Índice

PADRÕES DE PROGRAMAÇÃO (4GL E 4JS)

1. Definições

1.1. TAMANHO TIPOS DE DADOS

Abaixo estão citados os respectivos tamanhos em bytes ocupados em memória para cada tipo de dado na linguagem INFORMIX 4GL durante a execução de um programa:

DECIMAL = (n/2) + 1
Exemplo: DECIMAL(9,2) --> (9/2)+1=6
Esta coluna irá ocupar 6 bytes.
SMALLINT = 2 bytes
INTEGER = 4 bytes
DATE = 4 bytes
CHAR = n bytes
DATETIME = (n/2) + 1
onde “n” é a somatória dos dígitos
para: YEAR = 4 dígitos
MONTH = 2 dígitos
DAY = 2 dígitos
HOUR = 2 dígitos
MINUTE = 2 dígitos
SECOND = 2 dígitos
FRACTION = 1 a 5 dígitos
Exemplo: DATETIME YEAR TO FRACTION
[(4+2+2+2+2+2+3)/2] + 1 = 10

ARRAY = (n+3)*o
onde “n” é a somatória dos bytes das colunas
“o” é a quantidade de ocorrências
(n+3) é o tamanho de uma ocorrência da variável do tipo array.

OBSERVAÇÃO:

É muito importante ter uma noção do volume de memória que as variáveis alocadas em tempo de execução ocupam, para sempre tentar ocupar o mínimo de memória possível, com objetivo de deixar espaço para execução de um maior número de aplicações possível.

Para isso deve-se atentar para as seguintes regras:

- Definir variáveis globais somente quando for estritamente necessário. As variáveis globais são definidas para compartilhamento de dados entre 2 ou mais arquivos “fonte” (extensão .4gl) que são compilados e posteriormente *linkeditados* de forma que gerem somente um arquivo executável.

- Definir variáveis modulares somente quando o volume de funções que realizarem acesso a uma determinada informação for alto e/ou esta informação for acessada em momentos diferentes durante a execução do programa;

- Sempre que possível definir variáveis locais para processamento de dados, pois estas são alocadas na memória somente enquanto uma determinada função está em execução. Ao término da execução de uma função, a memória alocada para as variáveis locais definidas na função é liberada.

1.2. NOMENCLATURA

1.2.1. Fontes

O controle da numeração de programas deverá ser feito pelo CSL (Controle de Sistemas da Logocenter).

A nomenclatura de um fonte deve ter o formato **sssnnnn.4gl** onde:

- **sss** identifica o sistema, como por exemplo: cap (Contas a Pagar), che (Controle de Cheques), cmi (Correção Monetária Integral), cre (Contas a Receber), man (Manufatura), etc.
- **nnnn** numeração seqüencial de 0001 a 9999 por sistema.
- **4gl** sufixo que identifica o programa fonte (fixo)

IMPORTANTE:

Para criação de programas que são utilizados com objetivo principal a execução de uma rotina para emissão de algum relatório de verificação de erros no banco de dados ou então para realizar possíveis correções de registros no banco de dados, deve-se tomar cuidado para não definir a nomenclatura do fonte com a sigla do módulo do sistema padrão LOGIX e sim utilizar a sigla do módulo conversor do respectivo módulo padrão LOGIX, pois estes programas são criados com objetivo "conversor" e isto é determinado pelo analista.

Exemplo: Programa para acerto de dados numa tabela do módulo SUPRIMENTOS (SUP).

Neste caso deverá ser criado um programa com a sigla SUC que é a sigla para conversores do módulo SUP. Isto pode ser consultado via sistema CSL opção 2 (Sistemas) consultando o sistema SUP.

1.2.2. Variáveis

- Identificar sempre com um prefixo todos os objetos de programa (nomes de variáveis, windows, screen record, cursores) conforme tabela abaixo:

Objeto de programa	Prefixo	Exemplo
Variável global:		
Tipo registro (RECORD)	gr_	gr_funcionario
Tipo vetor (ARRAY)	ga_	ga_valores
Tipo simples	g_	g_dat_referencia
Variável modular:		
Registro (RECORD)	mr_	mr_funcionario
Vetor (ARRAY)	ma_	ma_valores
Simples	m_	m_dat_referencia
Variável local:		
Registro (RECORD)	lr_	lr_funcionario
Vetor (ARRAY)	la_	la_valores
Simples	l_	l_dat_referencia
Registro de tela (SCREEN RECORD)	sr_	sr_men0010

Cursores (CURSOR)		
para leitura (query)	cq_	cq_funcionario
para modificação	cm_	cm_funcionario
para inclusão	ci_	ci_funcionario
para exclusão	ce_	ce_funcionario
Janelas (WINDOWS)	w_	w_men0010
Sentenças SQL preparadas (PREPARE)	var_	var_funcionario
Critérios de seleção (CONSTRUCT)		where_clause
Sentenças SQL a preparar (SELECT)		sql_stmt

OBSERVAÇÃO: Refere-se a tipo “simples” os tipos de variáveis diferentes de “record” e “array”. Exemplos: date, char, smallint, integer, datetime.

- Utilizar palavras auto-explicativas para formar o nome de uma variável, para que seja fácil de interpretar o objetivo para qual está sendo definida.

Exemplos:

Variável local para indicar o salário de um funcionário.

```
DEFINE l_salario_funcio DECIMAL(17,2)
```

Variável modular para indicar o registro de dados de um aluno:

```
DEFINE mr_aluno RECORD LIKE aluno.*
```

- Utilizar sempre que possível tipo de dado associado a uma coluna de tabela, utilizando a cláusula “LIKE”.

Exemplos:

Variável local que indica o login do usuário da tabela USUÁRIOS (coluna COD_USUARIO)

```
DEFINE l_cod_usuario LIKE usuarios.cod_usuario
```

Variável modular que indica o registro da tabela ALUNO

```
DEFINE mr_aluno   RECORD LIKE aluno.*
```

- Utilizar a definição das variáveis dentro do escopo do programa, sendo primeiramente as variáveis globais, depois as modulares. As variáveis locais devem ser definidas sempre no início de cada função.

- Sempre inicializar o valor de variáveis antes de atribuir algum valor a ela pela primeira vez ou quando for necessário.

1.2.3. Tabelas do banco de dados

Nome da tabela não poderá exceder 18 caracteres e o nome do arquivo que contém a definição da tabela (CREATE TABLE) não poderá exceder 8 caracteres. O nome do arquivo com a definição da tabela deverá ter extensão igual a “sql”.

Exemplo:

```
nome da tabela  = funcionario
nome do arquivo = funcio.sql
```

Observação: O controle dos nomes dos arquivos com extensão “sql” deverá ser feito pelo sistema CSL.

1.2.3.1. Nomes de índices de tabelas

O nome do índice da tabela deverá seguir o padrão ix1_aaaaaaaa.

Onde “aaaaaaaa” é o nome do arquivo com a definição da tabela, sem a extensão “sql” e “n” é um número seqüencial.

Exemplo:

```
nome da tabela          = funcionario
nome do arquivo         = funcio.sql
nome do índice          = ix1_funcio
nome da chave primária  = pk_funcio
```

Observação:
Digitar os comandos SQL em letra maiúscula e o nome da tabela, colunas e índices em letra minúscula.

1.3. DEFINIÇÃO DE TELAS

Na definição da tela (arquivo com extensão “.per”) devem ser levados em consideração os seguintes padrões:

1.3.1. Tamanho

- O tamanho das telas poderá ser de, no máximo, 78 colunas por 18 linhas, considerando as linhas que estiverem entre chaves “{}”. Este padrão é obrigatório para execução no ambiente caracter, pois no ambiente gráfico a limitação fica em 78 colunas e 21 linhas;

- Cuidado para não definir telas com um número de colunas menor que as mensagens emitidas pelo programa utilizando o comando MESSAGE. O texto de help das opções de menu (COMMAND) também não podem ultrapassar a largura total definida para a tela.

1.3.2. Posição

- Na execução de programas, a tela principal sempre deverá ser apresentada de forma centralizada na posição horizontal, de acordo como o número de colunas definido para esta. A posição vertical para as telas principais também deverá respeitar a posição centralizada, podendo também serem apresentadas mais próximas da margem superior da tela.

1.3.3. Formato

- Todos os campos de tela deverão ser delimitados por colchetes “[]”. Por motivo de espaço no desenho da tela, pode ser utilizado o símbolo pipe “|”. Neste caso e nas telas de “zoom” (popup) deverá ser definida a instrução DELIMITERS=' ' no bloco “INSTRUCTIONS” de definições da tela;

- As telas que apresentam opções de menu na execução deverão ser definidas com uma linha tracejada (hífens) como primeira linha, para delimitar a largura da tela a ser apresentada;

Exemplo de tela com apresentação de menu:

```
{
------------------------------------------------------------
 [a0]                  TÍTULO TELA

      Título primeiro campo: [a  ]
       Título segundo campo: [b                  ]

}
```

- As telas que NÃO apresentarem opções de menu na execução, mas tiverem a descrição de título, deverão ter uma linha tracejada (hífens) logo abaixo da linha onde foi definido o título, sendo que o título deverá obrigatoriamente ser descrito na primeira linha de definição da tela. Para as telas que não apresentam opções de menu na execução e não possuem descrição de título, não deverá conter a linha tracejada no início da definição da tela;

Exemplo de tela sem apresentação de menu mas com título:

```
{
 [a0]                  TÍTULO TELA
------------------------------------------------------------

      Título primeiro campo: [a  ]
       Título segundo campo: [b                  ]

}
```

Exemplo de tela sem apresentação de menu e sem título:

```
{

 Título primeiro campo: [a  ]
  Título segundo campo: [b                  ]

}
```

- As telas desenvolvidas para funções de “zoom” (popup) deverão ter seu título descrito na primeira linha de definição da tela e NÃO deverão conter a linha tracejada abaixo do título;

Exemplo de tela para funções de zoom:

```
{
 [a0]     TÍTULO TELA

 Coluna1  Coluna2
 -------  --------------------
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
```

```
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
[a1     ][a2                  ]
}
```

- Quando se tratar de um programa que utiliza dados provenientes de tabelas do banco de dados que possuem controle de informações por empresa (tabela possui coluna de código da empresa pertencente a chave primária), no canto superior esquerdo da tela, na mesma linha do início da descrição do título da tela e uma posição a direita, deverá ser definido um campo reservado para apresentar o código da empresa ativa no sistema (variável p_cod_empresa carregada pela função *log001_acessa_usuario()*);

- Tentar desenvolver telas de fácil visualização e interpretação pelo usuário. Atentar para estética visual da tela em execução, de forma que as informações sejam agrupadas, sempre levando em consideração a ordem mais coerente em que os dados devem ser informados. Por exemplo, tentar agrupar objetos dos tipos CHECKBOX e RADIO, agrupar campos que determinam intervalos, mas somente para os campos que não sejam dependentes de outra informação anterior definida em outro formato;

- Os campos que tiverem como domínio as opções “SIM” ou “NÃO” (Objeto do tipo CHECKBOX na versão gráfica do 4JS) deverão ser apresentados no formato CHECKBOX na versão gráfica 4JS;

- Para os campos do tipo CHECKBOX ou RADIO, deverá ser definido o atributo COMMENTS apresentando conteúdos diferentes entre a versão caracter e a gráfica. Na versão caracter deverá apresentar a lista de opções válidas para o respectivo campo, sendo que estas opções também deverão ser devidamente acentuadas. Na versão gráfica, somente o campo do tipo CHECKBOX é que deverá apresentar um texto explicativo. O atributo COMMENTS para descrever a lista de domínios válidos deverá respeitar o seguinte formato para versão caracter: <LETRA> - <Descrição>, sendo que na falta de espaço para apresentar a lista de domínios válidos pode-se abolir o espaço em branco antes e depois do “-” (hífen).

  Exemplo:

```
ATTRIBUTES
x = formonly.ck1 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(S,N),
    COMMENTS='S – Sim   N - Não'                  #para versão caracter
--#, WIDGET='CHECK', CONFIG='S N {}'
--# COMMENTS='Sim (marcado)   Não (desmarcado)'    #para versão gráfica
;
y = formonly.rd2 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(C,S,V,P,D,M),
    COMMENTS='C-Casado  S-Solteiro  V-Viúvo  P-Separado  D-Divorciado'
--#, WIDGET='RADIO'
--#, CONFIG='C {Casado} S {Solteiro} V {Viúvo} P {Separado} D
{Divorciado}'
--#, COMMENTS=''     #Versão gráfica não deverá apresentar COMMENTS
                #para objeto do tipo Radio
;
END
```

- Não elaborar tela onde na mesma tela apresente informações de parâmetros e informações de processamento, sabendo que são informações utilizadas em processos diferentes, ou seja, uma parte das informações é utilizada pela opção “Informar” e a outra somente pela opção “Processar”;

Exemplo:

Forma incorreta: registrar todos dados em uma mesma tela, mesmo se tratando de informações de processamentos diferentes.

```
+-----------------------------------------------------+
| [  ]      EMISSÃO NOTAS FISCAIS POR PERÍODO         |
|                                                     |
|      Período: [          ] até [          ]         |
|                                                     |
|                                                     |
|                P R O C E S S A N D O                |
|                                                     |
|   Nota: [       ]   Série: [   ]   Subsérie: [  ]   |
|                                                     |
+-----------------------------------------------------+
```

Forma correta: criar 2 telas distintas para cada informação.

Tela principal

```
+-----------------------------------------------------+
| [  ]     EMISSÃO NOTAS FISCAIS POR PERÍODO          |
|                                                     |
|      Período: [          ] até [          ]         |
|                                                     |
+-----------------------------------------------------+
```

Tela acionada somente na função de processamento

```
+-----------------------------------------------------+
|                     PROCESSANDO                     |
|                                                     |
|   Nota: [       ]   Série: [   ]   Subsérie: [  ]   |
|                                                     |
+-----------------------------------------------------+
```

- Quando um campo admitir 'Zoom', neste deverão ser codificadas instruções em 4JS para apresentar a figura do botão de zoom ao lado do respectivo campo. Este deverá ser codificado na definição do campo que possui ‘Zoom’ na tela (arquivo com extensão .PER).

  Exemplo:

```
b  = FORMONLY.descricao TYPE CHAR, UPSHIFT
  --#, WIDGET="FIELD_BMP", CONFIG="combo.bmp Control-z"
;
```

- Após o delimitador de final do campo “]” (colchete) para qual foi definido o botão de ‘Zoom’, no arquivo de definição de tela (arquivo com extensão .PER), obrigatoriamente deverá ser deixada uma posição em branco, para que exista espaço disponível para apresentação do botão de zoom no canto direito do campo no momento da sua execução em ambiente gráfico.

1.3.4. Título principal

- Deve estar centralizado na linha 2 da tela, logo abaixo da linha tracejada, quando esta existir, caso contrário deverá centralizar o título na linha 1 da tela;

- Deverá ser totalmente escrito em letras maiúsculas sem espacejamento entre as letras de cada palavra do título;

- Todas as palavras deverão ser devidamente acentuadas;

- Evitar uso de preposições, somente utilizando caso interfira na interpretação do texto pelo usuário;

- Utilizar abreviações somente em caso de falta de espaço. O uso de siglas também pode ser utilizado;

- Não utilizar termos técnicos na definição dos títulos de tela. Exemplo: No lugar da palavra "Default" deve-se utilizar "Padrão". Outros exemplos de palavras que não devem ser utilizadas: "Imprimir", "Digitar", "Deletar", "Inicializar", "Tabela" quando tiver como significado uma tabela do banco de dados, nomes de tabelas e ou colunas do banco de dados;

- No título de tela utilizada para emissão de relatório, não utilizar o termo "RELATÓRIO" ou "LISTAGEM". Neste caso deve-se utilizar o termo "EMISSÃO";

- No título de tela utilizada para geração de informações, não utilizar o termo "CRIAÇÃO" ou "ELABORAÇÃO". Neste caso deve-se utilizar o termo "GERAÇÃO";

- No título de telas utilizadas para cadastros de informações, não utilizar o termo "CADASTRO" ou "INCLUSÃO", apenas deve ser citada a descrição da tabela a ser cadastrada no sistema. Exemplo: Para "CADASTRO DE PEDIDOS DE COMPRA" deve ser apenas "PEDIDOS DE COMPRA";

- Após o título principal da tela, quando houver espaço disponível, obrigatoriamente deverá ser deixada **pelo menos 1 linha em branco** após o título.

- Sempre que possível deve-se registrar o título principal da tela no plural, principalmente no caso de programas que fazem referência a um conjunto de informações e não a uma informação única.

Exemplos:

"ITENS" para programa de cadastro de itens, pois pode-se cadastrar e consultar todos os itens pelo mesmo programa.

"EMISSÃO NOTAS FISCAIS" para programa que emite notas fiscais, pois em tela permite informar uma faixa de notas fiscais que serão emitidas num mesmo processamento. Caso a tela somente permitisse informar uma única nota fiscal, o título seria "EMISSÃO NOTA FISCAL", visto que num processamento somente iria emitir uma única nota fiscal por vez.

"ORÇAMENTOS POR PERÍODO" para programa de cadastro de orçamentos por período, pois em um mesmo período pode-se registrar vários orçamentos.

"FECHAMENTO MENSAL" para programa de fechamento mensal de estoque. Neste caso é um título que não pode ser colocado no plural, pois se trata de uma rotina mensal que gera um resultado único que é o fechamento mensal, e não seria correto citar como "FECHAMENTOS MENSAIS".

"GERAÇÃO PEDIDOS AUTOMÁTICOS" para programa que gera pedidos automáticos com base em ordens já existentes e a cada processamento é gerado vários pedidos. Caso o processamento resultasse em apenas um único pedido o título seria "GERAÇÂO PEDIDO AUTOMÁTICO".

1.3.5. Título de campos

- Os títulos de campos devem ser finalizados por ":" (dois pontos) não deixando espaços entre o título do campo e os dois pontos;

- Os campos que tiverem como domínio as opções "SIM" ou "NÃO" (Objeto do tipo CHECKBOX na versão gráfica do 4JS) deverão ter seu título finalizado por "?" (interrogação), não deixando espaços entre o título do campo e o ponto de interrogação;

- Os campos que tiverem como domínio as opções "SIM" ou "NÃO" e no seu título existir o uso de algum verbo indicando uma ação, tais verbos deverão ser escritos no modo infinitivo;

- Os títulos de cada campo de tela deverão estar alinhados a direita pelo caracter separador ":" (dois pontos) ou "?" (interrogação), de forma que a posição inicial do valor apresentado para cada campo fique alinhada. Isto também é válido para o caracter delimitador de início de campo "[" (colchete), pois este deverá estar alinhado com o "[" (colchete) do campo definido acima, quando já existir o alinhamento pelo ":" (dois pontos) ou "?" (interrogação). O alinhamento dos ":" ou "?" poderá ser feito também em blocos de informações de uma mesma tela, separados por uma linha em branco entre os blocos, desde que não comprometa a estética de apresentação da tela.

- Entre o caracter de finalização de título de campo, ":" (dois pontos) ou "?" (interrogação), e o caracter delimitador de início de campo, "[" (colchete), deverá existir sempre 1 espaço em branco, podendo ser abolido em caso de falta de espaço em tela. Se em alguma situação de tela ocorrer algum título de campo em que este espaço em branco tenha sido abolido, todos os demais títulos deverão respeitar a mesma regra, abolindo também o espaço em branco entre o final do título do campo (dois pontos ou interrogação) e o caracter delimitador de início de campo "[" (colchete);

- Os títulos dos campos deverão ser escritos apenas com a primeira letra maiúscula. Se o título estiver descrito com duas ou mais palavras, apenas a primeira letra da primeira palavra deverá ser escrita em letra maiúscula, às demais devem ficar em letras minúsculas, com exceção de siglas que devem ser escritas totalmente em letras maiúsculas (Ex: INSS, IR, NF) ou palavras que mereçam destaque que podem ter as iniciais em letra maiúscula;

  Como descrever a sigla na documentação:

  1. Caso a sigla se repita por várias vezes num mesmo documento, a mesma deve ser informada uma única vez, no início do documento.

  2. As siglas devem ser precedidas pelo nome por extenso
     Ex: Declaração do Imposto Retido na Fonte (Dirf); Relação Anual de Informações Sociais (Rais).

  3. Siglas consagradas pelo uso corrente não precisam de explicação.
     Ex: CPF, CNPJ, CEP, UF.

  4. O significado das siglas de palavras estrangeiras deve ser dado em português, exceto as siglas que não tiverem a tradução para o português reconhecida.
     Ex: RP (Rendezvous point); PIM-SM (Protocol Independent Multicast – Sparse Mode).

  5. Use maiúsculas nos seguintes casos:

     - Siglas - ABNT, RNP , UFRJ, PUC, MCT, CGEE (Exceções: PoP, QoS, CNPq, CPqD, UnB).

     - Acrônimos (siglas que formam palavras) com até três letras - ONU, RAU, MEC. A partir de quatro letras, use minúsculas - Banerj, Unicef.

6. Use minúsculas para os casos de unidades de medidas. Ex: Gbps, Mbps, Hz, Km.

- As descrições dos títulos dos campos deverão ser devidamente acentuadas e deverá ser evitados o uso de preposições. Somente utilizar preposições caso interfira na interpretação do texto pelo usuário.

- As abreviações, quando utilizadas em caso de não existir espaço suficiente em tela, NÃO devem ser finalizadas por “.” (ponto);

- Quando o título de campo for formado por mais de uma palavra, entre uma palavra e outra deverá existir somente 1 (um) espaço;

- Não utilizar o termo “Código” e em alguns casos o termo “Número”. Exemplo:”Número Seqüência”. Neste caso a palavra “Número” deverá ser abolida;

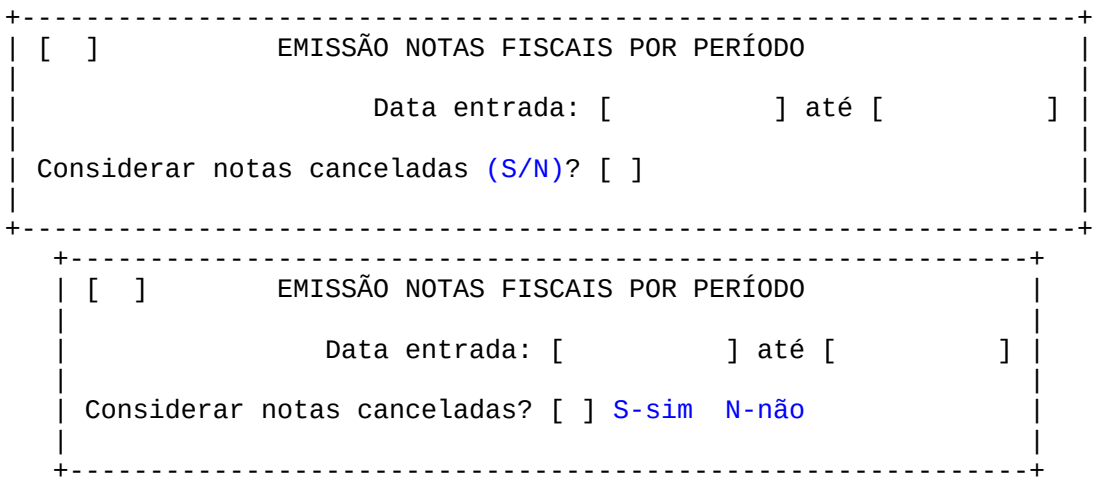
- Não utilizar termos técnicos na definição dos títulos de campos. Exemplo: No lugar da palavra “Default” deve-se utilizar “Padrão”. Outros exemplos de palavras que não devem ser utilizadas: “Imprimir”, “Digitar”, “Deletar”, “Inicializar”. O termo “Tabela” não pode ser utilizado quando tiver como significado uma tabela do banco de dados, nomes de tabelas e ou colunas do banco de dados. Um exemplo comum permitido de uso do termo “tabela” é “Tabela de IR”;

- Cuidar para que em uma mesma tela não sejam utilizados 2 termos de mesmo significado (sinônimos). Por exemplo: “Emitir” e “Listar”, “Somente” e “Apenas”, “Excluir” e “Eliminar”;

- Para campos que contenham uma lista de domínios válidos (onde geralmente utiliza-se a função log0830_listbox para zoom das opções válidas), não deve-se apresentar a lista de opções válidas junto ao título do campo ou ao lado deste, e sim, apresentar tais opções no arquivo de definição da tela (arquivo .PER), utilizando o atributo COMMENTS, no caso da execução do sistema no ambiente caracter;

  Exemplo:

  Forma incorreta: Incluir os domínios válidos junto ao título do campo.



  Forma correta: Título do campo sem domínio na sua descrição. Neste caso na definição da tela (arquivo com extensão .PER), para este campo deverá ser definido o atributo COMMENTS para apresentar a lista de domínios válidos, conforme o padrão gráfico ou caracter;

```
+-------------------------------------------------------------+
| [  ]       EMISSÃO NOTAS FISCAIS POR PERÍODO                |
|                                                             |
|                 Data entrada: [          ] até [          ] |
|                                                             |
| Considerar notas canceladas? [ ]                            |
|                                                             |
+-------------------------------------------------------------+
```

- Para títulos de campos que determinam intervalos (faixas de valores) deve-se definir apenas 1 título de campo para indicar a faixa inicial e final, sendo que entre os 2 campos deverá existir a palavra “até”, escrita em letras minúsculas e sem finalização com “:” (dois pontos) como forma de representação de um intervalo;

  Exemplo:

  Forma incorreta: registrar 2 títulos para indicar um intervalo.

```
+-------------------------------------------------------+
| [  ]   EMISSÃO PEDIDOS PENDENTES DE APROVAÇÃO         |
|                                                       |
|             Período inicial: [          ]             |
|                                                       |
|               Período final: [          ]             |
|                                                       |
+-------------------------------------------------------+
```

  Forma correta: registrar apenas 1 título para indicar um intervalo e este título deverá representar a informação real a que se faz referência.

  No exemplo acima utilizou-se o termo “Período” que internamente é utilizado para referenciar a data de emissão dos pedidos. Neste caso deve-se utilizar o título como “Data emissão”.

  Tela principal

```
+-------------------------------------------------------+
| [  ]   EMISSÃO PEDIDOS PENDENTES DE APROVAÇÃO         |
|                                                       |
|     Data emissão: [          ] até [          ]       |
|                                                       |
+-------------------------------------------------------+
```

  Além de definir um único título de campo para determinação de intervalo de valores em tela, a definição de help também deverá ser única para o campo que determina a faixa inicial e final, ou seja, no help deverá ser descrito que tal informação trata-se do valor inicial e final de uma faixa de valores, conforme o descritivo que for mais conveniente para cada situação. Um exemplo de descrição para help do campo acima poderia ser da seguinte forma:

```
.101
Data emissão

Data inicial e final de emissão de pedidos.
Somente os pedidos emitidos dentro do período de datas informado serão
considerados no processamento.

NOTA:
Quando o período de datas não for informado, todos os pedidos,
independente de sua data de emissão, serão considerados no
processamento.
```

Exemplo de tela:

```
{
-----1-------------------------------------------------------------------------
2[a0]                           3TÍTULO TELA
                               4
           5Título primeiro campo: [a  ]

             Título segundo campo:6[b                                  ]

}
```

1 – telas que apresentam opção de menu na execução, devem ter a definição da linha tracejada na primeira linha da definição de tela;

2 – um espaço em branco antes da apresentação do campo que identifica o código da empresa (variável p_cod_empresa), quando este for definido na mesma linha do título da tela;

3 – título da tela centralizado, acentuado e todo escrito em letras maiúsculas;

4 – pelo menos uma linha em branco separando o título principal dos títulos dos campos;

5 – título do campo todo em letras minúsculas, com apenas a primeira letra da primeira palavra em maiúscula, com espacejamento simples, devidamente acentuado e finalizado por ":" (dois pontos);

6 – Alinhamento dos campos "a" e "b" pelo sinal de ":" (dois pontos) e existência de espaço simples após o sinal de ":" pelo fato da tela ter espaço disponível.

Observação:
Para os comandos reconhecidos somente pelo 4JS, no início da linha sempre deverá existir os caracteres '--#' que são assumidos pelo compilador 4GL como uma linha de comentário, mas para o 4JS é assumido como uma linha normal de comandos.

1.3.6. Screen Record

Para a definição dos registros de tela (SCREEN RECORD) que possuírem algum campo com "Zoom", deve-se acrescentar a opção do 4JS **`--#OPTIONS="-nolist"`** a todos os campos integrantes do SCREEN RECORD para que não ocorra deslocamento de campos da lista durante a execução no ambiente gráfico.

Exemplo: No caso abaixo, pelo fato do campo "f09" (cod_item) possuir "zoom" e fazer parte de uma lista (screen record), todos os demais campos pertencentes a lista (screen record) precisaram ter o atributo OPTIONS="-nolist".

```
ATTRIBUTES
a   = dev_mestre.cod_empresa, NOENTRY;
f01 = dev_mestre.num_nff, AUTONEXT;
f02 = dev_mestre.dat_lancamento, AUTONEXT;
f03 = dev_mestre.cod_cliente, UPSHIFT, AUTONEXT
--#, WIDGET="FIELD_BMP", CONFIG="combo.bmp Control-z"
;
f04 = FORMONLY.nom_cliente TYPE CHAR;
b   = dev_item.num_sequencia, NOENTRY
--#, OPTIONS="-nolist"
;
f09 = dev_item.cod_item, UPSHIFT, AUTONEXT
--#, WIDGET="FIELD_BMP", CONFIG="combo.bmp Control-z"
--#, OPTIONS="-nolist"
```

```
;
f10 = dev_item.qtd_item, AUTONEXT
--#, OPTIONS="-nolist"
;
f11 = dev_item.pre_unit, AUTONEXT
--#, OPTIONS="-nolist"
;

INSTRUCTIONS
SCREEN RECORD sr_dev_item[09] (dev_item.num_sequencia,
                               dev_item.cod_item,
                               dev_item.qtd_item,
                               dev_item.pre_unit);
END
```

A opção do 4JS **--#OPTIONS="-nolist"** também pode ser utilizada nas listas que não possuírem campos com zoom, pois pode ser necessário dar um visual estético melhor para as listas que permitem edição de dados pelo usuário.

1.4. DEFINIÇÃO DE VARIÁVEIS (PADRÃO LOGIX)

Existe um grupo de variáveis globais que não respeitam o padrão de nomenclatura de variáveis globais atuais por se tratarem de variáveis definidas quando este padrão ainda não tinha sido estabelecido, mas que são utilizadas praticamente em todo o sistema pelo fato de armazenarem informações globais do sistema.

As principais variáveis globais que devem ser implementadas na maior parte dos programas são:

p_cod_empresa
p_user
p_versao

A seguir está a lista com os nomes de variáveis globais e modulares padrões que devem ser definidas nos programas, quando da sua necessidade.

1.4.1. Variáveis Globais

```
GLOBALS
  DEFINE p_cod_empresa         LIKE empresa.cod_empresa,
         p_user                LIKE usuario.nom_usuario

  DEFINE p_versao              CHAR(018)

  DEFINE g_ies_ambiente        LIKE tipos_processo_v2.ies_ambiente,
         g_ies_grafico         SMALLINT

  DEFINE g_cod_impressora      LIKE impressoras.cod_impressora

  DEFINE p_nom_arquivo         CHAR(100),
         p_nom_arquivo_back    CHAR(100),
         g_caminho_browser     CHAR(100)

  DEFINE p_ies_impressao       CHAR(001)

  DEFINE g_tem_config_impress  SMALLINT,
         g_escolhe_tam_formul  SMALLINT,
```

```
         g_usa_visualizador    SMALLINT,
         g_usa_browser         SMALLINT,
         g_nao_usa_disco       SMALLINT,
         g_nao_usa_imp         SMALLINT

  DEFINE g_ies_imp_arquivo     CHAR(001)

  DEFINE g_apenas_impressao    SMALLINT

  DEFINE g_seq_configuracao    INTEGER

  DEFINE g_tipo_sgbd           CHAR(003)

  DEFINE g_borda_1             CHAR(01),
         g_borda_2             CHAR(01),
         g_borda_3             CHAR(01),
         g_borda_4             CHAR(01),
         g_borda_5             CHAR(01),
         g_borda_6             CHAR(01),
         g_borda_7             CHAR(01),
         g_borda_8             CHAR(01),
         g_borda_9             CHAR(01),
         g_borda_10            CHAR(01),
         g_borda_11            CHAR(01)
END GLOBALS
```

- **p_cod_empresa** LIKE empresa.cod_empresa

  p_user LIKE usuario.nom_usuario

Variáveis carregadas pela função *log001_acessa_usuário()* acionada no início de cada programa, responsável pela verificação do login do usuário atual (*p_user*), código da empresa atual ativa no menu LOGIX (*p_cod_empresa*) e permissão de acesso do usuário atual ao programa em questão (*p_status*), conforme cadastro de controles do menu LOGIX.

Existe ainda uma terceira variável de “status” que é retornada por esta função, sendo que na maioria dos programas está sendo utilizada como global a nomenclatura p_status, mas não deve ser definida como global, pois é utilizada apenas para usos esporádicos que não existe necessidade de ser utilizada em outros fontes “linkeditados”. Esta variável pode ser definida com qualquer outro nome no programa, podendo ser local ou modular.

Em resumo, os valores contidos nestas variáveis podem ser utilizados durante o desenvolvimento de programas para as seguintes situações:

- Sempre que for necessário acessar informações da empresa ativa do sistema, pode-se fazer uso do conteúdo da variável *p_cod_empresa*.

- Sempre que for necessário acessar o login do usuário conectado no sistema, pode-se fazer uso do conteúdo da variável *p_user*.

- Após a chamada da função *log001_acessa_usuário()*, o programa somente pode continuar o processamento caso a variável indicadora do *status* esteja igual a “0”, caso contrário indica que o usuário atual não possui autorização para execução do programa em questão.

- **p_versao** CHAR(18)

Obrigatória em todos os programas principais (fonte com extensão 4gl que contém o bloco de comando MAIN) que deverá ser inicializada sempre com o nome do programa atual, seguido do número de versão, número da release e número da alteração (Exemplo: “LOG5200-05.00.07”).

Esta variável é controlada automaticamente pelo sistema CSL (Controle de Sistemas Logocenter) no momento da transferência de programa (TRANPROG) e consistida no momento da liberação (LIBPROG).

- **g_ies_ambiente** LIKE tipos_processo_v2.ies_ambiente

Indica o ambiente operacional de execução do sistema. Inicializada automaticamente pela função log0010_acessa_usuário() que é processada no início de cada programa.

U - Unix/Linux

D - DOS

W - Windows

I - Internet

- **g_ies_grafico** SMALLINT

Indica se o sistema está em execução em ambiente gráfico ou não.

- 0 (Não. Execução em ambiente caracter)
- 1 (Sim. Execução em ambiente gráfico)

- **g_cod_impressora** LIKE impressoras.cod_impressora

Indica o código da impressora selecionada no momento da emissão de um relatório. Caso a emissão tenha sido enviada para disco, esta variável conterá o valor fixo “DISCO”. Esta variável é carregada automaticamente pela função log0280_saida_relat().

- **p_nom_arquivo** CHAR(100)

Indica o caminho de saída do relatório, sendo que:

- Quando o destino do relatório for DISCO, conterá o caminho de diretório LST e o nome de arquivo de saída informado para geração.
- Quando o destino do relatório for IMPRESSORA, conterá o comando de impressão utilizado para emitir na impressora selecionada, conforme comando cadastrado no MEN0210 (Cadastro de impressoras).

- **p_nom_arquivo_back** CHAR(100)

Indica o mesmo conteúdo da variável p_nom_arquivo, no entanto, com as seguintes diferenças:

- No lugar dos espaços em branco existirá o símbolo “@”;

- No lugar do símbolo “|” (pipe) existirá o símbolo “!” (exclamação).

Esta variável é utilizada somente quando existe necessidade de ser passada como argumento de execução, para um outro programa por meio do comando de execução de programas (RUN), e poder ser reconhecida como um único argumento pelo programa que recebe os argumentos, pois no caso de existir espaços em branco no conteúdo da variável p_nom_arquivo, o programa a ser executado irá reconhecer como 2 ou mais argumentos.

```
Exemplo: fglrun <programa> <parâmetro1> <parâmetro2>...<parâmetro N>
```

No início do programa executado, após realizar a leitura do comando de impressão enviado com o conteúdo da variável p_nom_arquivo_back utilizando a instrução ARG_VAL(), deverá ser feita a conversão dos símbolos “@” e “!” existentes em seu conteúdo para os seus respectivos valores originais por meio da execução da função *log028_converte_comando_impressao()* que irá converter o conteúdo da própria variável p_nom_arquivo_back, que também deverá ter sido definida como GLOBAL no programa executado.

- **p_ies_impressao** CHAR(01)

Indica o destino de emissão do relatório, carregado automaticamente pela função *log0280_saida_relat()*.

S – Impressora (Quando opção de geração de relatório for Impressora)

N – Disco (Quando opção de geração de relatório for Disco)

V – Vídeo (Quando opção de geração de relatório for Tela)

B – Browser (Quando opção de geração de relatório for WEB)

- **g_tem_config_impress** SMALLINT

Utilizada quando o programa usa função de impressão LOG5000 ao invés de utilizar LOG5211. Quando definida e inicializada como TRUE será solicitada confirmação do usuário “Grava configuração de impressão?”, de forma que se for confirmada o relatório será gerado em disco, mesmo que o usuário tenha optado pela opção “Impressora”, e neste arquivo irá conter os comandos de impressão reconhecidos pela impressora selecionada. **(Esta variável não é mais definida quando é utilizada função LOG5211 para impressão)**.

- **g_num_programa_impres** LIKE conf_imp_default.num_programa

Para que sejam registradas as configurações de impressão, informadas na função LOG5210 (Parâmetros de impressão) por programa/usuário, de forma que na próxima execução de um programa sejam apresentadas as configurações de impressão informadas pelo usuário na sua última execução, deve-se inicializar esta variável com o número do programa no início da sua execução, conforme abaixo.

```
LET g_num_programa_impres = “LOG5200”
```

Caso esta variável não seja inicializada com a numeração do programa em execução as configurações padrões de impressão, apresentadas ao usuário no momento da listagem de algum relatório, serão as configurações registradas para o sistema de impressão da impressora selecionada no cadastro LOG5200 (Comandos de impressão).

- **g_usa_visualizador** SMALLINT

Quando desejar que haverá opção para emitir o relatório em disco e apresentá-lo em tela ao final da sua emissão, deve-se ativar esta variável para TRUE antes de acionar a função log0280_saida_relat() para que, logo após o término de emissão do relatório (FINISH REPORT), possa ser acionada a função para edição em tela (pelo programa LOG0290) para edição de arquivos.

Para isso será necessário implementar o seguinte código logo após o término de emissão do relatório (FINISH REPORT):

```
IF p_ies_impressao = “V” THEN
   CALL log028_visualiza_arquivo(p_nom_arquivo)
END IF
```

Onde *p_nom_arquivo* é a variável que contém o caminho de diretório e nome do arquivo HTML a ser apresentado.

- **g_usa_browser** SMALLINT

Quando o sistema é executado em ambiente WEB ou via WINDOWS e está configurado para uso de browser para edição de relatórios no formato HTML (variável de ambiente USA_BROWSER=1), é possível acionar a visualização de um relatório gerado em disco diretamente no *browser*, sendo necessário apenas gerá-lo no formato e extensão HTML

Quando isto for possível, deverá ser feita ativação da variável global g_usa_browser no programa antes de acionar a função *log0280_saida_relat()* para que a opção “WEB” seja apresentada ao usuário. Ao final da emissão do relatório, após o comando FINISH REPORT, deve-se fazer a seguinte implementação para que seja possível acionar o *browser* para apresentação do relatório logo após o término de impressão do relatório:

```
IF p_ies_impressao = “B” THEN
   CALL log028_executa_browser(p_nom_arquivo)
END IF
```

Onde *p_nom_arquivo* é a variável global que contém o caminho de diretório e nome do arquivo HTML gerado.

- **g_nao_usa_disco** SMALLINT

Para que não seja permitida emissão de relatório em disco (opção DISCO da função LOG0280), deve-se ativar esta variável para TRUE antes de acionar a função *log0280_saida_relat()*, para que a opção “DISCO” não seja apresentada na sua execução.

- **g_nao_usa_imp** SMALLINT

Para que não seja permitida emissão de relatório em impressora (opção IMPRESSORA da função LOG0280), deve-se ativar esta variável para TRUE antes de acionar a função *log0280_saida_relat()*, para

que a opção “IMPRESSORA” não seja apresentada na sua execução.

- **g_seq_configuracao** INTEGER

Armazena uma seqüência automática gerada no momento da configuração de impressão pela tela LOG5210 (acionada a partir da função *log0280_saida_relat()*). Quando um programa principal aciona a chamada da função *log0280_saida_relat()* mas não realiza a emissão do relatório propriamente dita, pois isto é realizado por outro programa externo (background) que é o responsável pela impressão dos dados (REPORT), esta variável deve ser passada como argumento para o programa externo para que seja feita captura das configurações de impressão selecionadas pelo usuário.

- **g_nao_exclui_par** SMALLINT

Indicador utilizado para evitar que a configuração de impressão, selecionada pelo usuário na tela LOG5210 e armazenada na tabela TRAN_ARG_RELATORIO, seja excluída após a emissão da primeira página do relatório onde a função *log5211_retorna_configuracao()* for acionada. A ativação desta variável é necessária no caso de programas que tenham geração de múltiplos relatórios simultaneamente, com objetivo de que todos os relatórios respeitem a mesma configuração de impressão selecionada pelo usuário. Basta que esta variável seja inicializada antes do início da emissão do primeiro relatório pelo programa.

- **g_borda_1, g_borda_2, ..., g_borda_11** CHAR(1)

Variáveis utilizadas exclusivamente para formação das bordas gráficas em relatórios, que receberão os valores da tabela MEN_TIP_BORDA dos campos borda_1 a borda_11, respectivamente, caso a impressora suporte borda gráfica. Caso contrário receberão os valores para formação da borda caracter. Por meio do MEN0211 (Bordas para impressora) é efetuado o cadastro, indicando se a impressora suporta ou não borda gráfica (MEN0211 é acionado a partir do cadastro de impressoras MEN0210, opção “Borda”). Caso suporte deverá ser cadastrado no MEN0212 (acionado a partir da opção “conJunto caracter do MEN0211) o código da tabela ASCII correspondente à borda. O MEN0212 assume como padrão o grupo de caracteres 'PC-8', que possui os caracteres parcialmente gráficos necessários para formação da borda gráfica, mas quando forem necessárias outras bordas, estas poderão ser cadastradas.

- **g_tipo_sgbd** CHAR(03)

Sigla que indica o banco de dados utilizado. Esta variável é carregada pela chamada da função *log001_acessa_usuario()* acionada no início de cada programa, independentemente de ser programa “background” ou não. Esta variável poderá conter os seguintes valores:

IFX – Banco de dados Informix

ORA – Banco de dados Oracle

DB2 – Banco de dados DB2

MSV – Banco de dados SQL Server

1.4.2. Variáveis Modulares

```
DEFINE m_status                SMALLINT,
       m_den_empresa           LIKE empresa.den_empresa,
       m_comando               CHAR(150),
       m_caminho               CHAR(150)

DEFINE m_versao_funcao         CHAR(018)
```

- **m_status** SMALLINT

Utilizada para armazenar o retorno de status da função *log001_acessa_usuario()* acionada no início dos programas, mas que pode ser utilizada posteriormente em qualquer ponto do programa onde houver necessidade de utilizar para obtenção de algum valor booleano (TRUE/FALSE ou 1/0).

- **m_den_empresa** LIKE empresa.den_empresa

Utilizada para armazenar o nome da empresa que deve ser emitido na primeira linha do cabeçalho padrão de relatórios, ou seja, caso o programa possua rotina de listagem de relatório esta variável será definida, devendo ser carregada após a confirmação de impressão pela função *log0280_saida_relat()*.

- **m_comando, m_caminho** CHAR(150)

Utilizadas para armazenar um comando a ser executado (exemplo: chamada de um outro programa) ou então o caminho de diretório do sistema, conforme cadastro de caminhos de diretórios do sistema (LOG1100 - Controle Geral - Cadastros - Caminhos de Diretórios).

- **m_versao_funcao** CHAR(18)

Variável utilizada da mesma forma e com o mesmo objetivo da variável global p_versao, no entanto, esta é específica para definição em módulos de funções (fontes com extensão .4gl que não contenham o bloco MAIN..END MAIN implementado) que realizem algum tipo de interação com o usuário final (abertura de telas). Esta variável sempre deve ser definida para estes tipos de módulos de funções para identificar a versão, release e modificação atual que é apresentada por meio da execução da função *log006_exibe_teclas()*.

Exemplo:

```
DEFINE m_versao_funcao   CHAR(18) #Variável modular a ser definida
...
#Inicialização da variável m_versao_função no início do código da
#função principal, antes da abertura de tela
LET m_versao_funcao = 'LOG0090-04.10.03'
CALL log006_exibe_teclas("02 08 10 17 18", m_versao_funcao)
...
```

As demais variáveis modulares que forem definidas em um fonte 4gl deverão, no mínimo, respeitar uma nomenclatura condizente com o seu conteúdo para facilitar o entendimento do programa.

1.5. MENSAGENS PADRÕES

Todas as mensagens e erros apresentados para o usuário devem seguir as seguintes regras:

- Descrição clara e objetiva e com todas as palavras devidamente acentuadas;
- Deve ser finalizada por "." (ponto final);
- Escrita em letras minúsculas, sendo somente a primeira letra da frase em maiúscula, salvo as siglas que devem ser em letras maiúsculas ou palavras reservadas que podem vir a ser destacadas na mensagem com letras maiúsculas.
- Não acentuar palavras abreviadas;
- Não finalizar palavras abreviadas com "." (ponto final);
- Não utilizar "." (ponto final) na escrita de siglas;
- Para mensagens de confirmação apresentadas ao usuário, utilizando a função *log0040_confirm()*, deve ser utilizado verbo na 3ª pessoa e NÃO no infinitivo. Exemplo: Confirma, Inclui, Exclui.

A seguir algumas mensagens padrões que devem ser obedecidas:

- **Consistência de informação obrigatória em tela que foi deixada em branco.**

Exemplos:

```
ERRO: "Matrícula funcionário não preenchida."
ERRO: "Item não preenchido."
```

→ Utilizar função log0030_mensagem(<msg>,"exclamation")

- **Consistência de um intervalo/faixa de dados.**

Exemplos:

```
ERRO: "Data final anterior a data inicial."
ERRO: "Data encerramento anterior a data de abertura."
ERRO: "Código de item final maior que o inicial."
```

→ Utilizar função log0030_mensagem(<msg>,"exclamation")

- **Consistência de informação inexistente no banco de dados.**

Exemplos:

```
ERRO: "Funcionário 99 não cadastrado."
ERRO: "Empresa XX não cadastrada."
```

→ Utilizar função log0030_mensagem(<msg>,"exclamation")

- **Consistência de informação já existente no banco de dados.**

Exemplos:

```
ERRO: “Funcionário 99 já cadastrado.”
ERRO: “Empresa XX já cadastrada.”
```

→ Utilizar função log0030_mensagem(<msg>,”exclamation”)

- **Consistência de informação já informada em tela (Informação em duplicidade).**

Exemplos:

```
ERRO: “Funcionário já informado.”
ERRO: “Empresa já informada.”
```

→ Utilizar função log0030_mensagem(<msg>,”exclamation”)

- **Inclusão, modificação e exclusão de registro:**

Exemplos:

```
MENSAGEM: “Inclusão efetuada com sucesso.”
MENSAGEM: “Exclusão efetuada com sucesso.”
MENSAGEM: “Modificação efetuada com sucesso.”
```

→ Utilizar comando MESSAGE <msg> ATTRIBUTE(REVERSE)

```
ERRO: “Inclusão cancelada.”
ERRO: “Exclusão cancelada.”
ERRO: “Modificação cancelada.”
```

→ Utilizar comando ERROR.

- **Consulta:**

Antes de realizar a consulta do primeiro registro emitir a seguinte mensagem:

```
MENSAGEM: “Processando a consulta...”
```

→ Utilizar comando MESSAGE <msg> ATTRIBUTE(REVERSE)

Após realizar a pesquisa do primeiro registro, se não encontrar registro emitir:

```
ERRO: “Argumentos de pesquisa não encontrados.”
```

→ Utilizar função log0030_mensagem(<msg>,”exclamation”)

Caso a consulta tenha sido cancelada pelo usuário, deverá emitir:

```
ERRO: “Consulta cancelada.”
```

→ Utilizar comando ERROR.

Após realizar a pesquisa do primeiro registro, se encontrar registro emitir:

```
MENSAGEM: “Consulta efetuada com sucesso.”
```

→ Utilizar comando MESSAGE <msg> ATTRIBUTE(REVERSE)

- **Paginação:**

Caso não tenha encontrado mais registros no momento da pesquisa do registro seguinte ou anterior deve-se emitir:

`ERRO: "Não existem mais dados nesta direção."`

→ Utilizar comando ERROR.

Caso não tenha sido realizada nenhuma consulta anteriormente emitir:

`ERRO: "Não existe nenhuma consulta ativa."`

→ Utilizar função log0030_mensagem(<msg>,"exclamation")

- **Relatório:**

Quando a opção "Informar" do relatório é cancelado:

`ERRO: "Ação cancelada."`

→ Utilizar comando ERROR.

Quando processamento do relatório é cancelado a partir da tela LOG0280:

`ERRO: "Processamento cancelado."`

→ Utilizar comando ERROR.

Quando iniciar o processamento do relatório:

`MENSAGEM: "Processando a extração do relatório..."`

→ Utilizar comando MESSAGE <msg> ATTRIBUTE(REVERSE)

Quando concluída emissão do relatório para IMPRESSORA:

`MENSAGEM: "Relatório impresso com sucesso."`

→ Utilizar função log0030_mensagem(<msg>,"info")

Quando concluída emissão do relatório para DISCO:

`MENSAGEM: "Relatório gerado em <arquivo de saída>."`

→ Utilizar função log0030_mensagem(<msg>,"info")

Quando nenhum dado for emitido no relatório:

ERRO: “Não existem dados para serem listados.“

→ Utilizar função log0030_mensagem(<msg>,”exclamation”)

- **Processamento de rotinas:**

```
MENSAGEM “Processando...”
MENSAGEM “Processando <título rotina em andamento>...”
MENSAGEM “Processando <informação em andamento>...”
```

→ Utilizar comando MESSAGE <msg> ATTRIBUTE(REVERSE)

MENSAGEM: “Processamento concluído com sucesso.”

→ Utilizar função log0030_mensagem(<msg>,”info”)

ERRO: “Processamento cancelado.”

→ Utilizar comando ERROR.

- **Confirmações:**

PERGUNTA: “Confirma exclusão do material?”

→ Utilizar função log0040_confirm(<linha>, <coluna>, <pergunta>).

1.6. INSTRUÇÕES SQL

Para toda instrução SQL (SELECT, INSERT, DELETE, UPDATE, LOAD, UNLOAD) implementada em um fonte, seja este de forma direta, via CURSOR ou SQL preparado, deve-se atentar para o seguinte:

- Todas as instruções de acesso ao banco de dados, seja via “PREPARE“, “DECLARE <cursor>“, “OPEN <cursor>“, “FETCH <cursor>“, “EXECUTE”, “LOAD“, “UNLOAD“, “INSERT“, “DELETE“, “UPDATE“, “SELECT“, “CREATE“ e “DROP“ devem ser precedidos da cláusula WHENEVER ERROR CONTINUE e sucedidos pela cláusula WHENEVER ERROR STOP. Isto se deve ao fato de evitar que a aplicação seja interrompida indevidamente em caso de problema de execução da instrução no banco de dados.

- Após cada instrução de acesso ao banco de dados deve-se realizar o teste da variável SQLCA.SQLCODE (SELECT, INSERT, DELETE, UPDATE, CREATE, DROP, LOAD, UNLOAD, COMMIT WORK). Isto também é válido para os comandos “PREPARE “,“DECLARE <cursor>“ “OPEN <cursor>“, “FETCH <cursor>“, “FOREACH <cursor>“ e “EXECUTE”, para que possa ser feita a apresentação do respectivo erro ao usuário de forma clara.

Exemplos:

Instrução PREPARE

```
WHENEVER ERROR CONTINUE
PREPARE var_empresa FOR sql_stmt
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   <tratamento de erro>
END IF
```

Instrução DECLARE <cursor>

```
WHENEVER ERROR CONTINUE
DECLARE cq_empresa FOR var_empresa
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   <tratamento de erro>
END IF
```

Instrução OPEN <cursor>

```
WHENEVER ERROR CONTINUE
OPEN cq_empresa
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   <tratamento de erro>
END IF
```

Instrução FETCH <cursor>

```
WHENEVER ERROR CONTINUE
FETCH cq_empresa INTO ...
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 AND sqlca.sqlcode <> NOTFOUND THEN
   <tratamento de erro>
END IF
```

Instrução FOREACH <cursor>

```
WHENEVER ERROR CONTINUE
FOREACH cq_empresa INTO …
   IF sqlca.sqlcode <> 0 THEN
      <tratamento de erro>
   END IF
   ...
   <instruções>
   ...
   WHENEVER ERROR CONTINUE #Ativar WHENEVER antes do próximo LOOP
                          #caso exista alguma instrução anterior
                          #executada internamente no bloco FOREACH
                          #tenha ocasionado a desativação da
                          #diretiva WHENEVER ERROR
END FOREACH
WHENEVER ERROR STOP
FREE cq_empresa
```

- Os seguintes erros de banco de dados (SQLCA.SQLCODE) devem ser tratados internamente nos fontes, de forma fixa, para prever situações específicas, permitindo emitir uma mensagem coerente ao usuário:

  Ação de inclusão de registro (INSERT) para emitir mensagem de registro já cadastrado:

  Erro –239 (Valor duplicado para um índice único definido para tabela)
  Erro –268 (Valor duplicado para chave primária definida para tabela)

  ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior deverá ser utilizada a função

log0030_err_sql_registro_duplicado().

Exemplo para versão 4.10 ou inferior do sistema:

```
WHENEVER ERROR CONTINUE
INSERT INTO <tabela>
       (coluna1,coluna2,...,colunaN)
VALUES (valor1,valor2,...,valorN)
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   IF sqlca.sqlcode = -239 OR sqlca.sqlcode = -268 THEN
      <mensagem de registro duplicado>
   ELSE
      <tratamento de erro>
   END IF
END IF
```

Exemplo para versão 5.00 ou superior do sistema:

```
WHENEVER ERROR CONTINUE
INSERT INTO <tabela>
       (coluna1,coluna2,...,colunaN)
VALUES (valor1,valor2,...,valorN)
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   IF log0030_err_sql_registro_duplicado() THEN
      <mensagem de registro duplicado>
   ELSE
      <tratamento de erro>
   END IF
END IF
```

Ação de criação de tabela (CREATE TABLE/CREATE TEMP TABLE) para evitar erro de tabela duplicada quando esta já existir:

Erro –310 (Tabela física já existente no banco de dados)
Erro –958 (Tabela temporária já existente.)

ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior deverá ser utilizada a função **log0030_err_sql_tabela_duplicada()**.

Exemplo para versão 4.10 ou inferior do sistema:

```
WHENEVER ERROR CONTINUE
CREATE TEMP TABLE <tabela_temp>
       (coluna1   <tipo> NOT NULL,
        coluna2   <tipo>,
        ...,
        colunaN   <tipo>)
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   IF sqlca.sqlcode = -310 OR sqlca.sqlcode = -958 THEN
      #Eliminar todos registros da tabela temporária
      WHENEVER ERROR CONTINUE
      DELETE FROM <tabela_temp>
      WHENEVER ERROR STOP
```

```
            IF sqlca.sqlcode <> 0 THEN
               <tratamento de erro>
            END IF
         ELSE
            <tratamento de erro>
         END IF
      END IF
```

Observação: O teste de tabela existente deverá ser testado contra os valores –310 e –958, pois no banco de dados Oracle não existe tabela temporária, e esta é considerada como tabela física no banco, retornando o código -310.

Exemplo para versão 5.00 ou superior do sistema:

```
WHENEVER ERROR CONTINUE
CREATE TEMP TABLE <tabela_temp>
       (coluna1   <tipo> NOT NULL,
        coluna2   <tipo>,
        ...,
        colunaN   <tipo>)
WHENEVER ERROR STOP
IF sqlca.sqlcode <> 0 THEN
   IF log0030_err_sql_tabela_duplicada() THEN
      #Eliminar todos registros da tabela temporária
      WHENEVER ERROR CONTINUE
      DELETE FROM <tabela_temp>
      WHENEVER ERROR STOP
      IF sqlca.sqlcode <> 0 THEN
         <tratamento de erro>
      END IF
   ELSE
      <tratamento de erro>
   END IF
END IF
```

Ação de leitura de registro (SELECT) para quando existir uma situação proposital em que se queira realizar a verificação de 1 ou mais registros:

Erro –284 (Foi encontrado além de um registro para os filtros informados)

Acesso a uma tabela inexistente no banco de dados que queira consistir de forma a ignorar erro durante processamento:

Erro –206 (Tabela inexistente no banco de dados)

ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior deverá ser utilizada a função **log0030_err_sql_tabela_inexistente()**.

Acesso a uma coluna de tabela inexistente no banco de dados que queira consistir de forma a tratar de forma diferenciada durante processamento:

Erro –217 (Coluna de tabela inexistente no banco de dados)

ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior, deverá ser utilizada a função **log0030_err_sql_coluna_inexistente()**.

Tratamento de erro ocasionado pela diferença de “layout” de tabela no banco de dados (número de colunas da instrução INSERT está diferente do número de colunas reais da tabela no banco de dados) e há necessidade de consistência:

Erro –236 (Número de colunas da tabela no banco de dados está diferente do esperado)

ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior deverá ser utilizada a função **log0030_err_sql_layout_tabela_diferente()**.

Atualização de uma coluna de tabela no banco de dados com valor nulo (NULL) sendo que está definida como obrigatória (NOT NULL) e há necessidade de consistência:

Erro –391 (Coluna de tabela inexistente no banco de dados)

ATENÇÃO: Para versão 5.00 do sistema LOGIX ou superior deverá ser utilizada a função **log0030_err_sql_coluna_nula()**.

IMPORTANTE: É proibida a utilização da cláusula * (asterisco) em qualquer instrução SQL (SELECT, INSERT, UPDATE), ficando tais instruções da seguinte forma:

```
SELECT <coluna1,
        coluna2,
        ...,
        colunaN>
  INTO <variável1,
        variável2,
        ...,
        variávelN>
  FROM <tabela>
 WHERE <condição>
```

OU

```
SELECT <coluna1,
        coluna2,
        ...,
        colunaN>
  INTO <variável_record.coluna1,
        variável_record.coluna2,
        ...,
        variável_record.colunaN>
  FROM <tabela>
 WHERE <condição>
```

```
INSERT INTO <tabela>
       (coluna1,
        coluna2,
        ...,
        colunaN)
VALUES (valor1,
        valor2,
        ...,
        valorN)
```

```
UPDATE <tabela>
   SET <coluna1> = <valor1>,
       <coluna2> = <valor2>,
       ...,
       <colunaN> = <valorN>
 WHERE <condição>
```

Pode-se utilizar variáveis do tipo RECORD definidas com a cláusula RECORD LIKE, mas estas não poderão ser utilizadas nas instruções SQL com a cláusula “*”, pois deverá ser feita referência campo a campo da variável do tipo RECORD correspondente.

1.7. OPÇÕES DE MENU

O texto de uma opção de menu (COMMAND “texto”) não pode exceder 14 caracteres. Ultrapassando este limite, o texto será apresentado de forma incompleta quando exibido em botões da versão gráfica (4JS), na área de teclas de atalho (Hot Key Area). Veja item **6.7 NOMEAR HOT KEYS**

A letra da tecla de atalho da opção de menu deverá sempre ser escrita em maiúsculo e não pode se repetir, permitindo utilizar qualquer outra letra, sendo codificada da seguinte forma:

```
COMMAND KEY(“U”) “U-opção”
COMMAND KEY(“P”) “oPção”
```

No caso da tecla de atalho coincidir com uma letra acentuada (cedilha, acento agudo, acento circunflexo, outros), deve-se definir a tecla de atalho com a respectiva letra não acentuada no início da

descrição da opção de menu, seguida de um “-“ (hífen), utilizando a opção KEY(“<letra atalho>”) conforme a seguir:

```
COMMAND KEY(“O”) “O-cópia”
```

O texto de uma opção de menu, quando definido com mais de uma palavra estas devem ser separadas com o caracter “_” (underline), conforme o exemplo abaixo:

```
COMMAND KEY(“P”) “inf_comPl”
```

Isto se faz necessário devido a execução do sistema no ambiente gráfico, que possui uma lista fixa de opções de menu que fazem referência a uma figura do tipo BMP apresentada em forma de ícone. Esta lista é pré-definida no arquivo de configuração (fglprofile) do 4JS. Caso existam 2 ou mais opções de menu em um fonte onde a primeira palavra do texto coincida com uma palavra já reservada para alguma figura BMP no arquivo de configuração do 4JS, e a segunda palavra do texto da opção de menu esteja separada com um espaço em branco, na execução será apresentado um único ícone para as 2 ou mais opções de menu. A solução para este caso pode ser implementada de 2 formas:

Opções de menu:

```
COMMAND “listar Ordem”
COMMAND “listar Pedido”
```

- Descrever o texto das 2 opções de menu com as palavras separadas pelo caracter “_” (underline). Exemplo:

```
COMMAND “listar_Ordem”
COMMAND “listar_Pedido”
```

- **MELHOR SOLUÇÃO**: Criar uma única opção de menu com a palavra inicial que possui uma figura vinculada no arquivo de configuração do 4JS e para esta opção apresentar um outro menu contendo a lista de opções válidas. Exemplo:

```
COMMAND “Listar”
   MENU “Listar”
   COMMAND “Ordem”
      ...
   COMMAND “Pedido”
      ...
   END MENU
```

A descrição do HELP que é informada ao lado da opção de menu deve sempre ser iniciada com letra maiúscula e o restante em letras minúsculas, terminando a descrição com “.” (ponto). Esta descrição deve respeitar também as acentuações e utilização de siglas com letras maiúsculas.

Exemplo:

```
MENU “Opção”
COMMAND “Incluir” “Inclusão de novo fornecedor.”
COMMAND “Consultar” “Consulta de fornecedores cadastrados.”
COMMAND “O-Copiar” “Cópia de informações de um fornecedor para outro.”
END MENU
```

Observação: Quando não for informada a opção “KEY” para determinar a tecla de atalho, a tecla de atalho padrão assumida automaticamente pela linguagem é a primeira letra da opção.

1.7.1. Programas de consulta em tela

Os programas desenvolvidos com o objetivo principal de consulta de informações em tela deverão ter as seguintes opções no menu, respeitando também a ordem em que estão descritas:

Consultar "Consulta informações de ..."

Quando o programa possuir opção de filtro de informações de consulta em tela, para esta opção deverá ser utilizado, sempre que possível, o comando CONSTRUCT, visto que este comando permite várias formas de pesquisa.

Listar "Emite relatório dos dados apresentados na última consulta realizada."

* * * OPÇÃO NÃO OBRIGATÓRIA * * *

Esta opção, quando necessária em programas de consulta em tela, deverá gerar um relatório com as informações obtidas a partir da última consulta realizada pela opção "Consultar", ou seja, para realizar a opção "Listar" em um programa de consulta em tela, é pré-requisito a execução da opção "Consultar".

Fim "Retorna ao menu anterior."

KEY ("!") (permitir digitação de comandos a partir do sistema operacional)

Observação: As opções de menu com outras funcionalidades poderão ser incluídas sem problemas nos programas de consulta, não esquecendo da definição única da tecla de atalho e as devidas acentuações das palavras.

1.7.2. Programas de manutenção/cadastro

Os programas de manutenção e cadastro deverão ter as seguintes opções no menu, respeitando também a ordem em que estão descritas:

Incluir "Inclusão de novo(a) <título tabela>."

Consultar "Consulta de <título tabela> cadastrados(as)."

Para esta opção deverá ser utilizado, sempre que possível, a instrução CONSTRUCT, visto que permite várias formas de pesquisa.

Modificar "Modificação de informações de <título tabela> já cadastrado(a)."

Esta opção tem como pré-requisito a consulta de um registro cadastrado.

Excluir "Exclusão de um(a) <título tabela> cadastrado(a)."

Esta opção tem como pré-requisito a consulta de um registro cadastrado.

Primeiro "Exibe o(a) primeiro(a) <título tabela> encontrado(a) na consulta."

Esta opção tem como pré-requisito a consulta de registros cadastrados.

Anterior "Exibe o(a) <título tabela> anterior encontrado(a) na consulta."

Esta opção tem como pré-requisito a consulta de registros cadastrados.

Seguinte "Exibe o(a) próximo(a) <título tabela> encontrado(a) na consulta."

Esta opção tem como pré-requisito a consulta de registros cadastrados.

Último "Exibe o(a) último(a) <título tabela> encontrado(a) na consulta."

Esta opção tem como pré-requisito a consulta de registros cadastrados.

Listar "Emissão de relatório dos(as) <título tabela> cadastrados(as)."

Esta opção deverá, sempre que possível, permitir realizar um filtro dos registros que o usuário deseja listar, utilizando o comando CONSTRUCT que também é utilizado para a opção "Consultar", pois desta forma o usuário poderá informar se deseja a emissão de todos os registros da tabela ou não.

Fim "Retorna ao menu anterior."

KEY ("!") (permitir digitar comandos a partir do sistema operacional)

Observação: Opções com outras funcionalidades poderão ser incluídas sem problemas nos programas de manutenção, não esquecendo da definição única da tecla de atalho e as devidas acentuações das palavras.

1.7.3. Programas para emissão de relatório

Os programas desenvolvidos com o objetivo principal de emitir um relatório deverão ter as seguintes opções no menu, respeitando também a ordem em que estão descritas:

Informar "Parâmetros a serem considerados na emissão do relatório."

Esta opção somente será obrigatória caso exista uma entrada de dados para informar parâmetros para o processamento do relatório. Ao término da entrada de dados dos parâmetros, caso o usuário tenha confirmado os parâmetros pressionando <ESC>, o cursor deverá ser posicionado automaticamente na opção "Listar".

Listar "Emite o relatório conforme parâmetros informados."

Esta opção tem como pré-requisito a informação de parâmetros, quando definidos.

Caso os parâmetros possam ser informados todos em branco, deve-se permitir acionar a opção "Listar" sem necessidade de acionar a opção "Informar".

Fim "Retorna ao menu anterior."

KEY ("!") (permitir digitar comandos a partir do sistema operacional)

Observação: Opções com outras funcionalidades poderão ser incluídas sem problemas nos programas de consulta, não esquecendo da definição única da tecla de atalho e as devidas acentuações das palavras.

1.7.4. Programas de processamento de rotina

Os programas desenvolvidos com o objetivo principal de processamento de alguma rotina de atualização/geração de dados ou rotina mensal deverão ter as seguintes opções no menu, respeitando também a ordem em que estão descritas:

Informar "Parâmetros para processamento do(a) <nome da rotina/processamento>."

Esta opção somente será obrigatória caso exista uma entrada de dados para informar parâmetros para o processamento da rotina. Ao término da entrada de dados dos parâmetros, caso o usuário tenha confirmado os parâmetros pressionando <ESC>, o cursor deverá ser posicionado automaticamente na opção "Processar".

Processar "Inicia o(a) <nome da rotina/processamento> conforme parâmetros informados."

Esta opção tem como pré-requisito a informação dos parâmetros, quando definidos.

Caso os parâmetros possam ser informados todos em branco, deve-se permitir acionar a opção "Processar" sem necessidade de acionar a opção "Informar".

Fim

KEY ("!") (permitir digitar comandos a partir do sistema operacional)

Observação: Opções com outras funcionalidades poderão ser incluídas sem problemas nos programas de consulta, não esquecendo da definição única da tecla de atalho e as devidas acentuações das palavras.

2. Função "MAIN"

A função MAIN é definida para todos os fontes considerados como "programa" ou "módulo principal". Esta função é responsável pelo início da execução do programa e o código fonte padrão a ser implementado nesta função é conforme demonstra o exemplo abaixo:

```
DEFINE m_status SMALLINT

MAIN
  LET p_versao = "sssnnnnn-vv.rr.mm"

  CALL log0180_conecta_usuario() #Para programas a partir da versão 05.00

  CALL log1400_isolation()
  WHENEVER ERROR CONTINUE
  SET LOCK MODE TO WAIT #Apenas definido para programas que possuem
  WHENEVER ERROR STOP   #atualização de informações no banco de dados

  DEFER INTERRUPT #Inibe a ação padrão da tecla <CTRL+C> que é
                  #abortar a execução de um programa em execução.

  CALL log001_acessa_usuario("NOME_SISTEMA","PRODUTOS")
       RETURNING m_status,p_cod_empresa,p_user

  IF m_status = 0 THEN
     #Ativar as teclas de função necessárias
     OPTIONS PREVIOUS KEY control-b,  #Para posicionar na página anterior
             NEXT     KEY control-f,  #Para posicionar na próxima página
             INSERT   KEY control-i,  #Para incluir nova linha (ARRAY)
             DELETE   KEY control-e   #Para excluir uma linha (ARRAY)
```

```
     CALL sssnnn_controle()
  END IF
END MAIN
```

Observações:

Em todos os programas deve-se fazer a definição da variável p_versao, para que estes possam ser controlados pelo sistema CSL.. Esta variável é carregada por meio da função *log001_acessa_usuario()*.
A cláusula OPTIONS deve ser definida no início de todos os fontes que tenham algum tipo de interação com o usuário final. Esta cláusula é utilizada para definição de algumas teclas de atalho e para ativar o arquivo de help do programa em execução.

3. Funções

A definição de todas as funções (FUNCTION/REPORT) dentro de um código fonte deve seguir um padrão de nomenclatura, conforme a seguir.

Existem muitas funções com ações comuns que já estão disponibilizadas sob forma de módulos de função (fonte com extensão “4gl”), que podem ser reutilizadas sob forma de link com o fonte principal do programa. Estas funções são citadas neste documento no item **5. Funções padrões LOGIX.**

3.1. NOMENCLATURA

O nome de uma função (FUNCTION ou REPORT) deverá sempre ter como prefixo o número completo do programa.

```
<numero_programa>_<nome_função>
```

Exemplo:

```
Programa                = man0310.4gl
Função controle()       = man0310_controle()
Função entrada_dados()  = man0310_entrada_dados()
```

A linha de código que contém a definição da função deverá sempre ser precedida e sucedida por uma linha tracejada conforme abaixo para dar mais destaque a lista de funções existentes em um fonte:

```
#-------------------------------#
 FUNCTION man0310_entrada_dados()
#-------------------------------#
```

O nome dado a cada função no fonte deve refletir exatamente o que ela processa, de forma clara e objetiva, para facilitar a interpretação por todos os programadores.

3.2. FORMATO DE EDIÇÃO DO CÓDIGO

Todos os comandos do INFORMIX-4GL deverão estar em letras maiúsculas e os nomes de variáveis de programa em letras minúsculas.

A identação padrão a ser respeitada entre as estruturas de comandos deve ser de 3 posições, com exceção da primeira tabulação dentro de uma função que é de 2 posições.

Exemplo:

```
MAIN
12CALL log001_acessa_usuario(...)
```

```
        RETURNING m_status, p_cod_empresa, p_user

IF m_status = 0 THEN
   CALL sssnnnn_controle()
END IF
END MAIN

#------------------------------#
FUNCTION sssnnnn_nome_funcao()
#------------------------------#
DEFINE ...

WHILE...
   IF .... THEN
      ...
   ELSE
      ...
   END IF
END WHILE
END FUNCTION
```

3.3. ABERTURA DE TELA

Para abertura de uma tela (fonte com extensão “per”), não se deve referenciar o número do fonte definido para tela diretamente no comando OPEN WINDOW, pois se deve respeitar o caminho de diretório onde o fonte compilado da tela está armazenado, que pode não ser o diretório corrente. Para isso deve-se utilizar a função *log130_procura_caminho() ou log1300_procura_caminho()*, passando como parâmetro o número do fonte da tela desejada, sem especificar a extensão do fonte. Esta função tem como objetivo retornar o caminho de diretório completo onde está localizado o fonte compilado da tela, conforme o caminho de diretório cadastrado para a empresa ativa no sistema via LOG1100, seguido do nome do fonte compilado da tela (executável).

Exemplo:

```
LET m_caminho = log1300_procura_caminho (fonte_tela_padrão,
                                         {fonte_tela_gráfica|""})
OPEN WINDOW w_sssnnnnq AT 2,2 WITH FORM m_caminho
     ATTRIBUTE(BORDER, MESSAGE LINE LAST, PROMPT LINE LAST)
```

Onde:

Fonte_tela_padrão - Código da tela padrão para quando se tratar de uma execução em ambiente caracter ou quando a definição desta tela atender ambos os ambientes, tanto caracter quanto gráfico.

Fonte_tela_gráfica - Código da tela gráfica para quando houver uma tela específica para este ambiente, ou “” (NULL), onde passa a utilizar o código da tela padrão.

Observação:
O nome do fonte para definição da tela (arquivo com extensão “per”) obedece o padrão “sssnnnnq”, conforme definido em **Padrões de Identificação de Programas Geral (pelo portal Logocenter pode ser acessado em “ Documentação - Metodologia Desenvolvimento”)**.

3.4. CHAMADAS DE PROGRAMAS EXTERNOS

Para execução de um programa a partir de outro programa é necessário saber o comando utilizado para execução de programas, pois este pode mudar de acordo com a ferramenta utilizada (fglgo/fglrun/fglrungen). É preciso também saber o caminho de diretório onde o arquivo executável do programa se encontra. Para isso deve-se utilizar a função *log120_procura_caminho()* que irá retornar o comando completo para execução do programa que é repassado como parâmetro para a função. Esta função irá retornar um texto contendo a linha de comando para execução do programa (comando de

execução, seguido do caminho completo de diretório e nome do programa a executar), sendo necessário somente executar este texto retornado por meio da instrução RUN.

ATENÇÃO: A partir da versão 5.00 do sistema deve-se fazer o uso da função **log1200_executa_programa()**

Exemplo para versões inferiores a versão 5.00 do sistema:

```
DEFINE l_cancel SMALLINT

LET m_comando = log120_procura_caminho("sssnnnn")
RUN m_comando RETURNING l_cancel
LET l_cancel = l_cancel / 256
IF l_cancel <> 0 THEN
   PROMPT "Tecle <ENTER> para continuar..." FOR CHAR m_resposta
END IF
```

Exemplo para versão 5.00 ou superior do sistema:

```
CALL log1200_executa_programa("sssnnnn","<parâmetros>")
```

3.5. RÉGUA DE TECLAS DE ATALHO

Todas as telas precisam apresentar a régua de teclas de atalho padrão utilizando a função *log006_exibe_teclas*. A apresentação desta régua das teclas de atalho é apresentada no rodapé da tela, quando da execução do programa no ambiente caracter. Nesta régua são apresentadas uma lista de teclas de atalho, seguida do número e versão atual do programa em execução. Na execução do programa em ambiente gráfico (4JS), as teclas de atalho são disponibilizadas em forma de botão ou ícone e o número e versão do programa são apresentados como título da janela do windows.

Exemplo:

```
CALL log006_exibe_teclas("tt tt tt",p_versao)
```

Onde **tt** é o código da tecla de atalho que deverá ser apresentada. Para apresentar mais de uma tecla de atalho basta separar os códigos com um espaço em branco.

Os códigos padrões disponíveis de teclas de atalho são:

01 - Ctrl-W=Help
02 - Ctrl-C=Fim
03 - Ctrl-Z=Zoom
05 - Ctrl-I=Incl
06 - Ctrl-E=Excl
07 - Esc=Efetiva
08 - Esc=Seleciona
09 - Enter=Processa
17 - Ctrl_F=Página Seguinte (array)
18 - Ctrl-B=Página Anterior (array)
24 - Ctrl-O=Copia

Observações:

- Existem outros códigos disponíveis para teclas de atalho na função LOG0060, mas estas são consideradas específicas de acordo com o tipo de processamento que o programa deve realizar.

- Sempre após a chamada da função *log006_exibe_teclas()* deve ser processado o comando **CURRENT WINDOW IS** para indicar o foco atual da execução do programa, pois esta função faz abertura de uma outra janela no rodapé da tela onde é feita a apresentação da barra status do programa na execução em ambiente caracter. O comando **CURRENT WINDOW IS** somente poderá ser dispensado, quando logo após a execução

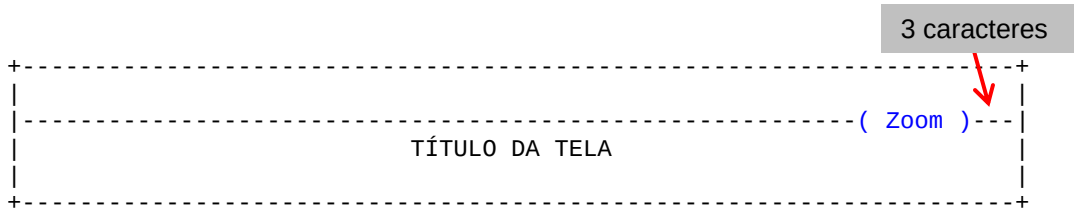
da função *log006_exibe_teclas()* for realizada a abertura de uma janela utilizando comando **OPEN WINDOW**, pois neste caso o foco de execução já é automaticamente atualizado.

3.6. FUNÇÃO ZOOM

- As aplicações que realizam algum tipo de entrada de dados e em algum momento algum campo de tela faz referência a informações provenientes de outra tabela (chave estrangeira), ou então possui uma lista de opções válidas (domínios), deverão possuir uma função de “zoom”, também chamada como “popup”, para permitir visualizar a lista de dados válidos que o usuário poderá informar como conteúdo do campo em questão;

- As mesmas opções de zoom disponíveis ao usuário no momento da inclusão e modificação **obrigatoriamente** devem estar disponíveis para acesso na função de consulta;

- Quando o cursor estiver posicionado sobre um campo que possui a opção de “zoom”, o texto fixo “(Zoom)” deverá ser apresentado numa posição fixa em relação a tela corrente. Esta posição fica a 3 (três) caracteres a esquerda da borda direita da tela, sobre a linha que delimita as opções de menu e as demais informações da tela. No entanto, quando o cursor for posicionado em um campo que não possui opção de “zoom”, este texto fixo “(Zoom)” deverá ser subtituído pelo texto “--------” (tracejado) de forma que não seja mais visualizado em tela, mas o texto tracejado somente é válido para exibição sobre a linha tracejada apresentada abaixo do menu. Podem existir situações em que não exista esta linha tracejada, sendo que neste caso deverá ser apresentado um texto que dê a impressão ao usuário de que o texto “(Zoom)” seja eliminado da tela. **Mas isto é válido somente para quando o sistema estiver sendo executado em ambiente caracter (não gráfico)**, conforme exemplo abaixo:



- Na execução do sistema em ambiente gráfico, para os campos com opção de zoom, deve-se definir o botão de zoom na definição do campo da tela (arquivo com extensão PER), quando houver espaço disponível, para que seja apresentado no momento da execução;

```
a = formonly.cod_item, UPSHIFT, AUTONEXT
--#, WIDGET=“FIELD_BMP”, CONFIG=“combo.bmp Control-z”
;
```

- Deve-se utilizar a função **fgl_keysetlabel()** para ativar o botão de ‘Zoom’ fora do escopo dos comandos de entrada de dados (INPUT, INPUT ARRAY, DISPLAY ARRAY, CONSTRUCT) utilizando a função *log006_exibe_teclas()* e dentro do escopo dos comandos de entrada de dados deve-se utilizar a função **fgl_dialog_setkeylabel()** para ativar e desativar o botão de zoom.

No campo onde existir ‘Zoom’ deve-se codificar da seguinte forma dentro do escopo da instrução utilizada para entrada de dados:

```
BEFORE FIELD <campo>
   CALL sssnnnn_ativa_zoom(TRUE)

AFTER FIELD <campo>
   CALL sssnnnn_ativa_zoom(FALSE)

AFTER INPUT
   CALL sssnnnn_ativa_zoom(FALSE) #Desabilitar zoom na conclusão da
                                  #entrada de dados pelo usuário
```

```
   IF int_flag = 0 THEN
      ...<consistências da entrada de dados>...
   END IF
...
#-----------------------------------#
 FUNCTION sssnnnn_ativa_zoom(l_ativa)
#-----------------------------------#
  DEFINE l_ativa SMALLINT

  IF l_ativa THEN
     IF g_ies_grafico THEN
        #Ativação do botão de zoom no ambiente gráfico
        --# CALL fgl_dialog_setkeylabel('control-z','Zoom')
     ELSE
        #Apresentação fixa no canto superior direito da tela
        DISPLAY "( Zoom )" AT <linha>,<coluna>
     END IF
  ELSE
     IF g_ies_grafico THEN
        #Desativação do botão de zoom no ambiente gráfico
        --# CALL fgl_dialog_setkeylabel('control-z',NULL)
     ELSE
        #Retirar texto fixo de zoom no canto superior direito da tela
        DISPLAY "--------" AT <linha>,<coluna>
     END IF
  END IF
 END FUNCTION
```

Caso o programa possua apresentação de mais de uma tela com opção de zoom, talvez seja necessário definir mais um argumento/parâmetro para a função *sssnnnn_ativa_zoom()* para indicar a tela a qual se refere o “zoom” pois na execução em ambiente caracter a posição de linha e coluna para apresentação do texto “(Zoom)” pode mudar de uma tela para outra devido a largura e/ou existência de opções de menu, ficando assim:

```
#-----------------------------------------#
 FUNCTION sssnnnn_ativa_zoom(l_ativa,l_tela)
#-----------------------------------------#
  DEFINE l_ativa  SMALLINT,
         l_tela   SMALLINT
  DEFINE l_coluna SMALLINT

  CASE l_tela
  WHEN 1 LET l_coluna = 68
  WHEN 2 LET l_coluna = 55
  WHEN 3 LET l_coluna = 63
  END CASE

  IF l_ativa THEN
     IF g_ies_grafico THEN
        #Ativação do botão de zoom no ambiente gráfico
        --# CALL fgl_dialog_setkeylabel('control-z','Zoom')
     ELSE
        #Apresentação fixa no canto superior direito da tela
        DISPLAY "( Zoom )" AT <linha>,l_coluna
     END IF
  ELSE
     IF g_ies_grafico THEN
        #Desativação do botão de zoom no ambiente gráfico
        --# CALL fgl_dialog_setkeylabel('control-z',NULL)
     ELSE
```

```
            #Retirar texto fixo de zoom no canto superior direito da tela
            DISPLAY "--------" AT <linha>,l_coluna
         END IF
      END IF
   END FUNCTION
```

- Sempre que um “zoom” for exigido, preferencialmente deve-se utilizar a função de “zoom genérico” *log009_popup()* para apresentação dos dados, não sendo necessário implementar uma nova função para apresentar dados de uma tabela estrangeira, a não ser que já exista uma função (fonte .4gl) para a tabela desejada. Para consulta de funções de zoom já existentes, utilize o sistema CSL, opção 4 (Manutenção de fontes).

```
CALL log009_popup(l_linha,l_coluna,l_cabecalho,l_tabela,l_coluna_1,
                  l_coluna_2,l_programa,l_testa_empresa,l_condição)
```

Observação: a descrição da lista de parâmetros da função log009_popup está melhor detalhada no manual de **Funções Padrões LOGIX** disponível no portal Logocenter**.**

3.7. FUNÇÃO DE ENTRADA DADOS

As consistências de obrigatoriedade dos campos, não permitindo valor em branco ou nulo, devem ser realizadas no bloco de comandos da cláusula AFTER INPUT. As demais consistências, quando não forem dependentes de outras informações devem ser feitas no bloco de comandos da cláusula AFTER FIELD do campo em questão.

As consistências de campos que envolvem dependência de informações em outros campos ou quando inicializadas automaticamente sem intervenção do usuário, devem ser realizadas também no bloco de comandos da cláusula AFTER INPUT.

Exemplos:

1) *Não permitir que o nome do usuário fique em branco ou nulo.*

   Esta consistência deve ser feita no AFTER INPUT, pois trata-se de consistência de obrigatoriedade de um campo, mas para que isto seja possível, na definição da tela (arquivo .PER) o campo precisa estar definido como não obrigatório.

   Um campo é automaticamente assumido como obrigatório quando é definido com o mesmo tipo de uma coluna de tabela do banco de dados que estiver definida como NOT NULL. Para resolver esta questão, deve-se definir as colunas como FORMONLY no arquivo de definição de tela (arquivo com extensão PER).

   Exemplo: a = FORMONLY.cod_empresa TYPE CHAR;

2) *O código do usuário informado deverá estar cadastrado na tabela usuários.*

   Esta consistência deverá ser realizada no AFTER FIELD da coluna de código do usuário, se a pesquisa na tabela USUARIOS não depender de nenhuma outra informação que ainda pode ser informada em tela, mas a pesquisa de existência na tabela USUARIOS somente deve ser realizada se o código de usuário da tela foi devidamente preenchido, ou seja, deve ser diferente de nulo e branco (teste de obrigatoriedade é definido no AFTER INPUT).

Exemplo de codificação:

```
INPUT BY NAME mr_dados.* WITHOUT DEFAULTS
   BEFORE INPUT
      ...
```

```
   BEFORE FIELD cod_usuario
      ...

   AFTER FIELD cod_usuario
      ...
      #Consistência do campo cod_usuario, quando estiver informado
      IF  mr_dados.cod_usuario IS NOT NULL
      AND mr_dados.cod_usuario <> " " THEN
         IF NOT sssnnnnn_usuario_existe(mr_dados.cod_usuario) THEN
            NEXT FIELD cod_usuario
         END IF
      END IF

   ON KEY (control-w, f1)
      CALL sssnnnn_help()

   ON KEY (control-z, f4)
      CALL sssnnnn_zoom()

   AFTER INPUT
      IF int_flag = 0 THEN #Se confirmou a entrada de dados
         #Consistência de obrigatoriedade apenas, pois a consistência de
         #existência do usuário já é feita no AFTER FIELD quando o campo é
         #devidamente preenchido. Mas isto se deve ao fato do código do
         #usuário ser a única informação necessária da tela para fazer tal
         #consistência, caso contrário deverá ser feita no AFTER INPUT.
         IF mr_dados.cod_usuario IS NULL
         OR mr_dados.cod_usuario = " " THEN
            CALL log0030_mensagem("Código do usuário não preenchido.",
                                  "exclamation")
            NEXT FIELD cod_usuario
         END IF
      END IF
END INPUT

#-----------------------------------------------#
 FUNCTION sssnnnnn_usuario_existe(l_cod_usuario)
#-----------------------------------------------#
  DEFINE l_cod_usuario LIKE usuarios.cod_usuario

  INITIALIZE mr_dados.nom_usuario TO NULL
  WHENEVER ERROR CONTINUE
  SELECT nom_funcionario
    INTO mr_dados.nom_usuario
    FROM usuarios
   WHERE cod_usuario = l_cod_usuario
  WHENEVER ERROR STOP

  DISPLAY BY NAME mr_dados.nom_usuario

  IF sqlca.sqlcode <> 0 THEN
     CALL log0030_mensagem("Usuário não cadastrado.","exclamation")
     RETURN FALSE
  END IF

  RETURN TRUE
 END FUNCTION
```

IMPORTANTE: Em nenhuma hipótese deve-se alterar o layout da apresentação dos dados da tela em tempo de execução, como por exemplo, tentar tornar um dos campos da tela invisível, apresentando

espaços em branco com o uso da instrução DISPLAY..AT. Isto se torna inviável pois somente terá o efeito desejado na execução do programa em ambiente caracter. Em ambiente gráfico, o conteúdo do campo que é visualizado em formato 3D é reapresentado a cada instrução DISPLAY..TO, o que fará com que seja apresentado um campo na tela sem título, distorcendo o visual da tela.

3.8. FUNÇÃO “HELP”

Todas as aplicações que possuem algum tipo de interação com o usuário, deverão possuir acesso para o texto de ajuda (help) pela tecla de atalho <CTRL+W> ou <F1> no caso da execução em ambiente gráfico, ou seja, deverá existir “help” acessível a partir de opções de menu e campos onde o cursor possa ser posicionado.

Este texto de ajuda deverá descrever de forma clara e objetiva o significado e objetivo da informação a ser incluída ou executada pelo usuário, bem como definições de regras de negócio do sistema.

O arquivo de ajuda deverá ser criado com o mesmo nome da aplicação (arquivo com extensão “4gl”), somente definindo-o com extensão ‘msg’. Este arquivo de ajuda também deve ser registrado no sistema CSL da mesma forma que os arquivos com extensão “4gl”, “per” e “sql”, no entanto, o tipo deste arquivo no sistema CSL é registrado como tipo “H” (Help). Para cada campo de tela com acesso em entrada de dados (INPUT, CONSTRUCT) ou opção de menu (COMMAND) implementados no fonte, deverá existir um número de identificação a ser associado no arquivo de ajuda.

Exemplo do arquivo ‘sssnnnn.msg’:

```
.001
Incluir

Permite incluir um novo registro.

.002
Modificar

Permite modificar o registro que está sendo exibido na tela.
Deve-se efetuar previamente uma consulta.

.101
<Título do campo 01 da tela>

Descrição do objetivo e funcionalidades do campo 01.

.102
<Título do campo 02 da tela>

Descrição do objetivo e funcionalidades do campo 02.
```

Exemplo de codificação da associação do help no programa:

```
CALL sssnnnn_ativa_help()
MENU “Opção”
COMMAND “Incluir” “Inclui novo registro.”
   HELP 001
COMMAND “Modificar” “Modifica registro atual.”
   HELP 002
END MENU
...
```

```
#-------------------------------#
 FUNCTION sssnnnn_entrada_dados()
#-------------------------------#
   ...
   INPUT BY NAME mr_dados.* WITHOUTH DEFAULTS
     ...
     AFTER FIELD <campo_tela_01>
     ...
     ON KEY (control-w, f1)
        CALL sssnnnn_help()
   END INPUT
   ...
END FUNCTION
...
#-----------------------------#
 FUNCTION sssnnnn_ativa_help()
#-----------------------------#
  DEFINE l_arquivo_help CHAR(100)

  #Busca do caminho do arquivo de help do programa
  LET l_arquivo_help = log140_procura_caminho("sssnnnn.iem")

  #Ativar o arquivo de help na memória
  OPTIONS HELP FILE l_arquivo_help
 END FUNCTION

#----------------------#
 FUNCTION sssnnnn_help()
#----------------------#
  CALL sssnnnn_ativa_help()

  CASE
  WHEN infield(<campo_tela_01>) CALL SHOWHELP(101)
  WHEN infield(<campo_tela_02>) CALL SHOWHELP(102)
  END CASE
END FUNCTION
```

***Observações*:**

Em relação aos padrões para descrição do arquivo de help **(help.txt)** verifique o manual de "**Padrões de documentação 4GL/4JS e Powerbuilder**" disponível no portal Logocenter.

A função *sssnnnn_ativa_help()* sempre deverá ser acionada após a ação executada nas opções de menu que realizarem a ativação de outro arquivo de help. Isto é necessário para que o arquivo de help do programa em execução seja reativado novamente na memória.

3.9. FUNÇÃO RELATÓRIO

Sempre que houver opção para emissão de relatório deve-se utilizar a função *log0280_saida_relat()* para gerenciar o destino da emissão do relatório, conforme descrito no item **5. Funções padrões LOGIX.**

`Sintaxe: log0280_saida_relat(<linha>,<coluna>)`

Onde <linha> e <coluna> indicam a posição da tela onde será apresentada uma janela para seleção do destino do relatório.

Exemplo:

```
IF log0280_saida_relat(14,40) IS NOT NULL THEN
```

```
      MESSAGE “Processando a extração do relatório...” ATTRIBUTE(REVERSE)

      INITIALIZE m_den_empresa TO NULL
      WHENEVER ERROR CONTINUE
      SELECT empresa.den_empresa
        INTO m_den_empresa
        FROM empresa
       WHERE empresa.cod_empresa = p_cod_empresa
      WHENEVER ERROR STOP

      IF p_ies_impressao = “S” THEN
         IF g_ies_ambiente = “W” THEN
            LET m_caminho = log150_procura_caminho('LST')
            LET m_caminho = m_caminho CLIPPED, “sssnnnn_”,p_user CLIPPED,“.tmp”
            START REPORT sssnnnn_relat TO m_caminho
         ELSE
            START REPORT sssnnnn_relat TO PIPE p_nom_arquivo
         END IF
      ELSE
         START REPORT sssnnnn_relat TO p_nom_arquivo
      END IF
      ...
      <instruções OUTPUT>
      ...
      FINISH REPORT sssnnnn_relat

      #Somente enviar os relatórios para impressora, na execução em ambiente
      #WINDOWS, quando alguma informação foi emitida no arquivo em disco.
      IF l_imprimiu_dados THEN
         IF g_ies_ambiente = “W” AND p_ies_impressao = “S” THEN
            LET m_comando = “LPDOS.BAT ”,m_caminho CLIPPED,“ ”,
                            p_nom_arquivo CLIPPED
            RUN m_comando
         END IF

         IF p_ies_impressao = “S” THEN
            CALL log0030_mensagem(“Relatório impresso com sucesso.”,”info”)
         ELSE
            LET l_mensagem = “Relatório gerado em “,p_nom_arquivo CLIPPED,”.”
            CALL log0030_mensagem(l_mensagem, “info”)
         END IF
      ELSE
         CALL log0030_mensagem(“Não existem dados para serem listados.“,
                               ”exclamation”)
      END IF
   END IF
```

Na execução a partir do ambiente Windows, quando for selecionado o destino como IMPRESSORA na execução da função *log0280_saida_relat()*, esta retorna na variável global *p_nom_arquivo* o comando de impressão válido para impressora selecionada, no entanto, deve-se atentar para que no programa MEN0210 (Controle Geral - Cadastros - Impressoras) esteja codificado o comando para impressão com o endereço da impressora na rede (nome do compartilhamento).

O arquivo de lote (shell) LPDOS.BAT é utilizado para enviar o relatório gerado em disco para a impressora quando o sistema é executado em ambiente WINDOWS, pois não é possível enviar diretamente para a impressora utilizando o comando START REPORT. Neste arquivo LPDOS.BAT deverá conter o seguinte código:

```
fglrun e:\log\log2000 %1 %TEMP%\gxrpt.tmp
```

```
copy %TEMP%\gxrpt.tmp %2
del %1
del %TEMP%\gxrpt.tmp
```

3.9.1. Visualização de relatório no BROWSER

Somente para execução do sistema LOGIX no ambiente WINDOWS existe opção de visualização de relatórios formatados em HTML no BROWSER. Para utilizar esta opção deve-se configurar o sistema da seguinte forma:

- Alterar o valor da variável de ambiente USA_BROWSER no sistema operacional para 1 (um), para indicar a utilização da opção de visualização de relatório pelo *browser*.

  USA_BROWSER = 1

- Alterar o código fonte do programa para que antes de acionar a função *log0280_saida_relat()*, o valor da variável global g_usa_browser seja inicializado como 1 (um), para indicar a utilização da opção de visualização de relatório pelo *browser*.

  LET g_usa_browser = 1

- Se o sistema operacional onde o banco de dados estiver instalado for Unix ou Linux devem ser tomadas as seguintes precauções:

  - Executar o LOGIX por terminal Windows e cadastrar o diretório LST para o ambiente “W”. A execução do LOGIX por terminal UNIX/LINUX não poderá acionar o browser.

  - Mapear o diretório do sistema LST no terminal Windows para o mesmo diretório do UNIXLINUX usando a conexão via ferramenta Samba (Verificado pela área de suporte).

- Será habilitado o item “Web” na função *log0280_saida_relat()* e se este for selecionado como destino do relatório, a variável global p_ies_impressao será retornada com o valor “B” (WEB).

- Após geração do relatório em formato HTML (deverá existir uma função implementada no fonte que gere o relatório no formato HTML), ou seja, após a instrução FINISH REPORT, deve-se executar a função *logg028_executa_browser()* conforme o exemplo abaixo, para que o aplicativo de visualização de arquivos no formato HTML seja acionado para apresentação do conteúdo do relatório:

  ```
  CALL log028_executa_browser(<arquivo a ser visualizado no browser>)
  ```

3.9.2. Visualização de relatório em tela utilizando LOG0290

Para permitir que entre as opções de destino do relatório, seja também disponibilizada a opção “TELA” para visualizar o relatório gerado em tela utilizando o programa LOG0290 deve-se proceder da seguinte forma:

- Quando o sistema estiver sendo executado em ambiente Windows é possível optar pelo visualizador desenvolvido na linguagem PowerBuilder ou na linguagem 4GL/4JS, pois para ambiente operacional UNIX somente pode ser utilizado o visualizador desenvolvido em 4GL/4JS. No caso de utilizar ambiente Windows, para optar pela utilização do visualizador desenvolvido em PowerBuilder deve-se indicar a variável de ambiente VIS_4JS para 0 (zero), caso contrário deve ter o valor igual a 1 (um).

VIS_4JS = 0 Para visualizador Powerbuilder

VIS_4JS = 1 Para visualizador 4GL/4JS

- Alterar no programa, antes da chamada da função *log0280_saida_relat()*, o valor da variável g_usa_visualizador para 1 (um), para indicar a utilização da opção de visualização de relatório em tela.

```
LET g_usa_visualizador = 1
```

- Será habilitado o item “Tela” na função *log0280_saida_relat()* e se este for selecionado como destino do relatório, a variável global p_ies_impressao será retornada com o valor “V” (Vídeo).

- Após geração do relatório, que deve ter sido feita em disco para permitir a posterior visualização em tela, ou seja, após execução da instrução FINISH REPORT, deve-se executar a função *log028_visualiza_arquivo()* conforme o exemplo abaixo, para que o programa de visualização de arquivos seja acionado para apresentação do conteúdo do relatório:

```
CALL log028_visualiza_arquivo(<arquivo a ser visualizado>)
```

3.9.3. Layout para emissão

Todas as informações emitidas nos relatórios em formato texto não devem ser acentuadas, pois a emissão a partir do sistema operacional UNIX/LINUX não será emitida corretamente.

O alinhamento dos dados no formato de colunas deve obedecer as seguintes regras:

- Campos numéricos alinhados pela direita;

- Campos alfanuméricos alinhados pela esquerda;

- Campos que identifiquem datas devem ser alinhados pela esquerda;

- Campos do tipo indicador, geralmente definido como alfanumérico de 1 ou 2 posições, podem ser centralizados quando título da coluna for maior que o conteúdo da informação.

Exemplo de relatório:

```
EMPRESA MODELO RELATORIO S.A
RHU0830                              FUNCIONARIOS                              FL.  1
                                                  EXTRAIDO EM 13/10/1995 AS 13:53:20 H

MATRICULA      NOME FUNCIONARIO                    ADMISSAO
---------      ------------------------------      ----------
  42343-2      TIO PATINHAS                        02/02/1993
  44334-2      ZE CARIOCA                          02/01/1994
   ...
   ...
   ...

* * * ULTIMA FOLHA * * *
```

Todos os relatórios gerados pelo sistema LOGIX devem seguir também as especificações de layout descritas a seguir.

3.9.3.1. Número de colunas

O padrão de largura de relatórios é 80 colunas, não podendo exceder 132 colunas, quando possível, ou seja:

- Todo relatório, mesmo que após a emissão dos dados desejados, as informações alinhadas mais a direita não chegarem a 80 colunas, o cabeçalho deverá respeitar 80 colunas como margem direita.

- A limitação de 132 colunas deve ser levada em consideração para relatórios que podem ser emitidos numa impressora do tipo matricial, que na maioria das vezes também é limitada a uma compressão máxima de 17CPP (caracteres por polegada), o que indica que para um formulário contínuo de 80 colunas a impressora será capaz de emitir até um total de 132 caracteres utilizando a compressão de 17CPP. Para casos de relatórios que não são emitidos em impressora do tipo matricial, o limite de 132 colunas pode ser ignorado.

O padrão de número de linhas por página é 66 linhas para o formulário no formato retrato e 46 linhas no formato paisagem utilizando configuração de 6 LPP (linhas por polegada), podendo variar de acordo com a situação a ser implementada em cada programa.

***Observação*:**
Quando é utilizada a função LOG5211 para configuração de impressão, para as impressoras estilo “jato de tinta” ou “laser” o número de linhas por polegada (LPP) e número de colunas por polegada (CPP) é automaticamente ajustado no formulário selecionado de acordo com o número de colunas e linhas repassado para a função.

3.9.3.2. Cabeçalho

Todo cabeçalho de página dos relatórios, emitidos pelo sistema LOGIX, deverá respeitar um layout padrão pré-definido para as primeiras 4 linhas de cada página.

- **Primeira linha**

Identificar a empresa ativa no instante da impressão alinhada pela margem esquerda. Deverá ser emitida a razão social da empresa (tabela EMPRESA, coluna DEN_EMPRESA).

- **Segunda linha**

O número do programa em letras maiúsculas alinhado pela margem esquerda (Ex: LOG0200);

O título do relatório escrito em letras maiúsculas e centralizado;

A numeração de página alinhada pela margem direita, respeitando o formato “FL. ###”, onde ### é a máscara numérica a ser utilizada.

- **Terceira linha**

Data e hora da emissão do relatório alinhados pela margem direita, utilizando o seguinte formato:

“EXTRAIDO EM DD/MM/AAAA AS HH:MM:SS H”

Onde: DD (dia atual), MM (mês atual), AAAA (ano atual com 4 dígitos), HH (hora atual), MM (minuto atual), SS (segundo atual).

- **Quarta linha**

Deixar em branco.

A partir da quinta linha, são emitidas as informações do relatório como cabeçalho de agrupamentos de informações, título das colunas de informações a serem impressas, entre outros dados.

3.9.3.3. Rodapé

Na última página do relatório deverá ser impresso o texto fixo “* * * ULTIMA FOLHA * * *” alinhado pela margem esquerda, no rodapé da página, sem acentuação.

3.9.4. Múltiplos relatórios gerados simultaneamente

Quando na mesma aplicação houver necessidade de gerar 2 ou mais relatórios em paralelo, a nomenclatura do arquivo no caso do destino da emissão ser em “DISCO”, deverá ter um sufixo numérico adicionado a sua nomenclatura para garantir a geração de arquivos distintos e esta numeração varia de 1 a 99, de acordo com o número de relatórios simultâneos necessários. Para emissão de relatório para impressora, no caso da execução em ambiente UNIX/LINUX, não existem diferenças já que a emissão será enviada diretamente para a impressora.

O sufixo numérico dos arquivos, quando da geração de relatório em DISCO, é gerado automaticamente pela função *log0280_saida_relat(.)* No entanto é necessária a execução da função *log028_determina_qtd_arquivos()* antes da execução da função *log0280_saida_relat()*, para determinar a quantidade de arquivos que serão gerados simultaneamente que é passado como parâmetro para tal função. No momento de iniciar o relatório, antes de executar o comando START REPORT de cada relatório utilizando a variável p_nom_arquivo, deve-se executar a função *log028_obtem_nome_arquivo()* para buscar o nome de arquivo a ser gerado de acordo com a seqüência numérica. A função *log028_obtem_nome_arquivo()* irá carregar o nome do arquivo a ser gerado na variável glogal *p_nom_arquivo*.

Exemplo:

```
...
#Determinação de 2 arquivos a serem gerados simultaneamente
CALL log028_determina_qtd_arquivos(2)

IF log0280_saida_relat(17,30) IS NOT NULL THEN
   ...
   #Como se trata da emissão de 2 relatórios simultaneamente, deve-se
   #ativar a variável g_nao_exclui_par para não permitir a exclusão dos
   #parâmetros de impressão selecionados pelo usuário após emissão da
   #primeira página do primeiro relatório utilizando a função
   #log5211_retorna_configuração().
   LET g_nao_exclui_par = TRUE

   #busca nome automático gerado para o primeiro arquivo.
   CALL log028_obtem_nome_arquivo(1)
   START REPORT sssnnnn_relat1 TO p_nom_arquivo

   #busca nome automático gerado para o segundo arquivo.
   CALL log028_obtem_nome_arquivo(2)
   START REPORT sssnnnn_relat2 TO p_nom_arquivo
   ...
   OUTPUT...
   ...
   FINISH REPORT sssnnnn_relat1
   FINISH REPORT sssnnnn_relat2
   ...
END IF
...
```

Observação:

O nome automático de arquivo com sufixo numérico é gerado da seguinte forma:

- Em ambiente UNIX/LINUX: Arquivo 1 => log0200:admlog:1
  Arquivo 2 => log0200:admlog:2
- Em ambiente WINDOWS: Arquivo 1 => log0200.lst1
  Arquivo 2 => log0200.lst2

3.9.5. Exemplo função REPORT

Todos os relatórios deverão possuir no bloco FIRST PAGE HEADER ou PAGE HEADER a chamada da função *log5211_retorna_configuracao()* para determinar a configuração de saída dos dados na impressora. Algumas das configurações previstas por esta função são:

- Tamanho de fonte;

- Configuração de condensação de caracteres (caracteres por polegada - CPP);

- Formato de formulário (Paisagem ou Retrato);

- Tamanho das margens;

- Outras.

Exemplo:

```
#------------------------------#
 REPORT sssnnnn_relat1(lr_relat)
#------------------------------#
  DEFINE lr_relat   RECORD
                    ...
                    END RECORD

  DEFINE l_last_row SMALLINT

  OUTPUT LEFT   MARGIN 0
         TOP    MARGIN 0
         BOTTOM MARGIN 1

  FORMAT
  PAGE HEADER #cabeçalho
     PRINT log5211_retorna_configuracao(PAGENO,66,80) CLIPPED;
     PRINT COLUMN 001, m_den_empresa #variável já carregada
     PRINT COLUMN 001, "LOG0200",
           COLUMN 036, "EMPRESAS",
           COLUMN 075, "FL.", PAGENO USING "###"
     ...
     PRINT COLUMN 045, "EXTRAIDO EM ",TODAY," AS ",TIME," H"
     SKIP 1 LINE
     PRINT COLUMN 001, "COD RAZAO"
     PRINT COLUMN 001, "--- --------------------------------"

  BEFORE GROUP OF ... #cabeçalho de um agrupamento
     ...

  ON EVERY ROW
     PRINT COLUMN 001, lr_relat.cod_empresa,
           COLUMN 005, lr_relat.den_empresa

  AFTER GROUP OF ... #rodapé de um agrupamento
     ...

  ON LAST ROW
     LET l_last_row = TRUE

  PAGE TRAILER
     IF l_last_row THEN
        PRINT "* * * ULTIMA FOLHA * * *",
              log5211_termino_impressao() CLIPPED
     ELSE
        PRINT " "
     END IF
END REPORT
```

***IMPORTANTE*:**

Estas especificações não se aplicam para formulários contínuos pré-impressos, pois na maior parte de relatórios não existe a possibilidade do usuário indicar o tipo de fonte, tamanho de fonte, margem, entre outras configurações, além de serem emitidos por impressoras matriciais. Neste caso é feita a emissão de comandos fixos para impressora, identificando compactação de caracteres por polegada e número de linhas por polegada, não utilizando as funções *log500_determina_cpp()* e *log5211_retorna_configuracao()*.

3.10. PROGRAMAS EXECUTADOS EM MODO “NORMAL” OU “BACKGROUND”

Programas executados a partir de outro programa em modo “background” podem ser executados quando não possuem interação com o usuário (entrada de dados) e não houver obrigatoriedade de conclusão de seu processamento para que as próximas instruções contidas no programa principal possam ser executadas.

Programas executados a partir de outro programa em modo “normal” podem ser executados independentemente de possuírem ou não interação com o usuário (entrada de dados), neste caso, o programa principal após acionar a instrução de execução de outro programa irá aguardar o término da execução deste outro programa para prosseguir com a execução da próxima instrução.

A execução de programas visando estes conceitos deve considerar o seguinte padrão de desenvolvimento:

- Para acionar um programa e aguardar o término de sua execução antes de acionar a próxima instrução (modo normal):

```
CALL log1200_executa_programa(...)
```

- Para acionar um programa e não aguardar o término de sua execução para acionar a próxima instrução (modo background):

```
CALL log1200_executa_programa_background(...)
```

No caso de execução de programas que necessitam de envio de argumentos para garantir sua correta execução, pode-se realizar a passagem destes argumentos de um programa para outro através da linha de comando de execução do programa ou então via registros gravados na tabela TRAN_ARG utilizando a função *log1200_parametro_programa_inclui()*.

Exemplos:

Passagem de argumentos via linha de comando que:

```
LET l_argumentos = p_ies_impressao CLIPPED,
                   " ",p_nom_arquivo_back CLIPPED,
                   " ",g_cod_impressora,
                   " ",g_seq_configuracao

#execução em modo background (não aguardar o término da execução)
CALL log1200_executa_programa_background("men0050",l_argumentos)

OU

#execução em modo normal (aguardar o término da execução)
CALL log1200_executa_programa("men0050",l_argumentos)
```

Passagem de argumentos via função *log1200_parametro_programa_inclui()* – tabela TRAN_ARG:

```
#Passagem do argumento 1 (p_ies_impressao)
IF NOT log1200_parametro_programa_inclui('men0050',1,0,p_ies_impressao) THEN
   #Abortar o acionamento do outro programa, pois a passagem de argumentos falhou
END IF

#Passagem do argumento 2 (p_nom_arquivo)
IF NOT log1200_parametro_programa_inclui('men0050',2,0,p_nom_arquivo) THEN
   #Abortar o acionamento do outro programa, pois a passagem de argumentos falhou
```

```
END IF

#Passagem do argumento 3 (g_cod_impressora)
IF NOT log1200_parametro_programa_inclui('men0050',3,0,g_cod_impressora) THEN
   #Abortar o acionamento do outro programa, pois a passagem de argumentos falhou
END IF

#Passagem do argumento 4 (g_seq_configuracao)
IF NOT log1200_parametro_programa_inclui('men0050',4,0,g_seq_configuracao) THEN
   #Abortar o acionamento do outro programa, pois a passagem de argumentos falhou
END IF

#execução em modo background (não aguardar o término da execução)
CALL log1200_executa_programa_background("men0050","")
```

OU

```
#execução em modo normal (aguardar o término da execução)
CALL log1200_executa_programa("men0050","")
```

Neste caso acima, como os argumentos foram repassados para outro programa (MEN0050) via registros da tabela TRAN_ARG, no programa que será executado deverá ser implementado um código para carga do conteúdo dos argumentos repassados que irão ser armazenados nas seguintes variáveis globais:

- p_ies_impressao
- p_nom_arquivo ou p_nom_arquivo_back
- g_cod_impressora
- g_seq_configuracao

Exemplo de recebimento de argumentos passados via registros da tabela TRAN_ARG:

```
#-------------------------------------------#
 FUNCTION sssnnnn_carga_parametros_tran_arg()
#-------------------------------------------#
  LET p_ies_impressao    = log1200_parametro_programa_le(1,0)
  LET p_nom_arquivo      = log1200_parametro_programa_le(2,0)
  LET g_cod_impressora   = log1200_parametro_programa_le(3,0)
  LET g_seq_configuracao = log1200_parametro_programa_le(4,0)
 END FUNCTION
```

Código padrão para rotina de impressão quando do recebimento de argumentos for via linha de comando:

```
LET p_ies_impressao    = arg_val(1)
LET p_nom_arquivo_back = arg_val(2)
LET g_cod_impressora   = arg_val(3)
LET g_seq_configuracao = arg_val(4)

#Quando for feita execução de programa "background" onde é passada a variável
#p_nom_arquivo_back como argumento, deve-se acionar a função
#log090_converte_comando_impressão para eliminar caracteres de edição
#("@" e "!") existentes na variável p_nom_arquivo_back.
LET p_nom_arquivo = log090_converte_comando_impressao(p_nom_arquivo_back)

IF p_ies_impressao = "S" THEN
   #emissão para impressora
   IF g_ies_ambiente = "W" THEN
```

```
      # Emissão utilizando ambiente Windows
      LET m_caminho = log150_procura_caminho('LST')
      LET m_caminho = m_caminho CLIPPED, “sssnnnn_”,p_user CLIPPED,“.tmp”
      START REPORT sssnnnn_relat TO m_caminho
   ELSE
      #Emissão utilizando ambiente UNIX/LINUX
      START REPORT sssnnnn_relat TO PIPE p_nom_arquivo
   END IF
ELSE
   #Emissão para disco
   START REPORT sssnnnn_relat TO p_nom_arquuivo
END IF
...
<instruções OUTPUT>
...
FINISH REPORT sssnnnn_relat

IF l_imprimiu_dados THEN
   IF p_ies_impressao <> “S” THEN
      LET l_msg = “Relatório gerado em ”, p_nom_arquivo CLIPPED,”.”
      CALL log0030_mensagem(l_msg,“info”)
   END IF

   CASE p_ies_impressao
   WHEN “B”
      #Browser
      CALL log028_executa_browser(p_nom_arquivo)

   WHEN “S”
      #Impressora
      IF g_ies_ambiente  = “W” THEN
         LET m_comando = “lpdos.bat ”,m_caminho CLIPPED,“ ”,
                                      p_nom_arquivo CLIPPED
         RUN m_comando
      END IF
      CALL log0030_mensagem(“Relatório impresso com sucesso.”,“info”)

   WHEN “V”
      #Tela (Vídeo)
      CALL log028_visualiza_arquivo(p_nom_arquivo)

   END CASE
ELSE
   CALL log0030_mensagem(“Não existem dados para serem listados.”,
                         “exclamation”)
END IF
```

4. Exemplos de aplicação

Existem alguns modelos de aplicações desenvolvidas em Informix 4GL disponíveis para consulta em \\pcnt1\documentacao\Metodologia_Desenvolvimento\Programacao\Logix1\Modelos.

4.1. DICAS DE PROGRAMAÇÃO

Existe um documento contendo uma lista de dicas de programação 4GL/4JS disponíveis para consulta em \\pcnt1\documentacao\Metodologia_Desenvolvimento\Programacao\Logix1\.

5. Funções padrões LOGIX

Existe um conjunto de funções desenvolvidas com intuito de minimizar e padronizar algumas rotinas comuns utilizadas, como por exemplo: apresentação de mensagens para o usuário, seleção de destino de emissão de relatório, tela para confirmação pelo usuário, entre outras.

Estas funções estão descritas e devidamente explicadas no documento disponível para consulta em \\pcnt1\documentacao\Metodologia_Desenvolvimento\Programacao\Logix1\funcoes_padroes_LOGIX.pdf ou via link no portal Logocenter.

6. Funcionalidades Gráficas 4JS

Aqui estão apresentadas diversas funcionalidades gráficas disponíveis no 4JS e que devem ser aplicadas ao padrão de desenvolvimento LOGIX.

6.1. MESSAGE BOX

Os “Message Boxes” são utilizados para exibir o resultado de algumas operações executadas pelo programa. Podem ser utilizados em casos onde existem processamentos de rotinas, geração de arquivos de saída e relatórios e erros de processamento.

Um exemplo de utilização desta funcionalidade é o uso da função **log0040_confirm()** para confirmação de execução de alguma operação.

Podem ainda ser utilizados para mensagens de erro grave onde se deseja chamar a atenção do usuário.

Sintaxe:

```
--# CALL fgl_winmessage(título,mensagem,ícone)
```

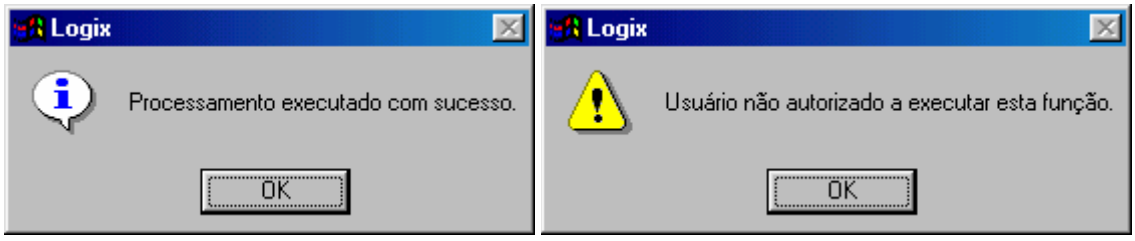
Ícone		
Info	Exclamation	Stop
		
Mensagem informativa de processamento executado(rotina, geração de relatório, geração de arquivo).	Mensagem de alerta para erros previstos na aplicação.	Mensagem de erro provocado por um erro no banco ou não previsto.
Título = Logix		
Mensagem = mensagem a ser exibida		

Exemplo código:

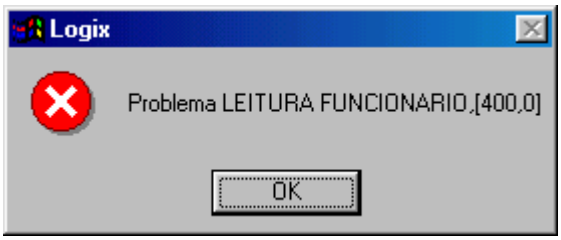
```
--# CALL fgl_winmessage(“Logix”,
--# “Processamento executado com sucesso.”,”info”)
```

Para se manter a portabilidade entre o 4GL e 4JS, deve-se usar a função **log0030_mensagem()** conforme descrito no item **5. Funções** padrões LOGIX.

Exemplo:



Exemplo erro:



6.2. DIALOG BOXES

As caixas de diálogos são utilizadas para confirmação de execução de alguma operação, permite selecionar uma ação a ser executada e permite também informar dados.

6.2.1. Confirmação

Utilizada para confirmar a execução de alguma operação ou função nos programas.

Sintaxe:

```
--# LET variável = fgl_winquestion(título,texto,
--#                 valor_default,valores_possíveis,ícone,0)
```

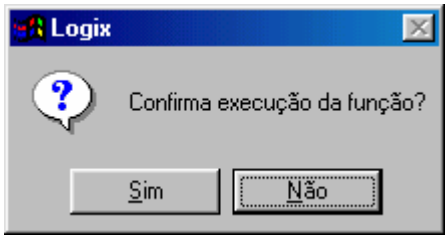
Título = Logix
Texto = texto a ser exibida
Valor_default = Opção pré-selecionada. “No”
Valores_possíveis = “Yes|No|Cancel” o texto é traduzido de forma automática.
Ícone = “question” 

Exemplo código:

```
--# LET l_resposta = fgl_winquestion(“Logix”,
--#                   “Confirma execução da função?”,
--#                   “No”,“Yes|No”,“question”,0)
```

Para manter a portabilidade entre o 4GL e 4JS, deve-se usar a função **log0040_confirm()** conforme descrito no item **5. Funções padrões LOGIX**.

Exemplo:



6.2.2. Selecionar opção

Utilizada para selecionar uma opção ou ação da função em andamento.

Sintaxe:

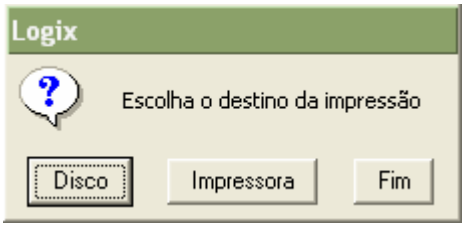
```
      IF g_ies_grafico THEN  #Verifica se execução é em ambiente gráfico
--#      LET variável = fgl_winbutton(título,texto,
--#                         valor_default,valores_possíveis,ícone,0)
      ELSE
         ...
      END IF
```

Título = Logix
Texto = texto a ser exibida
Valor_default = Opção pré-selecionada.
Valores_possíveis = as opções possíveis para seleção
Ícone = “question”

Exemplo código:

```
      IF g_ies_grafico THEN  #Verifica se execução é em ambiente gráfico
--#      LET l_resposta = fgl_winbutton(“Logix”,
--#                         “Escolha o destino da impressão”,
--#                         “Disco”,”Disco|Impressora|Fim”,
--#                         “question”,0)
      ELSE
         MENU "Opção"
         COMMAND "Disco"      "Envia relatório para disco."
            ...
         COMMAND "Impressora" "Envia relatório para impressora."
            ...
         COMMAND "Fim"        "Retorna ao menu anterior."
            EXIT MENU
         END MENU
      END IF
```

Exemplo:



6.2.3. Informa dados

Utilizada para entrada de dados pelo *prompt* do programa.

Sintaxe:

```
IF g_ies_grafico THEN  #Verifica se execução é em ambiente gráfico
   --# LET variável = fgl_winprompt(y,x,texto,
   --#                              valor_default,tamanho,tipo_do_dado)
ELSE
   ...
END IF
```

Y,X = posição de abertura da tela (linha,coluna)				
Texto = texto a ser exibida				
Valor_default = Não utilizado				
Tamanho = Tamanho máximo da informação a ser entrada				
Tipo_do_dado				
0	1	2	7	255
Caracter	Inteiro até 32767	Inteiro até 2 bilhões	Data	Invisível

Exemplo código:

```
IF g_ies_grafico THEN  #Verifica se execução é em ambiente gráfico
   --# LET m_comando = fgl_winprompt(20,3,"Digite o comando: ","",60,0)
ELSE
   PROMPT "Digite o comando: " FOR m_comando
END IF
```

Exemplo:

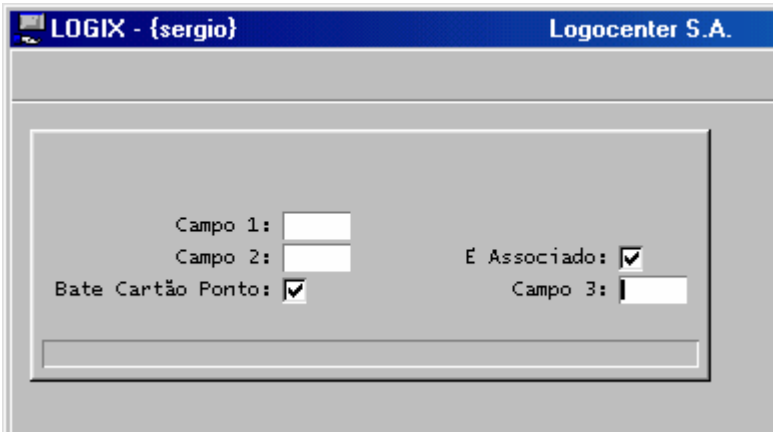
6.3. CHECK BOX

Os Check Boxes devem ser utilizados para campos onde existem apenas dois valores aceitos, do tipo "Sim" e "Não".

Estas alterações são efetuadas no fonte de definição da tela (arquivo com extensão PER). Em cada campo que existir um check box deverá ser acrescida a cláusula "COMMENTS" para descrever o valor aceito e o campo que está sendo referenciado para facilitar a identificação do usuário. Veja exemplo:

```
DATABASE formonly

SCREEN
{
           Campo 1: [a    ]
           Campo 2: [b    ]         É associado: [x]
 Bate cartão ponto: [y]                Campo 3: [c    ]
}

ATTRIBUTES
a = formonly.a;
b = formonly.b;
x = formonly.ck1 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(S,N),
    COMMENTS='Sim (marcado)  Não (desmarcado)
--#, widget='CHECK', config='S N { }'
;
y = formonly.ck2 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(S,N)
    COMMENTS='Sim (marcado)  Não (desmarcado)
--#, widget='CHECK', config='S N { }'
;
c = formonly.c;
END
```

Exemplo:



A opção **config** possui 3 parâmetros. Os dois primeiros parâmetros são respectivamente os valores retornados pelo "check box" (S – Sim e N – Não) quando marcado e desmarcado. O terceiro parâmetro é o nome do campo a ser exibido do lado direito do "check box". Este parâmetro não deverá conter nada entre os caracteres "{}".

Para marcar/desmarcar um campo do tipo checkbox, deve-se pressionar a barra de espaços no teclado ou clicar com o mouse sobre o objeto checkbox. Mas para que isto funcione de tal forma é necessário que no arquivo de configuração do 4JS (fglprofile) a diretiva **`gui.key.radiocheck.invokeexit`** esteja definida da forma:

```
gui.key.radiocheck.invokeexit = ""
```

6.4. RADIO BUTTON

Deve ser utilizado em campos cujo domínio seja fixo e com a possibilidade de escolher apenas uma opção dentro de um grupo de opções.

Cada opção do domínio, apresentada com o componente RADIO BUTTON, ocupa uma linha em tela. O tamanho do campo para a apresentação completa da descrição de todas as opções válidas deve ser correspondente ao número de caracteres da maior descrição entre a lista de opções válidas, acrescido de mais 2 espaços, reservados para apresentação gráfica em formato circula, próprio do componente RADIO BUTTON durante a execução de um programa.

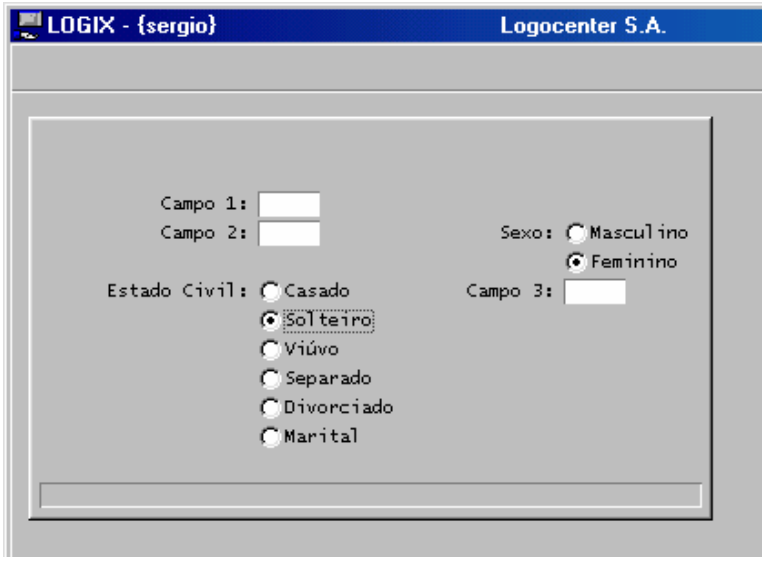
Exemplo de codificação:

```
DATABASE formonly

SCREEN
{
           Campo 1:[f01  ]
           Campo 2:[f02  ]                Sexo:[x          ]
                                                ..
      Estado Civil:[y           ]      Campo 3:[f03  ]
                   ..
                   ..
                   ..
                   ..
                   ..
}

ATTRIBUTES
f01 = formonly.a;
f02 = formonly.b;
x = formonly.rd1 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(M,F)
--#, widget='RADIO', config='M {Masculino} F {Feminino}'
;
y = formonly.rd2 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(C,S,V,P,D,M)
--#, widget='RADIO',
--#, config='C {Casado} S {Solteiro} V {Viúvo} P {Separado} D {Divorciado} M
{Marital}'
;
f03 = formonly.c;
END
```

Observe que neste exemplo o campo **rd1** está desenhado com 11 posições e duas linhas, pois existem duas opções a serem apresentadas (Masculino / Feminino) e a maior é ‘Masculino’ com 9 caracteres.

Já o campo **rd2** está desenhado com 12 posições e 6 linhas, pois existem 6 opções a serem apresentadas (Casado / Solteiro / Viúvo / Separado / Divorciado / Marital) e a maior é ‘Divorciado’ com 10 caracteres.

Exemplo:



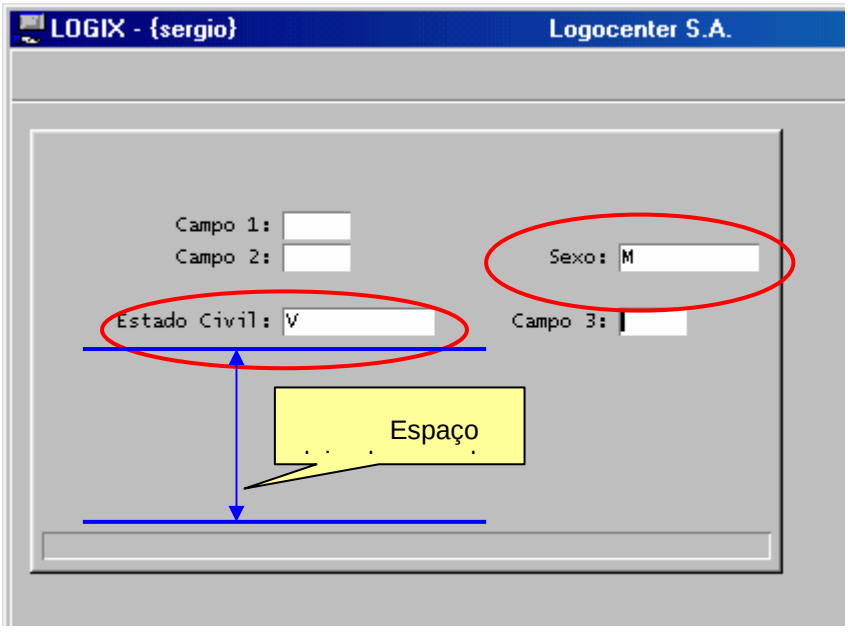
A opção **config** possui um conjunto de 2 parâmetros para cada opção. O primeiro parâmetro será o valor retornado pelo RADIO BUTTON quando selecionado. O segundo parâmetro é o texto a ser exibido do lado direito do RADIO BUTTON.

O valor retornado pelo RADIO BUTTON será nulo se nenhuma opção for selecionada. É possível definir um valor padrão para grupo de opções pela opção **default**.

Para marcar/desmarcar um campo do tipo RADIO BUTTON durante a execução, pode-se pressionar a barra de espaço do teclado ou dar um clique do mouse sobre a opção desejada. Para que isto funcione, é necessário que no arquivo de configuração do 4JS (fglprofile), a diretiva **`gui.key.radiocheck.invokeexit`** esteja definida da seguinte forma:

```
gui.key.radiocheck.invokeexit = “”
```

Um problema de portabilidade entre o ambiente caracter e gráfico é verificado em função da necessidade em redesenhar a tela, devido ao fato do objeto do tipo radio ocupar mais espaço no sentido horizontal e vertical. Diante desta situação, quando executamos em ambiente caracter, a tela seria apresentada da seguinte forma:



Como solução deve-se utilizar duas telas, uma para execução em ambiente gráfico e outra para execução em ambiente caracter. Para isso deve-se utilizar a função **log1300_procura_caminho()** informando as 2 telas (tela do ambiente caracter e tela do ambiente gráfico) no fonte do programa antes de realizar a instrução OPEN WINDOW.

Sintaxe:

```
LET m_caminho = log1300_procura_caminho(código_tela_padrão,
                                        {código_tela_gráfica|""})
```

Código_tela_padrão	Código da tela padrão específica quando o ambiente for caracter.
Código_tela_gráfica	Código da tela gráfica específica para quando o ambiente for gráfico, ou nula, onde então será utilizado o código da tela padrão.

Abaixo exemplo de código para manipular a abertura das telas nesta condição:

```
LET m_caminho = log1300_procura_caminho('log1100','log11001')
OPEN WINDOW w_log1100 AT 2,3 WITH FORM m_caminho
     ATTRIBUTE(BORDER, MESSAGE LINE LAST, PROMPT LINE LAST)
```

6.5. LIST BOX

O uso do **list box** poderá ser necessário quando existir uma lista muito grande para ser apresentada no formato RADIO BUTTON. O tamanho limite para uma lista ser definida como RADIO BUTTON é de 5 opções. Para um número maior que 5 opções é aconselhado o uso do list box.

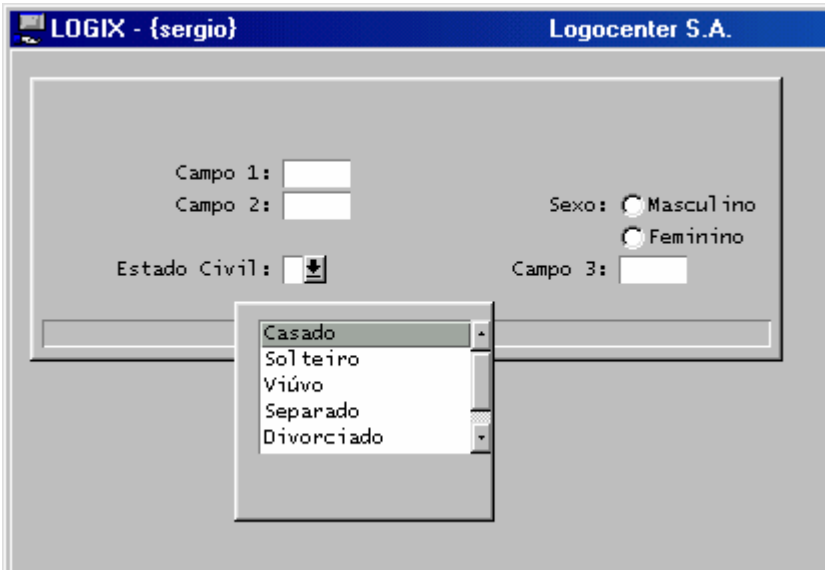
Exemplo de codificação tela:

```
DATABASE formonly

SCREEN
{
           Campo 1: [f01  ]
           Campo 2: [f02  ]                Sexo: [x          ]
                                                  ..
      Estado civil: [y]                 Campo 3: [f03  ]

}

ATTRIBUTES
f01 = formonly.a;
f02 = formonly.b;
x = formonly.rd1 TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(M,F)
--#, widget='RADIO', config='M {Masculino} F {Feminino}'
;
y = formonly.est_civil TYPE CHAR, UPSHIFT, AUTONEXT, INCLUDE=(C,S,V,P,D,M)
--#, widget='FIELD_BMP', CONFIG="combo.bmp Control-z"
;
f03 = formonly.c;
END
```

Exemplo de codificação no fonte:

```
INPUT BY NAME a, b, rd1, est_civil, c WITHOUT DEFAULTS
ON KEY (control-z, f4)
   CALL sssnnnn_zoom()
END INPUT
...
#----------------------#
 FUNCTION sssnnnn_zoom()
#----------------------#
  DEFINE l_opcao CHAR(01)

  CASE
  WHEN INFIELD(est_civil)
     LET l_opcao = log0830_list_box(10,6,
         'C {Casado},S {Solteiro},V {Viúvo},P {Separado},
          D {Divorciado},M {Marital}'
     IF l_opcao <> " " THEN
        LET mr_tela.est_civil = l_opcao
        DISPLAY BY NAME mr_tela.est_civil
     END IF
  END CASE

  LET int_flag = 0
  CALL sssnnnn_ativa_help()
END FUNCTION
```

Exemplo:



6.6. FOLDER TABS

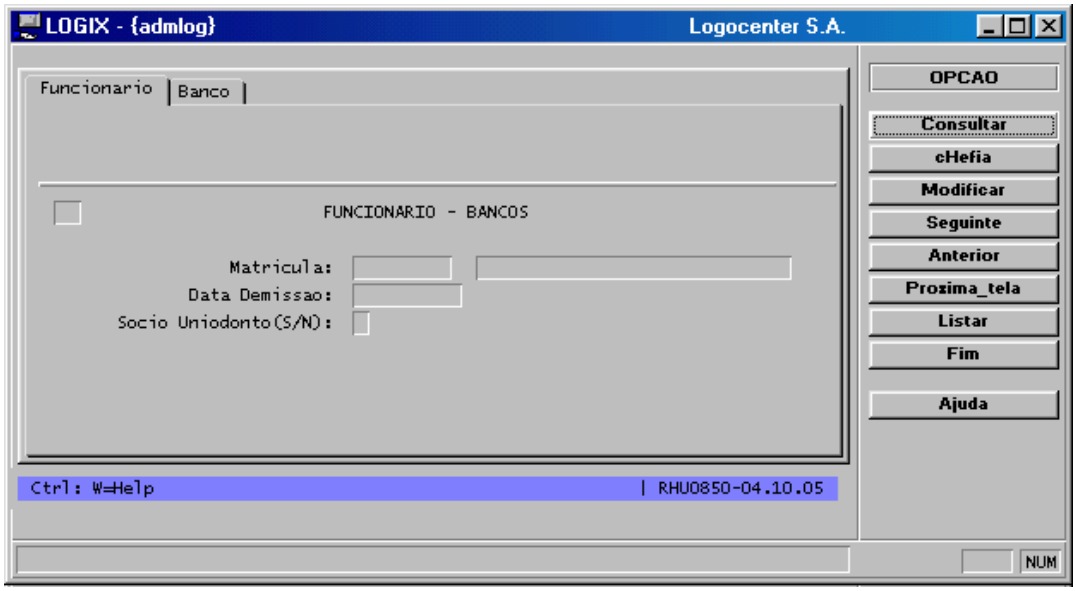
Pode ser utilizado para dividir telas com muitas informações, mas para a versão 4GL (ambiente caracter) é obrigatório trabalhar com mais de uma tela, pois não existe tal funcionalidade na versão caracter. O limite de tamanho de cada tela é o mesmo especificado no item **1.3. DEFINIÇÃO DE TELAS**.

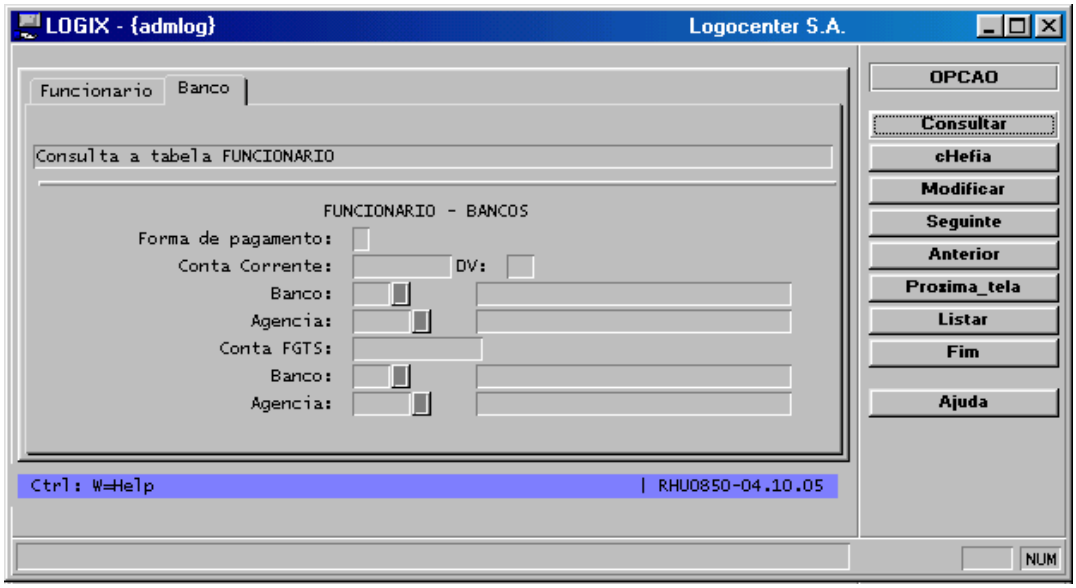
A pasta ativa (página do objeto FOLDER TABS em visualização) sempre será aquela que possui o campo com a última exibição executada. Para efetuar uma entrada de dados todos os campos devem constar na mesma instrução INPUT. Quando existirem entrada de dados utilizando listas (arrays), deve-se utilizar uma pasta específica (FOLDER) para tal entrada de dados. Desta forma, pode-se definir várias instruções INPUT.

Exemplo de codificação da tela:

```
DATABASE formonly

SCREEN --# TITLE "Tela 1/3"
{
 Campo 1: [f01 ]
 Campo 2: [f02 ]
}
--# SCREEN TITLE "Tela 2/3"
{
 Campo 3: [f03 ]
}
--# SCREEN TITLE "Tela 3/3"
{
 Tabela linha 1 [a01 ]
        linha 2 [a01 ]
        linha 3 [a01 ]
        linha 4 [a01 ]
}

ATTRIBUTES
F01 = formonly.f01;
F02 = formonly.f02;
--# F03 = formonly.f03;
--# A01 = formonly.a01;

--# INSTRUCTIONS
--# SCREEN RECORD sr_array[4] (a01)
```

Exemplo de tela:





Para utilizar este recurso é necessário que no arquivo de configuração 4JS (fglprofile), a diretiva **menu.style** esteja informada da seguinte forma:

```
menu.style=1
```

ou

```
menu.style=true
```

Isso fará com que as opções do menu sejam apresentadas no lado direito e de forma vertical na janela do windows no momento da execução do programa.

6.7. NOMEAR HOT KEYS

Os botões da área de “Hot Keys” são exibidos automaticamente quando definidos no fonte por meio das funções COMMAND KEY ou ON KEY. Desta forma, é exibida na tela a tecla de função e não um texto descritivo explicando a sua finalidade. Para adicionar um texto ao botão deve-se proceder da seguinte forma:

Sintaxe:

```
CALL fgl_setkeylabel('tecla','descrição_tecla')
```

Este comando deve ser executado antes de uma instrução INPUT, INPUT ARRAY, DISPLAY ARRAY ou CONSTRUCT.

A descrição da tecla de atalho não pode exceder 14 caracteres, que é o mesmo limite definido para o texto de opções de menu.

Se durante a execução de uma destas funções acima for necessário alterar a descrição da tecla de atalho, deve ser usado o seguinte comando:

Sintaxe:

```
CALL fgl_dialog_setkeylabel('tecla','descrição_tecla')
```

A função *fgl_dialog_setkeylabel()* é utilizada dentro do escopo de uma função de diálogo (INPUT, INPUT ARRAY, DISPLAY ARRAY, CONSTRUCT) quando se deseja alertar que um determinado campo, sobre o qual o cursor estiver posicionado, possui a opção de "Zoom". Neste caso, o botão é apresentado e, no momento do cursor ser posicionado em outro campo, o botão é desativado usando novamente a função *fgl_dialog_setkeylabel()* passando como parâmetro a descrição do botão com o valor NULL, conforme exemplo a seguir:

```
CALL fgl_dialog_setkeylabel('tecla',NULL)
```

As teclas de atalho especiais como ESC e <Control+C> não são passíveis de alteração por estas funções, pois já são pré-definidas no arquivo de configuração do 4JS (fglprofile).

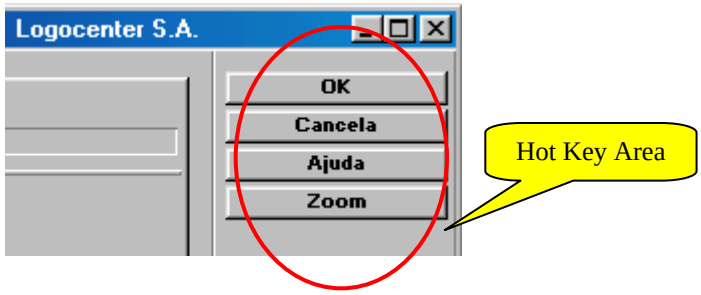
Para desabilitar um botão, deve-se atribuir a este a descrição nula, e para que o botão seja habilitado novamente basta indicar uma descrição válida. Para que isso tenha efeito é necessário que no arquivo de configuração do 4JS (fglprofile) seja definida a diretiva **gui.empty.button.visible** da seguinte forma:

```
gui.empty.button.visible = 0
```

ou

```
gui.empty.button.visible = false
```

Exemplo de tela:



Importante:

As informações constantes neste manual deverão ser consultadas com freqüência, pois sofrerão constantes atualizações.

Quadro de Revisões		
Revisão	**Descrição Alteração**	**Data**
01	Inicial	12/07/00
26	Exclusão dos quadros explicativos de funções do item **5 (Funções Padrões Logix)** e criação de novo documento para este tópico.	11/07/05
27	Alteração descrição item 3.10.	19/08/05
28	- Alteração item **1.6. Instruções SQL**, apresentando utilização das novas funções padrões do LOG0030. - Alteração item **3.4.Chamadas de programas externos**, citando a utilização da função log1200_executa_programa().	08/11/05
29	Inclusão do tópico “Como descrever a sigla na documentação” no capítulo explicativo sobre “Título de campos”.	11/05/06
30	Revisão sem alteração	08/11/06