 – Include Protheus.ch

```
#include "TOTVS.ch"

DEFINE MSDIALOG oDig TITLE "Janela em sintaxe ADVPL" FROM 000,000 TO 400,600 PIXEL ACTIVATE
MSDIALOG oDig CENTERED
```

Os componentes da interface visual que serão tratados neste tópico, utilizando a sintaxe ADVPL são:

BUTTON()

Sintaxe	@ nLinha,nColuna BUTTON cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef ACTION AÇÃO
Descrição	Define o componente visual Button, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados somente com um texto simples para sua identificação.

MSDIALOG()

Sintaxe	DEFINE MSDIALOGoObjetoDLGTITLE cTitulo FROMnLinIni,nColIni TO nLiFim,nColFim OF oObjetoRef UNIDADE
Descrição	Define o componente MSDIALOG(), o qual é utilizado como base para os demais componentes da interface visual, pois um componente MSDIALOG() é uma janela da aplicação.

MSGGET()

Sintaxe	@ nLinha, nColuna MSGGET VARIABEL SIZE nLargura,nAltura UNIDADE OF oObjetoRef F3 cF3 VALID VALID WHEN WHEN PICTURE cPicture
Descrição	Define o componente visual MSGGET, o qual é utilizado para captura de informações digitáveis na tela da interface.

SAY()

Sintaxe	@ nLinha, nColuna SAY cTexto SIZE nLargura,nAltura UNIDADE OF oObjetoRef
Descrição	Define o componente visual SAY, o qual é utilizado para exibição de textos em uma tela de interface.

SBUTTON()

Sintaxe	DEFINE SBUTTON FROM nLinha, nColuna TYPE N ACTION AÇÃO STATUS OF oObjetoRet
Descrição	Define o componente visual SButton, o qual permite a inclusão de botões de operação na tela da interface, os quais serão visualizados dependendo da interface do sistema ERP, utilizada somente com um texto simples para sua identificação, ou com uma imagem (BitMap) pré-definido.

Abaixo segue um código completo de interface, utilizando todos os elementos da interface visual descritos anteriormente:

```

DEFINE MSDIALOG oDig TITLE cTitulo FROM 000,000 TO 080,300 PIXEL
@ 001,001 TO 040, 150 OF oDig PIXEL
@ 010,010 SAY cTexto SIZE 55, 07 OF oDig PIXEL

@ 010,050 MSGGET cCGC SIZE 55, 11 OF oDig PIXEL PICTURE "@R 99.999.999/9999-99";
VALID !Vazio()

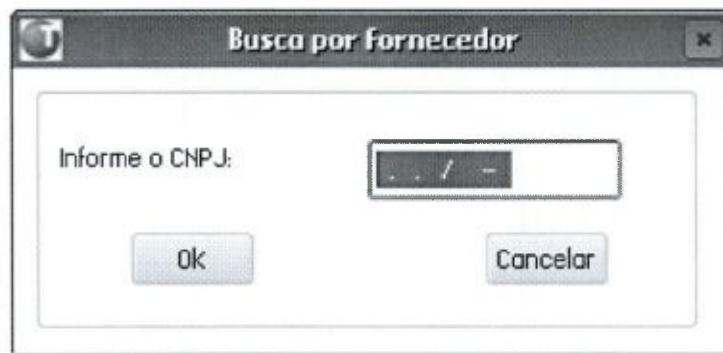
DEFINE SBUTTON FROM 010, 120 TYPE 1 ACTION (nOpca := 1,oDig:End());
ENABLE OF oDig

DEFINE SBUTTON FROM 020, 120 TYPE 2 ACTION (nOpca := 2,oDig:End());
ENABLE OF oDig

ACTIVATE MSDIALOG oDig CENTERED

```

O código demonstrado anteriormente é utilizado nos exercícios de fixação deste material e deverá produzir a seguinte interface:



7.13. Interfaces padrões para atualizações de dados

Os programas de atualização de cadastros e digitação de movimentos seguem um padrão que se apoia no Dicionário de Dados.

Basicamente são duas as interfaces quer permitem a visualizações das informações e a manipulação dos dados do Sistema.

7.13.1. Modelo1

Este modelo de programa exibe um Browse vertical de campos presentes no dicionário de dados. Genericamente as validações são herdadas do próprio dicionário de dados.

7.13.1.1 AxCadastro

O AxCadastro() é uma funcionalidade de cadastro simples, com poucas opções de customização, a qual é composta de:

- Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browse).
- Funções de pesquisa, visualização, inclusão, alteração e exclusão de padrões para visualização de registros simples, sem a opção de cabeçalho e itens.

Sintaxe: AxCadastro(cAlias, cTitulo, cVldExc, cVldAlt)

cAlias	Alias padrão do sistema para utilização, o qual deve estar definido no dicionário de dados – SX3.
cTitulo	Título da Janela.
cVldExc	Validação para Exclusão.
cVldAlt	Validação para Alteração.

```

Local cAlias := "SA2"
Local cTitulo := "Cadastro de Fornecedores"
Local cVldExc := ".T."

```

```

Local cVldAlt := ".T."
dbSelectArea(cAlias)
dbSetOrder(1)
AxCadastro(cAlias,cTitulo,cVldExc,cVldAlt)

```

7.13.1.2 MBrowse

A Mbrowse() é uma funcionalidade de cadastro que permite a utilização de recursos mais aprimorados na visualização e manipulação das informações do Sistema, possuindo os seguintes componentes:

- Browse padrão para visualização das informações da base de dados, de acordo com as configurações do SX3 – Dicionário de Dados (campo browser).
- Parametrização para funções específicas para as ações de visualização, inclusão, alteração e exclusão de informações, o que viabiliza a manutenção de informações com estrutura de cabeçalhos e itens.
- Recursos adicionais como identificadores de status de registros, legendas e filtros para as informações.

Sintaxe simplificada: Mbrowse(nLin1, nCol1, nLin2, nCol2, cAlias)

Variáveis Private adicionais:

	<p>Array contendo as funções que serão executadas pela Mbrowse. Este array pode ser parametrizados com as funções básicas da AxCadastro conforme abaixo:</p> <table border="1"> <tr> <td>aRotina</td><td> AADD(aRotina,{“Pesquisar”, “AxPesqui”,0,1}) AADD(aRotina,{“Visualizar”, “AxVisual”,0,2}) AADD(aRotina,{“Incluir”, “AxInclui”,0,3}) AADD(aRotina,{“Alterar”, “AxAltera”,0,4}) AADD(aRotina,{“Excluir”, “AxDeleta”,0,5}) </td></tr> <tr> <td>cCadastro</td><td>Título do browse que será exibido.</td></tr> </table>	aRotina	AADD(aRotina,{“Pesquisar”, “AxPesqui”,0,1}) AADD(aRotina,{“Visualizar”, “AxVisual”,0,2}) AADD(aRotina,{“Incluir”, “AxInclui”,0,3}) AADD(aRotina,{“Alterar”, “AxAltera”,0,4}) AADD(aRotina,{“Excluir”, “AxDeleta”,0,5})	cCadastro	Título do browse que será exibido.
aRotina	AADD(aRotina,{“Pesquisar”, “AxPesqui”,0,1}) AADD(aRotina,{“Visualizar”, “AxVisual”,0,2}) AADD(aRotina,{“Incluir”, “AxInclui”,0,3}) AADD(aRotina,{“Alterar”, “AxAltera”,0,4}) AADD(aRotina,{“Excluir”, “AxDeleta”,0,5})				
cCadastro	Título do browse que será exibido.				

Local cAlias := "SA2"

```

Private cCadastro := "Cadastro de Fornecedores"
Private aRotina := {}

```

```

AADD( aRotina,{“Pesquisar”, “AxPesqui”,0,1 })
AADD(aRotina,{“Visualizar”, “AxVisual”,0,2 })
AADD(aRotina,{“Incluir”, “AxInclui”,0,3 })
AADD(aRotina,{“Alterar”, “AxAltera”,0,4 })
AADD(aRotina,{“Excluir”, “AxDeleta”,0,5 })

```

```

dbSelectArea(cAlias)
dbSetOrder(1)

```

mBrowse(6,1,22,75,cAlias)

Return(NIL)

Ao definir as funções no array aRotina, se o nome da função não for especificado com "()", a Mbrowse passará como parâmetros as seguintes variáveis de controle:

- **cAlias:** Alias ativo definido para a Mbrowse.
- **nRecno:** Record number (recno) do registro posicionado no alias ativo.
- **nOpc:** Posição da opção utilizada na Mbrowse de acordo com a ordem da função no array a Rotina.

7.13.1.3 AxFunctions

Conforme mencionado nos tópicos sobre as interfaces padrão Mbrowse(), existem funções padrões da aplicação ERP que permitem a visualização, inclusão, alteração e exclusão de dados em formato simples.

AXALTERA()

Sintaxe	AxAltera(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lF3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)
Descrição	Função de alteração padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXDELETA()

Sintaxe	AXDELETA(cAlias, nReg, nOpc, cTransact, aCpos, aButtons, aParam,; aAuto, lMaximized)
Descrição	Função de exclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXINCLUI()

Sintaxe	AxInclui(cAlias, nReg, nOpc, aAcho, cFunc, aCpos, cTudoOk, lF3,; cTransact, aButtons, aParam, aAuto, lVirtual, lMaximized)
Descrição	Função de inclusão padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

AXPESQUI()

Sintaxe	AXPESQUI()
Descrição	Função de pesquisa padrão em registros exibidos pelos browses do Sistema, a qual posiciona o browse no registro pesquisado. Exibe uma tela que permite a seleção do índice a ser utilizado na pesquisa e a digitação das informações que compõe a chave de busca.

AXVISUAL()

Sintaxe	AXVISUAL(cAlias, nReg, nOpc, aAcho, nColMens, cMensagem, cFunc,; aButtons, lMaximized)
Descrição	Função de visualização padrão das informações de um registro, no formato Enchoice, conforme demonstrado no tópico sobre a interface AxCadastro().

7.13.1.4 MarkBrowse

A função MarkBrow() permite que os elementos de um browser, sejam marcados ou desmarcados. Para utilização da MarkBrow() é necessário declarar as variáveis cCadastro e aRotina como Private, antes da chamada da função.

Sintaxe: MarkBrow (cAlias, cCampo, cCpo, aCampos, lInvert, cMarca, cCtrlM, uPar8, cExplnI, cExpFim, cAval, bParBloco)

cAlias	Alias ativo definido para a Mbrowse()
cCampo	Campo do arquivo onde será feito o controle (gravação) da marca.
cCpo	Campo onde será feita a validação para marcação e exibição do bitmap de status.
aCampos	Vetor de colunas a serem exibidas no browse, deve conter as seguintes dimensões: [n][1] – nome do campo; [n][2] - Nulo (Nil); [n][3] - Título do campo; [n][4] - Máscara (picture).
lInvert	Inverte a marcação.
cMarca	String a ser gravada no campo especificado para marcação.
cCtrlM	Função a ser executada caso deseje marcar todos os elementos.
uPar8	Parâmetro reservado.
cExplnI	Função que retorna o conteúdo inicial do filtro baseada na chave de índice selecionada.
cExpFim	Função que retorna o conteúdo final do filtro baseada na chave de índice selecionada.
cAval	Função a ser executada no duplo clique em um elemento no browse.
bParBloco	Bloco de código a ser executado na inicialização da janela

7.13.1.5 Funções de Apoio

- **GetMark:** define a marca atual.
- **IsMark:** avalia se um determinado conteúdo é igual a marca atual.
- **ThisMark:** captura a marca em uso.
- **ThisInv:** indica se foi usado o recurso de selecionar todos (inversão).
- **MarkBRefresh:** atualiza exibição na marca do browser. Utilizada quando a marca é colocada ou removida em blocos e não pelo clique.

Exemplo: Função MarkBrow()

```
#include "Protheus.ch"

USER FUNCTION xMark()

Local aCpos := {}
Local aCampos := {}
Local nl := 0
Local cAlias := "SA2"
Private aRotina := {}
Private aRecDel := {}
Private cCadastro := "Cadastro de Fornecedores"

AADD(aRotina, {"Pesquisar" , "AxPesqui" ,0,1})
AADD(aRotina, {"Visualizar" , "AxVisual" ,0,2})
AADD(aRotina, {"Excluir Lote", "U_DelLote()",0,6})

AADD(aCpos, "A2_OK" )
AADD(aCpos, "A2_FILIAL" )
AADD(aCpos, "A2_COD" )
AADD(aCpos, "A2_LOJA" )
AADD(aCpos, "A2_NOME" )
AADD(aCpos, "A2_TIPO" )

dbSelectArea("SX3")
dbSetOrder(2)

For nl := 1 To Len(aCpos)
  If dbSeek(aCpos[nl])
    aAdd(aCampos,{X3_CAMPO,"",lif(nl==1,"",Trim(X3_TITULO)),Trim(X3_PICTURE)})
  Endif
Next

DbSelectArea(cAlias)
DbSetOrder(1)
MarkBrow(cAlias,aCpos[1], "A2_TIPO == \" \"", aCampos,.F.,; GetMark("SA2","A2_OK"))

Return()
```

USER FUNCTION DelLote()

```
Local cMarca := ThisMark()
Local nX := 0
Local lInvert := ThisInv()
DbSelectArea("SA2")
DbGoTop()
While SA2->( !EOF() )
  If SA2->A2_OK == cMarca .AND. lInvert
    AADD(aRecDel,SA2->(Recno()))
  Elseif SA2->A2_OK != cMarca .AND. lInvert
```

```

AADD(aRecDel,SA2->(Recno()))
Endif

SA2->(dbSkip())
Enddo

If Len(aRecDel) > 0 .AND. MsgYesNo( "Deseja excluir os :" + ;      cValToChar(Len(aRecDel)) + "
Fornecedores selecionados?" )

For nX := 1 to Len(aRecDel)
  SA2->(DbGoto(aRecDel[nX]))
  RecLock("SA2",.F.)
  dbDelete()
  MsUnLock()
Next nX

Endif

Return( NIL )

```

7.13.1.6 Enchoice

A função Enchoice é recurso baseado no dicionário de dados para verificar campos obrigatórios, validações, gatilhos, consulta padrão e etc. Assim também para criar pastas de cadastros.

Estes podem ser usados tanto com variáveis de memórias com o escopo Private como diretamente os campos da tabela que se refere. A diferença entre a função Enchoice e o objeto MsMGet é que a função não retorna o nome da variável de objeto exportável criado.

Sintaxe: Enchoice (cAlias [nReg]nOpc [aCRA] [cLetras] [cTexto] [aAcho] [aPos] [aCpos] [nModelo] [nColMens] [cMensagem] [cTudoOk] [oWnd] [IF3] [IMemoria] [IColumn] [caTela] [INoFolder] [IProperty] [aField] [aFolder] [ICreate] [INoMDIStrech] [cTela])

Nome	Tipo	Descrição	Obrigatório
cAlias	Caracter	Tabela cadastrada no Dicionário de Tabelas (SX2) que será editada	X
nReg	Nulo	Parâmetro não utilizado	
nOpc	Numérico	Número da linha do aRotina que definirá o tipo de edição (Inclusão, Alteração, Exclusão, Visualização)	X
aCRA	Nulo	Parâmetro não utilizado	
cLetras	Nulo	Parâmetro não utilizado	
cTexto	Nulo	Parâmetro não utilizado	
aAcho	Vetor	Vetor com nome dos campos que serão exibidos. Os campos de usuário sempre serão exibidos se não existir no parâmetro um elemento com a expressão "NOUSER"	
após	Vetor	Vetor com coordenadas para criação da enchoice no formato {, , }	
aCpos	Vetor	Vetor com nome dos campos que poderão ser editados	

Nome	Tipo	Descrição	Obrigatório
nModelo	Numérico	Se for diferente de 1 desabilita execução de gatilhos estrangeiros	
nColMens	Nulo	Parâmetro não utilizado	
cMensagem	Nulo	Parâmetro não utilizado	
cTudoOk	Nulo	Parâmetro não utilizado	
oWnd	Objeto	Objeto (janela, painel, etc) onde a enchoice será criada	
IF3	Lógico	Indica se a enchoice esta sendo criada em uma consulta F3 para utilizar variáveis de memória	
IMemoria	Lógico	Indica se a enchoice utilizará variáveis de memória ou os campos da tabela na edição	
IColumn	Lógico	Indica se a apresentação dos campos será em forma de coluna	
caTela	Caracter	Nome da variável tipo "private" que a enchoice utilizará no lugar da variável aTela	
INoFolder	Lógico	Indica se a enchoice não irá utilizar as Pastas de Cadastro (SX4)	
IProperty	Lógico	Indica se a enchoice não utilizará as variáveis tipo "private" aTela e aGets, somente suas propriedades com seus respectivos nomes	
aField	Vetor	Vetor com os campos que serão mostrados na Enchoice caso o SX3 não seja utilizado	
aFolder	Vetor	Vetor com o nome das pastas caso o SX3 não seja utilizado	
ICreate	Lógico	Indica se cria as pastas especificadas no parâmetro aFolder	
INoMDIStrech	Lógico	Define se o objeto não será alinhado conforme o espaço existente na janela	
cTela	Caracter	Campo reservado	

```
Local aSize := {}
Local aPObj := {}
```

```
// Retorna a area util das janelas Protheus
aSize := MsAdvSize()
```

```
// Será utilizado três áreas na janela
// 1º - Enchoice, sendo 80 pontos pixel
```

```
AADD( aObj, { 100, 080, .T., .F. })
AADD( aObj, { 100, 100, .T., .T. })
AADD( aObj, { 100, 015, .T., .F. })
```

```
// Cálculo automático da dimensões dos objetos (altura/largura) em pixel
```

```
aInfo := { aSize[1], aSize[2], aSize[3], aSize[4], 3, 3 }
aPObj := MsObjSize( aInfo, aObj )
```

```
// Cálculo automático de dimensões dos objetos MSGET
oDialog:=MSDialog():New(aSize[7],aSize[1],aSize[6],aSize[5],;
"MSDialog",,,CLR_BLACK,CLR_WHITE,,,T.)
EnChoice( cAlias1, nReg, nOpc, , , , aPObj[1])
```

oDialog:Activate(,,,T.)

Vale lembrar que nós programadores reaproveitamos muito o que já existe, isto é para simplesmente ganhamos tempo, e no caso da utilização da função Enchoice é preciso criar as variáveis de memórias que levam o mesmo nome dos campos da tabela em questão. Por exemplo o campo A2_NOME da tabela SA2 (cadastro de fornecedores) quando queremos referenciar o campo usa-se o prefixo da tabela e o campo em questão, desta forma:

- **SA2->A2_NOME**

Agora quando queremos referenciar a uma variável que está com o conteúdo do mesmo campo criamos outro recurso, desta forma:

- **M->A2_NOME**

E para criar variáveis com o nome do campo utilizamos a função:

RegToMemory()

Sintaxe	RegToMemory(cAlias,lInc , lDic , lInitPad , cStack)
	Esta função inicializa as variáveis de memoria utilizadas pela interfaces.
	<p>cAlias: Alias da tabela que terá suas variaveis inicializadas.</p> <p>lInc: Indica se a inicialização será baseada numa operações de inclusão (.T.) ou manutenção (.F.). A diferença entre elas é que na operação de inclusão os valores são inicializados vazios e na manutenção com os dados do registro posicionado.</p> <p>lDic: Indica se a inicialização das variaveis será baseada no dicionário de dados ou apenas nos dados da WorkArea aberta. A diferença entre elas são os campos virtuais que somente são inicializados com a opção .T. - Dicionário de dados.</p> <p>lInitPad: Indica se o inicializador padrão do dicionário de dados será executado. Este parametro somente será acionado se o parametro ExpL3 for configurado como .T. - Dicionário de Dados.</p> <p>cStack: Qualquer parâmetro reservado</p>

As informações contidas no SX3 definirão se o campo deverá ser exibido, editado e qual tipo de objeto será utilizado (Get, Combo, Memo, CheckBox).

Para facilitar as validações possui funções para ajudar nas validações na Enchoice, para verificar se todos os campos foram digitados podemos utilizar a função:

Obrigatório()

Formação Programação ADVPL



Sintaxe	obrigatório (aGets, aTela, uPar3, lShow)
Descrição	<p>aGets: Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.</p> <p>aTela: Variável PRIVATE tratada pelo objeto Enchoice(), previamente definida no fonte.</p> <p>uPar3: Não utilizado.</p> <p>lShow: Lógico determina se exibirá o help caso algum campo obrigatório não tenha sido preenchido. Default é .T.</p> <p>Função retorna um valor lógico .T. indicando se todos os campos obrigatórios foram preenchidos.</p>

FieldPos()

Sintaxe	FieldPos(X3_CAMPO)
Descrição	A função FieldPos tem a funcionalidade de retornar a posição de um determinado campo dentro da estrutura do alias corrente. Caso o campo não exista na estrutura, é retornado zero.

FieldGet()

Sintaxe	FieldGet(<nFieldPos>)
Descrição	nFieldPos: Número da posição ordinal do campo na tabela SX3

FieldName ()

Sintaxe	FieldName (<cNomeCampo >)
Descrição	cNomeCampo: Retorna uma string contendo o nome do campo especificado. Se o parâmetro (nPosição) for maior que o total de campos do alias atual ou não tenha um alias aberto na área de trabalho, a função retornará uma string vazia ("").

FieldPut ()

Sintaxe	FieldPut (<nFieldPos > ,<Conteudo>)
Descrição	<p>Permite definir o valor de um campo em uma tabela utilizando a posição ordinal do campo na estrutura da tabela.</p> <p>nFieldPos: Número da posição ordinal do campo na tabela SX3;</p> <p>Conteudo: Conteúdo de será atribuído no campo.</p>

FCount()

Sintaxe	FCount ()
Descrição	Retorna a quantidade de campos do arquivo de dados.

CriaVar()

Sintaxe	CriaVar (<X3_CAMPO>,<LinicioalidorPadrão>)
Descrição	Retorna a quantidade o tamanho campos, podendo trazer o conteúdo do inicializador padrão do campo. X3_CAMPO: Nome do campo LinicioalidorPadrão: Valor lógico .T. traz o conteúdo do inicializador padrão

7.13.2. EnchoiceBar

Função que cria uma barra de botões padrão de Ok Cancelar, permitindo a implementação de botões adicionais.
 Sintaxe: ENCHOICEBAR(oDlg, bOk, bCancelar, [IMensApag], [aBotoes])

Parâmetros:

oDlg	Objeto	Janela onde a barra será criada.
bOk	Objeto	Bloco de código executado quando clicado botão Ok.
bCancelar	Objeto	Bloco de código executado quando clicado.
IMensApag	Lógico	Indica se ao clicar no botão Ok aparecerá uma tela de confirmação de exclusão. Valor padrão falso.
Abotoes	Vetor	Vetor com informações para criação de botões adicionais na barra. Seu formato é {bitmap, bloco de código, mensagem}. AADD(aButtons,"CLIPS", AlwaysTrue(),"Usuário")

7.13.3. TcBrowser

Objeto tipo grid com uma ou mais colunas para cadastramento de dados baseado em um vetor.

Sintaxe: TCBrowse():New([nRow], [nCol], [nWidth], [nHeight], [bLine], [aHeaders], [aColSizes], [oWnd], [cField], [uValue1], [uValue2], [bChange], [bLDbClick], [bRClick], [oFont], [oCursor], [nClrFore], [nClrBack], [cMsg], [uParam20], [cAlias], [IPixel], [bWhen], [uParam24], [bValid], [IHScroll], [IVScroll])

Parâmetros:

Nome	Tipo	Descrição
nRow	numérico	Indica a coordenada vertical.
nCol	numérico	Indica a coordenada horizontal.
nWidth	numérico	Indica a largura em pixels do objeto.
nHeight	numérico	Indica a altura em pixels do objeto.
bLine	bloco de código	Indica o bloco de código da lista de campos. Observação: Esse parâmetro é utilizado somente quando o browse trabalha com array.
aHeaders	vetor	Indica o título dos campos no cabeçalho.
aColSizes	vetor	Indica a largura das colunas.
oWnd	objeto	Indica o controle visual onde o divisor será criado.
cField	caractere	Indica os campos necessários para o filtro.
uValue1	qualquer	Indica o início do intervalo para o filtro.
uValue2	qualquer	Indica o fim do intervalo para o filtro.
bChange	bloco de código	Indica o bloco de código que será executado ao mudar de linha.
bLDbClick	bloco de código	Indica o bloco de código que será executado quando clicar duas vezes, com o botão esquerdo do mouse, sobre o objeto.
bRClick	bloco de código	Indica o bloco de código que será executado quando clicar, com o botão direito do mouse, sobre o objeto.
oFont	objeto	Indica o objeto do tipo TFont utilizado para definir as características da fonte aplicada na exibição do conteúdo do controle visual.
oCursor	objeto	Indica o tipo de ponteiro do mouse.
nClrFore	numérico	Indica a cor do texto da janela.
nClrBack	numérico	Indica a cor de fundo da janela.
cMsg	caractere	Indica a mensagem ao posicionar o ponteiro do mouse sobre o objeto.
uParam20	lógico	Compatibilidade.
cAlias	caractere	Indica se o objeto é utilizado com array (opcional) ou tabela (obrigatório).
IPixel	lógico	Indica se considera as coordenadas passadas em pixels (.T.) ou caracteres (.F.).
bWhen	bloco de código	Indica o bloco de código que será executado quando a mudança de foco da entrada de dados, na janela em que o controle foi criado, estiver sendo efetuada. Observação: O bloco de código retornará verdadeiro (.T.) se o controle permanecer habilitado; caso contrário, retornará falso (.F.).
uParam24	lógico	Compatibilidade.

7.13.4. Modelo 2

Este modelo de programa exibe um cabeçalho com informações pré-determinadas, um Browse horizontal central (dependente do dicionário de dados) e um rodapé com variáveis de memória que são atualizadas de acordo com os valores preenchidos no Browse horizontal. As validações do cabeçalho são pré-determinadas no programa-fonte. Já as validações do browse horizontal são genericamente herdadas do dicionário de dados.

O nome Modelo 2 foi conceituado pela Microsiga por se tratar de um protótipo de tela para entrada de dados. Inicialmente vamos desmistificar dois pontos:

- **Função Modelo2()** – Trata-se de uma função pronta que contempla o protótipo Modelo 2, porém, este é um assunto que não iremos tratar aqui, visto que é uma funcionalidade simples que quando necessário intervir em algo na rotina não há muito recurso para tal.
- **Protótipo Modelo 2** – Trata-se de uma tela, onde seu objetivo é efetuar a manutenção em vários registros de uma só vez. Por exemplo: efetuar o movimento interno de vários produtos do estoque em um único lote.

7.13.4.1 Estrutura de um programa utilizando a Modelo2()

A função Modelo2() não implementa as regras de visualização, inclusão, alteração e exclusão, como uma AxCadastro() ou AxFunction(). A inicialização das variáveis Private utilizada nos cabeçalhos e rodapés, bem como a inicialização e gravação do aCols devem ser realizadas pela rotina que "suporta" a execução da Modelo2(). Da mesma forma, o Browse deve ser tratado por esta rotina, sendo comum a Modelo2() estar vinculada ao uso de uma MBrowse().

Sintaxe: Modelo2([cTitulo], [aCab], [aRoda], [aGrid], [nOpc], [cLinhaOk], [cTudoOk])

cTitulo	Título da janela Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice() aCab[n][1] (Caractere) := Nome da variável private que será vinculada ao campo da Enchoice(). aCab[n][2] (Array) := Array com as coordenadas do campo na tela {Linha, Coluna} aCab[n][3] (Caractere) := Título do campo na tela aCab[n][4] (Caractere) := Picture de formatação do get() do campo. aCab[n][5] (Caractere) := Função de validação do get() do campo. aCab[n][6] (Caractere) := Nome da consulta padrão que será executada para o campo via tecla F3 aCab[n][7] (Lógico) := Se o campo estará livre para digitação.
aRoda	Array contendo as informações que serão exibidas no cabeçalho na forma de Enchoice(), no mesmo formato que o aCab.
aGrid	Array contendo as coordenadas da GetDados() na tela. Padrão := {44,5,118,315}
nOpc	Opção selecionada na MBrowse, ou que deseja ser passada para controle da Modelo2, aonde: 2 - Visualizar 3 - Incluir 4 - Alterar 5 - Excluir
cLinhaOk	Função para validação da linha na GetDados()

Importante

A função Modelo2() possui um retorno lógico. Indicando se a tela da interface foi confirmada ou cancelada pelo usuário.

7.13.5. Modelo3

O nome Modelo 3, assim como a Modelo 2 foi conceituado pela Microsiga por se tratar de um protótipo de tela para entrada de dados. Inicialmente vamos desmistificar dois pontos:

- **Função Modelo3()** – Trata-se de uma função pronta que contempla o protótipo Modelo 3, porém, este é um assunto que não iremos tratar aqui, visto que é uma funcionalidade simples que quando necessário intervir em algo na rotina não há muito recurso para tal.
- **Protótipo Modelo 3** – Trata-se de uma tela, onde seu objetivo é efetuar a manutenção em vários registros de uma só vez relacionada a outro registro de outra tabela, ou seja, aqui teremos o relacionamento de registros "pai e filho", então é preciso se preocupar com este relacionamento. Por exemplo: efetuar a manutenção em um pedido de vendas, onde terá um registro em uma tabela referente ao cabeçalho do pedido e outra tabela com os registros referentes aos itens deste pedido de vendas.

Para montar a estrutura do modelo3 iremos trabalhar alguns framework e componentes visuais:

- MsDialog()
- Enchoice()
- EnchoiceBar()
- MsNewGetDados()

7.13.5.1 MsNewGetDados

Objeto tipo lista com uma ou mais colunas para cadastramento de dados baseado em um vetor. A consulta padrão, validação do usuário e gatilhos estarão habilitados se o campo estiver cadastrado no Dicionário de Dados (SX3/SX7) e apresentar estas opções disponíveis.

Nome	Tipo	Descrição
nTop	Numérico	Distância entre a MsNewGetDados e o extremidade superior do objeto que a contém.
nLeft	Numérico	Distância entre a MsNewGetDados e o extremidade esquerda do objeto que a contém.
nBottom	Numérico	Distância entre a MsNewGetDados e o extremidade inferior do objeto que a contém.
nRight	Numérico	Distância entre a MsNewGetDados e o extremidade direita do objeto que a contém.
nStyle	Numérico	Essa nova propriedade, passada via parâmetro, substitui a passagem das variáveis nOpc. Pode ser utilizada GD_INSERT + GD_UPDATE + GD_DELETE para criar a flexibilidade da MsNewGetdados.

Nome	Tipo	Descrição
cLinhaOk	Caracter	Função executada para validar o contexto da linha atual do aCols.
cTudoOk	Caracter	Função executada para validar o contexto geral da MsNewGetDados (todo aCols).
cIniCpos	Caracter	Nome dos campos do tipo caracter que utilizarão incremento automático. Este parametro deve ser no formato "+++".
aAlter	Array of Record	Vetor com os campos que poderão ser alterados.
nFreeze	Numérico	Congela a coluna da esquerda para a direita. Se 0 não congela, se 1 congela a primeira coluna. Obs: atualmente só é possível congelar a primeira coluna, devido a limitação do objeto.
nMax	Numérico	Número máximo de linhas permitidas. Valor padrão 99.
cFieldOk	Caracter	Função executada na validação do campo.
cSuperDel	Caracter	Função executada quando pressionada as teclas +.
cDelOk	Caracter	Função executada para validar a exclusão de uma linha do aCols.
oWnd	Objeto	Objeto no qual a MsGetDados será criada.
aPartHeader	Array of Record	aHeader
aParCols	Array of Record	Acols
uChange	Bloco de código	Bloco de execução a ser executado na propriedade bChange do Objeto.
cTela	Caracter	String contendo os campos contidos no X3_TELA.

7.13.5.2 Tratamento do aHeader e Acols

- **aHeader:** Existe um tratamento interno que inviabiliza a sobreposição de seu conteúdo após a criação do objeto. Vetor com informações das colunas no formato.

Elemento	Conteúdo
1	Título
2	Campo
3	Picture
4	Tamanho
5	Decimal
6	Validação
7	Reservado
8	Tipo
9	Reservado
10	Reservado

Formação Programação ADVPL



Exemplo :

```

dbSelectArea("SX3")
dbSetOrder(1)
dbSeek(cTabela)
While ! SX3->(EOF()) .And. SX3->X3_Arquivo == cAlias

If X3Uso(SX3->X3_Usado) .And. cNivel >= SX3->X3_Nivel

AAdd(aHeader, {Trim(SX3->X3_Titulo) ;
    SX3->X3_Campo ;
    SX3->X3_Picture ;
    SX3->X3_Tamanho ;
    SX3->X3_Decimal ;
    SX3->X3_Valid ;
    SX3->X3_Usado ;
    SX3->X3_Tipo ;
    SX3->X3_Arquivo ;
    SX3->X3_Context })
Endif

SX3->(dbSkip())

```

EndDo

- **aCols:** Vetor com as linhas a serem editadas. As colunas devem ser construídas com base no aHeader mais uma ultima com valor lógico que determina se a linha foi excluída.

If nOpc <> 3 // diferente de incluir

```

cChave := (cAlias)->B1_COD

dbSelectArea( cAlias )
dbSetOrder(1)
dbSeek( xFilial(cAlias) + cChave )

While ! (cAlias)->( EOF() ) .AND. ;
    ((Alias)->B1_FILIAL == xFilial(cAlias) .AND. ;
    (cAlias)->B1_COD == cChave )

        AADD( aREG, SZ3->( RecNo() ) )
        AADD( aCOLS, Array( Len( aHeader ) + 1 ) )

        For nl := 1 To Len( aHeader )
            If aHeader[nl,10] == "V"
                aCOLS[Len(aCOLS),nl] := CriaVar(aHeader[nl,2],.T.)
            Else
                aCOLS[Len(aCOLS),nl] := FieldGet(FieldPos(aHeader[nl,2]))
            Endif
        Next nl

        aCOLS[ Len( aCOLS ), Len( aHeader ) + 1 ] := .F. //Campo do Delete

```

```

(cAlias)->(dbSkip())
EndDo
Else
AADD( aCOLS, Array( Len( aHeader ) + 1 ) )
For nl := 1 To Len( aHeader )
    aCOLS[1, nl] := CriaVar( aHeader[nl], 2 ), .T. )
Next nl

aCOLS[1, AScan(aHeader,{|x| Trim(x[2])=="B1_ITEM"})] := "01" // Atribui no
//valor := 01 na array na posição

aCOLS[1, Len( aHeader ) + 1 ] := .F. //Criar o elemento para deletar a linha

Endif

```

7.14. Utilizando Querys no Protheus

Podemos utilizar querys no Protheus quando acessamos bancos de dados via TopConnect. As querys, quando bem construídas, melhoram enormemente a eficiência (velocidade) das consultas aos dados e reduzem a sobrecarga no servidor de aplicação, TopConnect e Banco de Dados. Normalmente uma query é utilizada em substituição a um Loop (While) na base de dados de programação convencional. Querys mais complexas utilizando joins poder ser construídas com a mesma função de vários loops.

O Protheus possui algumas funções:

TCQUERY()

Sintaxe	TCQUERY cSQL ALIAS cAlias NEW VIA "TOPCONN"	
	Executa uma Query no servidor e coloca seu retorno em uma WorkArea . Durante o processo de compilação	
	cSQL: Expressão SQL a ser enviada ao servidor.	
	ALIAS: cAlias especifica um nome para a WorkArea a ser aberta.	
Descrição	NEW: Abre a tabela <cTable> na próxima WorkArea disponível. Se esta cláusula não for especificada, <cTable> será na Work Área corrente. VIA: "TOPCONN" Este parâmetro indica ao ADVPL que esta Work Area será gerenciada pelo TOPconnect	
TCQUERY (cQuery) ALIAS (cAlias) NEW		

Importante

Durante o processo de compilação, a sintaxe TCQUERY() é substituída pelas expressão:

```
dbUseArea(T_,"TOPCONN",TcGenQry(,,cQuery), "ALIAS" ,T_,F_)
```

Esta substituição é realizada conforme as definições do include TOPCONN.CH. Desta forma é recomendável a utilização direta da sintaxe DbUseArea() + TcGeQry().

```
Local cAlias := GetNextAlias()
```

```
Local cSql := ""
```

```
cSql := " Select A2_COD, "
cSql +="     A2_LOJA, "
cSql +="     A2_NREDUZ, "
cSql +="     A2_EST, "
cSql +="     A2_MSBLQL "
cSql +=" FROM " + RetSQLName("SA2")
cSql +=" Where A2_FILIAL = " + xFilial('SA2') + ""
cSql +=" AND D_E_L_E_T = '' "
```

```
cSql := ChangeQuery(cSql)
```

```
TCQUERY ( cQuery ) ALIAS ( cAlias ) NEW
```

DbUseArea()

Sintaxe	<code>DBUSEArea(LNovaArea,cDriver ,cTabela,cAlias,ICompartilhado , ISomenteLeitura)</code>
Descrição	<p>LNovaArea: Caso verdadeiro, indica que a tabela deve ser aberta em uma nova workarea (Default=.F.)</p> <p>Caso o parâmetro seja .F. ou não especificado (NIL), a tabela será aberta na workarea atual. Caso já exista uma tabela aberta na WorkArea atual, a mesma é fechada.</p> <p>CDriver: Informa o Driver (RDD) a ser utilizada para a abertura da tabela. Caso não especificado (NIL), será usado o driver default de acesso a arquivos locais.</p> <p>cTabela: Nome da tabela a ser aberta. Caso o driver utilizado acesse tabelas no sistema de arquivos, deve ser informado um path no servidor de aplicação. Não é possível abrir tabelas de dados no SmartClient. Quando utilizada a RDD "TOPCONN", caso seja desejada a abertura de uma Query no SGDB, devemos passar como 3º parâmetro o retorno da função TcGenQry(), onde informamos a string contendo a Query como terceiro parâmetro da função TcGenQry()</p> <p>cAlias: Nome dado ao ALIAS desta tabela, para ser referenciado no programa Advpl.</p> <p>ICompartilhado: Caso verdadeiro, indica que a tabela deve ser aberta em modo compartilhado, isto é, outros processos também poderão abrir esta tabela.</p> <p>ISomenteLeitura: Caso verdadeiro, indica que este alias será usado apenas para leitura de dados. Caso contrário, estas operações serão perdidas</p>

Importante

```
TCGenQry ( xPar1, xPar2, cQuery )
```

Permite a abertura de uma query diretamente no banco de dados utilizado na conexão atual, mediante uso da RDD TOPCONN. O retorno desta função deve ser passado como o 3º parâmetro da função DbUseArea() -- que corresponderia ao nome da tabela.

```
Local cAliasFor := GetNextAlias()
Lcoal cSql    := ""
cSql := " Select A2_COD,   "
cSql += "     A2_LOJA,   "
cSql += "     A2_NREDUZ, "
cSql += "     A2_EST,   "
cSql += "     A2_MSBLQL "
cSql += " FROM " + RetSQLName("SA2")
cSql += " Where A2_FILIAL = '" + xFilial('SA2') + "'"
cSql += " AND D_E_L_E_T = ''"

cSql := ChangeQuery(cSql)
dbUseArea( .T.,"TOPCONN", TCGENQRY(,,cSql),(cAliasFor), .F., .T.)
```

Funções de apoio:

- **CHANGEQUERY()**
Função que efetua as adequações necessárias a query para que a mesma possa ser executada adequadamente no banco de dados em uso pela aplicação ERP através do TopConnect. Esta função é necessária pois a aplicação ERP Protheus pode ser utilizada com diversos bancos de dados, e cada banco possui particularidades em sua sintaxe, de forma que mesmo uma query escrita respeitando o padrão SQL ANSI podem necessitar de adequações.
Exemplo: CHANGEQUERY(cSql)
- **RetSQLName()**
Retorna o nome padrão da tabela para seleção no banco de dados através da query.
Exemplo: RetSQLName("SA1")
- **xFilial()**
Usada na construção da query para selecionar a filial corrente, da mesma forma que em ISAM.
Exemplo: xFilial("SA1")
- **TCSetField()**
Compatibiliza os tipos de dados diferentes de caractere retornados pela query aos tipos do ADVPL.
Exemplo: TCSetField (< cAlias>, < cCampo>, < cTipo>, [nTamanho], [nDecimais])
TCSetField("cAlias","D1_EMSSAO","D")
TCSetField("cAlias","D1_VUNIT" , "N",12,2)
- **GetNextAlias()**
Retorna um alias para ser utilizado no record set definido em dbUseArea()

7.14.1. Embedded SQL

O objetivo do Embedded SQL é facilitar a escrita e leitura de queries. Para isso, foi definida uma sintaxe para que se possa escrever a query diretamente no código AdvPL, sem a necessidade de ficar concatenando pedaços de string para compor a string final.

O bloco de código onde será escrito o comando SELECT, deve sempre ser iniciado com BeginSQL Alias e finalizado com EndSQL

Não é permitido incluir funções no meio do código embedded. Se precisar, o valor deve ser guardado em uma variável antes do início do BeginSQL.

```
Local cAliasProd := GetNextAlias()
```

```
BeginSql Alias cAlias
    Column D1_QUANT as Numeric(12,2)
    Column D1_EMISSAO as date
%NOPARSER%
    SELECT D1_DOC,
        D1_FORNECE,
        D1_LOJA,
        D1_Serie,
        D1_QUANT,
        D1_ITEM,
D1_EMISSAO
    FROM %TABLE:SD1%
    WHERE D1_FILIAL = %xFilial:SD1%
        AND D1_EMISSAO >= %EXP:pDatade%
        AND D1_EMISSAO <= %EXP:pDataAte%
        AND %NOTDEL%
EndSql
```

Características operacionais – Sintaxe

- **%Table:SB1%**: Retorna o nome padrão da tabela para seleção no banco de dados através da query.
- **%NotDel%**: é substituída por D_E_L_E_T_=''.
- **%xFilial%**: SB1% é substituída pela função xFilial("SB1")
- **%exp%**: Para informar uma variável no "Select" necessário informar %exp:NOME_DA_VARIAVEL%
- **Column**: Faz a transformação do campo igual TCSetField()
- **%NOPARSER%**: Indica que a query não deve passar pela função ChangeQuery() antes de ser enviada ao banco de dados. Caso não especificado, o padrão é a string da query ser passada automaticamente pela função ChangeQuery().

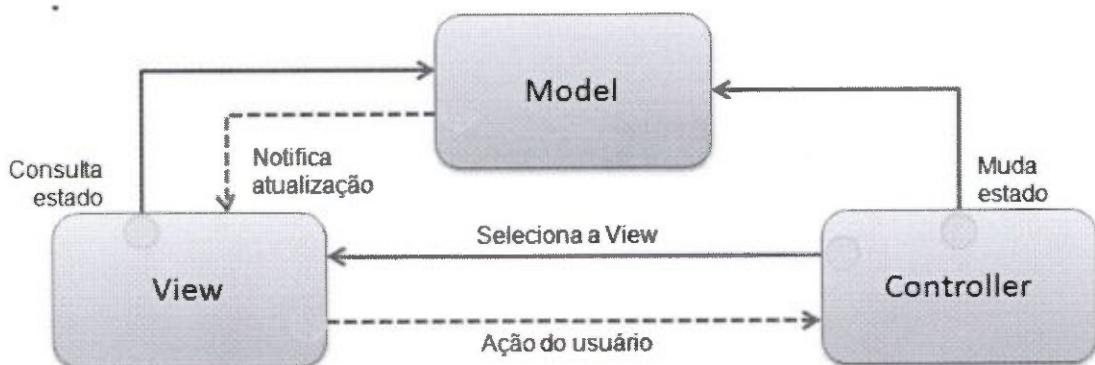
- **GetLastQuery():** Após a abertura do cursor, no alias especificado, a função GetLastQuery() retorna um array, com 5 elementos, onde estão disponíveis as seguintes informações sobre a query executada.
 - [1] cAlias - Alias usado para abrir o cursor.
 - [2] cQuery - Query executada.
 - [3] aCampos - Array de campos com critério de conversão especificados.
 - [4] INoParser - Caso verdadeiro (.T.), não foi utilizada a função ChangeQuery() na string original.
 - [5] nTimeSpend - Tempo, em segundos, utilizado para abertura do cursor.

8. Arquitetura MVC

A arquitetura **Model-View-Controller** ou **MVC**, como é mais conhecida, é um padrão de arquitetura de software que visa separar a lógica de negócio da lógica de apresentação (a interface), permitindo o desenvolvimento, teste e manutenção isolados de ambos.

Aqueles que já desenvolveram uma aplicação em AdvPL vão perceber, que justamente a diferença mais importante entre a forma de construir uma aplicação em MVC e a forma tradicional é essa separação. E é ela que vai permitir o uso da regra de negócio em aplicações que tenham ou não interfaces, como Web Services e aplicação automática, bem como seu reuso em outras aplicações.

A arquitetura MVC possui três componentes básicos:



- **Model ou modelo de dados:** representa as informações do domínio do aplicativo e fornece funções para operar os dados, isto é, ele contém as funcionalidades do aplicativo. Nele definimos as regras de negócio: tabelas, campos, estruturas, relacionamentos etc. O modelo de dados (Model) também é responsável por notificar a interface (View) quando os dados forem alterados.
- **View ou interface:** responsável por renderizar o modelo de dados (Model) e possibilitar a interação do usuário, ou seja, é o responsável por exibir os dados.
- **Controller:** responde às ações dos usuários, possibilita mudanças no Modelo de dados (Model) e seleciona a View correspondente.

Para facilitar e agilizar o desenvolvimento, na implementação do MVC feita no AdvPL, o desenvolvedor trabalhará com as definições de Modelo de dados (Model) e View, a parte responsável pelo Controller já está intrínseca.

Frisando bem, a grande mudança, o grande paradigma a ser quebrado na forma de pensar e se desenvolver uma aplicação em AdvPL utilizando MVC é a separação da regra de negócio da interface. Para que isso fosse possível foram desenvolvidas várias novas classes e métodos no AdvPL.

8.1. Principais funções da aplicação em AdvPL utilizando o MVC

Apresentamos agora o modelo de construção de uma aplicação em AdvPL utilizando o MVC. Os desenvolvedores em suas aplicações serão responsáveis por definir as seguintes funções:

- **ModelDef:** Contém a construção e a definição do Model, lembrando que o Modelo de dados (Model) contém as regras de negócio;
- **ViewDef:** Contém a construção e definição da View, ou seja, será a construção da interface;
- **MenuDef:** Contém a definição das operações disponíveis para o modelo de dados (Model).

Cada fonte em MVC (PRW) só pode conter uma de cada dessas funções. Só pode ter uma **ModelDef**, uma **ViewDef** e uma **MenuDef**. Ao se fazer uma aplicação em AdvPL utilizando MVC, automaticamente ao final, está aplicação já terá disponível.

- **Pontos de Entradas** já disponíveis;
- **Web Service** para sua utilização;
- **Importação ou exportação** mensagens XML.

8.1.1. O que é a função ModelDef?

A função ModelDef define a regra de negócios propriamente dita onde são definidas

- Todas as entidades (tabelas) que farão parte do modelo de dados (Model);
- Regras de dependência entre as entidades;
- Validações (de campos e aplicação);
- Persistência dos dados (gravação).

Para uma ModelDef não é preciso necessariamente possuir uma interface. Como a regra de negócios é totalmente separada da interface no MVC, podemos utilizar a ModelDef em qualquer outra aplicação, ou até utilizarmos uma determinada ModelDef como base para outra mais complexa.

As entidades da ModelDef não se baseiam necessariamente em metadados (dicionários), ela se baseia em estruturas e essas por sua vez é que podem vir do metadados ou serem construídas manualmente. A ModelDef deve ser uma Static Function dentro da aplicação.

8.1.2. O que é a função ViewDef?

A função ViewDef define como o usuário interage com o modelo de dados (Model) recebendo os dados informados pelo usuário, fornecendo ao modelo de dados (definido na ModelDef) e apresentando o resultado. A interface pode ser baseada totalmente ou parcialmente em um metadado (dicionário), permitindo:

- Reaproveitamento do código da interface, pois uma interface básica pode ser acrescida de novos componentes;
- Simplicidade no desenvolvimento de interfaces complexas. Um exemplo disso são aquelas aplicações onde uma GRID depende de outra. No MVC a construção de aplicações que tem GRIDs dependentes é extremamente fácil;
- Agilidade no desenvolvimento, a criação e a manutenção se tornam muito mais ágeis;
- Mais de uma interface por Business Object. Poderemos ter interfaces diferentes para cada variação de um segmento de mercado, como o varejo.

A ViewDef deve ser uma **Static Function** dentro da aplicação.

8.1.3. O que é a função MenuDef?

Uma função MenuDef define as operações quer serão realizadas pela aplicação, tais como inclusão, alteração, exclusão, etc.

Deve retornar um array em um formato específico com as seguintes informações:

- Título
- Nome da aplicação associada
- Reservado;
- Tipo de Transação a ser efetuada.

E que podem ser:

- 1 para Pesquisar
- 2 para Visualizar
- 3 para Incluir
- 4 para Alterar
- 5 para Excluir
- 6 para Imprimir
- 7 para Copiar
- Nível de acesso;
- Habilita Menu Funcional;

8.1.4. Novo comportamento na interface

Nas aplicações desenvolvidas em AdvPL tradicional, após a conclusão de uma operação de alteração fecha-se a interface e retorna ao Browse.

Nas aplicações em MVC, após as operações de inclusão e alteração, a interface permanece ativa e no rodapé exibe-se a mensagem de que a operação foi bem sucedida.

8.2. Aplicações com Browses (FWMBrowse)

Para a construção de uma aplicação que possui um Browse, o MVC utiliza a classe FWMBrowse. Esta classe exibe um objeto Browse que é construído a partir de metadados (dicionários). Esta classe não foi desenvolvida exclusivamente para o MVC, aplicações que não são em MVC também podem utilizá-la.

- Substituir componentes de Browse;
- Reduzir o tempo de manutenção, em caso de adição de um novo requisito;
- Ser independente do ambiente Microsiga Protheus.
- E apresenta como principais melhorias:
- Padronização de legenda de cores;
- Melhor usabilidade no tratamento de filtros;
- Padrão de cores, fontes e legenda definidas pelo usuário – Deficiente visual;
- Redução do número de operações no SGBD (no mínimo 3 vezes mais rápido);
- Novo padrão visual.

8.2.1. Construção básica de um Browse

Iniciamos a construção básica de um Browse. Primeiramente crie um objeto Browse da seguinte forma:

```
oBrowse := FWMBrowse():New()
```

Definimos a tabela que será exibida na Browse utilizando o método SetAlias. As colunas, ordens, etc. A exibição é obtida pelo metadados (dicionários).

```
oBrowse:SetAlias('SA1')
```

Definimos o título que será exibido como método SetDescription.

```
oBrowse:SetDescription('Cadastro de Cliente')
```

E ao final ativamos a classe.

```
oBrowse:Activate()
```

Com esta estrutura básica construímos uma aplicação com Browse.

O Browse apresentado automaticamente já terá:

- Pesquisa de registro;
- Filtro configurável;
- Configuração de colunas e aparência;
- Impressão.

8.2.2. Legendas de um Browse (AddLegend)

Para o uso de legendas no Browse utilizamos o método AddLegend, que possui a seguinte sintaxe:

```
AddLegend( <cRegra>, <cCor>, <cDescrição> )
```

Exemplo:

```
oBrowse:AddLegend( "A1_TIPO =='F'", "YELLOW", "Cons.Final" )
oBrowse:AddLegend( "ZA0_TIPO==L", "BLUE" , "Produtor Rural" )
```

cRegra: é a expressão em AdvPL para definir a legenda.

cCor: é o parâmetro que define a cor de cada item da legenda.

São possíveis os seguintes valores:

- GREEN: Para a cor Verde
- RED: Para a cor Vermelha
- YELLOW: Para a cor Amarela
- ORANGE: Para a cor Laranja
- BLUE: Para a cor Azul
- GRAY: Para a cor Cinza
- BROWN: Para a cor Marrom
- BLACK: Para a cor Preta
- PINK: Para a cor Rosa
- WHITE: Para a cor Branca
- cDescrição: a que será exibida para cada item da legenda

Importante

Cada uma das legendas se tornará automaticamente uma opção de filtro.

Cuidado ao montar as regras da legenda. Se houverem regras conflitantes será exibida a legenda correspondente à 1ª regra que for satisfeita.

8.2.3. Filtros de um Browse (SetFilterDefault)

Se quisermos definir um filtro para o Browse utilizamos o método SetFilterDefault, que possui a seguinte sintaxe:

```
SetFilterDefault ( <filtro> )
```

Exemplo:

```
oBrowse:SetFilterDefault("A1_TIPO=='F'")
ou
oBrowse:SetFilterDefault( "Empty(A1_ULTCOM)" )
```

A expressão de filtro é em AdvPL. O filtro definido na aplicação não anula a possibilidade do usuário fazer seus próprios filtros. Os filtros feitos pelo usuário serão aplicados em conjunto com o definido na aplicação (condição de AND).

Importante

O filtro da aplicação não poderá ser desabilitado pelo usuário.

8.2.4. Desabilitação de detalhes do Browse (DisableDetails)

Automaticamente para o Browse são exibidos, em detalhes, os dados da linha posicionada. Para desabilitar esta característica utilizamos o método **DisableDetails**.

Exemplo :

```
oBrowse:DisableDetails()
```

Importante

Normalmente, para se exibir campos virtuais nos Browses, fazemos uso da função **Posicione**. No novo Browse esta prática se torna ainda mais importante, pois, quando ele encontra a função **Posicione** na definição de um campo virtual e a base de dados é um Sistema de Gerenciamento de Base de Dados - SGBD (usa o TOTVSDbAccess), o Browse acrescenta um INNER JOIN na query que será enviada ao SGBD, melhorando assim o desempenho para a extração dos dados. Portanto, sempre utilize a função **Posicione** para exibir campos virtuais.

Exemplo:

```
User Function COMP012_MVC()
```

```
Local oBrowse
```

```
// Instanciamento da Classe de Browse
oBrowse := FWMBrowse():New()

// Definição da tabela do Browse
oBrowse:SetAlias('SA1')

// Definição da legenda
oBrowse:AddLegend( "A1_TIPO =='F'" , "YELLOW" , "Cons.Final"    )
oBrowse:AddLegend( "ZA0_TIPO=='L'" , "BLUE" , "Produtor Rural"  )

// Definição de filtro
oBrowse:SetFilterDefault( "A1_TIPO=='F'" )

// Título da Browse
oBrowse:SetDescription('Cadastro de Clientes')

// Opcionalmente pode ser desligado a exibição dos detalhes
//oBrowse:DisableDetails()

// Ativação da Classe
```

```
oBrowse:Activate()
```

```
Return NIL
```

8.2.5. Construção de aplicação ADVPL utilizando MVC

Iniciamos agora a construção da parte em MVC da aplicação, que são as funções de ModelDef, que contém as regras de negócio e a ViewDef que contém a interface. Um ponto importante que deve ser observado é que, assim como a MenuDef, só pode haver uma função ModelDef e uma função ViewDef em uma fonte.

Se para uma determinada situação for preciso trabalhar em mais de um modelo de dados (Model), a aplicação deve ser quebrada em vários fontes (PRW) cada um com apenas uma ModelDef e uma ViewDef.

8.2.6. Criando o MenuDef

A criação da estrutura do MenuDef deve ser uma **Static Function** dentro da aplicação.

```
Static Function MenuDef()
```

```
Local aRotina := {}
aAdd( aRotina, { 'Visualizar', 'VIEWDEF.COMP021_MVC', 0, 2, 0, NIL } )
aAdd( aRotina, { 'Incluir' , 'VIEWDEF.COMP021_MVC', 0, 3, 0, NIL } )
aAdd( aRotina, { 'Alterar' , 'VIEWDEF.COMP021_MVC', 0, 4, 0, NIL } )
aAdd( aRotina, { 'Excluir' , 'VIEWDEF.COMP021_MVC', 0, 5, 0, NIL } )
aAdd( aRotina, { 'Imprimir' , 'VIEWDEF.COMP021_MVC', 0, 8, 0, NIL } )
aAdd( aRotina, { 'Copiar' , 'VIEWDEF.COMP021_MVC', 0, 9, 0, NIL } )
```

```
Return aRotina
```

Sempre referenciaremos a ViewDef de um fonte, pois ela é a função responsável pela a interface da aplicação. Para facilitar o desenvolvimento, no MVC a MenuDef escreva-a da seguinte forma:

```
Static Function MenuDef()
```

```
Local aRotina := {}
```

```
ADD OPTION aRotina Title 'Visualizar' Action 'VIEWDEF.COMP021_MVC' OPERATION 2 ACCESS 0
ADD OPTION aRotina Title 'Incluir' Action 'VIEWDEF.COMP021_MVC' OPERATION 3 ACCESS 0
ADD OPTION aRotina Title 'Alterar' Action 'VIEWDEF.COMP021_MVC' OPERATION 4 ACCESS 0
ADD OPTION aRotina Title 'Excluir' Action 'VIEWDEF.COMP021_MVC' OPERATION 5 ACCESS 0
ADD OPTION aRotina Title 'Imprimir' Action 'VIEWDEF.COMP021_MVC' OPERATION 8 ACCESS 0
ADD OPTION aRotina Title 'Copiar' Action 'VIEWDEF.COMP021_MVC' OPERATION 9 ACCESS 0
```

```
Return aRotina
```

- **TITLE:** nome do item no menu
- **ACTION:** 'VIEWDEF.nome_do_arquivo_fonte' PRW
- **OPERATION:** Valor que define a operação a ser executada(inclusão, alteração,etc)

- **ACCESS:** Valor que define o nível de acesso

Podemos criar um menu com opções padrão para o MVC utilizando a função **FWMVCMENU**

```
Static Function MenuDef()  
Return FWMVCMenu('COMP012_MVC'))
```

Será criado um menu padrão com as opções: **Visualizar, Incluir, Alterar, Excluir, Imprimir e Copiar.**

8.2.7. Construção da função ModelDef

Nessa função são definidas as regras de negócio ou modelo de dados (Model). Elas contêm as definições de:

- Entidades envolvidas;
- Validações;
- Relacionamentos;
- Persistência de dados (gravação);

Iniciamos a função ModelDef:

```
Static Function ModelDef()  
Local oModel // Modelo de dados que será construído
```

Construindo o Model

```
oModel := MPFormModel():New( 'COMP012M' )
```

MPFormModel é a classe utilizada para a construção de um objeto de modelo de dados (**Model**).

Devemos dar um identificador (*ID*) para o modelo como um todo e também um para cada componente. Essa é uma característica do MVC, todo componente do modelo ou da interface devem ter um ID, como formulários, GRIDs, boxes, etc.

COMP012M é o identificador (*ID*) dado ao Model, é importante ressaltar com relação ao identificador (*ID*) do Model:

Importante

Se a aplicação é uma *Function*, o identificador (*ID*) do modelo de dados (**Model**) não pode ter o mesmo nome da função principal e esta prática é recomendada para facilitar a codificação. Por exemplo, se estamos escrevendo a função XPTO, o identificador (*ID*) do modelo de dados (**Model**) não poderá ser XPTO.

8.2.8. Construção de uma estrutura de dados (FWFormStruct)

A primeira coisa que precisamos fazer é criar a estrutura utilizada no modelo de dados (**Model**). As estruturas são objetos que contêm as definições dos dados necessárias para uso da **ModelDef** ou para a **ViewDef**.

- Estrutura dos Campos;
- Índices;
- Gatilhos;
- Regras de preenchimento;

O MVC não trabalha vinculado aos metadados (dicionários) do Microsiga Protheus, ele trabalha vinculado a estruturas. Essas estruturas, por sua vez, é que podem ser construídas a partir dos metadados. Com a função FWFormStruct a estrutura será criada a partir do metadado.

Sua sintaxe:

`FWFormStruct(<nTipo>, <cAlias>)`

- **nTipo:** Tipo da construção da estrutura: 1 para Modelo de dados (Model) e 2 para interface (View);
- **cAlias:** Alias da tabela no metadado;

Exemplo:

```
Local oStruSA1 := FWFormStruct( 1, 'SA1' )
```

No exemplo, o objeto oStruSA1 será uma estrutura para uso em um modelo de dados (**Model**). O primeiro parâmetro (1) indica que a estrutura é para uso no modelo e o segundo parâmetro indica qual a tabela dos metadados será usada para a criação da estrutura (SA1).

Importante

Para modelo de dados (**Model**), a função FWFormStruct, traz para a estrutura todos os campos que compõem a tabela independentemente do nível, uso ou módulo. Considera também os campos virtuais.

Para a interface (**View**) a função FWFormStruct, traz para a estrutura os campos conforme o nível, uso ou módulo.

8.2.9. Criação de componente no modelo de dados (AddFields)

O método **AddFields** adiciona um componente de formulário ao modelo.

A estrutura do modelo de dados (**Model**) deve iniciar, obrigatoriamente, com um componente de formulário.

Exemplo:

```
oModel:AddFields( 'SA1MASTER', /*cOwner*/ , oStruSA1 )
```

Devemos dar um identificador (ID) para cada componente do modelo.

- **SA1MASTER:** é o identificador (ID) dado ao componente de formulário no modelo, oStruZA0 é a estrutura que será usada no formulário e que foi construída anteriormente utilizando FWFormStruct, note que o segundo parâmetro (owner) não foi informado, isso porque este é o 1º componente do modelo, é o Pai do modelo de dados (**Model**) e portanto não tem um componente superior ou owner.

8.2.10. Criação de um componente na interface (AddField)

Adicionamos na interface (View) um controle do tipo formulário (antiga enchoice), para isso usamos o método **AddField**. A interface (View) deve iniciar, obrigatoriamente, com um componente do tipo formulário.

```
oView:AddField( 'VIEW_SA1', oStruSA1, 'SA1MASTER' )
```

Devemos dar um identificador (ID) para cada componente da interface (View).

VIEW_SA1 é o identificador (ID) dado ao componente da interface (View), **oStruSA1** é a estrutura que será usada e **SA1MASTER** é identificador (ID) do componente do modelo de dados (Model) vinculado a este componente da interface (View).

Cada componente da interface (View) deve ter um componente do modelo de dados (Model) relacionado, isso equivale a dizer que os dados do **SA1MASTER** serão exibidos na interface (View) no componente **VIEW_SA1**

8.2.11. Descrição dos componentes do modelo de dados (SetDescription)

Sempre definindo uma descrição para os componentes do modelo. Com o método **SetDescription** adicionamos a descrição ao modelo de dados (Model), essa descrição será usada em vários lugares como em Web Services por exemplo.

Adicionamos a descrição do modelo de dados:

```
oModel:SetDescription( 'Modelo de dados Cliente' )
```

Adicionamos a descrição dos componentes do modelo de dados:

```
oModel:GetModel( 'SA1MASTER' ):SetDescription( 'Dados dos Clientes' )
```

Para um modelo que só contém um componente parece ser redundante darmos uma descrição para o modelo de dados (Model) como um todo e uma para o componente, mas quando estudarmos outros modelos onde haverá mais de um componente esta ação ficará mais clara.

8.2.12. Finalização de ModelDef

Ao final da função **ModelDef**, deve ser retornado o objeto de modelo de dados (Model) gerado na função.

Return oModel

Exemplo completo da ModelDef

Static Function ModelDef()

Local oModel // Modelo de dados que será construído

```
// Cria o objeto do Modelo de Dados  
oModel := MPFormModel():New('COMP012M')
```

```

// Cria a estrutura a ser usada no Modelo de Dados
Local oStruSA1 := FWFormStruct( 1, 'SA1' )

// Adiciona ao modelo um componente de formulário
oModel:AddFields( 'SA1MASTER', /*cOwner*/, oStruSA1 )
// Adiciona a descrição do Modelo de Dados
oModel:SetDescription( 'Modelo de dados de Clientes' )

// Adiciona a descrição do Componente do Modelo de Dados
oModel:GetModel( 'SA1MASTER' ):SetDescription( 'Dados do Cliente' )

// Retorna o Modelo de dados
Return oModel

```

8.2.13. Construção da função ViewDef

A interface (View) é responsável por renderizar o modelo de dados (Model) e possibilitar a interação do usuário, ou seja, é o responsável por exibir os dados.

O **ViewDef** contém a definição de toda a parte visual da aplicação.

Iniciamos a função:

Static Function ViewDef()

A interface (View) sempre trabalha baseada em um modelo de dados (Model). Criaremos um objeto de modelo de dados baseado no **ModelDef** que desejamos.

Com a função **FWLoadModel** obtemos o modelo de dados (Model) que está definido em um fonte, no nosso caso é o próprio fonte, mas nada impediria o acesso do modelo de qualquer outro fonte em MVC, com isso podemos reaproveitar o modelo de dados (Model) em mais de uma interface (View).

```
Local oModel := FWLoadModel('COMP012M')
```

- **COMP012_MVC**: é nome do fonte de onde queremos obter o modelo de dados (Model). Iniciando a construção da interface (View).

```
oView := FWFormView():New()
```

- **FWFormView**: é a classe que deverá ser usada para a construção de um objeto de interface (View). Definimos qual o modelo de dados (Model) que será utilizado na interface (View).

```
oView:SetModel( oModel )
```

8.2.14. Exibição dos dados (CreateHorizontalBox / CreateVerticalBox)

Sempre precisamos criar um contêiner, um objeto, para receber algum elemento da interface (View). Em MVC criaremos sempre box horizontal ou vertical para isso.

O método para criação de um box horizontal é:

```
oView>CreateHorizontalBox( 'TELA' , 100 )
```

Devemos dar um identificador (*ID*) para cada componente da interface (View).

- TELA: é o identificador (*ID*) dado ao box e o número 100 representa o percentual da tela que será utilizado pelo Box.

No MVC não há referências a coordenadas absolutas de tela, os componentes visuais são sempre All Client, ou seja, ocuparão todo o contêiner onde for inserido.

8.2.15. Relacionando o componente da interface (SetOwnerView)

Precisamos relacionar o componente da interface (View) com um box para exibição, para isso usamos o método **SetOwnerView**.

```
oView:SetOwnerView( 'VIEW_SA1' , 'TELA' )
```

Desta forma o componente *VIEW_SA1* será exibido na tela utilizando o box *TELA*.

8.2.16. Finalização da ViewDef

Ao final da função ViewDef, deve ser retornado o objeto de interface (View) gerado

Return oView

Exemplo completo da ViewDef

Static Function ViewDef()

```
// Cria um objeto de Modelo de dados baseado no ModelDef() do fonte informado  
Local oModel := FWLoadModel( 'COMP012_MVC' )  
// Cria a estrutura a ser usada na View  
Local oStruZA0 := FWFormStruct( 2 , 'SA1' )  
  
// Interface de visualização construída  
Local oView  
  
// Cria o objeto de View  
oView := FWFormView():New()  
  
// Define qual o Modelo de dados será utilizado na View  
oView:SetModel( oModel )  
  
// Adiciona no nosso View um controle do tipo formulário  
// (antiga Enchoice)  
oView:AddField( 'VIEW_SA1' , oStruSA1 , 'SA1MASTER' )  
  
// Criar um "box" horizontal para receber algum elemento da view  
oView>CreateHorizontalBox( 'TELA' , 100 )
```

```
// Relaciona o identificador (ID) da View com o "box" para exibição  
oView:SetOwnerView( 'VIEW_SA1', 'TELA' )  
  
// Retorna o objeto de View criado  
Return oView
```

8.2.17. Carregar o modelo de dados de uma aplicação já existente (FWLoadModel)

Para criarmos um objeto com o modelo de dados de uma aplicação, utilizamos o função **FWLoadModel**.

FWLoadModel(<nome do fonte>)

Exemplo:

```
Static Function ModelDef()  
// Utilizando um model que ja existe em outra aplicacao  
Return FWLoadModel( 'COMP012_MVC' )
```

8.2.18. Carregar a interface de uma aplicação já existente (FWLoadView)

Para criarmos um objeto com o modelo de dados de uma aplicação, utilizamos o função **FWLoadView**.

FWLoadView (<nome do fonte>)

Exemplo:

```
Static Function ViewDef()  
// Utilizando uma view que ja existe em outra aplicacao  
Return FWLoadView( 'COMP012_MVC' )
```

8.2.19. Carregar a menu de uma aplicação já existente (FWLoadMenudef)

Para criarmos um vetor com as opções de menu de uma aplicação, utilizamos a função **FWLoadMenudef**.

– **FWLoadMenudef (<nome do fonte>)**

Exemplo:

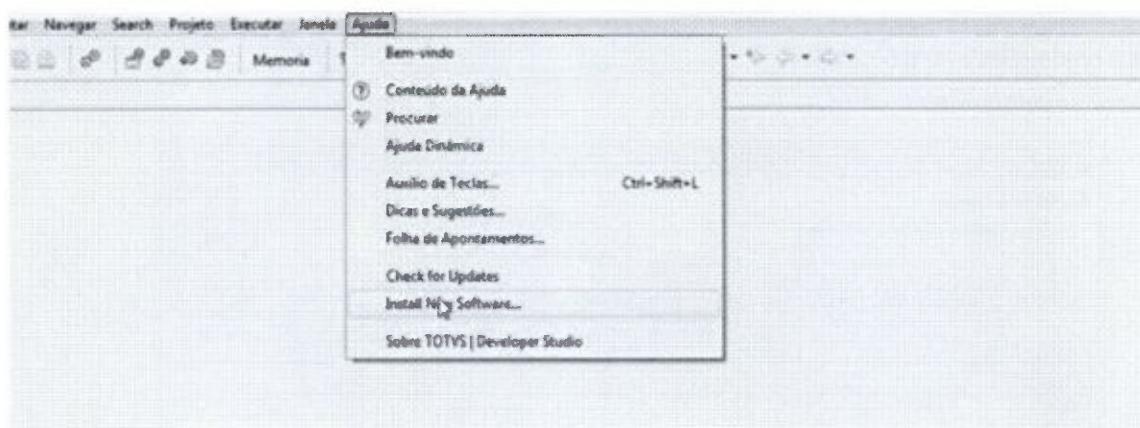
```
Static Function MenuDef()  
// Utilizando um menu que ja existe em outra aplicacao  
Return FWLoadMenuDef( 'COMP012_MVC' )
```

8.3. Instalação do Desenhador MVC

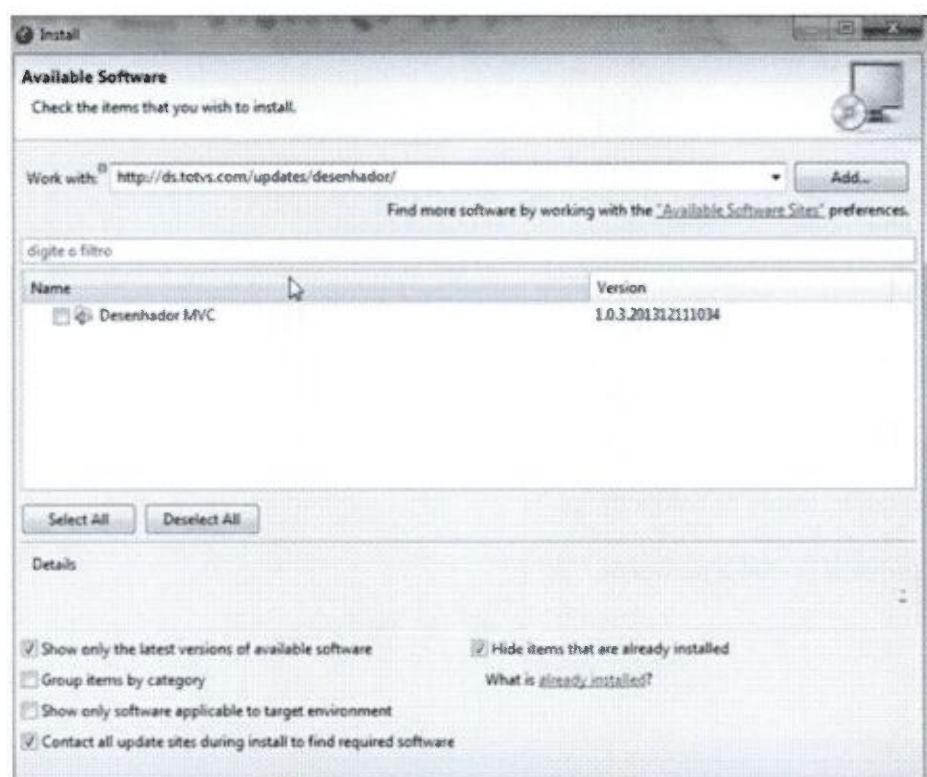
O **Desenhador MVC** é um conjunto de *plug-ins* que funcionam dentro do **TDS (Totvs Developer Studio)** sendo uma ferramenta de auxílio na criação e manutenção de códigos fontes escritos em AdvPL em conjunto com o framework de desenvolvimento.

Para fazer a instalação do Desenhador MVC é necessário ter instalado o TDS.

Iremos fazer a instalação do Plugin MVC. Ao acessar o TDS ir na opção Ajuda->Install New Software



Após acessar preencher Work With com endereço do instalador do desenhador MVC:
<http://ds.totvs.com/updates/desenhador/>



As opções tem que estar habilitadas:

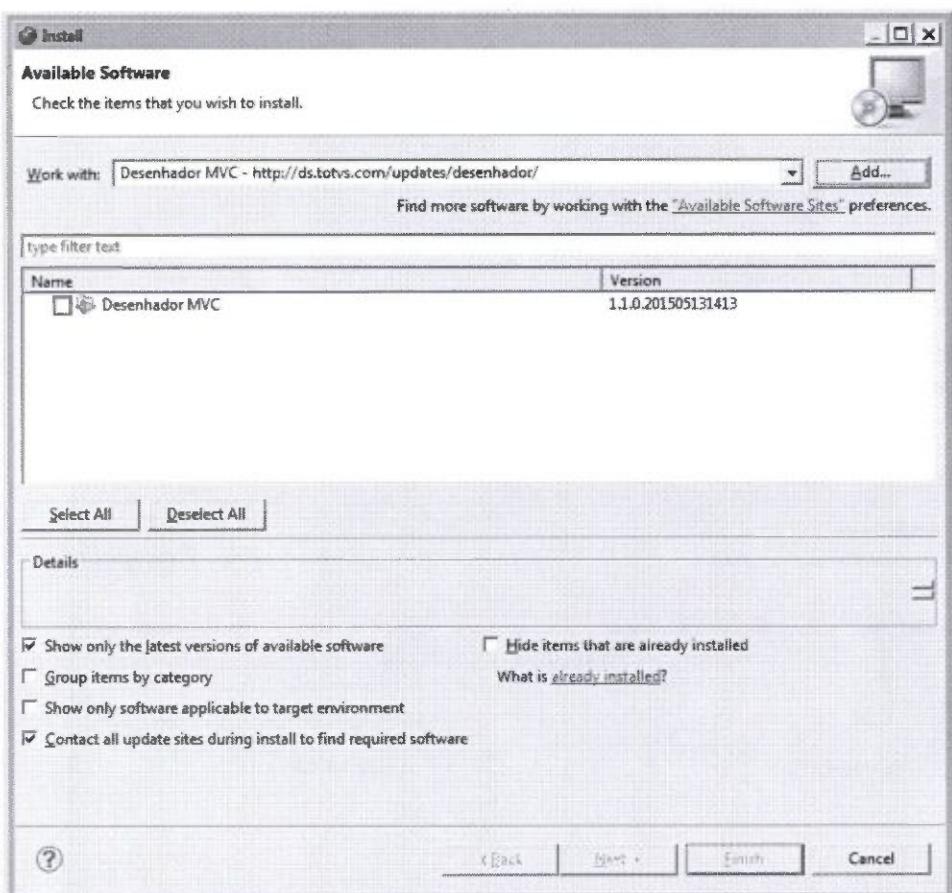
- Show Only the latest version of available software
- Contact all update sites during install to find required software

As demais opções desabilitada

Selecionar a opção:

- Desenhador MVC

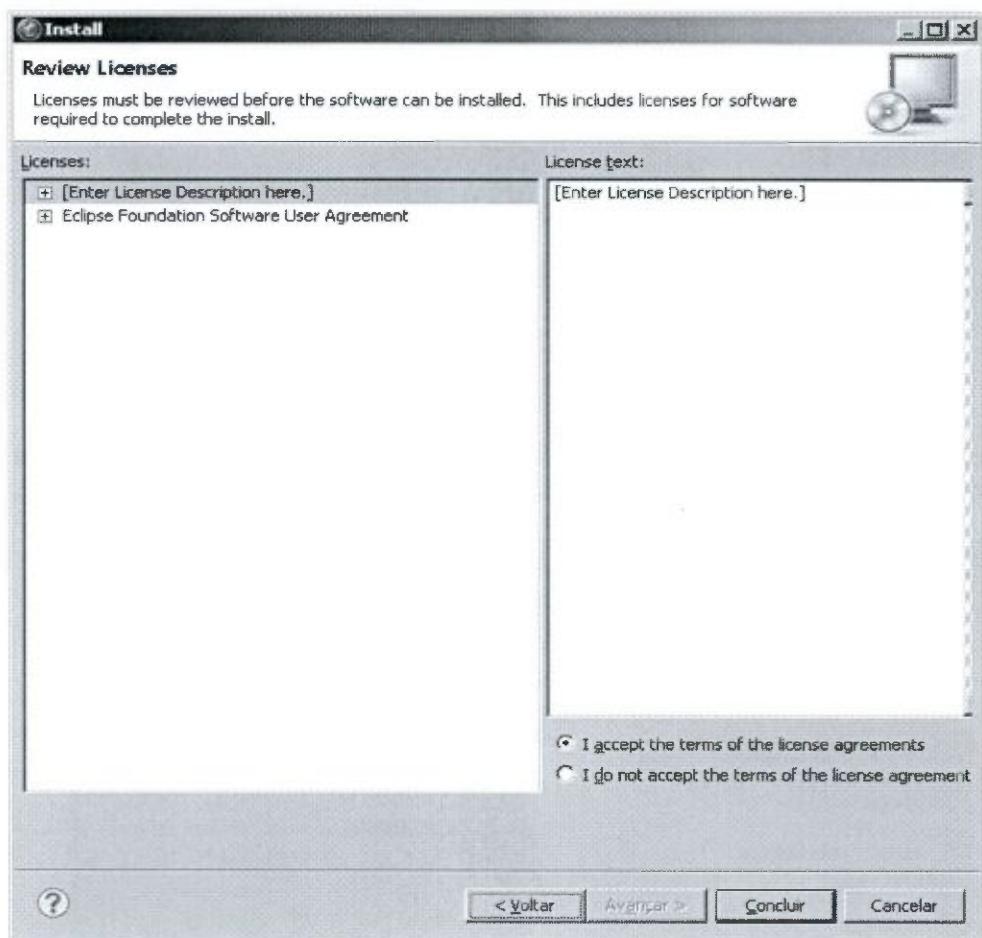
Após marcar a opção selecionar o botão Next.



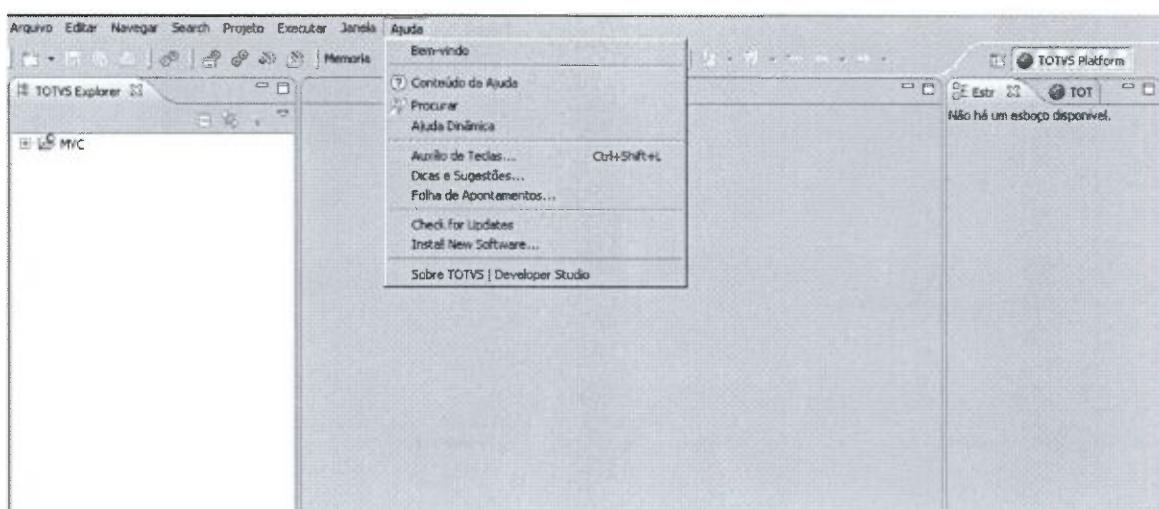
Seleciona a opção que aceita o termos de licença de uso.

- I accept the terms of license agreements

Formação Programação ADVPL



Selecionar o botão Concluir, após esse processo a desenhador sera instalado, para confirmar a instalação do desenhador acessar Ajuda selecionar a opção Sobre Totvs | Developer Studio.



Selecionar o botão "Detalhes da Instalação".



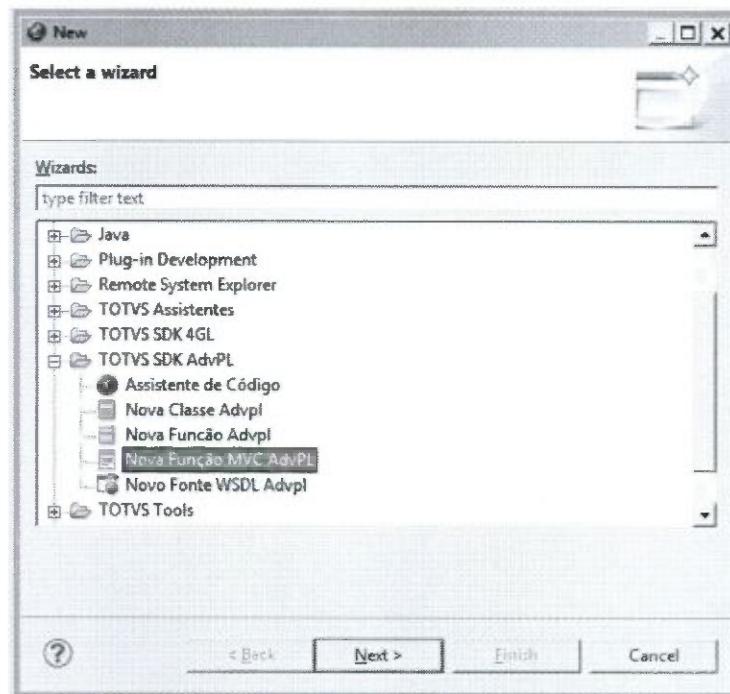
Ira listar todos os plug-ins instalado no TDS, localizar o “Desenhador MVC”, caso não esteja na lista refazer o processo novamente caso ocorrer algum erro na instalação contate o suporte da Totvs.



8.3.1. Criação de um novo fonte Desenhador MVC

Os fontes MVC AdvPL precisam necessariamente estar dentro de um projeto TOTVS.

Para criar um fonte novo, clique em Arquivos->Novo->Outras, escolha o item Nova Função MVC ADVPL do Totvs SDK AdvPL.



Na tela do novo fonte MVC, preencher os campos:



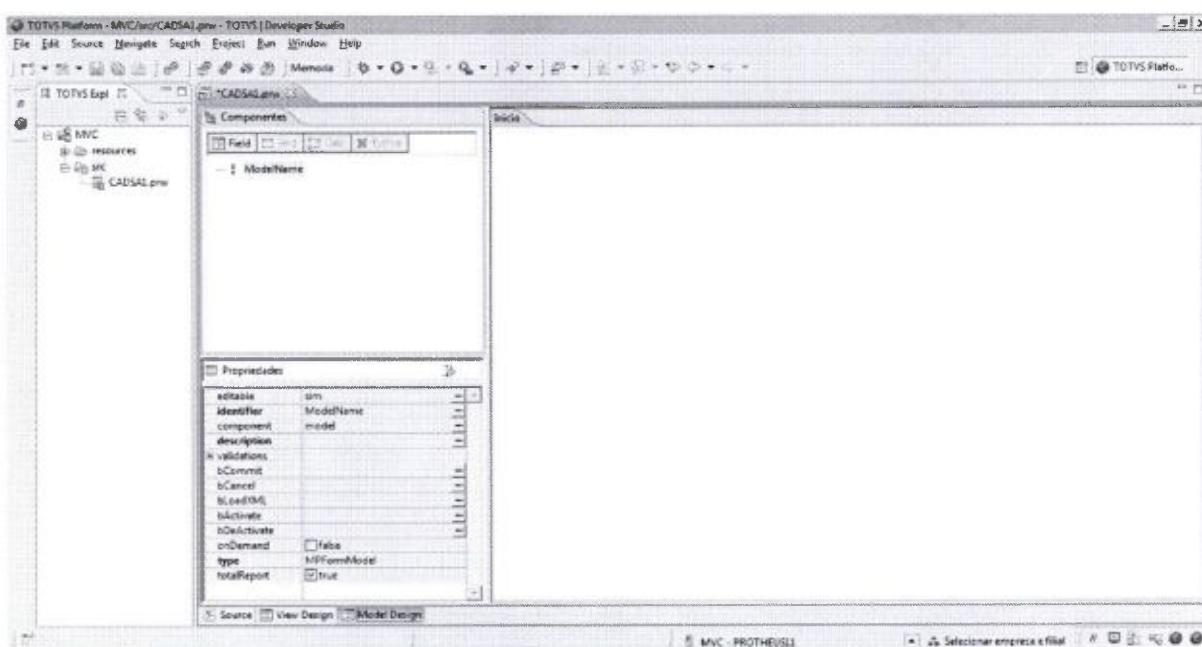
- **Local:** Informe a pasta do projeto que deseja salvar o fonte.
- **Nome da função:** Informe o nome do PRW.
- **Criar Arquivo com referência:** Para salvar o fonte fora do WorkSpace.

O Fonte é aberto com o Desenhador MVC possui três abas Source, View e Model.



O com novo plugin MVC possui novas validações do compilador facilitando o desenvolvimento.

Para criar uma estrutura MVC necessário criar Model onde iremos definir qual estrutura iremos trabalhar, selecionar a aba Model Design.



Na guia Model possui todos os componentes para criação do ModelDef:

Componentes:

- **Field** – Adiciona no Model um sub-Modelo de campos "Entoiche"
- **Grid** - Adiciona no Model um sub-Modelo de grid "GetDados"
- **Calc** – Adiciona no grid campos totalizadores

Propriedades:

- **Editable**: Se o modelo for carregado usando a função FWloadModel não é possível alterar as propriedades
- **Identifier**: Identificador do Modelo
- **Component**: Nome do componente
- **Description**: Atribui ao modelo um texto explicativo sobre o objetivo do Modelo. O objetivo é mostrado em diversos operações, tais como web services, relatórios e schemas (xsd).
- **Validations**:
bPre: Bloco de código de pré-validação do modelo. O bloco recebe como parametro o objeto de Model e deve retornar um valor lógico. Quando houver uma tentativa de atualização de valor de qualquer Submodelo o bloco de código será invocado. Caso o retorno seja verdadeiro, a alteração será permitida, se retornar falso não será possível concluir a alteração e um erro será atribuido ao model.

bPos: Bloco de código de pós-validação do modelo, equilave ao "TUDOOK". O bloco recebe como parametro o objeto de Model e deve retomar um valor lógico. O bloco será invocado antes da persistência dos dados para validar o model. Caso o retorno seja verdadeiro e não haja nenhum submodelo invalido, será feita a gravação dos dados. Se retornar falso não será possível realizar a gravação e um erro será atribuido ao model.

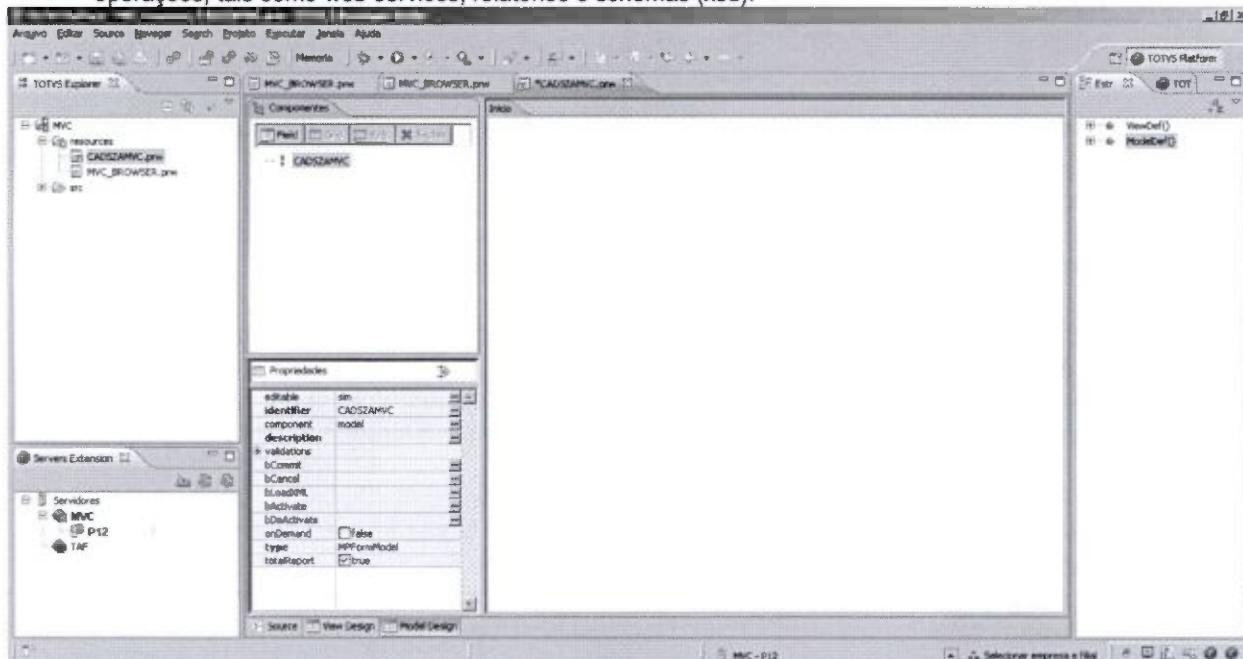
bVldActivate: Bloco de código que será chamado antes do Activate do model. Ele pode ser utilizado para inhibir a inicialização do model. Se o retorno for negativo uma exceção de usuário será gerada. O code-block recebe como parametro o objeto model.

bCommit: Bloco de código de persistência dos dados, ele é invocado pelo método CommitData. O bloco recebe como parametro o objeto do Model e deve realizar a gravação dos dados.

- **bCancel**: Bloco de código de cancelamento da edição, ele é invocado pelo método CancelData. O bloco recebe como parametro o objeto do Model.
- **bLoadXML**: Configura o modelo para ser ativado com uma folha de dados em XML. Bloco de código recebe o modelo por parametro e deve retornar um XML com os dados baseado no modelo.
- **bActivate**: Bloco de código que será chamado logo após o Activate do model. Esse bloco recebe como parametro o proprio model.
- **bDeActivate**: Bloco de código que será chamado logo após o DeActivate do model. Esse bloco recebe como parametro o proprio model.
- **onDemand**: Define se a carga dos dados será por demanda. Quando o model é por demanda, os dados do submodelo não serão carregados de uma vez no activate. Conforme eles forem manipulados eles serão carregados. O método é útil em modelos que contem muitos submodelos e que podem ficar lentos devido a grande quantidade de dados.
- **Type**: A opção MPFormModel realiza tratamento nas variaveis de memória e possibilita o uso do Help para as mensagens. A opção FWFormModel não realiza esses tratamentos, por isso é necessário utilizar o SetErrorMessage em todas as mensagens de erro. A opção mais utilizada é a MPFormModel.
- **TotalReport**: Configura se o relatório gerado pelo metodo ReportDef deverá totalizar todos os campos numéricos.

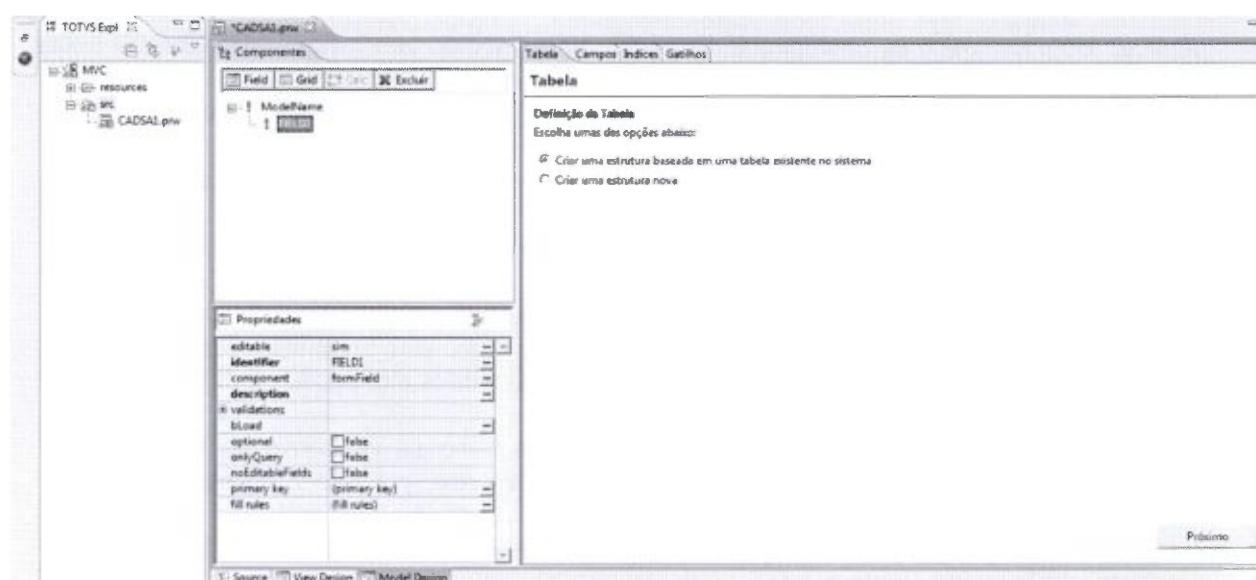
Para criar a estrutura básica do Model necessário informar os campos:

- Identifier:** Nome do Model, o identificador (ID) do modelo de dados (Model) pode ter o mesmo nome da função principal e esta prática é recomendada para facilitar a codificação.
- Component:** Nome do componente
- Description:** Adicionamos a descrição ao modelo de dados (Model). O objetivo é mostrado em diversos operações, tais como web services, relatórios e schemas (xsd).



Após criar estrutura do modelo de dados (Model), necessário adicionar um componente de formulário, iremos criar no exemplo o componente "Field".

Para criar o componente selecionar na guia de componentes a opção Field, após selecionar o componente, irá aparecer dentro da estrutura do Model.



Formação Programação ADVPL



Nas propriedade dos componente possui a estrutura:

- **Definições da Tabela:**

- **Cria a estrutura a ser usada no Modelo de Dados:** Lista as tabelas existentes no SX2
- **Cria uma uma estrutura nova:** Criar estrutura de tabela temporária

Na guia tabela irá lista todos os dados do arquivo SX2.

Alias	Descrição
AA1	Atendentes
AA2	Habilidades dos Atendentes
AA3	Base de Atendimento
AA4	Acessórios da Base Atendimento
AA5	Serviços
AA6	Kits de Atendimentos
AA7	Produtos x Ocorrências
AA8	Plano de Manutenção
AA9	Itens do Plano de Manutenção
AAA	Grupos de Cobertura
AA8	Itens do Grupo de Cobertura
AAC	Habilidades da Amarração
AAD	Índices
AAE	Índices - Taxas
AAF	Históricos
AAG	Ocorrências
AAH	Contrato de Manutenção
AAI	FAQ
AAJ	Preventiva
AAK	Obscuridão
AAK	Obscuridão

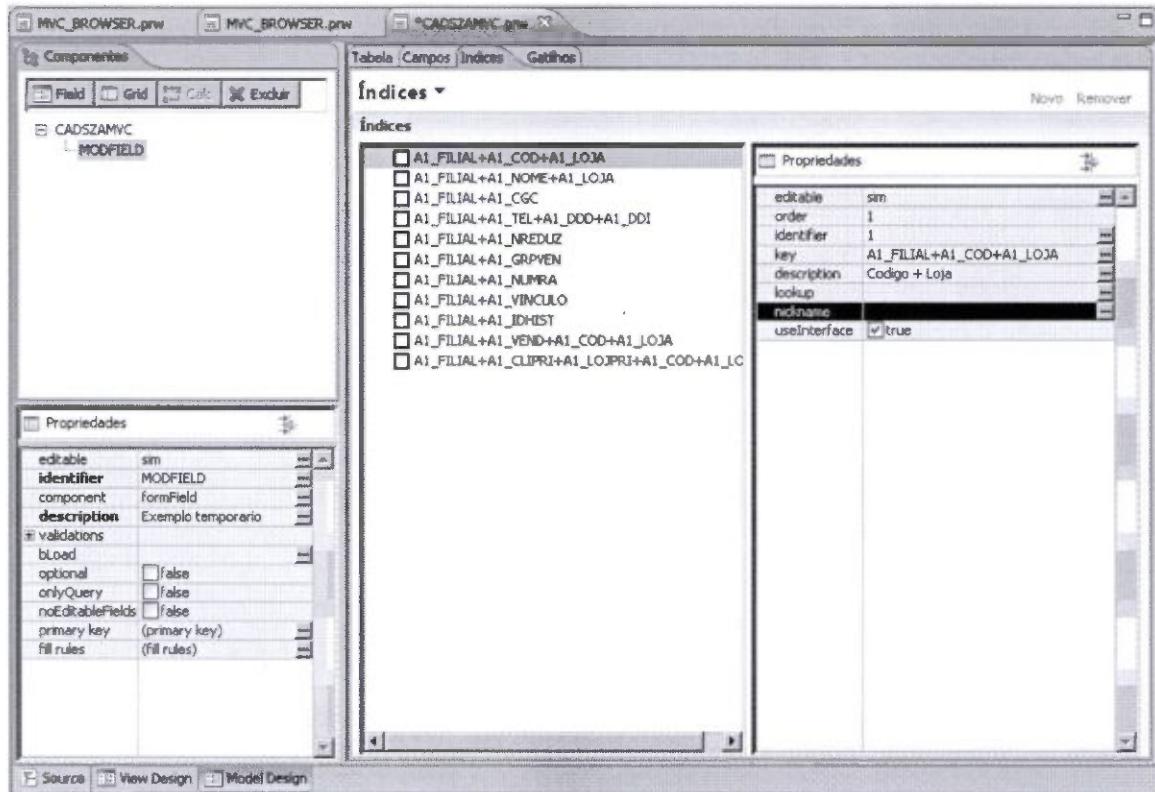
Para relacionar a tabela selecionar na lista

Na pasta "Campos" irá lista os campos com as propriedades dele, podemos interagir nos dados, mas não altera a estrutura no SX3.

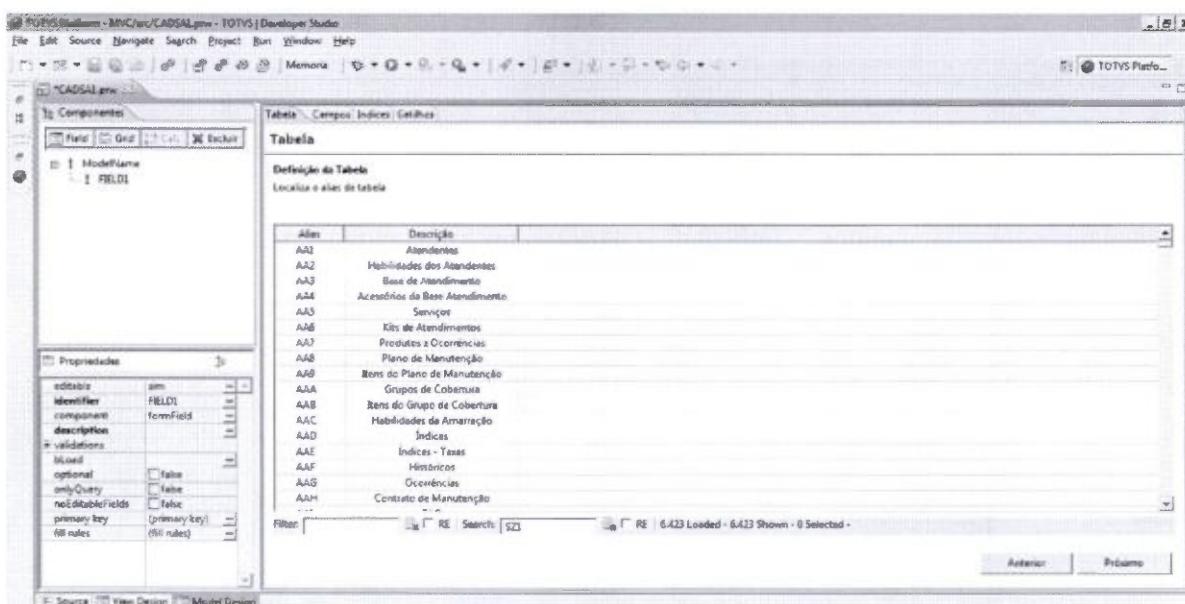
Nome	Propriedades
AI_FILIAL	editable: sim identifier: AI_FILIAL component: formField description: Exemplo temporario validators: blad: optional: false onlyQuery: false noEditableFields: false primary key: (primary key) Fill rules: (fill rules)
AI_FONE	
AI_FONE	
AI_NOME	
AI_PESSOA	
AI_MESMA2	
AI_SINR	
AI_BAIRRO	
AI_TIPO	
AI_EST	
AI_ESTACAO	
AI_CEP	
AI_COD_MUN	
AI_MUN	
AI_REGIAO	
AI_DSRES	
AI_NATUREZ	
AI_DSCOD	
AI_DSCOB	
AI_DSC	
AI_DIREC	
AI_DENDIT	
AI_TEI	
AI_TELPAN	
AI_CONTATO	
AI_FAX	
AI_CPF	

Todos os campos na lista irá aparecer na estrutura.

Na pasta Indices, lista os dados da SIX com as suas propriedades



Irá listar todas as tabelas do SX2 no grid, selecionar a tabela que irá trabalhar, após informar a tabela selecionar o botão Próximo.

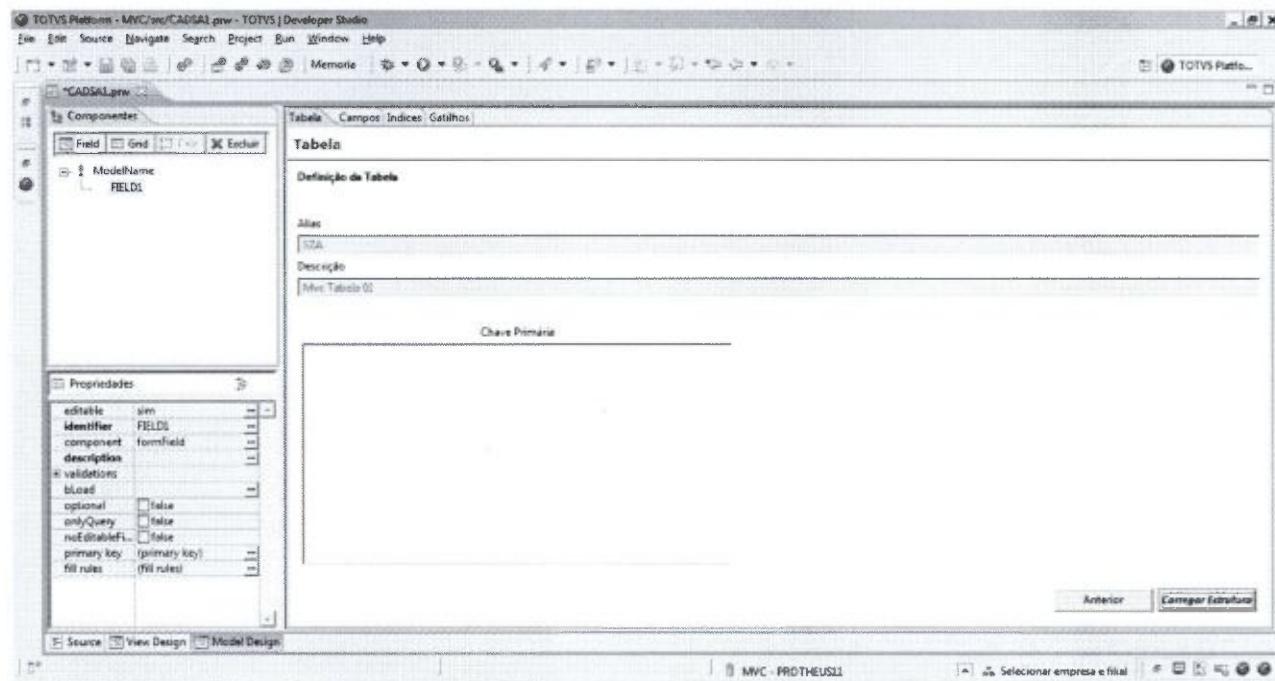


Formação Programação ADVPL

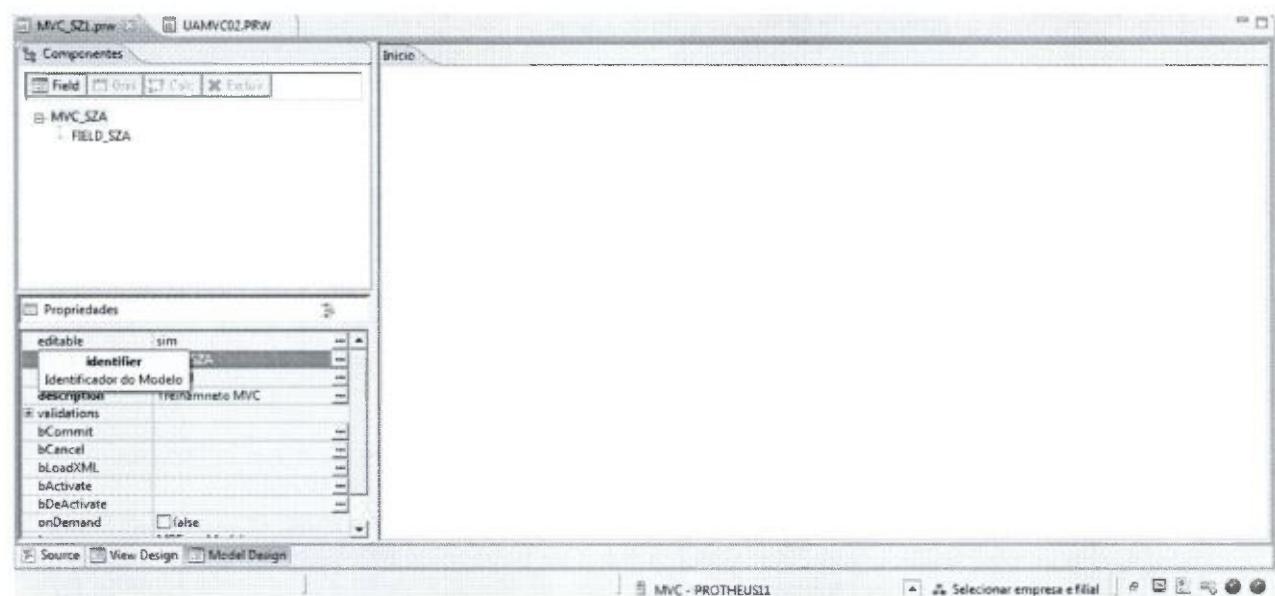


Os índices e gatilhos (SX7) não podem ter suas propriedades alteradas, os campos (SX3) você pode alterar as propriedades mas lembre-se que as alterações ficam no fonte, não refletem no dicionário de dados do Protheus. Preferencialmente faça as alterações no dicionário.

Ao informar a tabela, selecionar o botão “Carregar Estrutura”, todos os campos, índices e gatilhos da tabela foram carregados para a estrutura de dados.

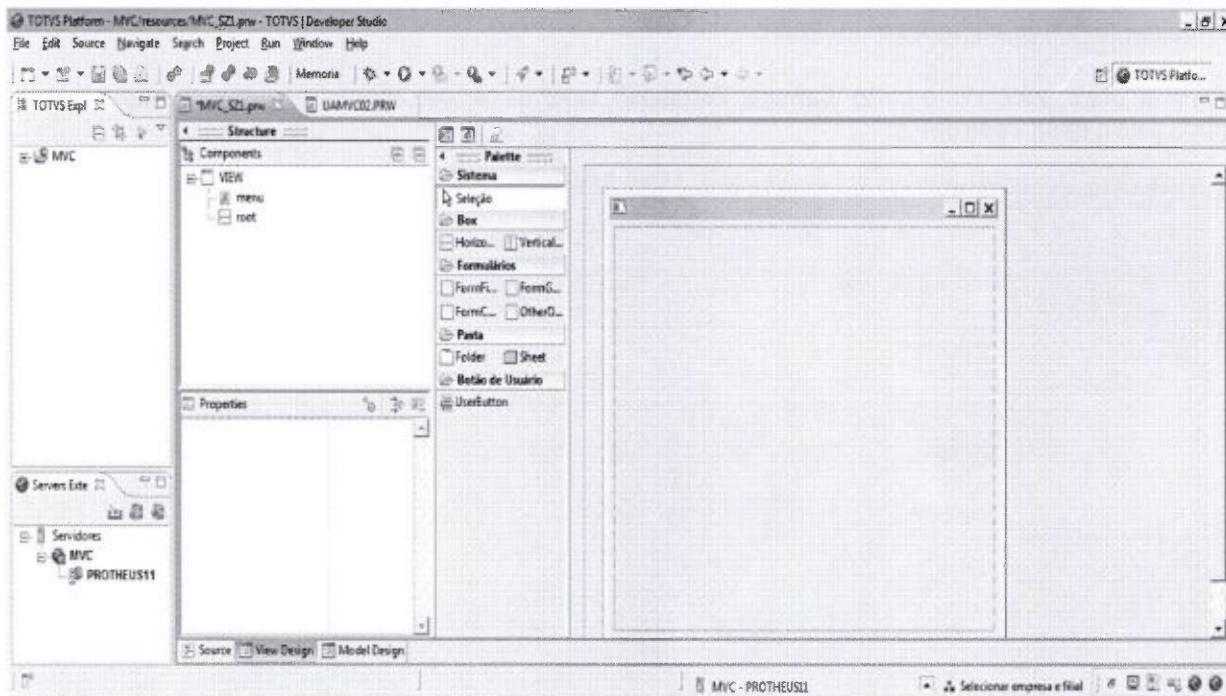


Na Guia de propriedades você tem todas as opções Model, ao passar o mouse sobre as propriedades exemplo: “Identifier”, sera aberto uma ajuda com as explicações do bloco de persistência dos dados.



Após criar Model iremos definir as propriedades do View, selecionar no fonte MVC a aba View Design.

Ira listar todas as propriedades para definir a função ViewDef.



Na árvore de componentes:

- **View:** Estrutura do View
- **Menu:** Botões no View ira listar todo os botões criado pelo “UserButton”
- **Root:** Lista todos os componentes visuais
- **Menu:** É um componente onde serão colocados os botões de usuário.
- **Root:** É o componente base onde os componentes de interface serão alocados.
- **Palette:** Possui todos os componentes de visualização do Model

Palette

- **Box:** Caixa de visualização do Model
HorizontalBox: O Box dever ser utilizado para dividir a tela. O Box Horizontal divide a tela na posição horizontal e todos os componentes que forem criados dentro dele também assumirão essa orientação. Atenção: Esse box somente pode ser colocado dentro de um box vertical.
VerticalBox: Box é um componente utilizado para dividir a tela. O Box Vertical divide a tela na posição vertical e todos os componentes que forem criados dentro dele também assumirão essa orientação. Atenção: Esse box somente pode ser colocado dentro de um box horizontal.
- **Formulários:**
FormField: Formulário que exibe campos no mesmo modelo da antiga enchoice. Para usar esse componente se faz necessário criar um submodelo do tipo field no Model e associar o mesmo ao formulário.
FormGrid: Formulário que exibe um grid, ele mostra diversos registros na tela. Para usar esse componente se faz necessário criar um submodelo do tipo grid no Model e associar o mesmo ao formulário.
FormCalc: Formulário que exibe campos calculados criados no Model. Para usar esse componente se faz necessário criar um submodelo do tipo calc no Model e associar o mesmo ao formulário.

Formação Programação ADVPL



OtherObject:: Cria um painel onde pode ser colocado qualquer coisa que o desenvolvedor desejar. Para criar as coisas use o bloco de código de activate do componente. Atenção: Tudo que for criado dentro desse componente NÃO será renderizado nesse plugin.

- **Pasta:**

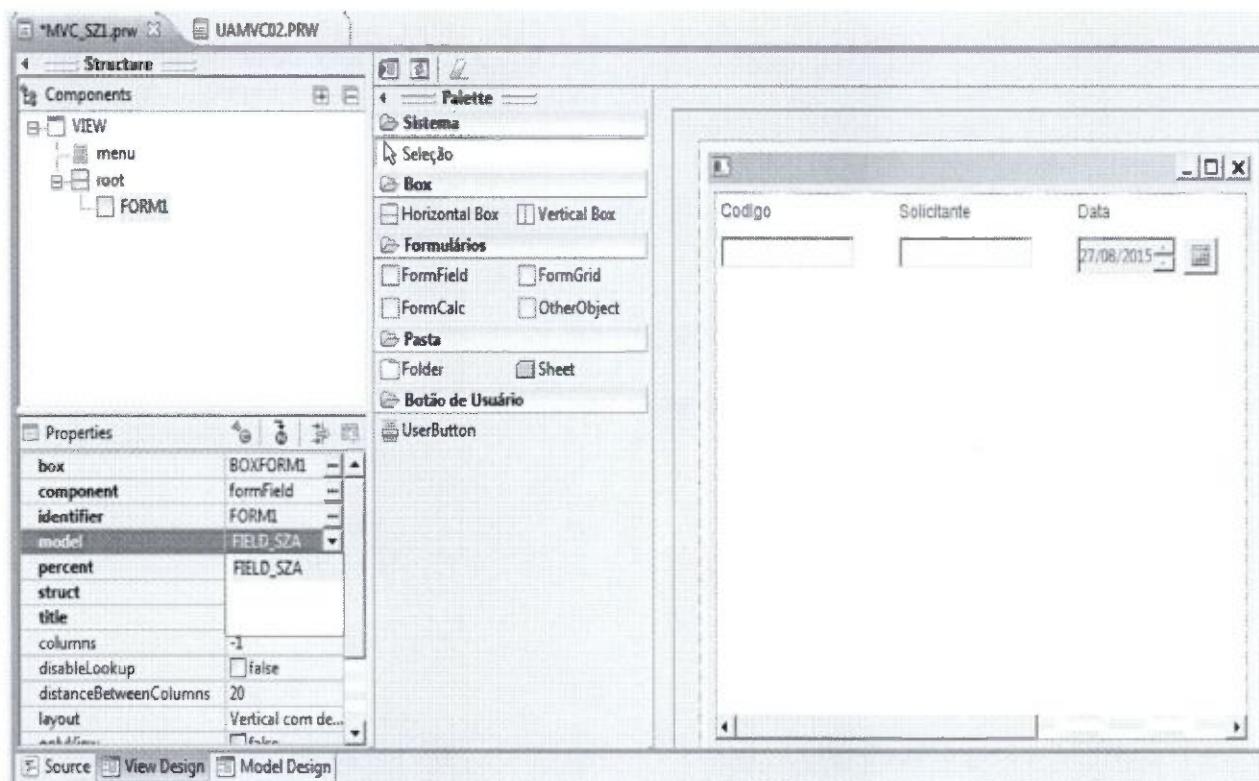
Folder: Cria um componente de pasta que é capaz de armazenar dentro dele diversas abas

Sheet: Cria uma aba dentro de uma Folder. Esse componente permite criar novas divisões dentro da aba, criando diversas opções na tela.

- **Botão de Usuário:**

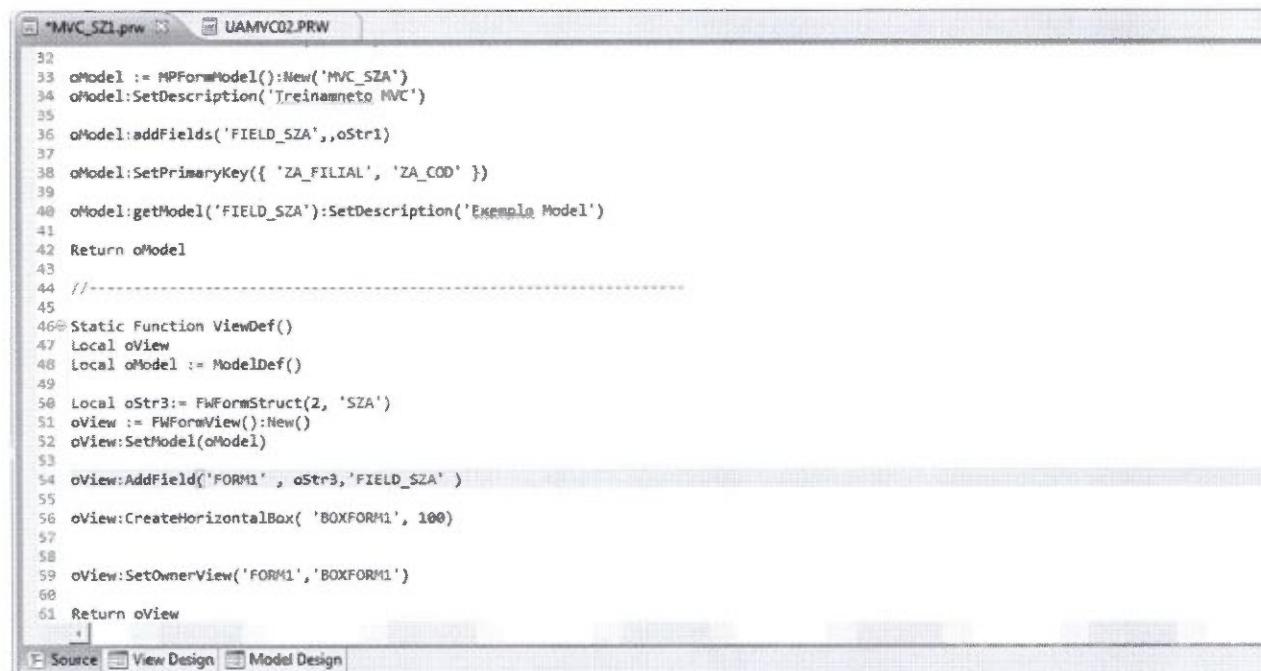
UserButton: Adiciona botões do desenvolvedor na barra de ferramentas do formulário.

Para criar a ViewDef necessário criar um componente de visualização, selecionar o componente "FormField" e adicionar no componte "root", em propriedades "Model" informar a identificador do Model criado.



Acabamos de criar um fonte em MVC com informações básicas.

Tudo que foi desenvolvido nas guias View e Model criou o fonte na guia Source.



```

32
33 oModel := MPFormModel():New('MVC_SZA')
34 oModel:SetDescription('Treinamento MVC')
35
36 oModel:addFields('FIELD_SZA',,oStr1)
37
38 oModel:SetPrimaryKey({ 'ZA_FILIAL', 'ZA_COD' })
39
40 oModel:getModel('FIELD_SZA'):SetDescription('Exemplo Model')
41
42 Return oModel
43
44 //-----
45
46@ Static Function ViewDef()
47 Local oView
48 Local oModel := ModelDef()
49
50 Local oStr3:= FWFormStruct(2, 'SZA')
51 oView := FWFormView():New()
52 oView:SetModel(oModel)
53
54 oView:AddField('FORM1', oStr3,'FIELD_SZA' )
55
56 oView>CreateHorizontalBox( 'BOXFORM1', 100)
57
58
59 oView:SetOwnerView('FORM1','BOXFORM1')
60
61 Return oView

```

Source View Design Model Design

8.4. Tratamentos para o Modelo de dados (Model)

Veremos alguns tratamentos que podem ser feitos no modelo de dados (*Model*) conforme a necessidade:

- Validações;
- Comportamentos;
- Manipulação da Grid.
- Obter e atribuir valores ao modelo de dados (*Model*);
- Gravação dos dados manualmente;
- Regras de preenchimento.

8.4.1. Retorno da operação que está sendo realizada (GetOperation)

Para sabermos a operação com que um modelo de dados (*Model*) está trabalhando, usamos o método **GetOperation**. Esse método retorna:

- O valor 3 quando é uma **inclusão**;
- O valor 4 quando é uma **alteração**;
- O valor 5 quando é uma **exclusão**.

Para acassser a operção necessario esta com o Model

Local nVar := oModel:GetOperation()

Para as operações do modelo de dados (*Model*) podem ser utilizados:

- **MODEL_OPERATION_INSERT**

- MODEL_OPERATION_UPDATE
- MODEL_OPERATION_DELETE

Exemplo:

```
If oModel:GetOperation() == MODEL_OPERATION_INSERT  
    Help(,, 'Help', 'Não permitido incluir.', 1, 0)  
Endif
```

8.4.2. Retorno do componente do modelo de dados (GetModel)

Durante o desenvolvimento teremos que manipular o modelo de dados (Model), para facilitar essa manipulação podemos ao invés de trabalhar como o modelo todo pode trabalhar com uma parte específica (um componente) de cada vez.

```
Local oModel := oModel:GetModel( 'NOME DO ID DO MODELO DE DADOS' )
```

Podemos capturar o modelo completo

```
Local oModel := oModel:GetModel()
```

8.4.3. Componente do modelo de dados Ativo FWModelActive()

Esta função fornece o último objeto da classe FWFormModel ativo, para ser utilizado nas regras de validação do sistema.

Retorno do model de dado Ativo:

```
Local oModel := FwModelActive()
```

Para definir qual o modelo de dados (Model) ativo:

```
Local oModelBkp := FWModelActive()  
FWModelActive( oModelBkp )
```

8.4.4. Carregar o modelo de dados (FWLoadModel)

Para criarmos um objeto com o modelo de dados de uma aplicação, utilizamos o função FWLoadModel.
FWLoadModel(<nome do fonte>)

Exemplo:

```
Static Function ModelDef()  
// Utilizando um model que ja existe em outra aplicacao  
Return FWLoadModel( 'NOMEDOPRW' )
```

8.5. Obtendo os valores do modelo de dados

As operações mais comuns que faremos em um modelo de dados (*Model*) é obter valores.

Para isso utilizamos um dos métodos abaixo:

- **GetValue:** Obtém um dado do modelo de dados (*Model*). Podemos obter o dado a partir do modelo completo ou a partir de um componente dele.

A partir do modelo de dados (*Model*) consegue acessar as estruturas dos campos podendo capturar ou manipular o dado no (*Model*).

```
cVar := oModel:getValue( 'ID do componente', 'Nome do campo para obter o dado' )
```

Ou em casos que não tenha recuperado o modelo da tabela em uma variável:

```
cVar := oModel:GetModel('ID'):GetValue('Nome do campo do qual se deseja obter o dado')
```

- **FWFIdGet():** Retorna o valor de um campo do Modelo.

Nome	Tipo	Descrição	Default	Obrigatório
cCampo	Caracteres	Nome do campo que se deseja pegar o conteúdo		X
nLinha	Numérico	Número da linha do campo que se deseja pegar o conteúdo quando ele está numa FORMGRID	0	
oModel	Objeto	Modelo de dados de referência	FWMODELACTIVE()	
lShowMsg	Lógico	.T./.F. Exibe ou não mensagem de erro quando o campo não é encontrado no Model	.T.	

Exemplo:

```
cNome := FWFIdGet("A1_NOME")
```

8.5.1. Atribuindo valores no modelo de dados

As operações mais comuns que faremos em um modelo de dados (*Model*) é atribuir valores.

Para isso utilizamos um dos métodos abaixo:

- **SetValue:** Atribui um dado ao modelo de dados (*Model*). Podemos atribuir o dado a partir do modelo completo ou a partir de uma parte dele. Quando utilizamos o SetValue para atribuir um dado a um campo as validações deste campo são executadas e também são disparados os seus gatilhos.

O SetValue retorna .T. (verdadeiro) se a atribuição foi bem sucedida, os motivos para que não seja podem ser que o dado não satisfez a validação ou o modo de edição (WHEN) não foi satisfeito, etc.

- **LoadValue:** Atribui um dado ao modelo de dados (Model) sem executar as validações nos campo.

A partir do modelo de dados (*Model*) completo

Exemplo:

```
oModel:SetValue( 'ID', 'Nome do Campo', 'Valor para o campo' )
```

Ou em casos que não tenha recuperado o modelo da tabela em uma variável:

Exemplo:

```
oModel:SetValue('Nome do Campo', 'Valor para o campo' )
```

- **FWFIdPut():** Atribui um conteúdo a de um campo do Modelo.

Nome	Tipo	Descrição	Default	Obrigatório
cCampo	Caracteres	Nome do campo que se deseja atribuir o conteúdo		X
xConteudo	Qualquer	Conteúdo a ser atribuído		X
nLinha	Numérico	Número da linha do campo que se deseja colocar o conteúdo quando ele está numa FORMGRID	0	
oModel	Objeto	Modelo de dados de referência	FWMODELACTIVE()	
lShowMsg	Lógico	.T./.F. Exibe ou não mensagem de erro quando o campo não é encontrado no Model	.T.	
lLoad	Lógico	.T./.F. Se "força" a entrada do conteúdo, executando o LoadValue ao invés do SetValue.	.F.	

Exemplo:

```
FWFIdPut("A1_NOME", "TOTVS")
```

8.5.2. Mensagens exibidas na interface

As mensagens são usadas principalmente durante as validações feitas no modelo de dados. A validação é um processo executado dentro da regra de negócio e uma eventual mensagem de erro que será exibida ao usuário, é um processo que deve ser executado na interface, ou seja, não pode ser executado na regra de negócios camada modelo de dados.

Para trabalhar essa situação foi feito um tratamento para a função **help**. A função **help** poderá ser utilizada nas funções dentro do modelo de dados (Model), porém o MVC irá guardar essa mensagem e ela só será exibida quando o controle voltar para a interface. Por exemplo, uma determinada função conterá:

Exemplo:

```
If nPrcUnit == 0 // Preço unitário
  Help(,, 'Título do campo',,, 'Mensagem a ser exibida no help.', 1, 0 )
EndIf
```

Supondo que a mensagem de erro foi acionada porque o preço unitário é 0 (zero), neste momento não será exibido nada ao usuário, isso pode ser observado ao debugar o fonte. Você verá que ao passar pela função Help nada acontece, porém, quando o controle interno volta para a interface, a mensagem é exibida.

8.6. Manipulação do Modelo de dados

Fornece um objeto com o modelo hierárquico dos campos de edição para o Microsiga Protheus. Esta classe é uma herança da classe modelo FWFormModel, utilizada apenas para facilitar o desenvolvimento do padrão MVC no Protheus

```
oModel:=New(<cID>, <bPre>, <bPost>, <bCommit>, <bCancel>)
```

- **bPre:** Bloco de código de pré-validação do modelo. O bloco recebe como parametro o objeto de Model e deve retornar um valor lógico. Quando houver uma tentativa de atualização de valor de qualquer Submodelo o bloco de código será invocado. Caso o retorno seja verdadeiro, a alteração será permitida, se retornar falso não será possível concluir a alteração e um erro será atribuido ao model.

Exemplo:

```
bPre := {} | ValidPre(oModel)
oModel := MPFormModel():New('SA1MODEL', , bPre).
```

```
Static Function ValidPre(oModel)
Local nOperation := oModel:GetOperation()
Local IRet := .T.
```

```
If nOperation == MODEL_OPERATION_UPDATE
  If Empty( oModel:GetValue( 'SA1MASTER', 'A1_CGC' ) )
    Help(,, 'HELP',, 'Informe o CNPJ', 1, 0)
    IRet := .F.
  EndIf
EndIf
Return IRet
```

- **bPos:** A validação pós-validação do modelo, equilava ao "TUDOOK". O bloco recebe como parametro o objeto de Model e deve retornar um valor lógico. O bloco será invocado antes da persistência dos dados para validar o model. Caso o retorno seja verdadeiro e não haja nenhum submodelo invalido, será feita a gravação dos dados. Se retornar falso não será possível realizar a gravação e um erro será atribuido ao model, sendo necessário indicar a natureza do erro através da função Help.

Exemplo:

```
bPosModel: { | oModel | fPosModelo(oModel) }
oModel := MPFormModel():New('SA1MODEL', ,bPosModel)
```

```

Static Function fPosModelo ( oModel )
Local nOperation := oModel:GetOperation()
Local lRet := .T.

If nOperation == MODEL_OPERATION_UPDATE
  If Empty( oModel:GetValue( 'SB1MASTER', 'B1_COD' ) )
    Help(,, 'HELP', 'Informe o código do Produto', 1, 0)
    lRet := .F.
  Endif
Endif

Return lRet

```

8.6.1. Gravação manual de dados (FWFormCommit)

A gravação dos dados do modelo de dados (Model) (persistência) é realizada pelo MVC onde são gravados todos os dados das entidades do model. Porém, pode haver a necessidade de se efetuar gravações em outras entidades que não participam do modelo. Por exemplo, quando incluímos um Pedido de Vendas é preciso atualizar o valor de pedidos em aberto do Cadastro de Clientes. O cabeçalho e itens do pedido fazem parte do modelo e serão gravados, o cadastro de Cliente não faz parte, mas precisa ser atualizado também.

Para este tipo de situação é possível intervir no momento da gravação dos dados. Para isso definimos um bloco de código no 4º. parâmetro da classe de construção do modelo de dados (Model) MPFormModel.

- MPFORMMODEL():New(<cID>, <bPre>, <bPost>, <bCommit>, <bCancel>)

O bloco de código recebe como parâmetro um objeto que é o modelo e que pode ser passado à função que fará a gravação. Diferentemente dos blocos de código definidos no modelo de dados (Model) para validação que complementam a validações feitas pelo MVC, o bloco de código para gravação substitui a gravação dos dados. Então ao ser definido um bloco de código para gravação, passa ser responsabilidade da função criada, a gravação de todos os dados inclusive os dados do modelo de dados em uso.

Para facilitar o desenvolvimento foi criada a função FWFormCommit que fará a gravação dos dados do objeto de modelo de dados (Model) informado.

Exemplo:

```

Static Function COMP012GRV ( oModel )
Local lGravo

lGravo := FWFormCommit( oModel )

If lGravo
  // Efetuar a gravação de outros dados em entidade que
  // não são do model
Endif

Return(lGravo)

```

Não devem ser feitas atribuições de dados no modelo (Model) dentro da função de gravação. Conceitualmente ao se iniciar a gravação, o modelo de dados (Model) já passou por toda a validação, ao tentar atribuir um valor, esse valor pode não satisfazer a validação do campo tornando o modelo de dados (Model) invalidado novamente e o que ocorrerá é a gravação de dados inconsistentes.

8.6.2. Cancelamento da gravação de dados

Essa validação realiza os tratamentos necessários ao cancelamento dos formulários de edição. O bloco recebe como parâmetro o objeto do Model.

Quando esse bloco é passado o tratamento de numeração automática não é mais realizado, a menos que o bloco chame a função FWFormCancel.

- MPFORMMODEL():New(<cID>, <bPre>, <bPost>, <bCommit>, <bCancel>)

Exemplo:

```
bCancel:= {|oModel| fCancelar(oModel)}
oModel := MPFormModel():New("MODELO",,,,{ | oModel | fCancelar (oModel)})

Static Function fCancelar(oModel)
Local IRet := .T.
If oModel:GetOperation() == MODEL_OPERATION_INSERT
    RollBackSX()
Endif
Return(IRet)
```

8.6.3. Validação da Ativação do Modelo de Dados

É a validação realizada no momento da ativação do modelo, permitindo ou não a sua ativação.

- **SetVldActivate:** O bloco de código recebe como parâmetro um objeto que é o do modelo correspondente, porém, o modelo ainda não tem os dados carregados, pois a carga dos dados é feita após a sua ativação.

A função chamada pelo bloco de código deve retornar um valor lógico, onde se .T. (verdadeiro) a ativação é realizada e .F. (falso) não é realizada.

Exemplo:

```
oModel:SetVldActivate( { |oModel| fValActicveModel( oModel ) } )

Static Function fValActicveModel( oModel )
Local IRet := .T.

If oModel:GetOperation() == MODEL_OPERATION_UPDATE .AND. oModel:GetOperation() == MODEL_OPERATION_DELETE

    If SZ1->Z1_NOME <> cUserName
        MsgBox("Usuario não possui permissão para visualizar esse pedido")
    Endif
Endif

Return IRet
```

8.6.4. Método de desativação do Modelo de Dados

Após a desativação do Model é permito executar função, será chamado logo após o DeActivate do model. Esse bloco recebe como parametro o proprio model.

- oModel:SetDeActivate ()

Exemplo:

```
oModel:SetDeActivate ({|oModel | MsgInfo("SetDeActivate","Model") })
```

8.6.5. Tratamentos de estrutura de dados(FWFormStruct)

Esta função fornece o objeto com as estruturas de metadado do dicionário de dados, utilizadas pelas classes Model e View.

Sintaxe	FWFormStruct(<nTipo>, <cAlias>)
nTipo	Tipo da construção da estrutura: 1 para Modelo de dados (Model) e 2 para interface (View);
cAlias	Alias da tabela no metadado;
bSX3	Bloco de Código de avaliacao do dicionário de dados. O bloco recebe como parametro o ID do campo e deve retornar um lógico. Se retornar verdadeiro o campo é incluido na estrutura.

Ao criarmos uma estrutura baseada no metadados (dicionários), utilizando a função FWFormStruct, ela leva em consideração todos os campos da entidade, respeitando nível, módulo, uso, etc. Se quisermos selecionar quais os campos do metadados (dicionários) que farão parte da estrutura, devemos utilizar o 3º parâmetro da FWFormStruct, que é um bloco de código que será executada para cada campo que a função trouxer do metadados (dicionários) e que recebe como parâmetro o nome do campo. O bloco de código deve retornar um valor lógico, onde se, .T. (verdadeiro) o campo fará parte da estrutura, se, .F. (falso) não fará.

Exemplo:

```
Local oStruZA0 := FWFormStruct( 2, 'ZA0', { |cCampo| COMP12STRU(cCampo) } )
```

```
Static Function COMP12STRU( cCampo )
```

```
Local IRet := .T.
```

```
If cCampo == 'ZA0_QTDMUS'  
  IRet := .F.  
EndIf  
Return IRet
```

Esse tratamento pode ser feito tanto para as estruturas que serão usadas no modelo de dados (Model) quanto na interface (View). Mas tome o seguinte cuidado: Se for removido da estrutura da interface (View) um campo obrigatório, mesmo ele não sendo exibido para o usuário, o modelo de dados (Model) fará a sua validação dizendo que um campo obrigatório não foi preenchido. Ao remover o campo do Model não executa o inicializador padrão

RemoveField ()camada (Model) e (View)

Sintaxe	ObjetoStruct: RemoveField('X3_CAMPO')
X3_CAMPO	Nome do campo que deseja remover da estrutura

Exemplo:

```
Local oStruSA1 := FWFormStruct(1, 'SA1')
oStruSA1: RemoveField('A1_DDI')
```

Esse tratamento pode ser feito tanto para as estruturas que serão usadas no modelo de dados (Model) quanto na interface (View).

Mas tome o seguinte cuidado: Se for removido da estrutura da interface (View) um campo obrigatório, mesmo ele não sendo exibido para o usuário, o modelo de dados (Model) fará a sua validação dizendo que um campo obrigatório não foi preenchido.

SetProperty() camada (Model)

Propriedade para campos da estrutura do modelo de dados na camada (Model). Ao criarmos uma estrutura baseada no metadados (dicionários), utilizando a função FWFormStruct, são respeitadas as propriedades que o campo tem como validação, inicializador padrão, modo de edição e etc.

Para os define funcionar é necessário utilizar o INCLUDE 'FWMVCDEF.CH'

ID	Tipo	Descrição
MODEL_FIELD_TITULO	C	Titulo
MODEL_FIELD_TOOLTIP	C	Descrição completa do campo
MODEL_FIELD_IDFIELD	C	Nome (ID)
MODEL_FIELD_TIPO	C	Tipo
MODEL_FIELD_TAMANHO	N	Tamanho
MODEL_FIELD_DECIMAL	N	Decimais
MODEL_FIELD_VALID	B	Validação
MODEL_FIELD_WHEN	B	Modo de edição
MODEL_FIELD_VALUES	A	Lista de valores permitido do campo (combo)

ID	Tipo	Descrição
MODEL_FIELD_OBRIGAT	L	Indica se o campo tem preenchimento obrigatório
MODEL_FIELD_INIT	B	Inicializador padrão
MODEL_FIELD_KEY	L	Indica se o campo é chave
MODEL_FIELD_NOUPD	L	Indica se o campo pode receber valor em uma operação de update
MODEL_FIELD_VIRTUAL	L	Indica se o campo é virtual

Se houver a necessidade de mudar alguma propriedade do campo na estrutura, usaremos o método SetProperty para isso.

Exemplo:

```
oStruSA1:SetProperty('A1_EMAIL', MODEL_FIELD_WHEN, 'INCLUI' )
```

Onde o 1º parâmetro é o nome do campo que se deseja mudar ou atribuir a propriedade o 2º é a propriedade que está sendo tratada e o 3º é o valor para a propriedade. No exemplo o campo A1_EMAIL só poderá ser editado na operação de inclusão.

SetProperty() camada (View)

As propriedades para os campos da estrutura do modelo de dados na camada (View). Ao criarmos uma estrutura baseada no metadados (dicionários), utilizando a função FWFormStruct, são respeitadas as propriedades que o campo tem como Mascara, Ordem de campo, Consulta Padrão (F3) e etc.

Para os define funcionar é necessário utilizar o INCLUDE 'FWMVCDEF.CH'

ID	Tipo	Descrição
MVC_VIEW_IDFIELD	C	Nome do Campo
MVC_VIEW_ORDEM	C	Ordem
MVC_VIEW_TITULO	C	Titulo do campo
MVC_VIEW_DESCR	C	Descrição do campo
MVC_VIEW_HELP	A	Array com Help
MVC_VIEW_PICT	C	Picture
MVC_VIEW_PVAR	B	Bloco de Picture Var
MVC_VIEW_LOOKUP	C	Consulta F3
MVC_VIEW_CANCHANGE	L	Indica se o campo é editável
MVC_VIEW_FOLDER_NUMBER	C	Pasta do campo
MVC_VIEW_GROUP_NUMBER	C	Agrupamento do campo
MVC_VIEW_COMBOBOX	A	Lista de valores permitido do campo (Combo)
MVC_VIEW_MAXTAMCMB	N	Tamanho Máximo da maior opção do combo
MVC_VIEW_INIBROW	C	Inicializador de Browse
MVC_VIEW_VIRTUAL	L	Indica se o campo é virtual
MVC_VIEW_PICTVAR	C	Picture Variável

Exemplo:

```
oStruSA1:SetProperty('A1_EMAIL', MVC_VIEW_PICTVAR, '@!' )
```

Onde o 1º parâmetro é o nome do campo que se deseja mudar ou atribuir a propriedade o 2º é a propriedade que está sendo tratada e o 3º é o valor para a propriedade. No exemplo o campo A1_EMAIL foi alterada a máscara.

AddField() camada (Model)

Se quisermos criar um campo em uma estrutura já existente, utilizamos o método Addfield. Há diferenças na sequência de parâmetros deste método para adicionar campos para a estrutura do modelo de dados (Model) ou da interface (View). Sua sintaxe para o modelo de dados (Model) é:

```
AddField( cTitulo, cTooltip, cldField, cTipo, nTamanho, nDecimal, bValid, bWhen, aValues, lObrigat, blnt, lKey, lNoUpd, lVirtual, cValid)
```

- **cTitulo:** título do campo;
- **cTooltip:** tooltip do campo;
- **cldField:** ID do Field;
- **cTipo:** tipo do campo;
- **nTamanho:** tamanho do campo;
- **nDecimal:** decimal do campo;
- **bValid:** code-block de validação do campo;
- **bWhen:** code-block de validação do modo de edição do campo;
- **aValues:** lista de valores permitido do campo;
- **lObrigat:** indica se o campo tem preenchimento obrigatório;
- **blnt:** code-block de inicialização do campo;
- **lKey:** indica se trata de um campo chave;
- **lNoUpd:** indica se o campo não pode receber valor em uma operação de update;
- **lVirtual:** indica se o campo é virtual;

Exemplo:

```
oStruSA1:AddField( ; // Ord. Tipo Desc.
    AllTrim( 'Exemplo 1' ) , ; // [01] C Titulo do campo
    AllTrim( 'Campo Exemplo 1' ) , ; // [02] C ToolTip do campo
    'A1_XEXEM1' , ; // [03] C identificador (ID) do Field
    'C' , ; // [04] C Tipo do campo
    1 , ; // [05] N Tamanho do campo
    0 , ; // [06] N Decimal do campo
    FwBuildFeature( STRUCT_FEATURE_VALID,"Pertence('12')") , ; // [07] B Code-block de validação do campo
    NIL , ; // [08] B Code-block de validação When
    {'1=Sim','2=Não'} , ; // [09] A Lista de valores permitido do campo
    NIL , ; // [10] L Indica se o campo tem preenchimento obrigatório
    FwBuildFeature( STRUCT_FEATURE_INIPAD, "2" ) , ; // [11] B Code-block de inicializacao do campo
    NIL , ; // [12] L Indica se trata de um campo chave
    NIL , ; // [13] L Indica se o campo pode receber valor ADvPI utilizando o MVC
    .T. ) // [14] L Indica se o campo é virtualber valor em uma operação de update.
```

AddField() camada (View)

e quisermos criar um campo em uma estrutura já existente, utilizamos o método Addfield. Há diferenças na sequência de parâmetros deste método para adicionar campos para a estrutura do modelo de dados (Model) ou da interface (View). Sua sintaxe para o modelo de dados (View) é:

```
AddField( cldField, cOrdem, cTitulo, cDescric, aHelp, cType, cPicture, bPictVar, cLookUp, lCanChange, cFolder, cGroup, aComboValues, nMaxLenCombo, clnIBrow, lVirtual, cPictVar, lInsertLine )
```

- **cIdField:** Nome do Campo
- **cOrdem:** Ordem
- **cTitulo:** Título do campo
- **cDescriç:** Descrição completa do campo
- **aHelp:** Array com help
- **cType:** Tipo do campo
- **cPicture:** Picture
- **bPictVar:** Bloco de PictureVar
- **cLookUp:** Consulta F3
- **lCanChange:** Indica se o campo é editável
- **cFolder:** Pasta do campo
- **cGroup:** Agrupamento do campo
- **aComboValues:** Lista de valores permitido do campo (combo)
- **nMaxLenCombo:** Tamanho máximo da maior opção do combo
- **cIniBrow:** Inicializador de Browse
- **lVirtual:** Indica se o campo é virtual
- **cPictVar:** Picture Variável

Exemplo:

```
oStruSA1:AddField( ; // Ord. Tipo Desc.  
'1A_XEXEM1' , ; // [01] C Nome do Campo  
'50' , ; // [02] C Ordem  
AllTrim( 'Exemplo 1' ) , ; // [03] C Título do campo  
AllTrim( 'Campo Exemplo 1' ) , ; // [04] C Descrição do campo  
{ 'Exemplo de Campo de Manual 1' } , ; // [05] A Array com Help  
'C' , ; // [06] C Tipo do campo  
'@!' , ; // [07] C Picture  
NIL , ; // [08] B Bloco de Picture Var  
" , ; // [09] C Consulta F3  
.T. , ; // [10] L Indica se o campo é evitável  
NIL , ; // [11] C Pasta do campo  
NIL , ; // [12] C Agrupamento do campo  
{'1=Sim','2=Não'} , ; // [13] A Lista de valores permitido do campo (Combo)  
NIL , ; // [14] N Tamanho Máximo da maior opção do combo  
NIL , ; // [15] C Inicializador de Browse  
.T. , ; // [16] L Indica se o campo é virtual  
NIL ) // [17] C Picture Variável
```

SetNoFolder() camada (View)

Se quisermos retirar as pastas que estão configuradas em uma estrutura, por exemplo, pelo uso da função FWFormStruct, usamos o método **SetNoFolder**.

Exemplo:

```
oStruSA1:SetNoFolder()
```

SetNoGroups() camada (View)

Se quisermos retirar as pastas que estão configuradas em uma estrutura, por exemplo, pelo uso da função FWFormStruct, usamos o método **SetNoGroups**.

Exemplo:

```
oStruSA1:SetNoGroups()
```

8.6.5.1 Criação manual de gatilho

Com a nova estrutura MVC é possível cirar novas gatilhos sem interagir com o dicionário de dados utilizando a o método AddTrigger.

Esse estrutura faz parte do Model

```
AddTrigger( cldField, cTargetIdField, bPre, bSetValue )
```

- **cldField**: nome (ID) do campo de origem;
- **cTargetIdField**: nome (ID) do campo de destino;
- **bPre**: bloco de código de validação da execução do gatilho;
- **bSetValue**: bloco de código de execução do gatilho.

Os blocos de código deste método pendem uma construção específica. Ao se atribuir ou manipular essas propriedades devem ser informadas no padrão que o MVC espera.

Exemplo:

```
oStr2:AddTrigger( 'Z1_VALOR', 'Z1_TOTAL',{|| .T. }, {||oModelo| fSoma(oModelo) } )
```

```
Static Function fSoma(oModel)
Local xRet := 0
Local nQtd := oModel:GetValue("Z1_QTD" )
Local nValor := oModel:GetValue("Z1_VALOR" )

xRet := Round(nQtd*nValor,2)

Return(xRet)
```

Para facilitar a construção do gatilho foi criada a função FwStruTrigger, ela retorna um array com 4 elementos já formatados para uso no AddTrigger.

FwStruTrigger: (cDom, cCDom, cRegra, !Seek, cAlias, nOrdem, cChave, cCondic)

- **cDom**: campo domínio;
- **cCDom**: campo de contradomínio;
- **cRegra**: regra de preenchimento;
- **!Seek**: se posicionará ou não antes da execução do gatilhos;
- **cAlias**: alias da tabela a ser posicionada;
- **nOrdem**: ordem da tabela a ser posicionada;
- **cChave**: chave de busca da tabela a ser posicionada;

- **cCondic:** condição para execução do gatilho.

Exemplo:

```
aAux := FwStruTrigger();
  'Z1_VALOR' ;;
  'Z1_TOTAL' ;;
  Round('Z1_VALOR**Z1_QTD',2))

oStr2:AddTrigger(aAux[1], aAux[2], aaAux[3], aAux[4])
```

8.6.6. Código para a estrutura (FWBuildFeature)

Algumas propriedades dos campos da estrutura pedem uma construção específica de bloco de código. Ao se atribuir ou manipular essas propriedades elas devem ser informadas no padrão que o MVC espera. Ao se tratar essa propriedade para o uso em aplicações deve-se usar a função FWBuildFeature para construí-la.

Exemplo:

```
oStruct := FWFormStruct(1, "SA1")
cValid := "ExistCpo('SA1')"
bValid := FWBuildFeature( STRUCT_FEATURE_VALID, cValid )
oStruct:SetProperty('A1_COD',MODEL_FIELD_VALID,bValid)
```

As propriedades que precisam ser tratadas com esta função são:

- **STRUCT_FEATURE_VALID:** para a validação
- **STRUCT_FEATURE_WHEN:** para o modo de edição
- **STRUCT_FEATURE_INIPAD:** para o inicializador padrão
- **STRUCT_FEATURE_PICTVAR:** configuração da máscara

Utilize sempre esta função FWBuildFeature para a construção das propriedades do contrário poderão ocorrer erros na aplicação, tal como a não atualização das variáveis de memória para os componentes de formulário.

8.7. Manipulação do componente AddFields

Um submodelo do tipo Field permite manipular somente um registro por vez. Ele tem um relacionamento do tipo 1xN ou 1x1 com outros SubModelos ou então não tem nenhum relacionamento.

MPFORMMODEL():AddFields(< cId >, < cOwner >, < oModelStruct >, < bPre >, < bPos >, < bLoad >) -> NIL

- **bPre:** Bloco de Código de pré-validação do submodelo. Esse bloco é invocado quando há uma tentativa de atribuição de valores. O bloco recebe por parâmetro o objeto do FormField(FWFormFieldsModel), a identificação da ação e a identificação do campo que está sofrendo a atribuição. As identificações que podem ser passadas são as seguintes:
CANSETVALUE: valida se o submodelo pode ou não receber atribuição de valor.
SETVALUE: valida se o campo do submodelo pode receber aquele valor. Nesse caso, o bloco recebe um quarto parâmetro que contém o valor que está sendo atribuído ao campo.

Para todos os casos, o bloco deve retornar um valor lógico, indicando se a ação pode ou não ser executada.

Se o retorno for falso, um erro será atribuído ao Model, sendo necessário indicar a natureza do erro através da função Help.

Exemplo:

```
bPre:={|oFields,cAcao,cCampo,xVALOR|bPreSB1(oFields,cAcao,cCampo,xVALOR)}
oModel:addFields('SB1FIELD',,oStr1, bPre)
```

```
Static Function bPreSB1(oFields,cAcao,cCampo,xValor)
Local lRet := .T.
```

```
If cAcao == "SETVALUE" .AND. cCampo == "B1_DESC"
    If xVALOR == oFIELDS:GetValue("B1_DESC")
        Help(,, "HELP", "Não pode manter o mesmo valor", 1, 0 )
        lRet := .F.
    Endif
Endif
```

```
Return lRet
```

- **bPos:** Bloco de Código de pós-validação do submodelo. É equivalente ao "TUDOOK". O bloco de código recebe como parâmetro o objeto de model do FormField(FWFormFieldsModel) e deve retornar um valor lógico. Este bloco é invocado antes da persistência (gravação) dos dados, validando o submodelo. Se o retorno for verdadeiro, a gravação será realizada se os demais submodelos também estiverem válidos, do contrário um erro será atribuído ao Model, sendo necessário indicar a natureza do erro através da função Help.

Exemplo:

```
bPos:={|oFields|bPosSB1(oFields)}
oModel:addFields('SB1FIELD',,oStr1, ,bPos)
```

```
Static Function bPosSB1 (oFields)
Local lRet := .T.
```

```
If oModel:GetValue("B1_MSBLQL") == "1"
    Help(,, "HELP", "Produto não pode ser bloqueado", 1, 0 )
lRet := .F.
Endif
```

```
Return lRet
```

- **bLoad:** Bloco de carga dos dados do submodelo. Este bloco será invocado durante a execução do método activate desta classe. O bloco recebe por parâmetro o objeto de model do FormField(FWFormFieldsModel) e um valor lógico indicando se é uma operação de cópia. Espera-se como retorno um array com os dados que serão carregados no objeto.,

O array deve conter todos os campos do formulario respeitando a mesma ordem dos campos.

Exemplo:

```
bLoad:={|oFields,ICopy|bLoadSB1(oFields, ICopy)}
oModel:addFields('SB1FIELD',,oStr1,,ICopy)

Static Function bLoadSB1 (oFields, ICopy)

Local aLoad := {}

aAdd(aLoad, {xFilial("SB1"), "000001", "Musica 1", "UN", "01"})
aAdd(aLoad, 0) //recno

Return aLoad
```

8.8. Manipulação do componente FormGrid

Um submodelo do tipo Grid permite manipular diversos registros por vez. Ele tem um relacionamento do tipo Nx1 ou NxM com outros Submodelos. Para criar o componente FormGrid é necessário ter o componte AddFields.

MPFORMMODEL():AddGrid(<cld>, <cOwner>, <oModelStruct>, <bLinePre>, <bLinePos>, <bPre>, <bPos>, <bLoad>)

- **cld:** Id do componente, para identificar o componente para manipulação do Grid.
- **cOwner:** Id do componente superior "Pai".
- **oModelStruct:** Retorno da função FWFormStruct estrutura todos os campos que compõem a tabela.
- **bLinePre:** Bloco de Código de pré-validação do submodelo. O bloco é invocado na deleção de linha, no undelete da linha, na inserção de uma linha e nas tentativas de atribuição de valor. Recebe como parametro o objeto de modelo do FormGrid(FWFormGridModel), o número da linha atual e a identificação da ação. A Identificação da ação pode ser um dos seguintes itens: "UNDELETE", "DELETE", "ADDLINE" - nesse caso não será passado nada para o parametro de numero de linha, "SETVALUE" - nesse caso, serão passados mais três parametros. O 4º parametro é o identificador do campo que está sendo atualizado, o 5º parametro é o valor que está sendo atribuido e o 6º parametro é o valor que está atualmente no campo. "CANSETVALUE" : nesse caso será passado mais um parametro. O 4º parametro é o identificador do campo que está tentando ser atualizado. O retorno do bloco deve ser um valor lógico que indique se a linha está valida para continuar com a ação. Se retornar verdadeiro, executa a ação do contrário atribui um erro ao Model.

Exemplo:

```
bLinePre      := {|oModelGrid, nLinha, cAcao, cCampo, xValue, xValueAntigo | fLinePreGrid(oModelGrid,
nLinha, cAcao, cCampo, xValue, xValueAntigo )}

oModel:addGrid('SZ3DETAIL','SZ2MASTER',oStr2, bLinePre)

Static Function fLinePreGrid(oModelGrid, nLinha, cAcao, cCampo, xValue, xValueAntigo)
Local IRet := .T.
If cAcao == "SETVALUE"
    If cCampo $ "Z3_QTD|Z3_VALOR" .AND. xValue <= 0
        Help(,, "HELP", "O campo: " + cCampo + " é obrigatório", 1, 0 )
    IRet := .F.
    Endif
Endif
```

Return IRet

- **bLinePos:** Bloco de código de pós validação da linha do grid, equivale ao "LINHAOK". Recebe como parametro o objeto de modelo do FormGrid(FWFormGridModel) e o número da linha que está sendo validada. O bloco será invocado antes da gravação dos dados e na inclusão de uma linha. Espera-se um retorno lógico do bloco indicando se a linha está ou não valida. Caso o retorno seja falso um erro será atribuido no Model e a gravação não será realizada.

Exemplo:

```
bLinePos := {[oModelGrid,nLinha | fLINHAOKGrid(oModelGrid,nLinha)}
oModel:addGrid('SZ3DETAIL', 'SZ2MASTER', oStr2, , bLinePos)
```

Static Function fLINHAOKGrid (oModelGrid)

```
If ! oModelGrid:IsDeleted()
  If Empty(oModelGrid:GetValue("Z3_CODPRO"))
    Help(,, 'CAMPO OBRIGATORIO', 'Informe o codigo do Produto', 1, 0)
    Return .F.
  Endif
  If oModelGrid:GetValue("Z3_QTD") == 0
    Help(,, 'CAMPO OBRIGATORIO', 'Informar a Quantidade', 1, 0)
    Return .F.
  Endif
  If oModelGrid:GetValue("Z3_VALOR") == 0
    Help(,, 'CAMPO OBRIGATORIO', 'Informar o Valor', 1, 0)
    Return .F.
  Endif
EndIf
```

Return .T.

- **bPre:** Bloco de Código de pré-validação do submodelo. O bloco é invocado na deleção de linha, no undelete da linha, na inserção de uma linha e nas tentativas de atribuição de valor. Recebe como parametro o objeto de modelo do FormGrid(FWFormGridModel), o número da linha atual e a identificação da ação. A Identificação da ação pode ser um dos seguintes itens: "UNDELETE", "DELETE", "ADDLINE" - nesse caso não será passado nada para o parametro de numero de linha, "SETVALUE" - nesse caso, serão passados mais três parametros. O 4º parametro é o identificador do campo que está sendo atualizado, o 5º parametro é o valor que está sendo atribuido e o 6º parametro é o valor que está atualmente no campo. "CANSETVALUE" : nesse caso será passado mais um parametro. O 4º parametro é o identificador do campo que está tentando ser atualizado. O retorno do bloco deve ser um valor lógico que indique se a linha está valida para continuar com a ação. Se retornar verdadeiro, executa a ação do contrário atribui um erro ao Model.

Exemplo:

```
bPre := {[oModelGrid, nLinha, cAcao, cCampo, xValue, xValueAntigo | fGridbPre(oModelGrid, nLinha,
cAcao, cCampo, xValue, xValueAntigo ) }
oModel:addGrid('SZ3DETAIL', 'SZ2MASTER', oStr2, , bPre)
```

```

Static Function fGridbPre(oModelGrid, nLinha, cAcao, cCampo, xValue, xValueAntigo)
Local IRet := .T.
    If cAcao == "SETVALUE"
        If cCampo $ "Z3_QTD|Z3_VALOR" .AND. xValue <= 0
            Help(,, "HELP", "O campo: " + cCampo + " é obrigatório", 1, 0 )
        IRet := .F.
    Endif
Endif
Return IRet

```

- **bPos:** Bloco de Código de pós-validação do submodelo, ele é equivalente ao "TUDOOK". O bloco de código recebe como parametro o objeto do submodelo e deve retornar um valor lógico. Este bloco é invocado antes da persistência(gravação) dos dados, validando o submodelo. Se o retorno for verdadeiro a gravação será realizada se os demais submodelos também estiverem validos, do contrário um erro será atribuido no Model.

Exemplo:

```

bPos := {|oModelGrid | fTudoOkGrid(oModelGrid)}

oModel:addGrid('SZ3DETAIL','SZ2MASTER',oStr2, , , bPos)

Static Function fbTudoOk(oModelGrid)
Local nLinha := 0
For nLinha := 1 To oModelGrid:Length()
    oModelGrid:GoLine( nLinha )
    If ! oModelGrid:IsDeleted()
        If Empty(oModelGrid:GetValue("Z3_CODPRO"))
            Help(,, 'CAMPO OBRIGATORIO', 'Informe o codigo do Produto', 1, 0)
            Return .F.
        Endif
        If oModelGrid:GetValue("Z3_QTD") == 0
            Help(,, 'CAMPO OBRIGATORIO', 'Informar a Quantidade', 1, 0)
            Return .F.
        Endif
        If oModelGrid:GetValue("Z3_VALOR") == 0
            Help(,, 'CAMPO OBRIGATORIO', 'Informar o Valor', 1, 0)
            Return .F.
        Endif
    Endif
    Next
Return .T.

```

- **bLoad:** Bloco de carga dos dados do submodelo. Este bloco será invocado durante a execução do método activate do objeto. O bloco recebe por parametro o objeto de submodelo e um valor lógico indicando se é uma operação de cópia. Espera-se como retorno um array com os dados que serão carregados no submodelo. O array deve conter todos os campos do formulario respeitando a mesma ordem dos campos.

Exemplo:

```

bLoad := {| oModelGrid,ICopy| floadGrid (oModelGrid,ICopy) }
oModel:addGrid('SZ3DETAIL','SZ2MASTER',oStr2, , , bLoad)

```

```

Static Function lLoadGrid(oModelGrid, lCopy)
Local aLoad := {}
aAdd(aLoad,{0,{xFilial("ZA2"), "000001", "01", "000100","AUTOR1","AUTOR"}})
aAdd(aLoad,{0,{xFilial("ZA2"), "000001", "02", "000102","AUTOR2","AUTOR"}})
aAdd(aLoad,{0,{xFilial("ZA2"), "000001", "03", "000104","AUTOR3","AUTOR"}})
aAdd(aLoad,{0,{xFilial("ZA2"), "000001", "04", "000105","AUTOR4","AUTOR"}})
aAdd(aLoad,{0,{xFilial("ZA2"), "000001", "05", "000106","AUTOR5","AUTOR"}})
Return aLoad

```

8.8.1. Criação de relação entre as entidades do modelo (SetRelation)

Dentro do modelo devemos relacionar todas as entidades que participam dele. No nosso exemplo temos que relacionar a entidade Detail com a entidade Master.

Uma regrinha bem simples para entender isso é: Toda entidade do modelo que possui um superior (*owner*) dever ter seu relacionamento para ele definido. Em outras palavras, é preciso dizer quais as chaves de relacionamento do filho para o pai. O método utilizado para esta definição é o SetRelation.

Exemplo:

```
oModel:SetRelation( 'FMRDETAIL',{{ 'C6_FILIAL', 'xFilial( "SC6" )},{ 'C6_NUM', 'C5_NUM' }},SC6->(IndexKey( 1 )))
```

- **FMRDETAIL:** é o identificador (ID) da entidade Detail, o segundo parâmetro é um vetor bidimensional onde são definidos os relacionamentos entre cada campo da entidade filho para a entidade Pai. O terceiro parâmetro é a ordenação destes dados no componente.

O relacionamento sempre é definido do Detail (Filho) para o Master (Pai), tanto no identificador (ID) quanto na ordem do vetor bidimensional.

8.8.2. Quantidade de linhas do componente de grid

Para se obter a quantidade de linhas do *grid* devemos utilizar o método **Length**. As linhas apagadas também são consideradas na contagem.

Exemplo:

```

For nl := 1 To oModel:Length()
  // Segue a funcao ...
Next nl

```

Se for passado um parâmetro no método **Length**, o retorno será apenas a quantidade de linhas não apagadas da *grid*.

Exemplo:

```
nLinhas := oModel:Length(.T.) // Quantidade linhas não apagadas
```

8.8.3. Status da linha de um componente de grid

Quando estamos falando do modelo de dados (Model) temos 3 operações básicas: Inclusão, Alteração e Exclusão. Quando a operação é de inclusão, todos os componentes do modelo de dados (Model) estão incluídos, esse raciocínio também se aplica à exclusão, se esta é a operação, todos os componentes terão seus dados excluídos. Porém, quando falamos da operação de alteração, não é bem assim.

Em um modelo de dados onde existam componentes do grid, na operação de alteração o grid pode ter linhas incluídas, alteradas ou excluídas, ou seja, o modelo de dados (Model) está em alteração mas um grid pode ter tido as 3 operações em suas linhas.

Em MVC é possível saber que operações uma linha sofreu pelos seguintes métodos de status:

- **IsDeleted:** Informa se uma linha foi apagada. Retornando, .T. (verdadeiro) a linha foi apagada.
- **IsUpdated:** Informa se uma linha foi alterada. Retornando, .T. (verdadeiro) a linha foi alterada.
- **IsInserted:** Informa se uma linha foi inserida, ou seja, se é uma linha nova no grid. Retornando, .T. (verdadeiro) a linha foi inserida.

Exemplo:

```
Static Function COMP23ACAO()
Local oModel := FWModelActive()
Local oModelZA2 := oModel:GetModel( 'ZA2DETAIL' )
Local nl := 0
Local nCtInc := 0
Local nCtAlt := 0
Local nCtDel := nCtInc := nCtAlt := 0
Local aSaveLines := FWSaveRows()

For nl := 1 To oModel:Length()
oModel:GoLine( nl )
If oModel:IsDeleted()
nCtDel++
Elseif oModel:IsInserted()
nCtInc++
Elseif oModel:IsUpdated()
nCtAlt++
Endif
Next
FwRestRows(aSaveLines)
```

8.8.4. Adição uma linha a grid

O grid inicia sempre com uma linha. Esse método AddLine deve ser usado para adicionar novas linhas. O método retorna a quantidade total de linhas da grid. Se a grid já possui 2 linhas e tudo correu bem na adição da linha, o AddLine retornará 3, se ocorreu algum problema retornará 2, pois a nova linha não foi inserida. Os motivos para a inserção não ser bem sucedida podem ser algum campo obrigatório não informado, a pós-validação da linha retornou falso ou o grid atingiu a quantidade máxima de linhas, por exemplo.

Se a linha for incluida com sucesso o grid posiciona na mesma.

Exemplo:

```
nLinha++
If oModel:AddLine() == nLinha
// Segue a função
Endif
```

Os motivos para a inserção não ser bem sucedida poderá ser algum campo obrigatório não informado, a pós-validação da linha retornou .F. (falso), atingiu a quantidade máxima de linhas para o *grid*, por exemplo.

8.8.5. Apagando e recuperando uma linha

Para apagarmos uma linha de um componente de *grid* do modelo de dados (*Model*) utilizamos o método

- **DeleteLine:** O método **DeleteLine** retorna .T. (verdadeiro) se a deleção foi bem sucedida. Um motivo para que não seja é a pré-validação da linha retornar .F. (falso).

Exemplo:

```
Local oModel := FWModelActive()
Local oModel := oModel:GetModel( 'ZA2DETAIL' )
Local nl := 0

For nl := 1 To oModel:Length()
  oModel:GoLine( nl )
  If !oModel:isDeleted()
    oModel:DeleteLine()
  Endif
Next
```

- **UnDeleteLine:** O método **UnDeleteLine** retorna .T. (verdadeiro) se a recuperação foi bem sucedida. Um motivo para que não seja é a pré-validação da linha retornar .F. (falso).

Exemplo:

```
Local oModel := oModel:GetModel( 'SA1DETAIL' )
Local nl := 0
For nl := 1 To oModel:Length()

  If oModel:isDeleted()
    oModel:GoLine( nl )
    oModel:UnDeleteLine()
  Endif
Next
```

8.8.6. Guardando e restaurando o posicionamento do grid

Um cuidado que devemos ter quando escrevemos uma função, mesmo que não seja para uso em MVC, é restaurarmos as áreas das tabelas que desposicionamos.

- **FWSaveRows:** Salva as posicoes das FWFormGrids do Model

Exemplo:

```
Local aSaveLines := FWSaveRows()
```

- **FwRestRows:** Restaura as posicoes das FWFormGrids do Model deve ficar antes do Return para restaurar o valor:

Exemplo:

```
FwRestRows( aSaveLines )
```

8.8.7. Validação de linha duplicada (SetUniqueLine)

Em um modelo de dados onde existam componentes de grid podem ser definidos quais os campos que não podem se repetir dentro deste grid. Por exemplo, imaginemos o Pedido de Vendas e não podemos permitir que o código do produto se repita, podemos definir no modelo este comportamento, sem precisar escrever nenhuma função específica para isso.

O método do modelo de dados (Model) que dever ser usado é o SetUniqueLine.

No exemplo o campo **B1_COD** não poderá ter seu conteúdo repetido no *grid*. Também pode ser informado mais de um campo, criando assim um controle com chave composta.

Exemplo:

```
oModel:GetMethod( 'SB1DETAIL' ):SetUniqueLine( { 'B1_COD' } )
```

8.8.8. Campo Incremental (AddIncrementField)

Podemos fazer com que um campo do modelo de dados (Model) que faça parte de um componente de *grid*, possa ser incrementado unitariamente a cada nova linha inserida.

Por exemplo, imaginemos o Pedido de Vendas, nos itens, o número do item pode ser um campo incremental.

Para isso utilizamos o método AddIncrementField.

Exemplo:

```
oView:AddIncrementField( 'VIEW_SC6', 'C6_ITEM' )
```

Onde **VIEW_SC6** é o identificador (ID) do componente da interface (View), onde se encontra o campo e **C6_ITEM** o nome do campo que será incrementado.

8.9. Ações com a interface View

Existe no MVC a possibilidade de se executar uma função em algumas ações da interface (View). Esse recurso pode ser usado quando queremos executar algo na interface e que não tem reflexo no modelo de dados (Model) como um Refresh de tela por exemplo.

Isso é possível nas seguintes ações:

- Refresh da interface;
- Acionamento do botão confirmar da interface;
- Acionamento do botão cancelar da interface;

8.9.1. Método na SetViewAction

A sua sintaxe é:

- `oView:SetViewAction(<cActionID>, <bAction>)`

Onde **cActionID**:

- **cActionID**: do ponto onde a ação será executada que podem ser:
 - **REFRESH**: executa a ação no Refresh da View;
 - **BUTTONOK**: executa a ação no acionamento do botão confirmar da View;
 - **BUTTONCANCEL**: executa a ação no acionamento do botão cancelar da View;
 - **DELETELINE**: executa a ação na deleção da linha da grid;
 - **UNDELETELINE**: executa a ação na restauração da linha da grid;
 - **REFRESH**: recebe como parâmetro o objeto de View;
 - **BUTTONOK**: recebe como parâmetro o objeto de View;
 - **BUTTONCANCEL**: recebe como parâmetro o objeto de View;
 - **DELETELINE**: recebe como parâmetro o objeto de View, identificador (ID) da View e número da linha.
 - **UNDELETELINE**: recebe como parâmetro o objeto de View, identificador (ID) da View e número da linha

bAction: bloco com a ação a ser executada. Recebe como parâmetro:

Exemplo:

```
oView:SetViewAction( 'BUTTONOK' ,{ |oView| SuaFuncao( oView ) } )
oView:SetViewAction( 'BUTTONCANCEL',{ |oView| OutraFuncao( oView ) } )
```

Essas ações são executadas apenas quando existe uma interface (View). O que não ocorre quando temos o instânciamento direto do modelo, rotina automática ou Web Service. Deve-se evitar então colocar nestas funções ações que possam influenciar a regra de negócio, pois na execução da aplicação sem interface essas ações não serão executadas.

8.9.2. Ação de interface do campo (SetFieldAction)

Existe no MVC a possibilidade de se executar uma função após a validação de campo de algum componente do modelo de dados (*Model*). Esse recurso pode ser usado quando queremos executar algo na *interface* e que não tem reflexo no modelo, como um Refresh de tela ou abrir uma tela auxiliar, por exemplo.

- `FWFORMVIEW():SetFieldAction()`

```
oView:SetFieldAction( <cIDField>, <bAction> )
```

cIDField: ID do campo (nome);

bAction: bloco com a ação a ser executada, recebe como parâmetro:

- Objeto De View
- O identificador (ID) Da View
- O identificador (ID) Do Campo
- Conteúdo Do Campo

Exemplo:

```
oView:SetFieldAction('A1_COD',{||oView,cIDView,cField,xValue|SuaFuncao(oView,cIDView,cField,xValue)})
```

8.9.3. Método SetAfterOkButton

Método que seta um bloco de código que será chamado no final da execução do botão OK. O bloco recebe como parâmetro o objeto de View e não precisa retornar nenhum valor.

- FWFORMVIEW():SetAfterOkButton(<bBlock >)

Exemplo:

```
oView:SetAfterOkButton({|| MsgInfo("Após evento de OK") })
```

8.9.4. Método SetViewCanActivate

Método que seta um Code-block para ser avaliado antes de se ativar o View, caso precisamos avaliar, ou questionar o usuário sobre algo. No momento de chamada desse bloco o Model e o View não estão ativos, portanto não é possível recuperar dados. O bloco recebe como parâmetro o objeto oView e deve retornar verdadeiro para permitir a ativação. Se retornar falso a janela será fechada.

- FWFORMVIEW():SetViewCanActivate(<bBlock >)

Exemplo:

```
oView:SetViewCanActivate({|| MsgInfo('Pode Ativar'), .F. })
```

8.9.5. Método SetAfterViewActivate

Seta um bloco de código que será chamado depois do Activate do View. Esse bloco será apenas executado, o retorno dele não será observado. O bloco de código recebe como parâmetro o objeto View.

Exemplo:

```
oView:SetAfterViewActivate({|| MsgInfo('Depois de ativado') })
```

8.9.6. Adicionando botão na tela

Para adicionar botões na barra de ferramentas do formulário necessário utilizar a função:

- FWFORMVIEW():addUserButton(<cTitle>, <cResource>, <bBloco>, [cToolTip], [nShortCut], [aOptions])

Exemplo:

```
oView:AddUserButton('NomeBotão','CLIPS';
{|| fValSto(),'NomeBotao',/*nShortCut*/,
{MODEL_OPERATION_INSERT, MODEL_OPERATION_UPDATE})
```

8.9.7. Criação de pastas (CreateFolder)

Em MVC podemos criar pastas onde serão colocados os componentes da interface (View). Para isso utilizamos o método CreateFolder.

Exemplo:

```
oView>CreateFolder( 'PASTAS' )
```

Devemos dar um identificador (ID) para cada componente da interface (View). PASTAS é o identificador (ID) dado às pastas. Após a criação da pasta principal, precisamos criar as abas desta pasta. Para isso é usado o método AddSheet.

Exemplo:

```
oView>AddSheet( 'PASTAS', 'ABA01', 'Cabeçalho' )
oView>AddSheet( 'PASTAS', 'ABA02', 'Item' )
```

Onde PASTAS é o identificador (ID) da pasta, e ABA01 e ABA02 são os IDs dados a cada aba e Cabeçalho e Item são os títulos de cada aba. Para que possamos colocar um componente em uma aba, precisamos criar um box, um objeto, para receber os elementos da interface (View). A forma para se criar um box em uma aba é:

Exemplo:

```
oView>CreateHorizontalBox( 'SUPERIOR', 100,,, 'PASTAS', 'ABA01' )
oView>CreateHorizontalBox( 'INFERIOR', 100,,, 'PASTAS', 'ABA02' )
```

Devemos dar um identificador (ID) para cada componente da interface (View).

- SUPERIOR e INFERIOR são os IDs dados a cada box.
- 100 indica o percentual que o box ocupará da aba.
- PASTAS é o identificador (ID) da pasta.
- ABA01 e ABA02 os IDs das abas.

Precisamos relacionar o componente da interface (View) com um box para exibição, para isso usamos o método SetOwnerView.

Exemplo:

```
oView:SetOwnerView( 'VIEW_SB1' , 'SUPERIOR' )
oView:SetOwnerView( 'VIEW_SB5' , 'INFERIOR' )
```

8.9.8. Validação das pasta SetVldFolder

Executa ação quando uma aba é selecionada em qualquer folder da View

- oView:SetVldFolder({|cFolderID, nOldSheet, nSelSheet| ValFolder(cFolderID, nOldSheet, nSelSheet)})

Exemplo:

```
Static Function VldFolder(cFolderID, nOldSheet, nSelSheet)
Local IRet := .T.

If nOldSheet == 1 .And. nSelSheet == 2
  Help(,, 'Help',;
  'Não é permitido selecionar a aba 2 se você estiver na aba 1.', 1, 0 )
  IRet := .F.
Endif

Return IRet
```

8.9.9. Outros objetos (AddOtherObjects)

Na construção de algumas aplicações pode ser que tenhamos que adicionar à interface um componente que não faz parte da interface padrão do MVC, como um gráfico, um calendário, etc.

Para isso usaremos o método AddOtherObject

AddOtherObject(<id>, <Code Block a ser executado>)

- **Id:** parâmetro é o identificador (ID)
- **Código de bloco:** para a criação dos outros objetos. O MVC se limita a fazer a chamada da função, a responsabilidade de construção e atualização dos dados cabe ao desenvolvedor em sua função.

Exemplo:

```
AddOtherObject( "OTHER_PANEL", { |oPanel| COMP23BUT( oPanel ) } )
```

Para o botão parecer na camada viewr necessário associa ao box que ira exibir os outros objetos

Exemplo:

```
oView:SetOwnerView("OTHER_PANEL",'EMBAIXODIR')
```

Note que o 2º parâmetro recebe como parâmetro um objeto que é o **container** onde o desenvolvedor deve colocar seus objetos.

Exemplo:

```
Static Function COMP23BUT ( oPanel )
```

```
TButton():New( 020, 020, "TESTE", oPanel, {|| MsgInfo('TESTE') }, 030, 010, , , .F., .T., .F., , .F., , .F. )
```

```
Return( Nil )
```

8.9.10. Carregar a interface de uma aplicação já existente (FWLoadView)

Para criarmos um objeto com o modelo de dados de uma aplicação, utilizamos o função **FWLoadView**.
FWLoadView (<nome do fonte>)

Exemplo:

```
Static Function ViewDef()
// Utilizando uma view que ja existe em outra aplicacao
Return FWLoadView( 'COMP012_MVC' )
```

8.10. Criação de campos de total ou contadores (AddCalc)

Em MVC é possível criar automaticamente um novo componente composto de campos totalizadores ou contadores, um componente de cálculos. Os campos do componente de cálculos são baseados em componentes de grid do modelo. Atualizando o componente de grid automaticamente os campos do componente de cálculos serão atualizados. O Addcalc é o componente de modelo de dados (Model) responsável por isso. Sua sintaxe é

AddCalc: (cld, cOwner, cldForm, cldField, cldCalc, cOperation, bCond, bInitValue, cTitle, bFormula, nTamanho, nDecimal)

- **cld:** Identificador do componente de cálculos;
- **cOwner:** identificador do componente superior (owner). Não necessariamente é o componente de grid de onde virão os dados.
- **cldForm:** código do componente de grid que contém o campo, a que se refere o campo calculado;
- **cldField:** nome do campo do componente de grid a que se refere o campo calculado;
- **cldCalc:** identificador (nome) para o campo calculado;
- **cOperation:** identificador da operação a ser realizada.

As operações podem ser:

- **SUM:** Faz a soma do campo do componente de grid;
- **COUNT:** Faz a contagem do campo do componente de grid;
- **AVG:** Faz a média do campo do componente de grid;
- **FORMULA:** Executa uma fórmula para o campo do componente de grid;
- **bCond:** Condição para avaliação do campo calculado. Recebe como parâmetro o objeto do modelo. Retornando .T. (verdadeiro) Exemplo: {!oModel} teste (oModel);
- **bInitValue:** bloco de código para o valor inicial para o campo calculado. Recebe como parâmetro o objeto do modelo. Exemplo: {!oModel} teste (oModel);
- **cTitle:** título para o campo calculado;
- **bFormula:** fórmula a ser utilizada quando o parâmetro cOperation é do tipo formula.

Recebe como parâmetros: o objeto do modelo, o valor da atual do campo fórmula, o conteúdo do campo do componente de grid, campo lógico indicando se é uma execução de soma (.T. (verdadeiro)) ou subtração (.F. (falso)). O valor retornado será atribuído ao campo calculado

Exemplo:

```
{|oModel,nTotalAtual,xValor,lSomando|Calculo( oModel,nTotalAtual, xValor, lSomando )}
```

- **nTamanho:** tamanho do campo calculado (Se não for informado usa o tamanho padrão).
- **SUM:** será o tamanho do campo do componente de grid + 3;
- **COUNT:** será o tamanho será fixo em 6;
- **AVG:** será o tamanho do campo do componente de grid. Se o campo do componente de grid tiver o tamanho de 9, o campo calculado terá 9;
- **FORMULA:** será o tamanho do campo do componente de grid + 3. Se o campo do componente de grid tiver o tamanho de 9, o campo calculado terá 12;
- **nDecimal:** número de casas decimais do campo calculado;

Para adicionar contador de registro, adicione a seguinte linha no ModelDef:

Exemplo:

```
oModel:AddCalc( 'COUNTTOTAL', 'MODEL_AAA', 'MODEL_BBB', 'AAA_ID', 'Total', 'COUNT' )
```

- **COUNTTOTAL:** é o identificador do componente de cálculos
- **MODEL_AAA:** é o identificador do componente superior (owner)
- **MODEL_BBB:** é o código do componente de grid de onde virão os dados
- **AAA_ID:** é o nome do campo do componente de grid a que se refere o campo calculado
- **Total:** é o identificador (nome) para o campo calculado
- **COUNT:** é o identificador da operação a ser realizada

Na ViewDef também temos que fazer a definição do componente de cálculo. Os dados usados em um componente de cálculo são baseados em um componente de grid, porém, a sua exibição se dá da mesma forma que um componente de formulário, por utilizarmos para o componente de cálculo o **AddField** e para obtermos a estrutura que foi criada na ModelDef usamos **FWCALCStruct**.

Exemplo:

```
Local oCount := FWCALCStruct( oModel:GetModel('COUNTTOTAL') )
oView:AddField( 'VCALC_ID', oCount, 'INFERIOR' )
oView>CreateHorizontalBox( 'INFERIOR', 24 )
oView:SetOwnerView('VCALC_ID','INFERIOR')
```

8.11. FWExecView

Função que apresenta uma janela exibindo o View(FWFormView) de um determinado programa-fonte. Esta função instancia a interface (View) e consequentemente o modelo de dados (Model) com as operações de visualizar, incluir, alterar ou excluir. A intenção dela é fazer similarmente o que faziam as funções AXVISUAL, AXINCLI, AXALTERA e AXDELETA.

FWExecView (cTitulo, cPrograma, nOperation, oDlg, bCloseOnOk, bOk, nPercReducao, aEnableButtons, bCancel)

Nome	Tipo	Descrição
cTitulo	Caracteres	Titulo da janela
cPrograma	Caracteres	Nome do programa-fonte
nOperation	Numérico	Indica o código de operação (Inclusão, alteração ou exclusão)
oDlg	Objeto	Objeto da janela em que o View deve ser colocado, aceita somente Dialog e Window. Se não informado, uma nova janela será criada.
bCloseOnOK	Bloco de Código	Bloco de validação do fechamento da janela. O bloco recebe como parametro o objeto da View e deve retornar um lógico. Se retornar .T. a janela é fechada após clicar no botão OK, do contrário será mantida aberta.
bOk	Bloco de Código	Bloco de validação do botão OK. Se o bloco retornar .T., executa o OK do MVC, do contrário não.
nPercReducao	Numérico	Reduz a janela em nPercReducao/100 do tamanho da janela Owner.
aEnableButtons	Array	Botões que serão habilitados na janela. Verifique as observações para maiores informações.
bCancel	Bloco de Código	Bloco de validação do botão Cancelar. Se o bloco retornar .T., executa o Cancelar do MVC, do contrário não.
cOperatId	Caracteres	Identificador da opção do menu, usado em conjunto com o MenuDef para saber qual a opção escolhida. O parametro é usado quando existe, por exemplo, mais de uma opção 4 no menu.
cToolBar	Caracteres	Indica o relacionamento com os botões da tela (FormBar)
oModelAct	Objeto	Model que será usado pelo View. Quando esse parametro é informado, o View não cria um novo model, ele usa o model que foi enviado. Desconsiderando inclusive a operação informada no 3º parametro, o que vale é o que está no model. Importante, nesse caso o model já deve estar ativo.

O retorno desta função será:

- Se o usuário finalizar a operação com o botão confirmar
- Se o usuário finalizar a operação com o botão cancelar

Exemplo:

```
oModel := FwLoadModel("TST01")// NOME DO PRW
oModel:SetOperation(3)
```

```
oModel:Activate()
```

```
IOk := FWExecView('Exemplo','VIEWDEF.TST01',MODEL_OPERATION_INSERT,,|| .T. )

If IOk
  Help(,, 'Help',, 'Foi confirmada a operação, 1, 0 )
Else
  Help(,, 'Help',, 'Foi cancelada a operação, 1, 0 )
EndIf
```

8.12. Pontos de entrada no MVC

Pontos de entrada são desvios controlados executados no decorrer das aplicações. Ao se escrever uma aplicação utilizando o MVC, automaticamente já estarão disponíveis pontos de entrada pré-definidos.

A ideia de ponto de entrada, para fontes desenvolvidos utilizando-se o conceito de MVC e suas classes, é um pouco diferente das aplicações desenvolvidas de maneira convencional.

Nos fontes convencionais temos um nome para cada ponto de entrada criado, por exemplo, na rotina MATA010 – Cadastro de Produtos, temos os pontos de entrada: MT010BRW, MTA010OK, MT010CAN, etc.

Em MVC criamos um único ponto de entrada e este é chamado em vários momentos dentro da aplicação desenvolvida. Este ponto de entrada único deve ser uma User Function e ter como nome o identificador (ID) do modelo de dados (Model) da fonte.

O ponto de entrada criado recebe via parâmetro (PARAMIXB) um vetor com informações referentes à aplicação. Estes parâmetros variam para cada situação, em comum todos eles têm os 3 primeiros elementos que são listados abaixo, no quadro seguinte existe a relação de parâmetros para cada ID:

Posições do array de parâmetros comuns a todos os IDs:

Pos	Tipo	Descrição
1	O	Objeto do formulário ou do modelo, conforme o caso
2	C	ID do local de execução do ponto de entrada
3	C	ID do formulário

O ponto de entrada é chamado em vários momentos dentro da aplicação, na 2^a posição da estrutura do vetor é passado um identificador (*ID*) que identifica qual é esse momento. Ela pode ter como conteúdo

ID	MOMENTO DE EXECUÇÃO DO PONTO DE ENTRADA
MODELPRE	Antes da alteração de qualquer campo do modelo. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário Retorno: Requer um retorno lógico

MODELPOS Na validação total do modelo. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário Retorno: Requer um retorno lógico
FORMPRE Antes da alteração de qualquer campo do formulário. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário Retorno: Requer um retorno lógico
FORMPOS Na validação total do formulário. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário Retorno: Requer um retorno lógico
FORMLINEPRE Antes da alteração da linha do formulário FWFORMGRID. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário 4 N Número da Linha da FWFORMGRID 5 C Ação da FWFORMGRID 6 C Id do campo Retorno: Requer um retorno lógico
FORMLINEPOS Na validação total da linha do formulário FWFORMGRID. Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário 4 N Número da Linha da FWFORMGRID Retorno: Requer um retorno lógico

MODELCOMMITTS	<p>Após a gravação total do modelo e dentro da transação.</p> <p>Parâmetros Recebidos:</p> <p>1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário</p> <p>Retorno:</p> <p>Não espera retorno</p>
MODELCOMMITNTTS	<p>Após a gravação total do modelo e fora da transação.</p> <p>Parâmetros Recebidos:</p> <p>1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário</p> <p>Retorno:</p> <p>Não espera retorno</p>
FORMCOMMITTSPRE	<p>Antes da gravação da tabela do formulário.</p> <p>Parâmetros Recebidos:</p> <p>1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário 4 L Se .T. indica novo registro (Inclusão) se .F. registro já existente (Alteração / Exclusão)</p> <p>Retorno:</p> <p>Não espera retorno</p>
FORMCOMMITTSPOS	<p>Após a gravação da tabela do formulário.</p> <p>Parâmetros Recebidos:</p> <p>1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário 4 L Se .T. indica novo registro (Inclusão) se .F. registro já existente (Alteração / Exclusão)</p> <p>Retorno:</p> <p>Não espera retorno</p>
FORMCANCEL	<p>No cancelamento do botão.</p> <p>Parâmetros Recebidos:</p> <p>1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário</p> <p>Retorno:</p> <p>Requer um retorno lógico</p>

BUTTONBAR	<p>Para a inclusão de botões na ControlBar. Para criar os botões deve-se retornar um array bi-dimensional com a seguinte estrutura de cada item:</p> <table style="margin-left: 20px; border-collapse: collapse;"> <tr><td style="width: 20px;">1</td><td style="width: 20px;">C</td><td>Titulo para o botão</td></tr> <tr><td>2</td><td>C</td><td>Nome do Bitmap para exibição</td></tr> <tr><td>3</td><td>B</td><td>CodeBlock a ser executado</td></tr> <tr><td>4</td><td>C</td><td>ToolTip (Opcional)</td></tr> </table> <p>Parâmetros Recebidos: 1 O Objeto do formulário ou do modelo, conforme o caso 2 C ID do local de execução do ponto de entrada 3 C ID do formulário</p> <p>Retorno: Requer um array de retorno com estrutura pré definida</p>	1	C	Titulo para o botão	2	C	Nome do Bitmap para exibição	3	B	CodeBlock a ser executado	4	C	ToolTip (Opcional)
1	C	Titulo para o botão											
2	C	Nome do Bitmap para exibição											
3	B	CodeBlock a ser executado											
4	C	ToolTip (Opcional)											

Exemplo:

```

#include 'Protheus.ch'
#include 'FWMVCDEF.CH'

User Function TSTMDO3()// Nome do ID do MODEL
Local lRet := .T.
Local aParam := PARAMIXB
Local oModel := "
Local oStruct := "
Local cldPonto := "
Local cldModel := "
If aParam <> NIL
    oModel := aParam[1] // Retorno do Model
    cldPonto := aParam[2] // Nome do id de execução do ponto de entrada
    cldModel := aParam[3] // Id do Modelo que esta sendo executado
    // 1o Ponto de entrada
If cldPonto == "MODELVLDACTIVE" // Verifica se pode ativação do modelo
    // Podemos ultizar para adicionar alguma manutenção na estrutura dos campos
    // Validação para ativação do Model,
    // Exemplo podemos Removendo o Inicializador padrão
    //oModel:GetStruct() Recuperando a estrutura dos campos
    //oModel:GetStruct(): SetProperty() // Propriedades dos campos
If oModel:GetOperation() == MODEL_OPERATION_INSERT
    oStruct := oModel:GetModel( "SZ2MASTER" ):GetStruct()
    oStruct:SetProperty("Z2_COD",MODEL_FIELD_INIT,;
        FWBuildFeature( STRUCT_FEATURE_INIPAD , NIL ))
Endif
Return .T.
Endif
// 2o Ponto de entrada
If cldPonto == "FORMPRE" .AND. cldModel == "SZ2MASTER" // FormPre do Enchoice
    // PARAMIXB[4] AÇÃO | ISENABLE | SETVALUE |
    // PARAMIXB[5] Campo |

```

```
// PARAMIXB[4] xValor| Conteudo digitado no Enchoice
Return .T.
Endif
// 3o Ponto de entrada
If cldPonto == "FORMPRE" .AND. cldModel == "SZ3DETAIL" // FormPre do Grid
// PARAMIXB[4] Numero da Linha
// PARAMIXB[5] AÇÃO | "ISENABLE" | "CANSETVALUE" | "SETVALUE" | "ADDLINE"
|"DELETE"|"UNDELETE"
// PARAMIXB[6] CAMPO
// M-> Pode trabalhar com Memoria para validar o conteudo qe esta posicionado
Return .T.
Endif
// 4o Ponto de entrada
If cldPonto == "FORMLINEPRE" .AND. cldModel == "SZ3DETAIL"
// FormPre do Grid | FORMLINEPRE
// PARAMIXB[4] Numero da Linha
// PARAMIXB[5] AÇÃO | "CANSETVALUE" | "SETVALUE" | "DELETE" |"UNDELETE"
// PARAMIXB[6] CAMPO
// M-> Pode trabalhar com Memoria para validar o conteudo qe esta posicionado
Return .T.
Endif
// 5o Ponto de entrada
If cldPonto == "BUTTONBAR" // Adiciona botão no enchoiceBar
Return {{'Exemplo', 'CLIPS', { || Msginfo( 'New Botton' ) }, 'Exemplo novo Botão' } }
Endif
// 6o Ponto de entrada
If cldPonto == "FORMPRE" .AND. cldModel == "TSTMD03" // FormPre do Model
Return .T.
Endif
// 7o Ponto de entrada
If cldPonto == "MODELPRE" .AND. cldModel == "TSTMD03" // MODELPRE do Model |
// M-> Pode trabalhar com Memoria para validar o conteudo qe esta posicionado
Return .T.
Endif
// 8o Ponto de entrada
If cldPonto == "MODELCANCEL" // Valida se pode fechar a tela
Return .T.
Endif
// 9o Ponto de entrada
If cldPonto == "FORMPOS" .AND. cldModel == "SZ2MASTER" // FORMPOS do Enchoice |Validação
TudoOK do Enchoice
Return .T.
Endif
// 10o Ponto de entrada
If cldPonto == "FORMPOS" .AND. cldModel == "TSTMD03" // FORMPOS do Modelo | Validação TudoOK
Return .T.
Endif
```

```

// 11o Ponto de entrada
If cldPonto == "FORMLINEPOS" .AND. cldModel == "SZ3DETAIL" // FORMLINEPOS do Gr
// PARAMIXB[4]Numero da Linha
Return .T.
Endif

// 12o Ponto de entrada
// Antes da gravação da tabela do formulário.
If cldPonto == "FORMCOMMITTSPRE"
// PARAMIXB[4]Se .T. indica novo registro (Inclusão) | .F. Alterar ou Excluir
Return .T.
Endif

// 13o Ponto de entrada
// Após a gravação total do modelo e dentro da transação.
If cldPonto == "MODELCOMMITTS"
Return .T.
Endif

// 14o Ponto de entrada
//Após a gravação total do modelo e fora da transação.
If cldPonto == "MODELCOMMITNTTS"
Return .T.
Endif
Return IRet
  
```

8.13. Rotina automática

Quando uma aplicação é desenvolvida utilizando-se o conceito de MVC e suas classes, pode-se fazer uso de seu modelo de dados em outras aplicações, similmente ao que seria uma rotina automática no desenvolvimento tradicional. Não existe mais o uso da função MSExecAuto. A ideia básica é instanciar o modelo de dados (Model) que se deseja, atribuir os valores a ele e fazer a validação. Para melhor entendimento, usaremos de exemplo o fonte abaixo, onde se faz em MVC o que seria uma rotina automática para importação de um cadastro simples.

Exemplo:

```

// Importamos outro registro
Local aCampos := {}
Local cFonte := "TST06"
Local cAlias := "SA1"

aAdd( aCampos, { 'A1_COD' , '000104' })
aAdd( aCampos, { 'A1_LOJA' , '02' })
aAdd( aCampos, { 'A1_NOME' , 'MVC EXEC' })
aAdd( aCampos, { 'A1_NREDUZ' , 'Emilio Rosa' })
aAdd( aCampos, { 'A1_TIPO' , 'F' })
aAdd( aCampos, { 'A1_EST' , 'SP' })
aAdd( aCampos, { 'A1_MUN' , 'SÃO PAULO' })
aAdd( aCampos, { 'A1_END' , 'Av Braz Leme' })

// Carrega modelo
  
```

Formação Programação ADVPL



```
oModel := FWLoadModel( cFonte )
// informa operação Incluir
oModel:SetOperation( MODEL_OPERATION_INSERT )
oModel:Activate()

// ID do componente addfields
oAux := oModel:GetModel( "SA1FIELD" ):GetStruct()
// Removendo todas as validações do modo de edições dos campos

oAux:SetProperty("*",MODEL_FIELD_WHEN, FWBuildFeature( STRUCT_FEATURE_WHEN , NIL ) )

For nl := 1 To Len( aCampos )
If oAux:HasField(aCampos[nl][1])
// É feita a atribuição do dado ao campo do Model
If !( lAux := oModel:SetValue( "SA1FIELD" , aCampos[nl][1], aCampos[nl][2] ) )
// Caso a atribuição não possa ser feita, por algum motivo
// (validação, por exemplo) o //método SetValue retorna .F.
    lRet := .F.
    Exit
Endif
Endif
Next nl

If lRet
// neste momento os dados não são gravados, são somente validados.
If ( lRet := oModel:VldData() )
// Se os dados foram validados faz-se a gravação efetiva dos dados (commit)
    oModel:CommitData()
Endif
Endif

If ! lRet
// Se os dados não foram validados obtemos a descrição do erro para gerar LOG //
ou mensagem de aviso
    aErro := oModel:GetErrorMessage()
    // A estrutura do vetor com erro é:
    // [1] identificador (ID) do formulário de origem
    // [2] identificador (ID) do campo de origem
    // [3] identificador (ID) do formulário de erro
    // [4] identificador (ID) do campo de erro
    // [5] identificador (ID) do erro
    // [6] mensagem do erro
    // [7] mensagem da solução
    // [8] Valor atribuído
    // [9] Valor anterior

    AutoGrLog( "Id do formulário de origem:" + '[' + AllToChar( aErro[1] ) + ']' )
    AutoGrLog( "Id do campo de origem:" + '[' + AllToChar( aErro[2] ) + ']' )
    AutoGrLog( "Id do formulário de erro:" + '[' + AllToChar( aErro[3] ) + ']' )
    AutoGrLog( "Id do campo de erro:" + '[' + AllToChar( aErro[4] ) + ']' )
    AutoGrLog( "Id do erro:" + '[' + AllToChar( aErro[5] ) + ']' )
    AutoGrLog( "Mensagem do erro:" + '[' + AllToChar( aErro[6] ) + ']' )
```

```

AutoGrLog( "Mensagem da solução: " + '[' + AllToChar( aErro[7] ) + ']' )
AutoGrLog( "Valor atribuído: " + '[' + AllToChar( aErro[8] ) + ']' )
AutoGrLog( "Valor anterior: " + '[' + AllToChar( aErro[9] ) + ']' )
MostraErro()
Else
    MsqInfo("Registro gavado com sucesso")
EndIf

// Desativamos o Model
oModel:DeActivate()

```

Uma situação que poderá ser encontrada, nos casos em que se está convertendo uma aplicação já existente para a estrutura de MVC, é o fato da aplicação já estar preparada para trabalhar com rotina automática e consequentemente pode haver várias outras aplicações que já utilizam essa rotina automática.

9. Relatórios Gráficos

O Protheus oferece o recurso personalização para alguns relatórios de cadastros e movimentações do sistema. Ele tem como principais funcionalidades a definição de cores, estilos, tamanho, fontes, quebras, máscara das células para cada seção, criação de fórmulas e funções (Soma, Média, etc.), possibilidade de salvar as configurações por usuário e criação de gráficos.

Com a funcionalidade de Relatórios Personalizáveis, o usuário pode modificar os relatórios padrões, criando seu próprio layout.

Vale lembrar que nem todos os relatórios são personalizáveis. Por exemplo, relatórios que tenham layout pré-definidos por lei e formulários (boletos, notas-fiscais, etc) não poderão ser alterados.

O TReport é uma classe de impressão que substitui as funções SetPrint, SetDefault, RptStatus e Cabec.

A classe TReport permite que o usuário personalize as informações que serão apresentadas no relatório, alterando fonte (tipo, tamanho, etc.), cor, tipo de linhas, cabeçalho, rodapé, etc. Estrutura do componente TReport:

- O relatório (TReport) contém 1 ou mais seções (TRSection);
- Uma seção (TRSection) pode conter 1 ou mais seções;
- A seção (TRSection) contém células pré-definidas e células selecionadas pelo usuário;
- A seção (TRSection) também contém as quebras (TRBreak) para impressão de totalizadores (TRFunction);

Os relatórios em TReport poderão ser impressos em diversos tipos como:

- Arquivo: o relatório será salvo em um arquivo local ou no servidor;
- Spool: onde o relatório será exibido na tela do próprio sistema;
- E-mail: o relatório é enviado por e-mail;
- Planilha: o relatório pode ser aberto em Excel ou no BrOffice;
- HTML: exibido no browser definido como padrão no sistema operacional;
- PDF: é salvo na extensão .PDF, e pode ser salvo no local definido pelo usuário;
- ODF: editor de texto da BrOffice esta disponível no relatório (TReport).

Relatório Simples:

User Function RSimples()

Local oReport := nil

//Chama a função para carregar a Classe tReport

oReport := RptDef()

oReport:PrintDialog()

Return()

Static Function RptDef()

Local oReport := Nil

Local oSection1:= Nil

Local oBreak

Local oFunction

//Sintaxe:

//Classe TReport

//cNome: Nome físico do relatório

//cTitulo: Titulo do Relatorio

//cPergunta: Nome do grupo de pergunta que sera carreto em parâmetros

//bBlocoCodigo: Execura a função que ira alimentar as TRSection

//TReport():New(cNome,cTitulo,cPerguntas,bBlocoCodigo,cDescricao)

oReport:=TReport():New("Exemplo01", "Cadastro Produtos",/* cPergunta *, {oReport| ReportPrint(oReport), "Descrição do meu relatório")

// Relatorio em retrato

oReport:SetPortrait()

// Define se os totalizadores serão impressos em linha ou coluna

oReport:SetTotalInLine(.F.)

//Monstando a primeira seção

oSection1:= TRSection():New(oReport, "Produtos", {"SB1"}, NIL, .F., .T.)

TRCell():New(oSection1, "B1_COD" , "SB1","Produto" , "@!",30)

TRCell():New(oSection1, "B1_DESC" , "SB1","Descrição","@!",100)

TRCell():New(oSection1, "B1_LOCPAD" , "SB1","Arm.Padrao" , "@!",20)

TRCell():New(oSection1, "B1_POSIPI" , "SB1","NCM" , "@!",30)

//O parâmetro que indica qual célula o totalizador se refere ,

//será utilizado para posicionamento de impressão do totalizador quando

//estiver definido que a impressão será por coluna e como conteúdo para a

//função definida caso não seja informada uma fórmula para o totalizador

TRFunction():New(oSection1:Cell("B1_COD"),NIL,"COUNT",,,,F.,T.)

Return(oReport)

Static Function ReportPrint(oReport)

Local oSection1 := oReport:Section(1)

Local cQuery := ""

Local cNcm := ""

```

Local lPrim      := .T.
Local cAlias := GetNextAlias()

cPart := "% AND B1_COD >= '" + MV_PAR01 + "' "
cPart += " AND B1_COD <= '" + MV_PAR02 + "' %"

BeginSql alias cAlias
    SELECT B1_COD,B1_DESC,B1_LOCPAD,B1_POSIPI
    FROM %table:SB1% SB1
    WHERE B1_FILIAL = %xfilial:SB1%
        AND B1_MSBLQL <> '1'
        AND SB1.%notDel%
    %exp:cPart%
    ORDER BY B1_COD

EndSql

dbSelectArea(cAlias)
(cAlias)->(dbGoTop())

oReport:SetMeter((cAlias)->(LastRec()))

While !(cAlias)->( EOF() )
    If oReport:Cancel()
        Exit
    EndIf

    oReport:IncMeter()

    IncProc("Imprimindo " + altrim((cAlias)->B1_DESC))

    //inicializo a primeira seção
    oSection1:Init()

    //imprimo a seção, relacionando os campos da section com os
    //valores da tabela

    oSection1:Cell("B1_COD" ):SetValue((cAlias)->B1_COD )
    oSection1:Cell("B1_DESC" ):SetValue((cAlias)->B1_DESC )
    oSection1:Cell("B1_LOCPAD"):SetValue((cAlias)->B1_LOCPAD )
    oSection1:Cell("B1_POSIPI"):SetValue((cAlias)->B1_POSIPI)
    oSection1:Printline()

    (cAlias)->(dbSkip())

    //imprimo uma linha
    oReport:ThinLine()

```

```

Enddo

//finalizo seção
oSection1:Finish()

Return( NIL )

```

9.1.1. Pergunte()

A função PERGUNTE() inicializa as variáveis de pergunta (mv_par01,...) baseada na pergunta cadastrado no Dicionário de Dados (SX1). Se o parâmetro lAsk não for especificado ou for verdadeiro será exibida a tela para edição da pergunta e se o usuário confirmar as variáveis serão atualizadas e a pergunta no SX1 também será atualizada.

Sintaxe: Pergunte(cPergunta , [lAsk] , [cTitle])

cPergunta	Pergunta cadastrada no dicionário de dados (SX1) a ser utilizada.
Ask	Indica se exibirá a tela para edição.
cTitle	Título do diálogo.

Retorno:

Lógico	Indica se a tela de visualização das perguntas foi confirmada (.T.) ou cancelada (.F.)
--------	--

9.1.2. AjustaSX1()

A função AJUSTASX1() permite a inclusão simultânea de vários itens de perguntas para um grupo de perguntas no SX1 da empresa ativa.

Sintaxe: AJUSTASX1(cPerg, aPergs)

cPerg	Grupo de perguntas do SX1 (X1_GRUPO)
aPergs	Array contendo a estrutura dos campos que serão gravados no SX1.

Estrutura – Item do array aPerg:

Posição	Campo	Tipo	Descrição
1	X1_PERGUNT	Caractere	Descrição da pergunta em português
2	X1_PERSPA	Caractere	Descrição da pergunta em espanhol

Posição	Campo	Tipo	Descrição
3	X1_PERENG	Caractere	Descrição da pergunta em inglês
4	X1_VARIAVL	Caractere	Nome da variável de controle auxiliar (mv_ch)
5	X1_TIPO	Caractere	Tipo do parâmetro
6	X1_TAMANHO	Numérico	Tamanho do conteúdo do parâmetro
7	X1_DECIMAL	Numérico	Número de decimais para conteúdos numéricos
8	X1_PRESEL	Numérico	Define qual opção do combo é a padrão para o parâmetro.
9	X1_GSC	Caractere	Define se a pergunta será do tipo G – Get ou C – Choice (combo)
10	X1_VALID	Caractere	Expressão de validação do parâmetro
11	X1_VAR01	Caractere	Nome da variável MV_PAR+ "Ordem" do parâmetro
12	X1_DEF01	Caractere	Descrição da opção 1 do combo
13	X1_DEFSPA1	Caractere	Descrição da opção 1 do combo em espanhol
14	X1_DEFENG1	Caractere	Descrição da opção 1 do combo em inglês
15	X1_CNT01	Caractere	Conteúdo padrão ou último conteúdo definido como respostas para a pergunta.
16	X1_VAR02	Caractere	Não é informado
17	X1_DEF02	Caractere	Descrição da opção X do combo em português
18	X1_DEFSPA2	Caractere	Descrição da opção X do combo em espanhol
19	X1_DEFENG2	Caractere	Descrição da opção X do combo em inglês
20	X1_CNT02	Caractere	Não é informado
21	X1_VAR03	Caractere	Não é informado
22	X1_DEF03	Caractere	Descrição da opção X do combo em português
23	X1_DEFSPA3	Caractere	Descrição da opção X do combo em espanhol
24	X1_DEFENG3	Caractere	Descrição da opção X do combo em inglês
25	X1_CNT03	Caractere	Não é informado
26	X1_VAR04	Caractere	Não é informado
27	X1_DEF04	Caractere	Descrição da opção X do combo em português
28	X1_DEFSPA4	Caractere	Descrição da opção X do combo em espanhol
29	X1_DEFENG4	Caractere	Descrição da opção X do combo em inglês
30	X1_CNT04	Caractere	Não é informado
31	X1_VAR05	Caractere	Não é informado

Posição	Campo	Tipo	Descrição
32	X1_DEF05	Caractere	Descrição da opção X do combo em português
33	X1_DEFSPA5	Caractere	Descrição da opção X do combo em espanhol
34	X1_DEFENG5	Caractere	Descrição da opção X do combo em inglês
35	X1_CNT05	Caractere	Não é informado
36	X1_F3	Caractere	Código da consulta F3 vinculada ao parâmetro
37	X1_GRPSXG	Caractere	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
38	X1_PYME	Caractere	Se a pergunta estará disponível no ambiente Pyme
39	X1_HELP	Caractere	Conteúdo do campo X1_HELP
40	X1_PICTURE	Caractere	Picture de formatação do conteúdo do campo.
41	aHelpPor	Array	Vetor simples contendo as linhas de help em português para o parâmetro. Trabalhar com linhas de até 40 caracteres.
42	aHelpEng	Array	Vetor simples contendo as linhas de help em inglês para o parâmetro. Trabalhar com linhas de até 40 caracteres.
43	aHelpSpa	Array	Vetor simples contendo as linhas de help em espanhol para o parâmetro. Trabalhar com linhas de até 40 caracteres.

9.1.3. PutSX1()

A função PUTSX1() permite a inclusão de um único item de pergunta em um grupo de definido no Dicionário de Dados (SX1). Todos os vetores contendo os textos explicativos da pergunta devem conter até 40 caracteres por linha.

Sintaxe: PutSx1(cGrupo, cOrdem, cPergunt, cPerSpa, cPerEng, cVar, cTipo, nTamanho, nDecimal, nPresel, cGSC, cValid, cF3, cGrpSxg ,cPyme, cVar01, cDef01, cDefSpa1 , cDefEng1, cCnt01, cDef02, cDefSpa2, cDefEng2, cDef03, cDefSpa3, cDefEng3, cDef04, cDefSpa4, cDefEng4, cDef05, cDefSpa5, cDefEng5, aHelpPor, aHelpEng, aHelpSpa, cHelp)

cGrupo	Grupo de perguntas do SX1 (X1_GRUPO)
cOrdem	Ordem do parâmetro no grupo (X1_ORDEM)
cPergunt	Descrição da pergunta em português
cPerSpa	Descrição da pergunta em espanhol
cPerEng	Descrição da pergunta em inglês
cVar	Nome da variável de controle auxiliar (X1_VARIAVL)
cTipo	Tipo do parâmetro
nTamanho	Tamanho do conteúdo do parâmetro

nDecimal	Número de decimais para conteúdos numéricos
nPresel	Define qual opção do combo é a padrão para o parâmetro.
cGSC	Define se a pergunta será do tipo G – Get ou C – Choice (combo)
cValid	Expressão de validação do parâmetro
cF3	Código da consulta F3 vinculada ao parâmetro
cGrpSxg	Código do grupo de campos SXG para atualização automática, quando o grupo for alterado.
cPyme	Se a pergunta estará disponível no ambiente Pyme
cVar01	Nome da variável MV_PAR+“Ordem” do parâmetro.
cDef01	Descrição da opção 1 do combo em português
cDefSpa1	Descrição da opção 1 do combo em espanhol
cDefEng1	Descrição da opção 1 do combo em inglês
cCnt01	Conteúdo padrão ou último conteúdo definido como respostas para este item
cDef0x	Descrição da opção X do combo em português
cDefSpax	Descrição da opção X do combo em espanhol
cDefEngx	Descrição da opção X do combo em inglês
aHelpPor	Vetor simples contendo as linhas de help em português para o parâmetro.
aHelpEng	Vetor simples contendo as linhas de help em inglês para o parâmetro.
aHelpSpa	Vetor simples contendo as linhas de help em espanhol para o parâmetro.
cHelp	Conteúdo do campo X1_HELP

9.1.4. ParamBox()

Implementa uma tela de parâmetros, que não necessita da criação de um grupo de perguntas no SX1, e com funcionalidades que a Pergunte() não disponibiliza, tais como CheckBox e RadioButtons.

Cada componente da ParamBox será associado a um parâmetro Private denominado MV_PARxx, de acordo com a ordem do componente na tela. Os parâmetros da ParamBox podem ser utilizados de forma independente em uma rotina específica, ou complementando opções de uma rotina padrão.

Importante

A PARAMBOX define os parâmetros seguindo o princípio das variáveis MV_PARxx. Caso ela seja utilizada em uma rotina em conjunto com parâmetros padrões (SX1 + Pergunte()) é necessário salvar os parâmetros padrões, chamar a Parambox(), salvar o retorno da Parambox() em variáveis Private específicas (MVPARBOXxx) e depois restaurar os parâmetros padrões, conforme o exemplo desta documentação.

Definir variáveis Private próprias para a ParamBox, as quais irão armazenar o conteúdo das MV_PARs que esta retorna.

- **Sintaxe:** ParamBox (aParamBox, cTitulo, aRet, bOk, aButtons, lCentered,; nPosx, nPosy, oMainDlg, cLoad, lCanSave, lUserSave)
- **Retorno:** IOK à indica se a tela de parâmetros foi cancelada ou confirmada

aParametros: Array contendo os parâmetros

[1] Tipo do parâmetro (numérico) primeiro elemento não se altera, é sempre o tipo os demais se alteram conforme os tipos abaixo:

1. MsGet

- [2] : Descrição
- [3] : String contendo o inicializador do campo
- [4] : String contendo a Picture do campo
- [5] : String contendo a validação
- [6] : Consulta F3
- [7] : String contendo a validação When
- [8] : Tamanho do MsGet
- [9] : Flag .T./.F. Parâmetro Obrigatório ?

2. Combo

- [2] : Descrição
- [3] : Numérico contendo a opção inicial do combo
- [4] : Array contendo as opções do Combo
- [5] : Tamanho do Combo
- [6] : Validação
- [7] : Flag .T./.F. Parâmetro Obrigatório ?

3. Radio

- [2] : Descrição
- [3] : Numérico contendo a opção inicial do Radio
- [4] : Array contendo as opções do Radio
- [5] : Tamanho do Radio
- [6] : Validação
- [7] : Flag .T./.F. Parâmetro Obrigatório ?
- [8] : String contendo a validação When

4. CheckBox (Com Say)

- [2] : Descrição
- [3] : Indicador Lógico contendo o inicial do Check

- [4] : Texto do CheckBox
 - [5] : Tamanho do Radio
 - [6] : Validação
 - [7] : Flag .T./.F. Parâmetro Obrigatório ?
5. **CheckBox (linha inteira)**
- [2] : Descrição
 - [3] : Indicador Lógico contendo o inicial do Check
 - [4] : Tamanho do Radio
 - [5] : Validação
 - [6] : Flag .T./.F. Indica se campo é editável ou não
6. **File**
- [2] : Descrição
 - [3] : String contendo o inicializador do campo
 - [4] : String contendo a Picture do campo
 - [5] : String contendo a validação
 - [6] : String contendo a validação When
 - [7] : Tamanho do MsGet
 - [8] : Flag .T./.F. Parâmetro Obrigatório ?
 - [9] : Texto contendo os tipos de arquivo Ex.: "Arquivos .CSV |*.CSV"
 - [10]: Diretório inicial do cGetFile
 - [11]: PARAMETROS do cGETFILE
7. **Montagem de expressão de filtro**
- [2] : Descrição
 - [3] : Alias da tabela
 - [4] : Filtro inicial
 - [5] : Opcional - Cláusula When Botão Editar Filtro
8. **MsGet Password**
- [2] : Descrição
 - [3] : String contendo o inicializador do campo
 - [4] : String contendo a Picture do campo
 - [5] : String contendo a validação
 - [6] : Consulta F3
 - [7] : String contendo a validação When
 - [8] : Tamanho do MsGet
 - [9] : Flag .T./.F. Parâmetro Obrigatório ?
9. **MsGet Say**
- [2] : String Contendo o Texto a ser apresentado
 - [3] : Tamanho da String
 - [4] : Altura da String
 - [5] : Negrito (lógico)
10. **Range**
- [2] : Descrição
 - [3] : Range Inicial
 - [4] : ConsultaF3
 - [5] : Largo em pixels do Get
 - [6] : Tipo

[7] : Tamanho do campo (em chars)
 [8] : String contendo a validação When

11. MultiGet (Memo)

[2] : Descrição
 [3] : Inicializador padrao
 [4] : String contendo a validação
 [5] : String contendo a validação When
 [6] : Flag .T./.F. Parâmetro Obrigatório ?

12. Filtro de usuário por Rotina

[2] : Titulo do filtro
 [3] : Alias da tabela aonde será aplicado o filtro
 [4] : Filtro inicial
 [5] : Validação When

Parametros:

aParamBox	Array de parâmetros de acordo com a regra da ParamBox
cTitulo	Titulo da janela de parâmetros
aRet	Array que será passado por referencia e retomado com o conteúdo de cada parâmetro
bOk	Bloco de código para validação do OK da tela de parâmetros
aButtons	Array contendo a regra para adição de novos botões (além do OK e Cancelar) // AADD(aButtons,{nType,bAction,cTexto})
ICentered	Se a tela será exibida centralizada, quando a mesma não estiver vinculada a outra janela
nPosx	Posição inicial -> linha (Linha final: nPosX+274)
nPosy	Posição inicial -> coluna (Coluna final: nPosY+445)
oMainDlg	Caso o ParamBox deva ser vinculado a uma outra tela
cLoad	Nome do arquivo aonde as respostas do usuário serão salvas / lidas
ICanSave	Se as respostas para as perguntas podem ser salvas
IUserSave	Se o usuário pode salvar sua própria configuração.

User Function ParamBox()

```
Local aRet := {}
Local aParamBox := {}
Local aCombo := { "Janeiro", ;
  "Fevereiro", ;
  "Março", ;
  "Abril", ;
  "Maio", ;
  "Junho", ;
  "Julho", ;
  "Agosto", ;
```

```

"Setembro", ;
"Outubro", ;
"Novembro", ;
"Dezembro"}

Local i           := 0
Private cCadastro := "xParambox"

AADD(aParamBox,{1,"Produto",Space(15),"""",SB1,"",0,.F.})
AADD(aParamBox,{2,"Tipo de cliente",1,aCombo,50,"",.F.})
AADD(aParamBox,{3,"Mostra deletados",;
  IIF( Set(_SET_DELETED),1,2),{"Sim","Não"},50,"",.F.})

AADD(aParamBox,{4,"Marca todos ?",.F.,"Marque todos",50,"",.F.})
AADD(aParamBox,{5,"Marca todos ?",.F.,50,"",.F.})
AADD(aParamBox,{6,"Arquivo?",Space(50),"""","""",50,.F.,"Arquivo .DBF |*.DBF"})
AADD(aParamBox,{7,"Monte o filtro", "SX5","X5_FILIAL==xFilial('SX5')"})
AADD(aParamBox,{8,"Digite a senha",Space(15),"""","""",80,.F.})

If ParamBox(aParamBox,"Teste ParamBox...",@aRet)
  For i := 1 To Len(aRet)
    MsgInfo(aRet[i],"Opção escolhida")
  Next
Endif

Return( NIL )

```

9.2. Rotinas automáticas

9.2.1. MsExecAuto

A função MsExecAuto faz as manutenções automáticas de (inclusão, alteração e exclusão) das rotinas de manipulação de dados do sistema, automatizando o processo de entrada de dados sem a necessidade de desenvolver rotinas específicas

Vantagens:

- **Interface:** Os dados de entrada são enviados a rotina em forma de campos e conteúdos (array) e desta forma não é necessário a apresentação de nenhuma interface ao usuário.
- **Segurança:** A utilização de rotinas automáticas aumenta consideravelmente a segurança do sistema, uma vez que utiliza as validações padrões e diminui os problemas causados por atualização de versão ou inclusão de customizações nas rotinas padrões do sistema.
- **Agilidade no processo:** Aumenta consideravelmente o tempo de desenvolvimento das customizações que necessitam de entrada de dados. Exemplo: Importação de pedido de venda.
- **IMsHelpAuto:** Variável logica deverá ser declarada como private, direciona as mensagens de help para o arquivo de log.

- **IMsErroAuto:** Variável logica deverá ser declarada como private, na execução da função msexecauto retorno o valor.T. nessa variável informando que ocorreu um erro na execução, com a função MostraErro()
- **MostraErro():** Esta função mostra os erros gerados na execução dos processos automáticos.

Sintaxe: MsExecAuto({ |X,Y| ROTINA(X,Y)}, aMatriz, nOPC)

```

Local aRotAuto := {}
Local nOpc := 3 // inclusao
Private IMsHelpAuto := .T.
Private IMsErroAuto := .F.

PREPARE ENVIRONMENT EMPRESA '99' FILIAL '01' MODULO  'FAT'

Begin Transaction
aRotAuto:= {{'B1_COD' , '1010'      ,Nil},;
            {'B1_DESC' , 'Produto teste' ,Nil},;
            {'B1_TIPO' , 'PA'          ,Nil},;
            {'B1_UM'   , 'UN'          ,Nil},;
            {'B1_LOCPAD' , '01'        ,Nil},;
            {'B1_PICM' , 0             ,Nil},;
            {'B1_IPI'  , 0             ,Nil},;
            {'B1_PRV1' , 100           ,Nil},;
            {'B1_LOCALIZ' , 'N' ,Nil},;
            {'B1_CODBAR' , '789888800001' ,Nil}}}
MSExecAuto({|x,y| mata010(x,y)},aProduto,nOpc)

```

```

If IMsErroAuto
  DisarmTransaction()
  MostraErro()
Endif

End Transaction
Return()

```

9.2.2. Prepare Environment

O comando PREPARE ENVIRONMENT tem o objetivo de realizar a inicialização de um determinado ambiente assim que executado. PREPARE ENVIRONMENT permite definir a empresa e filial no qual desejamos fazer a inicialização do ambiente, especificando o usuário e senha para determinar os direitos e ações a serem realizadas. É possível também especificar quais as tabelas que serão abertas para uso no ambiente e qual módulo será utilizado.

Sintaxe:

Prepare Environment - [EMPRESA] [FILIAL] [USER] [PASSWORD] [TABLES] [MODULO]

Nome	Tipo	Descrição
EMPRESA	caracter	Código da empresa.
FILIAL	caracter	Código da filial.
USER	caracter	Nome de usuário.
PASSWORD	caracter	Senha do usuário.
TABLES	caracter	Nomes das tabelas.
MODULO	caracter	Sigla do módulo.

Necessario fechar o ambiente após sua execução com a função:

- RESET ENVIRONMENT

Importante

Para o função PREAPRE ENVIRONMENT funcionar necessário informar o #INCLUDE TBICONN.CH

9.2.3. RpcSetEnv

O comando RpcSetEnv() é utilizado para abertura de ambiente em rotinas automáticas, permitindo definir a empresa e filial no qual desejamos fazer a inicialização do ambiente, especificando o usuário e senha para determinar os direitos e ações a serem realizadas. É possível também especificar quais as tabelas que serão abertas para uso no ambiente e qual módulo será utilizado.

RpcSetEnv([cRpcEmp] [cRpcFil] [cEnvUser] [cEnvPass] [cEnvMod] [cFunName] [aTables] [IShowFinal] [IAbend] [IOpenSX] [IConnect])--> IRet

Retorno:

.T.: Conectou na empresa e filial informada no parâmetro
.F.: Não conseguiu se conectar

Após sua execução necessário fechar o ambiente:

- RpcClearEnv()

9.3. Manipulação de arquivos texto

Arquivos do tipo texto(também conhecidos como padrão TXT) são arquivos com registros de tamanho variável. A indicação do final de cada registro é representada por dois bytes, "0D 0A" em hexadecimal ou "13 10" em decimal ou, ainda, "CR LF" para padrão ASCII.

Apesar do tamanho dos registros ser variável, a maioria dos sistemas gera este tipo de arquivo com registros de tamanho fixo, de acordo com um layout específico que indica quais são os dados gravados. Para ilustrar estes procedimentos, serão gerados arquivos textos, com duas famílias de funções:

- Família: nesta família serão utilizadas as funções: FCreate(), FWrite(), FClose(), FSeek(), FOpen() e FRead().
- Família: nesta família serão utilizadas as funções: FT_FUse(), FT_FGoTop(), FT_FLastRec(), FT_FEof(), FT_FReadLn(), FT_FSkip(), FT_FGoto(), FT_FRecno().

FCREATE()

Função de baixo-nível que permite a manipulação direta dos arquivos textos como binários. Ao ser executada FCREATE() cria um arquivo ou elimina o seu conteúdo, e retorna o handle (manipulador) do arquivo, para ser usado nas demais funções de manutenção de arquivo. Após ser utilizado, o arquivo deve ser fechado através da função FCLOSE().

Na tabela abaixo, estão descritos os atributos para criação do arquivo , definidos no arquivo header fileio.ch

- Atributos definidos no include FileIO.ch

Constante	Valor	Descrição
FC_NORMAL	0	Criação normal do Arquivo (default/padrão).
FC_READONLY	1	Cria o arquivo protegido para gravação.
FC_HIDDEN	2	Cria o arquivo como oculto.
FC_SYSTEM	4	Cria o arquivo como sistema.

FWRITE()

Função que permite a escrita em todo ou em parte do conteúdo do buffer, limitando a quantidade de Bytes através do parâmetro nQtdBytes. A escrita começa a partir da posição corrente do ponteiro de arquivos, e a função FWRITE retornará a quantidade real de bytes escritos. Através das funções FOPEN(), FCREATE(), ou FOPENPORT(), podemos abrir ou criar um arquivo ou abrir uma porta de comunicação, para o qual serão gravados ou enviados os dados do buffer informado. Por tratar-se de uma função de manipulação de conteúdo binário, são suportados na String cBuffer todos os caracteres da tabela ASCII, inclusive caracteres de controle (ASC 0 , ASC 12 , ASC 128 , etc.).

Caso aconteça alguma falha na gravação, a função retornará um número menor que o nQtdBytes. Neste caso, a função FERROR() pode ser utilizada para determinar o erro específico ocorrido. A gravação no arquivo é realizada a partir da posição atual do ponteiro, que pode ser ajustado através das funções FSEEK(), FREAD() ou FREADSTR().

Sintaxe: FWRITE (<nHandle> , <cBuffer> , [nQtdBytes])

Parâmetros:

nHandle	É o manipulador de arquivo retornado pelas funções FOPEN(), FCREATE(), ou FOPENPORT().
cBuffer	<cBuffer> é a cadeia de caracteres a ser escrita no arquivo especificado. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes (caso seja informado o tamanho).

nQtyBytes

<nQtyBytes> indica a quantidade de bytes a serem escritos a partir da posição corrente do ponteiro de arquivos. Caso seja omitido, todo o conteúdo de <cBuffer> é escrito.

Retorno: FWRITE() retorna a quantidade de bytes escritos na forma de um valor numérico inteiro. Caso o valor retornado seja igual a <nQtyBytes>, a operação foi bem sucedida. Caso o valor de retorno seja menor que <nBytes> ou zero, ou o disco está cheio ou ocorreu outro erro. Neste caso , utilize a função FERROR() para obter maiores detalhes da ocorrência.

FCLOSE()

Função de tratamento de arquivos de baixo nível utilizada para fechar arquivos binários e forçar que os respectivos buffers do DOS sejam escritos no disco. Caso a operação falhe, FCLOSE() retorna falso (.F.). FERROR() pode então ser usado para determinar a razão exata da falha. Por exemplo, ao tentar-se usar FCLOSE() com um handle (tratamento dado ao arquivo pelo sistema operacional) inválido retorna falso (.F.) e FERROR() retorna erro 6 do DOS, invalid handle. Consulte FERROR() para obter uma lista completa dos códigos de erro.

Sintaxe: FCLOSE (< nHandle >)

nHandle

Handle do arquivo obtido previamente através de FOPEN() ou FCREATE().

Retorno: Retorna falso (.F.) se ocorre um erro enquanto os buffers estão sendo escritos; do contrário, retorna verdadeiro (.T.).

FSEEK()

Função que posiciona o ponteiro do arquivo para as próximas operações de leitura ou gravação. As movimentações de ponteiros são relativas à nOrigem que pode ter os seguintes valores, definidos em fileio.ch:

Tabela A: Origem a ser considerada para a movimentação do ponteiro de posicionamento do Arquivo

Origem	Constate (fileio.ch)	Operação
0	FS_SET	Ajusta a partir do início do arquivo. (Default)
1	FS_RELATIVE	Ajuste relativo a posição atual do arquivo.
2	FS_END	Ajuste a partir do final do arquivo.

Sintaxe: FSEEK (< nHandle > , [nOffSet] , [nOrigem])

Parâmetros:

nHandle	Manipulador obtido através das funções FCREATE,FOPEN.
nOffSet	nOffSet corresponde ao número de bytes no ponteiro de posicionamento do arquivo a ser movido. Pode ser um numero positivo, zero ou negativo, a ser considerado a partir do parâmetro passado em nOrigem.
nOrigem	Indica a partir de qual posição do arquivo, o nOffset será considerado.

Retorno: Retorna a nova posição do ponteiro de arquivo com relação ao início do arquivo (posição 0) na forma de um valor numérico inteiro. Este valor não leva em conta a posição original do ponteiro de arquivos antes da execução da função FSEEK().

FOPEN()

Função de tratamento de arquivo de baixo nível que abre um arquivo binário existente para que este possa ser lido e escrito, dependendo do argumento <nModo>. Toda vez que houver um erro na abertura do arquivo, FERROR() pode ser usado para retornar o código de erro do Sistema Operacional. Por exemplo, caso o arquivo não exista, FOPEN() retorna -1 e FERROR() retorna 2 para indicar que o arquivo não foi encontrado. Veja FERROR() para uma lista completa dos códigos de erro.

Caso o arquivo especificado seja aberto, o valor retornado é o handle (manipulador) do Sistema Operacional para o arquivo. Este valor é semelhante a um alias no sistema de banco de dados, e ele é exigido para identificar o arquivo aberto para as outras funções de tratamento de arquivo. Portanto, é importante sempre atribuir o valor que foi retornado a uma variável para uso posterior, como mostra o exemplo desta função.

Sintaxe: FOPEN (< cArq > , [nModo])

cArq	Nome do arquivo a ser aberto que inclui o path caso haja um.
nModo	Modo de acesso DOS solicitado que indica como o arquivo aberto deve ser acessado. O acesso é de uma das categorias relacionadas na tabela A e as restrições de compartilhamento relacionada na Tabela B. O modo padrão é zero, somente para leitura, com compartilhamento por Compatibilidade. Ao definirmos o modo de acesso, devemos somar um elemento da Tabela A com um elemento da Tabela B.

Retorno: FOPEN() retorna o handle de arquivo aberto na faixa de zero a 65.535. Caso ocorra um erro, FOPEN() retorna -1.

FREAD()

Função que realiza a leitura dos dados a partir um arquivo aberto, através de FOPEN(), FCREATE() e/ou FOPENPORT(), e armazena os dados lidos por referência no buffer informado. FREAD() lerá até o número de bytes informado em nQtdBytes; caso aconteça algum erro ou o arquivo chegue ao final, FREAD() retornará um número menor que o especificado em nQtdBytes. FREAD() lê normalmente caracteres de controle (ASC 128, ASC 0, etc.) e lê a partir da posição atual do ponteiro atual do arquivo , que pode ser ajustado ou modificado pelas funções FSEEK(), FWRITE() ou FREADSTR().

A variável String a ser utilizada como buffer de leitura deve ser sempre pré-alocado e passado como referência. Caso contrário, os dados não poderão ser retornados.

Sintaxe: FREAD (< nHandle > , < cBuffer > , < nQtdBytes >)

nHandle	É o manipulador (Handle) retornado pelas funções FOPEN(), FCREATE(), FOPENPORT(), que faz referência ao arquivo a ser lido.
cBuffer	É o nome de uma variável do tipo String , a ser utilizada como buffer de leitura , onde os dados lidos deverão ser armazenados. O tamanho desta variável deve ser maior ou igual ao tamanho informado em nQtdBytes. Esta variável deve ser sempre passada por referência. (@ antes do nome da variável), caso contrário os dados lidos não serão retornados.
nQtdBytes	Define a quantidade de Bytes que devem ser lidas do arquivo a partir posicionamento

do ponteiro atual.

Retorno: Quantidades de bytes lidos. Caso a quantidade seja menor que a solicitada, isto indica erro de leitura ou final de arquivo. Verifique a função FERROR() para maiores detalhes.

FT_FUSE()

Função que abre ou fecha um arquivo texto para uso das funções FT_F*. As funções FT_F* são usadas para ler arquivos texto, onde as linhas são delimitadas pela sequência de caracteres CRLF ou LF (*) e o tamanho máximo de cada linha é 1022 bytes. O arquivo é aberto em uma área de trabalho, similar à usada pelas tabelas de dados.

Sintaxe: FT_FUSE ([cTXTFile])

Parâmetros:

cTXTFile	Corresponde ao nome do arquivo TXT a ser aberto. Caso o nome não seja passado, e já exista um arquivo aberto, o mesmo é fechado.
----------	--

Retorno: A função retorna o Handle de controle do arquivo. Em caso de falha de abertura, a função retornará -1

FT_FGOTOP()

A função tem como objetivo mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

Sintaxe: FT_FGOTO (< nPos >)

Parâmetros:

nPos	Indica a posição que será colocado o ponteiro para leitura dos dados no arquivo.
------	--

FT_FLASTREC()

Função que retorna o número total de linhas do arquivo texto aberto pela FT_FUse. As linhas são delimitadas pela sequencia de caracteres CRLF o LF.

Sintaxe: FT_FLASTREC()

Retorno: Retorna a quantidade de linhas existentes no arquivo. Caso o arquivo esteja vazio, ou não exista arquivo aberto, a função retornará 0 (zero).

FT_FEOF()

Função que retorna verdadeiro (.t.) se o arquivo texto aberto pela função FT_FUSE() estiver posicionado no final do arquivo, similar à função EOF() utilizada para arquivos de dados.

Sintaxe: FT_FEOF()

Retorno: Retorna true caso o ponteiro do arquivo tenha chegado ao final, false caso contrário.

FT_FREADLN()

Função que retorna uma linha de texto do arquivo aberto pela FT_FUse. As linhas são delimitadas pela sequência de caracteres CRLF (chr(13) + chr(10)), ou apenas LF (chr(10)), e o tamanho máximo de cada linha é 1022 bytes.

Sintaxe: FT_FREADLN()

Retorno: Retorna a linha inteira na qual está posicionado o ponteiro para leitura de dados.

FT_FSKIP()

Função que move o ponteiro do arquivo texto aberto pela FT_FUSE() para a próxima linha, similar ao DBSKIP() usado para arquivos de dados.

Sintaxe: FT_FSKIP ([nLinhas])

nLinhas

nLinhas corresponde ao número de linhas do arquivo TXT ref. movimentação do ponteiro de leitura do arquivo.

FT_FGOTO()

Função utilizada para mover o ponteiro, que indica a leitura do arquivo texto, para a posição absoluta especificada pelo argumento <nPos>.

Sintaxe: FT_FGOTO (< nPos >)

nPos

Indica a posição que será colocado o ponteiro para leitura dos dados no arquivo.

FT_FRECNO()

A função tem o objetivo de retornar a posição do ponteiro do arquivo texto. A função FT_FRecno retorna a posição corrente do ponteiro do arquivo texto aberto pela FT_FUse.

Sintaxe: FT_FRECNO ()

Retorno: Retorna a posição corrente do ponteiro do arquivo texto.

10. Webservices

10.1. O Que É Um Webservices Wsdl

Web service é uma solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. Os Web services são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato XML.

Para as empresas, os Web services podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de logística. Toda e qualquer comunicação entre sistemas passa a ser dinâmica e principalmente segura, pois não há intervenção humana.

Essencialmente, o Web Service faz com que os recursos da aplicação do software estejam disponíveis sobre a rede de uma forma normalizada. Outras tecnologias fazem a mesma coisa, como por exemplo, os browsers da Internet acedem às páginas Web disponíveis usando por norma as tecnologias da Internet, HTTP e HTML. No entanto, estas tecnologias não são bem sucedidas na comunicação e integração de aplicações. Existe uma grande motivação sobre a tecnologia Web Service pois possibilita que diferentes aplicações comuniquem entre si e utilizem recursos diferentes.

Utilizando a tecnologia Web Service, uma aplicação pode invocar outra para efetuar tarefas simples ou complexas mesmo que as duas aplicações estejam em diferentes sistemas e escritas em linguagens diferentes. Por outras palavras, os Web Services fazem com que os seus recursos estejam disponíveis para que qualquer aplicação cliente possa operar e extrair os recursos fornecidos pelo Web Service.

Os Web Services são identificados por um URI (Uniform Resource Identifier), descritos e definidos usando XML (Extensible Markup Language). Um dos motivos que tornam os Web Services atrativos é o facto deste modelo ser baseado em tecnologias standards, em particular XML e HTTP (Hypertext Transfer Protocol). Os Web Services são utilizados para disponibilizar serviços interativos na Web, podendo ser acessados por outras aplicações usando, por exemplo, o protocolo SOAP (Simple Object Access Protocol).

O objetivo dos Web Services é a comunicação de aplicações através da Internet. Esta comunicação é realizada com intuito de facilitar a EAI (Enterprise Application Integration) que significa a integração das aplicações de uma empresa, ou seja, interoperabilidade entre a informação que circula numa organização nas diferentes aplicações como, por exemplo, o comércio electrónico com os seus clientes e seus fornecedores. Esta interação constitui o sistema de informação de uma empresa. E para além da interoperabilidade entre as aplicações, a EAI permite definir um workflow entre as aplicações e pode constituir uma alternativa aos ERP (Enterprise Resource Planning). Com um workflow é possível otimizar e controlar processos e tarefas de uma determinada organização.

As bases para a construção de um Web service são os padrões XML e SOAP. O transporte dos dados é realizado normalmente via protocolo HTTP ou HTTPS para conexões seguras (o padrão não determina o protocolo de transporte). Os dados são transferidos no formato XML, encapsulados pelo protocolo SOAP.

10.1.1. O Que É Um Xml

XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais. É um dos subtipos da SGML (acrônimo de Standard Generalized Markup Language ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da internet.

Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SDMX, SMIL, MathML (formato para expressões matemáticas), NCL, XBRL, XSIL e SVG (formato gráfico vetorial). A principal característica do XML, de criar uma infraestrutura única para diversas linguagens, é que linguagens desconhecidas e de pouco uso também podem ser definidas sem maior trabalho e sem necessidade de ser submetidas aos comitês de padronização.

Em meados da década de 1990, o World Wide Web Consortium (W3C) começou a trabalhar em uma linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da HTML.

O princípio do projeto era criar uma linguagem que pudesse ser lida por software, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

- Separação do conteúdo da formatação
- Simplicidade e legibilidade, tanto para humanos quanto para computadores
- Possibilidade de criação de tags sem limitação
- Criação de arquivos para validação de estrutura (chamados DTDs)
- Interligação de bancos de dados distintos
- Concentração na estrutura da informação, e não na sua aparência

O XML é um formato para a criação de documentos com dados organizados de forma hierárquica, como se vê, frequentemente, em documentos de texto formatados, imagens vetoriais ou bancos de dados.

Pela sua portabilidade, já que é um formato que não depende das plataformas de hardware ou de software, um banco de dados pode, através de uma aplicação, escrever em um arquivo XML, e um outro banco distinto pode ler então estes mesmos dados.

10.1.2. O Que É Soap

SOAP, originado do acrônimo inglês Simple Object Access Protocol, e em português Protocolo Simples de Acesso a Objetos, é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída.

Ele se baseia na Linguagem de Marcação Extensível (XML) para seu formato de mensagem, e normalmente baseia-se em outros protocolos da Camada de aplicação, mais notavelmente em Chamada de Procedimento Remoto (RPC) e Protocolo de Transferência de Hipertexto (HTTP), para negociação e transmissão de mensagens.

SOAP pode formar a camada base de uma pilha de protocolos de webservices, fornecendo um framework de mensagens básico sob o qual os serviços web podem ser construídos.

10.1.3. O Servidor Protheus como um servidor WEBSERVICES

O servidor Protheus pode ser configurado para trabalhar como um servidor WEBSERVICES.

O Protheus, a partir da versão AP7, possui ferramentas nativas e integradas com a LIB de Infra-Estrutura do ERP, para desenvolvimento de aplicações 'Cliente' e 'Server', utilizando a tecnologia dos Web Services. Para melhor compreensão do assunto, os tópicos relacionados a ambos foram didaticamente separados em Aplicações Server e Aplicações Cliente, respectivamente.

Nos tópicos 'Comandos' e 'Funções', são abordadas respectivamente as diretivas e funções da Lib de Infraestrutura do ERP disponibilizadas para o desenvolvimento de ambas as aplicações, Cliente e Server.

10.1.4. Configurando servidor de WEBSERVICES

Nos serviços HTTP e HTTPS, é possível especificar as configurações padrões deste protocolo e propriedades gerais aplicadas a todos os hosts e URLs de acesso utilizadas pelos projetos WEBSERVICES.

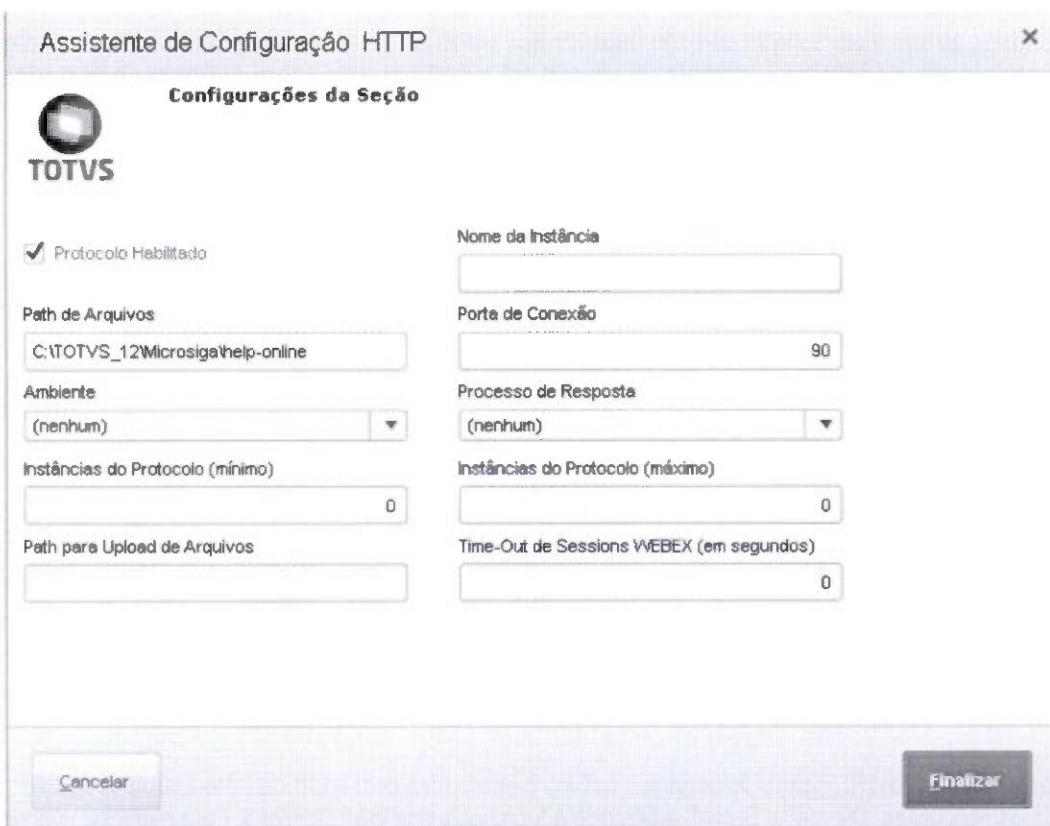
A habilitação do serviço de HTTP é necessária para a instalação dos módulos WebService. Durante a instalação de um módulo WebService, caso o serviço de HTTP não esteja habilitado, esta operação será executada automaticamente.

Ao expandir o tópico "Servidor HTTP", são mostrados os itens HTTP, HTTPS e FTP, que permitem a edição independentemente das configurações de cada um destes protocolos. Para cada um deles, são permitidas as operações de edição e exclusão da respectiva configuração.

Neste tópico iremos abordar somente a criação do Serviço HTTP e WEBSERVICES Editando a Configuração HTTP

Para inserir / editar uma configuração HTTP:

1. Abrir o Wizard localizado na pasta smartclient
2. No tópico "Servidor HTTP", posicionado o cursor sobre o item "HTTP" e clique em - "Editar Configuração", nas Ações Relacionadas.
3. Selecione o botão editar para alterar as configurações do protocolo HTTP



Protocolo Habilitado: Através deste campo é possível desabilitar a utilização do protocolo http, sem deletar as configurações atuais desta seção.

Nome da Instância: Este campo não está disponível para edição. Caso esteja preenchido, informa que um módulo Web foi instalado no host "HTTP [default]"; neste caso, não é possível alterar as informações de path, porta de conexão, ambiente e processo de resposta. Se necessário alterar as informações de configuração padrão do protocolo, deve-se utilizar o assistente de edição de Módulos Web, editando a instância que está utilizando a seção http [default].

Path de Arquivos: Especifica o diretório raiz a ser utilizado pelo protocolo "HTTP" para o acesso a arquivos estáticos e imagens. Deve ser informado com unidade de disco e caminho completo.

Porta de Conexão: Informa a porta de conexão utilizada. Para HTTP, não é aconselhável deixar a porta padrão 80.

Ambiente: Permite selecionar um ambiente (Environment) neste ByYou Application Server para atender às solicitações de processamento de links ".apl".

Processo de Resposta: Permite selecionar um processo WEB/WEBEX configurado neste ByYou Application Server para atender às solicitações de processamento de links ".apw".

Instâncias de Protocolo (mínimo e máximo): Nestas configurações, é possível especificar um número mínimo e máximo de processos internos referentes ao serviço de HTTP. Estes processos internos são utilizados para o atendimento simultâneo das requisições de conteúdo estático, arquivos, imagens, e demais arquivos disponíveis a partir da pasta definida em "Path de Arquivos", através deste protocolo(*)).

Path para Upload de Arquivos: Caso o host HTTP [default] esteja sendo utilizado com um processo de resposta de um projeto Web que suporte à funcionalidade de Upload de arquivos via HTTP, através desta chave, é possível configurar a partir de qual diretório serão gravados os arquivos enviados via http (relativo ao diretório raiz do ambiente utilizado pelo processo de resposta).

Esta configuração é atualizada, automaticamente, conforme o módulo web instalado.

Time-out de Sessões WEBEX (em segundos): Ao configurar um ou mais módulos Web que utilizem sessões de usuário através de um Processo WEBEX, é possível definir qual será o tempo de permanência em inatividade em memória das variáveis de sessões utilizadas pelos usuários do módulo web. Caso seja não especificado, o valor padrão é equivalente a 3600 segundos (uma hora).

(*) Vale ressaltar que uma thread HTTP não possui, necessariamente, ligação implícita com uma Thread AdvPL. Um Web Browser, quando solicita um arquivo HTML ou uma imagem, estabelece uma conexão HTTP com o ByYou Application Server, para receber o dado solicitado. Quando o browser recebe a informação desejada, fecha esta conexão, mantendo a Thread HTTP do Protheus disponível para atender a outras requisições HTTP, oriundas deste ou de outro Web Browser.

Módulos Web

4. Instalação do Modulo Web, no Wizard selecionar na opção "Modulo Web" e clicar no botão "Novo Modulo"



Módulo Web: Selecione o módulo Web que deve ser instalado. Para instalação do módulo PP - Portal Protheus, GPR - Gestão de Pesquisa e Resultado e GAC - Gestão de Acervos, é necessária a instalação prévia do módulo Web Services.

Nome da Instância: Informe o nome para identificação desta configuração do módulo Web, não utilize caracteres acentuados ou espaços. Este nome será utilizado para individualizar as configurações das instalações do módulo Web, assim, se a empresa necessita aplicar diferentes configurações para um mesmo módulo Web, é possível instalá-lo sob uma nova instância.

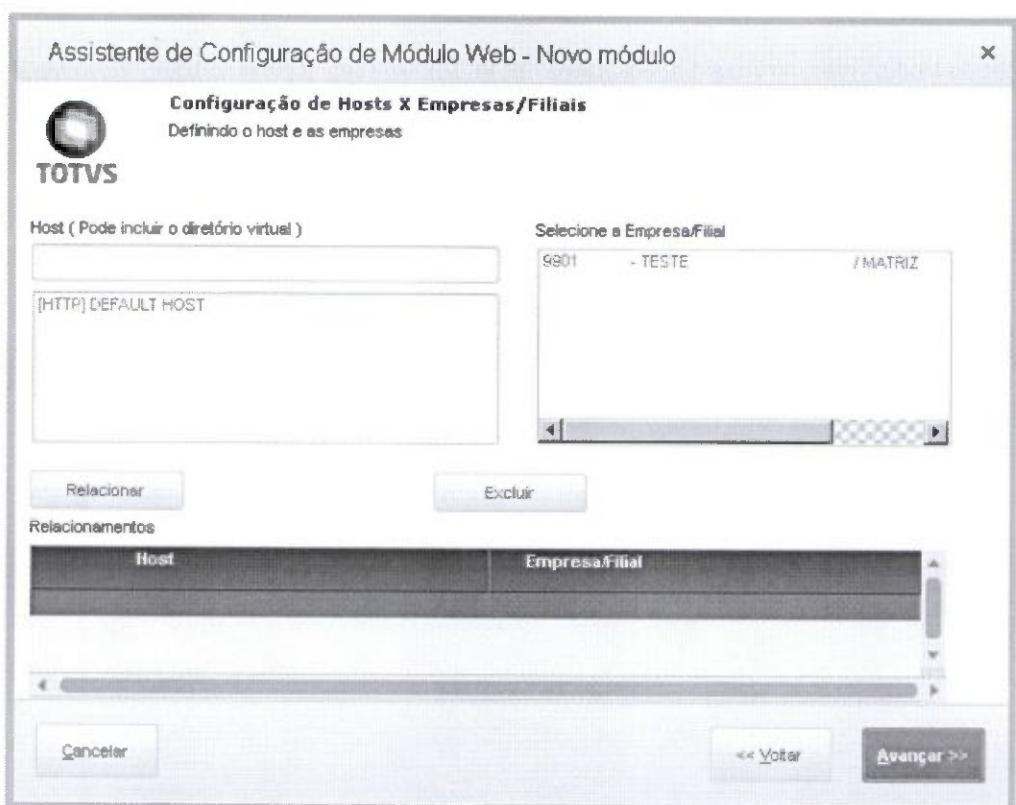
Exemplo: Na instalação do módulo GE - Educacional, cada unidade educacional pode utilizar um conjunto diferente de imagens para apresentação do seu site ou, ainda, um environment diferente no Server Protheus da versão correspondente; para isto, será necessário criar diferentes instâncias.

Diretório Raiz de Imagens (Web Path): Informe o diretório para instalação das imagens e dos demais arquivos (.css,.jar,.htm, etc.) deste módulo, que serão utilizados para apresentação no browser. Este diretório será criado abaixo do diretório raiz (RootPath) do Ambiente (environment) selecionado para a instalação.

Para cada instalação de módulo Web, deverá ser especificado um diretório diferente, iniciando com o sinal de "\\" (barra inversa).

Environment: Selecione o environment (ambiente) que será utilizado para execução do módulo. São relacionados todos os ambientes disponíveis no Server ativo.

Habilitar processos na inicialização do Servidor: Caso esta configuração seja selecionada, os processos WEB / WEBEX criados para esta configuração de módulo serão automaticamente inseridos na configuração OnStart do ByYou Application Server.



Formação Programação ADVPL



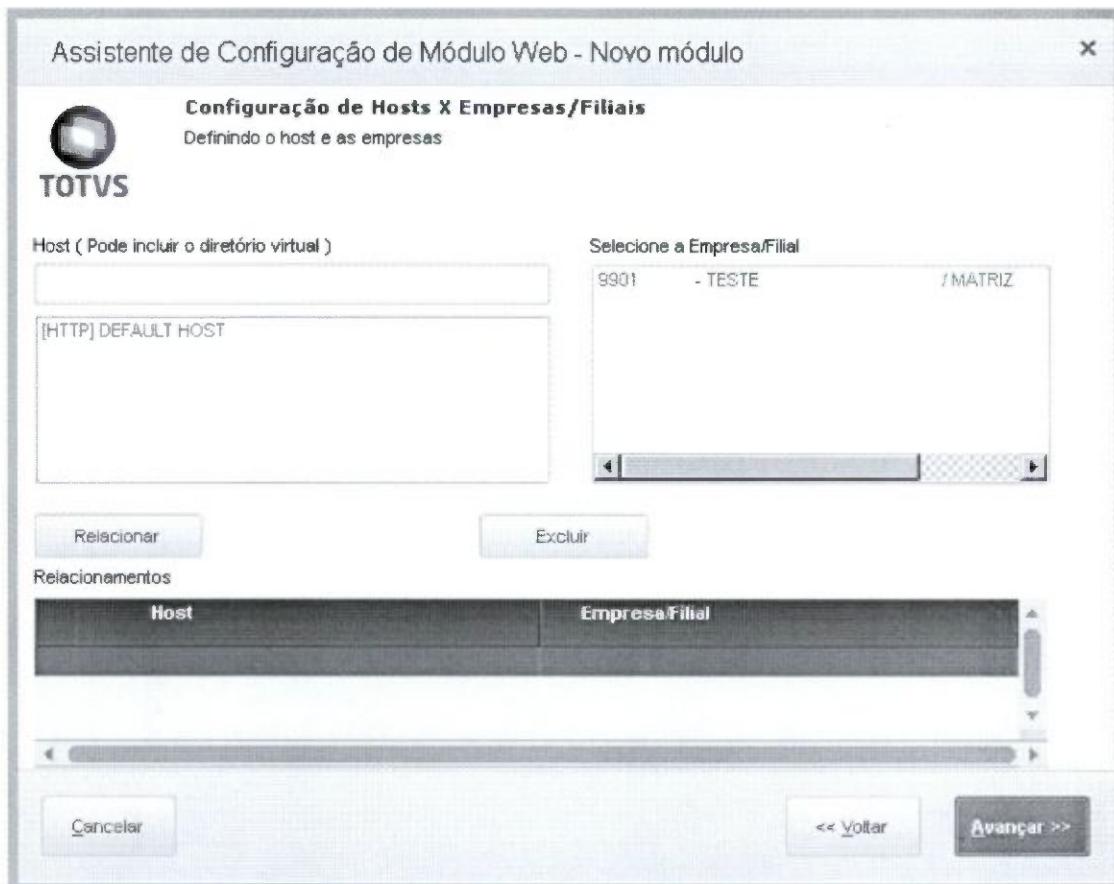
5. Para prosseguir para a segunda tela, clique em "Avançar".

Importante

O Assistente irá consistir as informações fornecidas e determinar se o ambiente está apto para instalação do módulo.

Deve-se observar que para instalação dos módulos Web, é necessário que os pacotes de instalação dos arquivos Web (.M2P) estejam disponíveis na pasta "SYSTEMLOAD" localizada abaixo do diretório raiz (RootPath) do Ambiente (environment). Caso os pacotes não sejam localizados, será apresentada uma janela de advertência.

6. A instalação poderá prosseguir; no entanto, os arquivos Web não serão descompactados, sendo apenas atualizada a configuração do servidor. Em seguida, será apresentada a janela "Configuração de Host x Empresas/Filiais".
7. Informe os dados conforme a orientação a seguir:



Host: Informe o endereço Web a partir do qual o módulo será acessado, por meio de um browser.

Exemplos:

"www.nomodosite.com.br" (para um ambiente Internet) "nomedoservidor" (para um ambiente Intranet).

Pode-se, adicionalmente, informar um diretório virtual após o Host, separando-os por uma barra "/". Isto permite que seja instalado, no mesmo host, mais de um módulo Web.

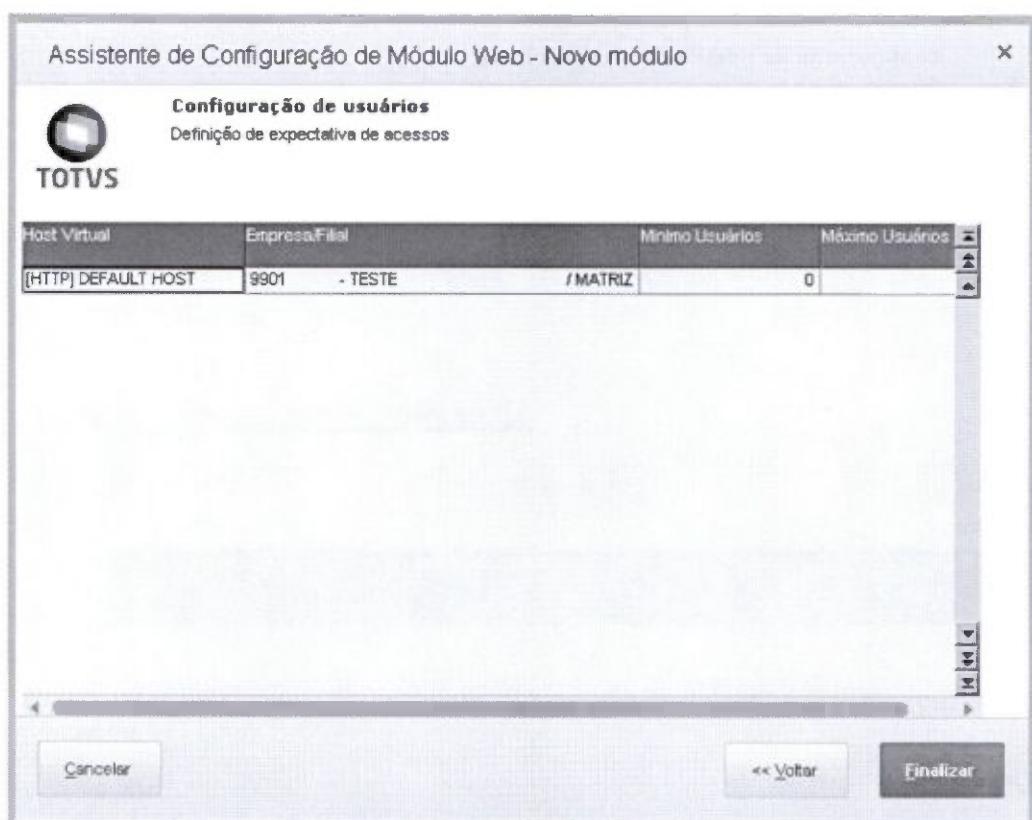
Exemplos:

"nomedoservidor/ws" (para webservices) "nomedoservidor/pp" (para o Portal)
Não se deve especificar o protocolo utilizado (como "HTTP://" ou "HTTPS://").

Vale ressaltar que é possível especificar um nome de host, não sendo obrigatoriamente o nome da estação servidor, desde que o nome especificado esteja registrado em um servidor de DNS (que relaciona um nome de host ao IP do equipamento servidor) e visível no âmbito do parque de máquinas-cliente da aplicação Web.

Selecione as Empresa/Filiais: Na área "Seleção Empresas/Filiais", selecione a empresa/filial para a qual está sendo configurado este host. Se a instalação for referente aos módulos BSC, PP e WPS, estará disponível apenas a opção "Todas as Empresas".

8. Após informar o Host e selecionar um item da área de "Seleção Empresa/Filiais", clique em "Relacionar". A amarração do host informado com este item será apresentada na janela "Relacionamentos". É possível criar diversos relacionamentos, o Assistente, automaticamente, irá criticar as amarrações, de acordo com as características operacionais do módulo em instalação.
9. Clique em "Avançar" para prosseguir.



Informe os dados conforme a orientação a seguir:

Host Virtual: Apresenta o host configurado.

Empresa/Filia: Apresenta a empresa/filial relacionada.

Mínimo Usuários: Informe a expectativa mínima de usuários que irão acessar o site.

Máximo Usuários: Informe a expectativa máxima de usuários que irão acessar o site.

Com base nos campos "Mínimo Usuários" e "Máximo Usuários", o Assistente irá determinar a quantidade média de processos (working threads) que o site irá utilizar.

Exemplo:

Mínimo: 5

Máximo: 10

Com base nesta configuração, o Protheus irá determinar um número de processos para atender a essa demanda. Considerando que a média de usuários por processo é de 10 para 1, neste caso, o Protheus irá configurar o número mínimo de 1 processos e o máximo de 3.

10. Para finalizar a instalação do módulo, clique em "Finalizar".

10.1.5. WSiNDEX - Índice de Serviços

Uma vez habilitada a configuração para Web Services, obtemos acesso a uma interface HTTP de consulta ao índice de serviços publicados. Para tal, basta reiniciar o servidor TOTVS | Application Server após a configuração ser realizada, abrir um Web Browser (por exemplo, Internet Explorer), e acessar o link <http://<servidor>/wsindex.apw>. No caso do exemplo de configuração acima, informe o endereço <http://localhost/wsindex.apw> e observe a interface de consulta ao índice dos serviços.

Exemplo:

Caso o host configurado para os Web Services seja o host local (localhost), deve-se acessar o endereço: <http://localhost/wsindex.apw>. Se estiver utilizando o sistema Microsiga Protheus 10, a tela apresentada será semelhante ao exemplo abaixo:

TOTVS Web Services

Web Services	HABILITADO
WSDL Version	ADVPL WSDL Server 1.110216
NameSpace (default)	http://localhost:90/
URL Location	http://localhost:90/ws/
Log de Chamada de Serviços	DESENLATADO
Empresa / Filial	99 / 01
Serviços Compilados	251
Serviços Abertos	0

Lista de Serviços Ativos

- ACSP
 - Consultas - ACSP
- ANALISAREC
 - Serviço de identificação de parcelas
- BANKACC
- BILL
 - Integração entre SisJuri e Microsiga-Protheus
- BINSECINTEGRATION
- CFGDICTIONARY
- CFGSTANDARDTABLES
- CFGTABLE
- CFVALIDATION
- CRCARTAO
- CRDEXTRATO
- CRDFILA
- CRDLIMCRED
- CRDLIMITE
- CRDLOGIN
- CRDORCAMENTO
- CRDBSTATUS
- CRDVENDA
- CRMCUSTOMERCONTACT
- CRMPROSPECT
- CRMSELLERCUSTOMERCONTACT

Observe que, na janela acima, são apresentados todos os serviços compilados e disponibilizados no repositório de objetos em uso no ambiente configurado. Através dessa janela, é possível obter mais detalhes de cada um dos serviços compilados. Pois, cada serviço ativo é um link para uma página que apresentará todos os métodos do serviço com um link do servidor TOTVS | Application Server que fornecerá a descrição do serviço (WSDL).

Observe, a seguir, um exemplo com os detalhes do serviço CFGTABLE.

TOTVS Web Services

NameSpace	http://webservices.microsiga.com.br/cfgtable.apw
URL Location	http://localhost:90/ws/
Nome do Serviço	CFGTABLE
Status	HABILITADO
Descrição do Serviço (WSDL)	CFGTABLE.apw?WSDL

Advertências de Carga dos Serviços

O Parâmetro [QUERYADDWHERE], do método [GETTABLE], é considerado opcional apenas para clients Protheus.
 O Parâmetro [BRANCH], do método [GETTABLE], é considerado opcional apenas para clients Protheus.
 O Parâmetro [LISTFIELDSVIEW], do método [GETTABLE], é considerado opcional apenas para clients Protheus.

Métodos do Serviço

GETTABLE

[índice de serviços](#)



Formação Programação ADVPL



Através desta janela, é possível obter a descrição do serviço WSDL ao clicar no link disponível em "Descrição do Serviço (WSDL)". Ao clicar neste link, uma nova janela será apresentada exibindo o documento WSDL do serviço. Além disso, cada método do serviço disponibilizado também é um link para uma página onde são apresentados os exemplos de pacotes SOAP que esse método especificamente espera para recepção de parâmetros e o modelo de retorno do serviço.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated 28/09/01 11:27:21 by ADVPL WSDL Server 1.110216 / Prometheus 7.00.1312274-28150602 -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:encoding="http://schemas.xmlsoap.org/wsdl/encoding/" xmlns:tm="http://schemas.microsoft.com/wsdl/2002/02/timestamp/"
  xmlns:udn="http://schemas.xmlsoap.org/wsdl/udn/" xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://webservices.micrsiga.com.br/cfgtable.apu">
  <types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="http://webservices.micrsiga.com.br/cfgtable.apu">
      <xsd:element name="GETTABLE">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element minOccurs="1" maxOccurs="1" name="USERCODE" type="xsd:string"/>
            <xsd:element minOccurs="1" maxOccurs="1" name="ALIAS" type="xsd:string"/>
            <xsd:element minOccurs="0" maxOccurs="1" name="QUERYADDINER" type="xsd:string"/>
            <xsd:element minOccurs="1" maxOccurs="1" name="BRANCH" type="xsd:string"/>
            <xsd:element minOccurs="0" maxOccurs="1" name="LISTFIELDVIEW" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="GETTABLERESPONSE">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element minOccurs="1" maxOccurs="1" name="GETTABLERESULT" type="s:TABLEVIEW"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:complexType name="TABLEVIEW">
        <xsd:sequence>
          <xsd:element minOccurs="1" maxOccurs="1" name="TABLEDATA" type="s:ARRAYOFFIELDVIEW"/>
          <xsd:element minOccurs="1" maxOccurs="1" name="TABLESTRUCT" type="s:ARRAYOFFIELDSTRUCT"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="FIELDVIEW">
        <xsd:sequence>
          <xsd:element minOccurs="1" maxOccurs="1" name="FLDTAG" type="s:ARRAYOFSTRINGS"/>
        </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="FIELDSTRUCT">
        <xsd:sequence>
          <xsd:element minOccurs="1" maxOccurs="1" name="FLDDEC" type="s:integer"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <operations>
    <operation name="GetTable" operationType="get">
      <input message="GetTableRequest" />
      <output message="GetTableResponse" />
    </operation>
  </operations>
</definitions>
```

Caso, o fonte-client AdvPL do serviço, seja gerado e esteja compilado no repositório atual, a interface de consulta habilita a funcionalidade de teste do Web Services, através da interface HTTP, apresentado no final da tela o botão testar.

Ao clicar nesse botão, será montada uma tela HTML para que os parâmetros do serviço sejam preenchidos. Após o preencher os parâmetros solicitados e submetê-los, o pacote de retorno do serviço e seu respectivo status é retornado no browser.

A operação de buscar o horário atual no servidor não necessita de nenhum parâmetro para execução. Porém, ela terá um retorno: O horário atual, no formato hh:mm:ss.

A especificação de um WebServices permite que um serviço seja declarado de modo a não receber nenhum parâmetro, mas exige que o Web Service sempre possua um retorno.

10.1.6. Estrutura do Serviço

Um método de um Web Service consiste em uma ação que será disponibilizada no serviço. Para isso, damos a ela um nome para identificação, declaramos a mesma na estrutura da classe do serviço, bem como seus parâmetros e respectivo retorno.

- **Parâmetros:** Ao declarar o fonte de um método, o mesmo pode receber um ou mais parâmetros, de tipo básico e/ou estruturas, é obrigatório ter um parâmetro.
- **Retorno :** Um método de Web Service deve, obrigatoriamente, ter uma propriedade de retorno. Não faz parte da especificação de Web Services a criação de um método que não possua retorno.

- Codificando o método em AdvPL:** Como visto anteriormente, tanto os parâmetros quanto o retorno de um método de Web Service devem ser declarados com um dado da classe (através da instrução WSDATA).

Ao escrever um método de um Web Service, e o mesmo receber uma solicitação de processamento, as propriedades declaradas como parâmetros do método são alimentadas e o método executado.

Por tratarem-se de propriedades, o código-fonte AdvPL deverá interagir com essas propriedades, prefixando-as com '..' (dois pontos seguidos) ou 'self:', sendo isto válido tanto para os parâmetros do método como para a propriedade de retorno.

Dada a existência de uma LIB de infra-estrutura, que realiza a comunicação, validação, montagem e desmontagem de pacotes.

Ao codificar um método de Web Service existem sempre dois retornos: A propriedade de retorno do método e o retorno efetivo do método ao final do processamento.

O retorno efetivo do método deve ser um valor booleano: Se verdadeiro (.T.), indica à LIB que o método foi executado com sucesso e, consequentemente, a propriedade de retorno foi alimentada. Logo, o pacote SOAP de retorno do método será montado pela LIB e devolvido, automaticamente, ao client que solicitou a chamada de processamento.

Caso o retorno efetivo do método seja falso (.F.), indica à LIB que, por alguma razão tratada no código-fonte do método, não foi possível a execução do método. Neste caso, deve-se especificar, antes do retorno, através da função SetSoapFault(), a causa da impossibilidade de processamento.

10.1.7. Tipos básicos de dados - 'Server'

Ao escrever um Web Service 'Server', deve-se especificar o tipo de informação de cada parâmetro e retorno, em conformidade com a especificação SOAP, utilizada nos pacotes XML de troca de dados. São considerados e suportados, pelo TOTVS | Application Server, quando da declaração dos parâmetros e retorno, os seguintes tipos básicos:

Tipo	Descrição
String	Dado AdvPL do tipo string.
Date	Dado AdvPL do tipo data.
Integer	Dado AdvPL do tipo numérico (apenas números inteiros).
Float	Dado AdvPL do tipo numérico (pode conter números inteiros e não-inteiros).
Boolean	Dado AdvPL do tipo booleano (lógico).
Base64Binary	Dado AdvPL do tipo string binária, aceitando todos os caracteres da tabela ASCII, de CHR(0) à CHR(255).

10.1.8. Estruturas - Tipos complexos

Uma estrutura (também conhecida por complex type), constitui um comando especial da linguagem AdvPL, chamado WSSTRUCT, criado especificamente para Web Services. Desta forma, deve-se criar uma estrutura quando tiver a necessidade de agrupar mais de uma informação, incluindo tipos básicos e/ou outras estruturas.

Ao criar um serviço que deverá receber como parâmetro um grupo de informações definidas como, por exemplo, os dados cadastrais de um cliente, deve-se criar uma estrutura para agrupar esses dados. No entanto, vale ressaltar que, a declaração de uma estrutura não amarra a mesma ao serviço em questão, de modo que a mesma estrutura pode ser utilizada para mais de um serviço compilado no repositório. Caso a estrutura criada seja específica para o serviço em questão, é recomendado que seja dado um nome à mesma que tenha a ver com o serviço ao qual ela pertença, pois não é possível compilar mais de uma estrutura de mesmo nome no repositório.

Para definir que a estrutura será tratada como um array, utiliza-se ARRAY OF antes do tipo da variável e/ou estrutura. Dentro da seção WSSERVICE não é permitido o uso de ARRAY OF.

10.1.9. Codificando o serviço

Para codificar um serviço, deve-se utilizar o TOTVS | Development Studio, e criar um novo arquivo de programa, e nele escrever o serviço. A numeração disposta à esquerda do código fonte é meramente ilustrativa, não devendo ser digitada. Essa numeração é utilizada mais abaixo, onde o código fonte exemplo é detalhado linha a linha.

```
1 #include "Protheus.ch"
2 #include "ApWebSRV.ch"
3 #include "TbiConn.ch"
4
5 WSSERVICE SERVERTIME Description "VEJA O HORARIO"
6 WSDATA Horário AS String
7 WSDATA Parametro AS String
8 WSMETHOD GetServerTime Description "METHOD DE VISUALIZAÇÃO DO HORARIO"
9
10 ENDWSSERVICE
11
12
13 WSMETHOD GetServerTime WSRECEIVE Parametro WSSEND Horario WSSERVICE SERVERTIME
14      ::Horário := TIME()
15 Return .T.
```

Linha 1 - É especificada a utilização do include Totvs.CH ou Protheus.ch, contendo as definições dos comandos AdvPL e demais constantes.

Linha 2 - É especificada a include APWebSrv.CH ou Totvswebsrv.ch, que contém as definições de comandos e constantes utilizados nas declarações de estruturas e métodos do Web Services. Ele é obrigatório para o desenvolvimento de Web Services.

Linha 3 - É especificada a utilização da include TBICONN.CH, contendo as definições dos comando AdvPL para conectar ao banco e Protheus Desejado.

Linha 5 - Com esta instrução, é definido o inicio da classe do serviço principal, ao qual damos o nome de SERVERTIME.

Linha 6 - Dentro da estrutura deste serviço, é informado que um dos parâmetros utilizados chama-se horário, e será do tipo string.

Linha 7 - Dentro da estrutura deste serviço, é informado que um dos parâmetros utilizados passar o parâmetro do tipo string.

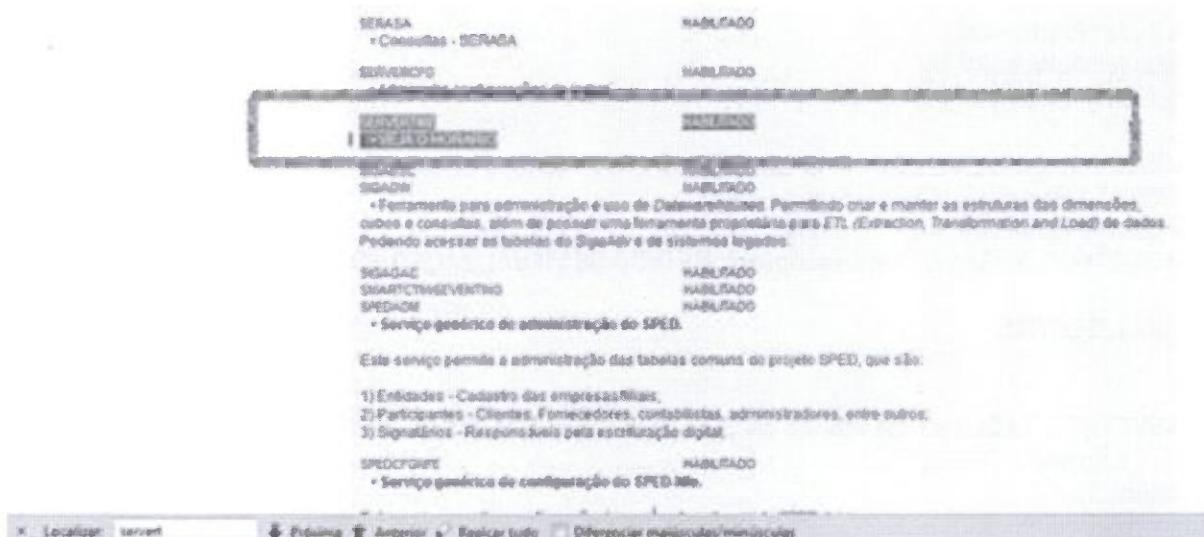
Linha 8 - Dentro da estrutura deste serviço, é informado que um dos métodos do serviço chama-se GetServerTime.

Linha 10 - Como não são mais necessárias mais propriedades ou métodos neste serviço, a estrutura do serviço é fechada com esta instrução.

Linha 13 - Aqui é declarado o fonte do método GetServerTime, que não receberá parâmetro nenhum (mas para efeitos de declaração, deve ser informado que ele receberá o parâmetro NULLPARAM), e é informado que seu retorno será o dado Horário (declarado na classe do serviço como uma propriedade, do tipo string).

Linha 14 - É atribuído na propriedade ::Horário da classe deste serviço, o retorno da função AdvPL Time(), que retoma a hora atual no servidor no formato HH:MM:SS. Observe que, deve-se utilizar "::" para alimentarmos a propriedade da classe atual.

Linha 15 - O método GetServerTime é finalizado nesta linha, retornando .T. (verdadeiro), indicando que o serviço foi executado com sucesso.



Ao acessar o serviço ira trazer as informações:

The screenshot shows the TOTVS Web Services interface. It displays the service details: NameSpace, UPL Location, Nome do Serviço, Status, and Descrição do Serviço (WSDL). It also lists the methods: GETSERVERTIME and VEJA O HORARIO. A link to 'Indice de serviços' is at the bottom right.

10.1.9.1 Testando o serviço

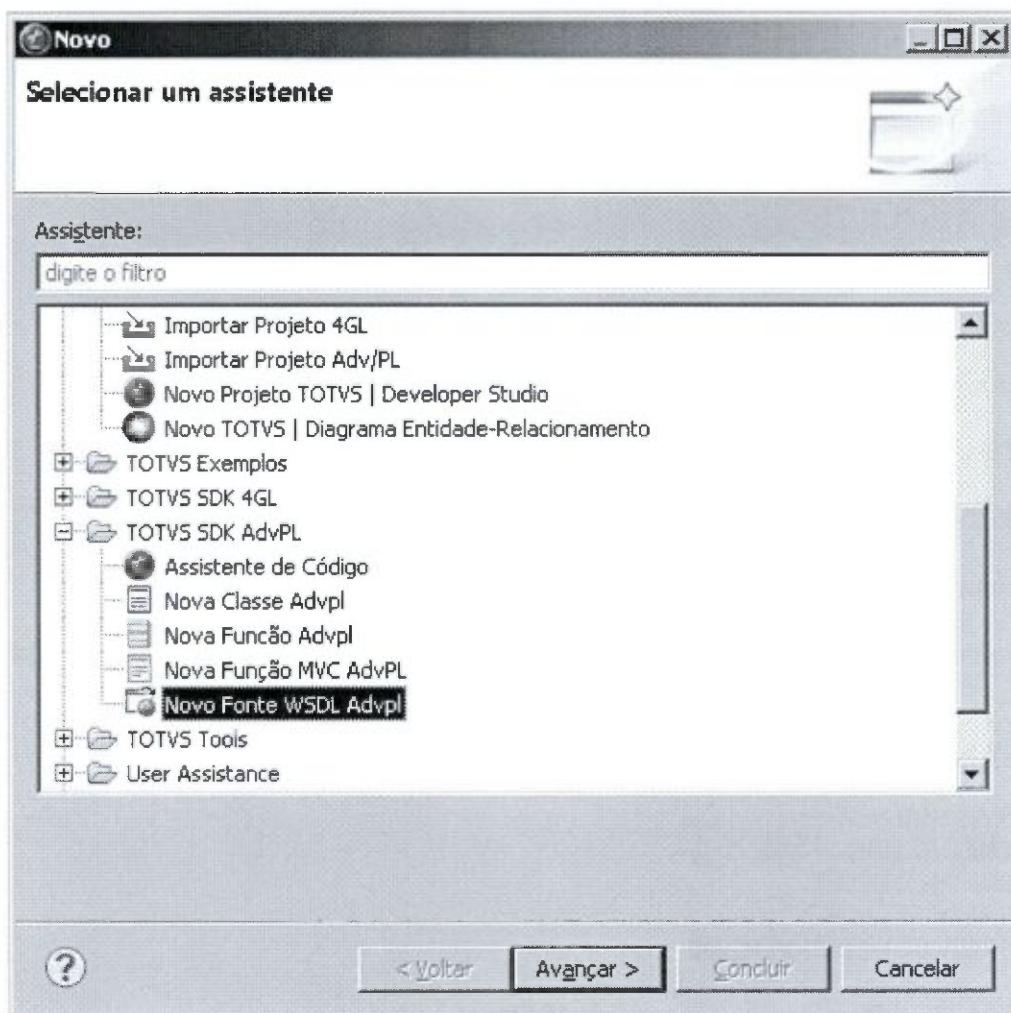
Ao acessar o serviço é possível realizar teste, para os teste ser realizado é necessário ter um Client do Web Service criado, para criar o cliente necessário obter o endereço do serviço disponível na Descrição do Serviço (WSDL);

Exemplo: <http://localhost:90/ws/SERVERTIME.apw?WSDL>

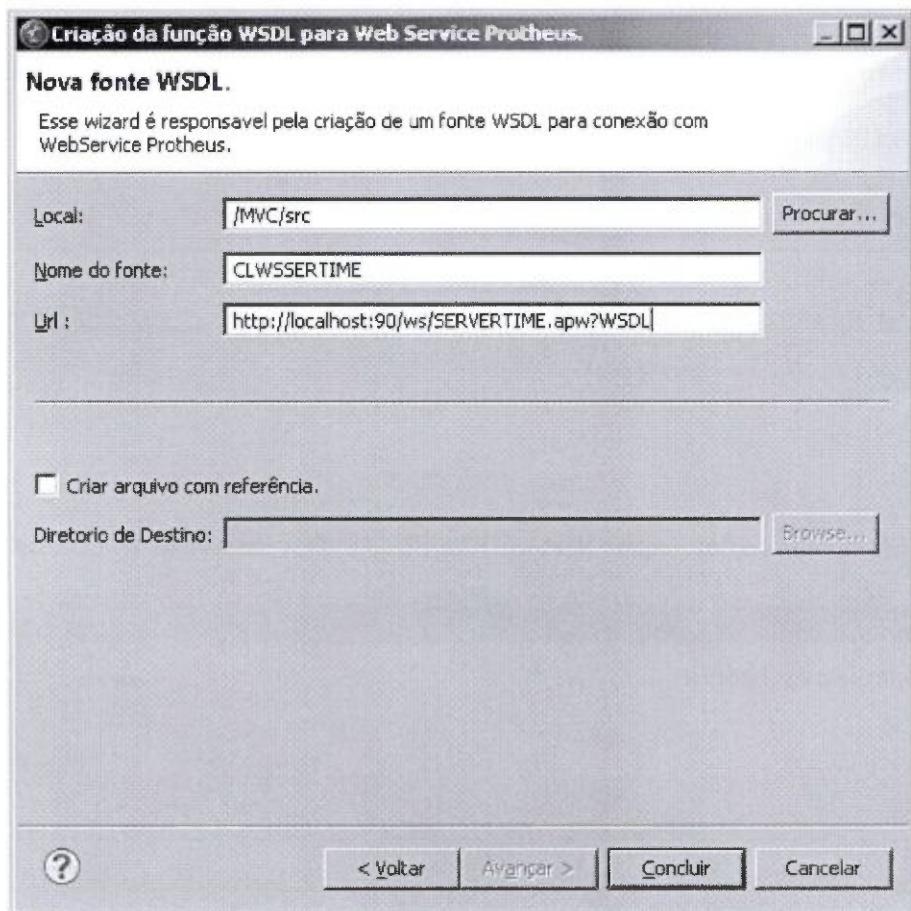
This XML file does not appear to have any style information associated with it. The document tree is shown below

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generated 20160301 14:55:09 by ADVPL WSDL Server 1.110216 / Protheus 7.00.131227A-20160114 NG
-->
<definitions xmlns:tns="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s2="http://localhost:90/"
  xmlns:encoding="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://localhost:90/">
  <types>
    <xsd:schema elementFormDefault="qualified" targetNamespace="http://localhost:90/">
      <xsd:element name="GETSERVERTIME">
        <xsd:complexType>
          <xsd:sequence>
```

Acessar o Developer Studio ir nas opções: Arquivo Novo – TOTVS SDK ADVPL – Novo Fonte WSDL Advpl



Na tela do Nova fonte WSDL.



Preencher com as informações:

- **Local:** Diretório que irá salvar o fonte;
- **Nome do fonte:** Nome do arquivo PRW que será criado;
- **Url:** Informar a URL do serviço que deseja criar o cliente.

Após preencher os dados como informado selecionar o botão "Concluir". Ao Clicar ok o sistema irá gerar o código fonte do cliente.

WSCLIENT WSSERVERTIME

WSMETHOD NEW
WSMETHOD INIT
WSMETHOD RESET
WSMETHOD CLONE
WSMETHOD GETSERVERTIME

WSDATA _URL
WSDATA _HEADOUT

AS String
AS Array of String

```
WSDATA _COOKIES           AS Array of String  
WSDATA cPARAMETRO        AS string  
WSDATA cGETSERVERTIMERESULT AS string
```

ENDWSCLIENT

Essa parte inicializa a criação do client do WebService podemos notar que o nome do cliente do WebService SERVERTIME é parecido com o nome do Client WSSERVERTIME

Vejamos que foi criado 4 methodos que não existiam no próprio Web Service criado.

```
WSMETHOD NEW WSCLIENT WSSERVERTIME  
::Init()  
If !FindFunction("XMLCHILDEX")  
    UserException("O Código-Fonte Client atual requer os executáveis do Protheus Build  
[7.00.131227A-20160114 NG] ou superior. Atualize o Protheus ou gere o Código-Fonte novamente  
utilizando o Build atual.")  
EndIf  
Return Self  
  
WSMETHOD INIT WSCLIENT WSSERVERTIME  
Return  
  
WSMETHOD RESET WSCLIENT WSSERVERTIME  
::cPARAMETRO := NIL  
::cGETSERVERTIMERESULT := NIL  
::Init()  
Return  
  
WSMETHOD CLONE WSCLIENT WSSERVERTIME  
Local oClone := WSSERVERTIME():New()  
oClone:_URL := ::_URL  
oClone:cPARAMETRO := ::cPARAMETRO  
oClone:cGETSERVERTIMERESULT := ::cGETSERVERTIMERESULT  
Return oClone
```

Pode analisando o próprio metodo chama outro método gerado pelo IDE INIT. Trata de um processo de criação do objeto WebService para disponibilizar a criação ou chamada de outros serviços disponível no repositório para complementar o WebService do cliente

Analissamos agora o Method RESET. Trata de um processo de limpeza de variáveis do WebService para que você possa utilizar novamente sem estar com as informações executadas anteriormente

Analissamos agora o Metodo CLONE.Tratamento de gerar uma nova variável com o Objeto criado do WebService "Duplicar a informação dos dados do WebService"

// WSDL Method GETSERVERTIME of Service WSSERVERTIME

```
WSMETHOD GETSERVERTIME WSSEND cPARAMETRO WSRECEIVE cGETSERVERTIMERESULT  
WSCLIENT WSSERVERTIME  
Local cSoap := "", oXmlRet
```

```
BEGIN WSMETHOD

cSoap += '<GETSERVERTIME xmlns="http://localhost:90/">'
cSoap += WSSoapValue("PARAMETRO", ::cPARAMETRO, cPARAMETRO, "string", .T., .F., 0, NIL, .F.)
cSoap += "</GETSERVERTIME>"
```

Analisamos agora o metodo GETSERVERTIME. Tratamento de executar o service disponível pelo WebService e retornar o processo executado por ele. Retornando na variável cGETSERVERTIMERESULT

```
oXmlRet := SvcSoapCall(      Self,cSoap,;
                           "http://localhost:90/GETSERVERTIME";
                           "DOCUMENT","http://localhost:90/", "1.031217";
                           "http://localhost:90/ws/SERVERTIME.apw")

::Init()
::cGETSERVERTIMERESULT      :=           WSAdvValue(          oXmlRet,
                           "_GETSERVERTIMERESONSE:_GETSERVERTIMERESULT:TEXT", "string",NIL,NIL,NIL,NIL,NIL,NIL)

END WSMETHOD

oXmlRet := NIL
Return .T.
```

Analizando o código fonte ele utiliza uma função chamada WSSoapValue essa função executa toda a estrutura do XML para dentro do WebService criando as suas tags respectivas que o metodo solicitado exige.

Logo a baixo é apresentada outra função WSADVVALUE que retorna o valor que o WebService está disponibilizando.

10.1.9.2 Consumindo um Web Service

Agora devemos gerar um código fonte "User Function" para capturar a informação disponível do nosso WebService.

```
#include "Protheus.ch"
User function XtempoX()

Local oBj := nil

oBj:= WSSERVERTIME():new()
oBj:GetServerTime() alert(oBj:cGetServerTimeResult)

Return()
```

Vamos definir que iremos gerar um WebServices para listar os dados do SIGAMAT.EMP

No primeiro momento veremos o código da declaração do WebService:

Esse código apresenta a criação do WebService chamado CTT apresentando a Descrição "Treinamento do WebService para o Curso CTT"

```
#include "Totvs.ch"
#include "Totvswebsrv.ch"
```

```
WsService CTT description "Curso CTT"
Wsdata cCodEmp as String      // código da empresa
Wsdata aEmpresa as array of EstruturaEmp    // sigamat.emp Wsdata
EndWsservice
```

Foi criado as Variáveis.

```
WsStruct EstruturaEmp
WsData      M0_CODIGO      as String
WsData      M0_CODFIL      as String
WsData      M0_FILIAL      as String
WsData      M0_NOME        as String
WsData      M0_NOMECOM     as String
WsData      M0_ENDCOB      as String
WsData      M0_CIDCOB      as String
WsData      M0_ESTCOB      as String
WsData      M0_CEPCOB      as String
WsData      M0_ENDENT      as String
WsData      M0_CIDENT      as String
WsData      M0_ESTENT      as String
WsData      M0_CEPENT      as String
WsData      M0_CGC          as String
WsData      M0_INSC         as String
WsData      M0_TEL          as String
WsData      M0_EQUIP        as String
WsData      M0_SEQUENC      as String
WsData      M0_DOCSEQ       as Integer
WsData      M0_FAX          as String
WsData      M0_PRODRUR     as String
WsData      M0_BAIRCOB     as String
WsData      M0_BAIRENT     as String
WsData      M0_COMPCOB     as String
WsData      M0_COMPENT      as String
WsData      M0_TPINSC       as Integer
WsData      M0_CNAE         as String
WsData      M0_FPas         as String
WsData      M0_ACTRAB       as String
WsData      M0_CODMUN       as String
WsData      M0_NATJUR       as String
WsData M0_DTBaste      as String
WsData      M0_NUMPROP      as Integer
WsData      M0_MODEND      as String
WsData      M0_MODINSC     as String
WsData      M0_CAUSA         as String
WsData      M0_INSCANT      as String
WsData      M0_TEL_IMP      as String
WsData      M0_FAX_IMP      as String
WsData      M0_TEL_PO        as String
WsData      M0_FAX_PO        as String
WsData      M0_IMP_CON      as String
WsData      M0_CODZOSE      as String
WsData      M0_DESZOSE      as String
```

```

WsData      M0_COD_ATV    as String
WsData      M0_INS_SUF    as String
WsData      M0_EMERGEN   as String
WsData      M0_LIBMOD     as String
WsData      M0_TPESTAB    as String
WsData      M0_DTAUTOR    as Date
WsData      M0_EMPB2B    as String
WsData      M0_CAIXA      as String
WsData      M0_LICENSA   as String
WsData      M0_CORPKEY   as String
WsData      M0_CHKSUM     as Integer
WsData      M0_DTVLD      as Date
WsData      M0_PSW        as String
WsData      M0_CTPSW      as String
WsData      M0_INTCTRL    as String
WsData      M0_INSCM      as String
WsData      M0_NIRE       as String
WsData      M0_DTRE        as date
WsData      M0_CNES        as String
WsData      M0_PSWSTRT   as String
WsData      M0_DSICCNA   as String
WsData      M0_asSPAT1    as String
WsData      M0_asSPAT2    as String
WsData      M0_asSPAT3    as String
WsData      M0_SIZEFIL   as integer
WsData      M0_LEIAUTE    as String
WsData      M0_PICTURE    as String
WsData      M0_STATUS     as String
WsData      M0_RNTRC      as String
WsData      M0_DTRNTRC    as Date
WsData      X_MENSAGEM   as String
EndWsStruct

```

Nesse Método está sendo apresentada todos campos do Sigamat.emp por final será criado um array e cada vetor desse array será o campo apresentado como variável nessa estrutura.

Essa estrutura irá montar um XML com o nome de cada variável Exemplo

```

<aEmpresa>
  <M0_CODIGO>01</M0_CODIGO>
  Etc...
</aEmpresa>

```

Logo a baixo foi criado o método de Lista empresa serviço que irá ler o sigamat e apresentar quantas empresas temos no sigamat.emp para o cliente

WsMethod LISTAEMPRESA DESCRIPTION "APRESENTA TODOS OS DADOS DO SIGAMAT. EMP DO CLIENTE"

Foi criado o Método disponibilizando o serviço proposto de apresentar as empresas:

WsMethod LISTAEMPRESA WsReceive cCodEmp WsSend aEmpresa WsService CTT

```

Local cEmp          := "99"
Local cFil          := "01"
Local aTab          := {"SA1"}
Local aRet          := {}
Local nDados        := 0

```

RpcSetEnv(cEmp,cFil,,,'FIN','ListEmpresa',aTab)//abre a conexão com o banco e a empresa padrão

Após validar a informação da Chave cCodEmp que definimos deverá ser "Abrir" o sistema analisa que a palavra está ou não está correta. Após a validação o Array ficara delimitado somente as colunas onde será obrigatória o seu preenchimento.

```

If cCodEmp != 'Abrir'
  ::cRet := "Palavra Chave Invalida"
  aadd(aEmpresa,WsClassNew("EstruturaEmp"))
  aEmpresa[1]:M0_CODIGO      := ""
  aEmpresa[1]:M0_CODFIL       := ""
  aEmpresa[1]:M0_FILIAL       := ""
  aEmpresa[1]:M0_NOME         := ""
  aEmpresa[1]:M0_NOMECON      := ""
  aEmpresa[1]:M0_ENDCOB       := ""
  aEmpresa[1]:M0_CIDCOB       := ""
  aEmpresa[1]:M0_ESTCOB       := ""
  aEmpresa[1]:M0_CEPCOB       := ""
  aEmpresa[1]:M0_ENDENT       := ""
  aEmpresa[1]:M0_CIDENT       := ""
  aEmpresa[1]:M0_ESTENT       := ""
  aEmpresa[1]:M0_CEPENT       := ""
  aEmpresa[1]:M0_CGC           := ""
  aEmpresa[1]:M0_INSC          := ""
  aEmpresa[1]:M0_TEL           := ""
  aEmpresa[1]:M0_EQUIP          := ""
  aEmpresa[1]:M0_SEQUENC       := ""
  aEmpresa[1]:M0_DOCSEQ        := 0
  aEmpresa[1]:M0_FAX           := ""
  aEmpresa[1]:M0_PRODRUR       := ""
  aEmpresa[1]:M0_BAIRCOB       := ""
  aEmpresa[1]:M0_BAIRENT       := ""
  aEmpresa[1]:M0_COMPCOB       := ""
  aEmpresa[1]:M0_COMPENT       := ""
  aEmpresa[1]:M0_TPINSC        := 0
  aEmpresa[1]:M0_CNAE          := ""
  aEmpresa[1]:M0_FPas          := ""
  aEmpresa[1]:M0_ACTRAB        := ""
  aEmpresa[1]:M0_CODMUN        := ""
  aEmpresa[1]:M0_NATJUR        := ""
  aEmpresa[1]:M0_DTBasE        := ""
  aEmpresa[1]:M0_NUMPROP        := 0
  aEmpresa[1]:M0_MODEND        := ""
  aEmpresa[1]:M0_MODINSC        := ""
  aEmpresa[1]:M0_CAUSA          := ""
  aEmpresa[1]:M0_INSCANT        := ""

```

```

aEmpresa[1]:M0_TEL_IMP      := ""
aEmpresa[1]:M0_FAX_IMP      := ""
aEmpresa[1]:M0_TEL_PO       := ""
aEmpresa[1]:M0_FAX_PO       := ""
aEmpresa[1]:M0_IMP_CON      := ""
aEmpresa[1]:M0_CODZOSE     := ""
aEmpresa[1]:M0_DESZOSE     := ""
aEmpresa[1]:M0_COD_ATV      := ""
aEmpresa[1]:M0_INS_SUF      := ""
aEmpresa[1]:M0_EMERGEN     := ""
aEmpresa[1]:M0_LIBMOD       := ""
aEmpresa[1]:M0_TPESTAB      := ""
aEmpresa[1]:M0_DTAUTOR     := STOD("")
aEmpresa[1]:M0_EMPB2B       := ""
aEmpresa[1]:M0_CAIXA        := ""
aEmpresa[1]:M0_LICENSA      := ""
aEmpresa[1]:M0_CORPKEY      := ""
aEmpresa[1]:M0_CHKSUM       := 0
aEmpresa[1]:M0_DTVLD        := STOD("")
aEmpresa[1]:M0_PSW          := ""
aEmpresa[1]:M0_CTPSW         := ""
aEmpresa[1]:M0_INTCTRL       := ""
aEmpresa[1]:M0_INSCM         := ""
aEmpresa[1]:M0_NIRE          := ""
aEmpresa[1]:M0_DTRE          := STOD("")
aEmpresa[1]:M0_CNES          := ""
aEmpresa[1]:M0_PSWSTRT       := ""
aEmpresa[1]:M0_DSICCNA       := ""
aEmpresa[1]:M0_asSPAT1       := ""
aEmpresa[1]:M0_asSPAT2       := ""
aEmpresa[1]:M0_asSPAT3       := ""
aEmpresa[1]:M0_SIZEFIL       := 0
aEmpresa[1]:M0_LEIAUTE        := ""
aEmpresa[1]:M0_PICTURE        := ""
aEmpresa[1]:M0_STATUS         := ""
aEmpresa[1]:M0_RNTRC          := ""
aEmpresa[1]:M0_DTRNTRC        := STOD("")
aEmpresa[1]:X_MENSAGEM      := ::cRet
Return (.T.)
EndIf

aRet      := SM0->(GETAREA())

While SM0->(!EOF())
  nDados += 1
  aadd( aEmpresa,WsClassNew( "EstruturaEmp" ) )
  aEmpresa[nDados]:M0_CODIGO      := SM0->M0_CODIGO
  aEmpresa[nDados]:M0_CODFIL      := SM0->M0_CODFIL
  aEmpresa[nDados]:M0_FILIAL      := SM0->M0_FILIAL
  aEmpresa[nDados]:M0_NOME        := SM0->M0_NOME
  aEmpresa[nDados]:M0_NOMECOM     := SM0->M0_NOMECOM
  aEmpresa[nDados]:M0_ENDCOB      := SM0->M0_ENDCOB

```

Formação Programação ADVPL



aEmpresa[nDados]:M0_CIDCOB	:= SM0->M0_CIDCOB
aEmpresa[nDados]:M0_ESTCOB	:= SM0->M0_ESTCOB
aEmpresa[nDados]:M0_CEPCOB	:= SM0->M0_CEPCOB
aEmpresa[nDados]:M0_ENDENT	:= SM0->M0_ENDENT
aEmpresa[nDados]:M0_CIDENT	:= SM0->M0_CIDENT
aEmpresa[nDados]:M0_ESTENT	:= SM0->M0_ESTENT
aEmpresa[nDados]:M0_CEPENT	:= SM0->M0_CEPENT
aEmpresa[nDados]:M0_CGC	:= SM0->M0_CGC
aEmpresa[nDados]:M0_INSC	:= SM0->M0_INSC
aEmpresa[nDados]:M0_TEL	:= SM0->M0_TEL
aEmpresa[nDados]:M0_EQUIP	:= SM0->M0_EQUIP
aEmpresa[nDados]:M0_SEQUENC	:= SM0->M0_SEQUENC
aEmpresa[nDados]:M0_DOCSEQ	:= SM0->M0_DOCSEQ
aEmpresa[nDados]:M0_FAX	:= SM0->M0_FAX
aEmpresa[nDados]:M0_PRODRUR	:= SM0->M0_PRODRUR
aEmpresa[nDados]:M0_BAIRCOB	:= SM0->M0_BAIRCOB
aEmpresa[nDados]:M0_BAIRENT	:= SM0->M0_BAIRENT
aEmpresa[nDados]:M0_COMPCOB	:= SM0->M0_COMPCOB
aEmpresa[nDados]:M0_COMPENT	:= SM0->M0_COMPENT
aEmpresa[nDados]:M0_TPINSC	:= SM0->M0_TPINSC
aEmpresa[nDados]:M0_CNAE	:= SM0->M0_CNAE
aEmpresa[nDados]:M0_FPas	:= SM0->M0_FPas
aEmpresa[nDados]:M0_ACTRAB	:= SM0->M0_ACTRAB
aEmpresa[nDados]:M0_CODMUN	:= SM0->M0_CODMUN
aEmpresa[nDados]:M0_NATJUR	:= SM0->M0_NATJUR
aEmpresa[nDados]:M0_DTBasE	:= SM0->M0_DTBasE
aEmpresa[nDados]:M0_NUMPROP	:= SM0->M0_NUMPROP
aEmpresa[nDados]:M0_MODEND	:= SM0->M0_MODEND
aEmpresa[nDados]:M0_MODINSC	:= SM0->M0_MODINSC
aEmpresa[nDados]:M0_CAUSA	:= SM0->M0_CAUSA
aEmpresa[nDados]:M0_INSCANT	:= SM0->M0_INSCANT
aEmpresa[nDados]:M0_TEL_IMP	:= SM0->M0_TEL_IMP
aEmpresa[nDados]:M0_FAX_IMP	:= SM0->M0_FAX_IMP
aEmpresa[nDados]:M0_TEL_PO	:= SM0->M0_TEL_PO
aEmpresa[nDados]:M0_FAX_PO	:= SM0->M0_FAX_PO
aEmpresa[nDados]:M0_IMP_CON	:= SM0->M0_IMP_CON
aEmpresa[nDados]:M0_CODZOSE	:= SM0->M0_CODZOSE
aEmpresa[nDados]:M0_DESZOSE	:= SM0->M0_DESZOSE
aEmpresa[nDados]:M0_COD_ATV	:= SM0->M0_COD_ATV
aEmpresa[nDados]:M0_INS_SUF	:= SM0->M0_INS_SUF
aEmpresa[nDados]:M0_EMERGEN	:= SM0->M0_EMERGEN
aEmpresa[nDados]:M0_LIBMOD	:= SM0->M0_LIBMOD
aEmpresa[nDados]:M0_TPESTAB	:= SM0->M0_TPESTAB
aEmpresa[nDados]:M0_DTAUTOR	:= SM0->M0_DTAUTOR
aEmpresa[nDados]:M0_EMPB2B	:= SM0->M0_EMPB2B
aEmpresa[nDados]:M0_CAIXA	:= SM0->M0_CAIXA
aEmpresa[nDados]:M0_LICENSA	:= SM0->M0_LICENSA
aEmpresa[nDados]:M0_CORPKEY	:= SM0->M0_CORPKEY
aEmpresa[nDados]:M0_CHKSUM	:= SM0->M0_CHKSUM
aEmpresa[nDados]:M0_DTVLD	:= SM0->M0_DTVLD
aEmpresa[nDados]:M0_PSW	:= SM0->M0_PSW
aEmpresa[nDados]:M0_CTPSW	:= SM0->M0_CTPSW

```

aEmpresa[nDados]:M0_INTCTRL      := SMO->M0_INTCTRL
aEmpresa[nDados]:M0_INSCM        := SMO->M0_INSCM
aEmpresa[nDados]:M0_NIRE         := SMO->M0_NIRE
aEmpresa[nDados]:M0_DTRE         := SMO->M0_DTRE
aEmpresa[nDados]:M0_CNES         := SMO->M0_CNES
aEmpresa[nDados]:M0_PSWSTRT     := SMO->M0_PSWSTRT
aEmpresa[nDados]:M0_DSCCNA       := SMO->M0_DSCCNA
aEmpresa[nDados]:M0_asSPAT1      := SMO->M0_asSPAT1
aEmpresa[nDados]:M0_asSPAT2      := SMO->M0_asSPAT2
aEmpresa[nDados]:M0_asSPAT3      := SMO->M0_asSPAT3
aEmpresa[nDados]:M0_SIZEFIL       := SMO->M0_SIZEFIL
aEmpresa[nDados]:M0_LEIAUTE       := SMO->M0_LEIAUTE
aEmpresa[nDados]:M0_PICTURE       := SMO->M0_PICTURE
aEmpresa[nDados]:M0_STATUS        := SMO->M0_STATUS
aEmpresa[nDados]:M0_RNTRC         := SMO->M0_RNTRC
aEmpresa[nDados]:M0_DTRNTRC       := SMO->M0_DTRNTRC
aEmpresa[nDados]:X_MENSAGEM      := "Sucesso "+strzero(nDados,2)
SMO->(dbSkip())

EndDo

RestArea(aRet)

Return(.T.)

```

10.1.10. TWsdlManager

A classe TWsdlManager faz o tratamento para arquivos WSDL (Web Services Description Language). Esta classe implementa métodos para identificação das informações de envio e resposta das operações definidas, além de métodos para envio e recebimento do documento SOAP.

A classe trabalha com 2 tipos de dados: os tipos complexos, que são as seções do XML, e os tipos simples, que são as tags que recebem valor. Por exemplo, numa operação de inserção de clientes, a tag "cliente" é um tipo complexo, pois contém os dados do cliente dentro, e uma tag "nome" é um tipo simples, pois recebe diretamente um valor (no caso, o nome do cliente).

TWsdManager irá realizar o parse de um WSDL, seja por uma URL ou por arquivo, e irá montar internamente uma lista com os tipos simples e outra com os tipos complexos. A lista de tipos complexos terá somente os tipos complexos que tenham número variável de ocorrências. Por exemplo, se tiver 2 tipos complexos onde um deles tem mínimo de 1 e máximo de 1, e outro com mínimo de 1 e máximo 2, só o que tem o valor mínimo diferente do valor máximo irá ser listado.

Através do método NextComplex é possível verificar quais são esses elementos de tipo complexo que necessitam de definição do número de ocorrências. Esse método deve ser chamado em recursão, até não existirem mais elementos retornados (irá retornar Nil). Para cada elemento retornado deve-se definir a quantidade de vezes que a tag irá aparecer no XML final (SOAP). Para isso utiliza-se o método SetComplexOccurs. Caso não seja especificado a quantidade de vezes que a tag irá aparecer, a classe irá considerar a quantidade como 0 (zero).

Caso seja passado zero no método SetComplexOccurs ele irá marcar os elementos simples e complexos dentro da tag para serem ignorados, o que fará com que os elementos complexo internos não apareçam no retorno do método NextComplex, e os elementos simples internos não serão retornados pelo método SimpleInput.

Uma vez definida a quantidade de vezes que os tipos complexos irão aparecer na mensagem SOAP, aí pode chamar o método SimpleInput, para poder retornar quais são os campos que irão receber valor. Os tipos simples podem ter seus valores definidos através dos métodos SetValue (para 1 valor apenas) ou SetValues (para mais de 1 valor). Para saber a quantidade de valores aceitos pelo tipo simples é só olhar a quantidade mínima e máxima informada no método SimpleInput, índices 3 e 4 de cada tag, respectivamente.

Exemplo:

```
Local oWsdl
Local lOk
Local aOps := {}, aComplex := {}, aSimple := {}

// Cria o objeto da classe TWsdlManager
oWsdl := TWsdlManager():New()

// Faz o parse de uma URL
lOk := oWsdl:ParseURL( "http://localhost:90/ws/SERVERTIME.apw?WSDL" )
if lOk == .F.
    MsgStop( oWsdl:cError , "ParseURL() ERROR" )
Return
endif

// Lista os Metodos do serviço
aOps := oWsdl>ListOperations()

// Seta a operação a ser utilizada listada pelo ListOperations nome do metodo
// que ira executar.

    lOk := oWsdl:SetOperation( "GETSERVERTIME" )

    if !lOk
        MsgStop( oWsdl:cError , "SetOperation(ConversionRate) ERROR" )
    Return
    Endif

// Verifica o tipo de parametro que vai ser usado pelo método, retornando quais
// são os campos que irão receber valor

aComplex := oWsdl:NextComplex()
aSimple := oWsdl:SimpleInput()

// Passando os valores para o parametro do metodo
xRet := oWsdl:SetValue( 0, "000" )

// Exibe a mensagem que será enviada
conout( oWsdl:GetSoapMsg() )

    // Faz a requisição ao WebService
    lOk := oWsdl:SendSoapMsg()

    if !lOk
        MsgStop( oWsdl:cError , "SendSoapMsg() ERROR" )
```

```

Return
endif

// Recupera os elementos de retorno, já parseados
cResp := oWsdl:GetParsedResponse()

// Monta um array com a resposta transformada, considerando
// as quebras de linha ( LF == Chr(10) )

aElem := StrTokArr(cResp,chr(10))
MsgInfo( SubStr(aElem[2], AT(":",aElem[2 ])+1))

Return( NIL )

```

10.2. Webservices MVC

Ao se desenvolver uma aplicação utilizando MVC, já estará disponível um Web Service para ser utilizado para o recebimento de dados. Todas as aplicações em MVC utilizarão o mesmo Web Service, independentemente de sua estrutura ou de quantas entidades ele possua. O Web Service que está disponível para o MVC é o FWWSMODEL. A ideia básica é que iremos instanciar o Web Service, informar qual a aplicação que será utilizada e informar os dados em um formato XML.

Veremos como construir uma aplicação que utilize o Web Service FWWSMODEL com um modelo de dados (Model) que possui apenas uma entidade.

O instânciamento se dá seguinte forma:

```

// Instancia o WebService Genérico para Rotinas em MVC
oMVCWS := WsFwWsModel():New()
// URL onde esta o WebService FWWSModel do Protheus
oMVCWS:_URL := http://127.0.0.1:8080/ws/FWWSMODEL.apw
// Seta Atributos do WebService
oMVCWS:cModelId := 'COMP012_MVC' // Fonte de onde se usara o Model
// Exemplo de como pegar a descrição do Modelo de Dados
//If oMVCWS:GetDescription()
// MsgInfo( oMVCWS:cGetDescriptionResult )
//Else
// MsgStop( 'Problemas em obter descrição do Model' + CRLF + WSError() )
//Endif
// Obtém a estrutura dos dados do Model
If oMVCWS:GetXMLData()
    // Retorno da GetXMLData
    cXMLEstrut := oMVCWS:cGetXMLDataResult
    // Retorna
    //<?xml version="1.0" encoding="UTF-8"?>
    //<COMP012M Operation="1" version="1.01">
    // <ZA0MASTER modeltype="FIELDS" >
    // <ZA0_FILIAL order="1"><value></value></ZA0_FILIAL>
    // <ZA0_CODIGO order="2"><value></value></ZA0_CODIGO>
    // <ZA0_NOME order="3"><value></value></ZA0_NOME>
    // </ZA0MASTER>
    //</COMP012M>

```

```
// Obtém o esquema de dados XML (XSD)
If oMVCWS:GetSchema()
    cXMLEsquema := oMVCWS:cGetSchemaResult
Endif
// Cria o XML
cXML := '<?xml version="1.0" encoding="UTF-8"?>'
cXML += '<COMP012M Operation="1" version="1.01">'
cXML += ' <ZA0MASTER modeltype="FIELDS" >'
cXML += ' <ZA0_FILIAL order="1"><value>01</value></ZA0_FILIAL>'
cXML += ' <ZA0_CODIGO order="2"><value>000100</value></ZA0_CODIGO>'
cXML += ' <ZA0_NOME order="3"><value>Tom Jobim</value></ZA0_NOME>'
cXML += ' </ZA0MASTER>'
cXML += '</COMP012M>'
// Joga o XML para o atributo do WebService
oMVCWS:cModelXML := cXML
// Valida e Grava os dados
If oMVCWS:PutXMLData()
    If oMVCWS:IPutXMLDataResult
        MsgInfo( 'Informação Importada com sucesso.' )
    Else
        MsgStop( 'Não importado' + CRLF + WSError() )
    Endif
Else
    MsgStop( AllTrim( oMVCWS:cVldXMLDataResult ) + CRLF + WSError() )
Endif
```