i/400 Consulting

Always Evolving

ESPECIFICAÇÕES TÉCNICAS DE PROGRAMAÇÃO
(Em RPG ILE /free)

CONTEÚDO

OBJETIVO

Este documento tem por objetivo estabelecer normas e especificações técnicas a serem seguidos pelos trabalhadores de i/400 Consulting na linguagem RPG ILE no formato "free".

ÂMBITO DE APLICAÇÃO

Manter um padrão para o desenvolvimento de sistemas, nomenclatura a ser utilizada e as técnicas de programação, resultando em:

- Um desenvolvimento de sistemas mais rápido
- facilidade na manutenção de programas e
- facilidade na leitura dos programas.

CAPÍTULO I: REGRAS GERAIS PARA DESENVOLVIMENTO DE SISTEMAS EM RPG ILE "FREE".

1. A aparência dos programas é muito importante e as regras de escritura devem ser respeitadas. Portanto, é necessário seguir as instruções dadas aqui para o desenvolvimento de sistemas.

2. Cada colaborador é responsável por respeitar as bibliotecas assignadas ao projeto, de acordo com as especificações descritas no capítulo II sobre este documento.

3. Todo programa terminado deve ser submetido ao líder do projeto que será responsável por testes adicionais e aprovação.

4. O líder do projeto é responsável em manter o controle sobre os arquivos e objetos nas bibliotecas. Qualquer solicitação de substituição, criação ou supressão de fontes ou objetos deve ser dirigida a ele.

5. Um programa interativo será considerado concluído quando os correspondentes painéis de ajuda (em UIM) estiverem prontos e testados (quando aplicável).

6. Todo o programa desenvolvido deve ser documentado com a respectiva "Ficha Técnica", que deve ser armazenada na pasta do projeto correspondente.

7. Todas as telas desenvolvidas devem estar de acordo com as **normas SAA** de **IBM**, (a menos que o cliente solicite o contrário).

8. Os Passes a Produção devem respeitar as instruções dadas no "Formato de Passe a Produção".

CAPÍTULO II: ESPECIFICAÇÕES TÉCNICAS.

2.1. DESENVOLVIMENTO DE SISTEMAS

- Cada projeto terá na sua biblioteca de fontes todos os arquivos de fontes padrão utilizados pela IBM e mais alguns específicos de i/400 Consulting:

  QRPGLESRC - Fontes RPG ILE
  QCMDSRC - Fontes de Comandos
  QDDSSRC - Fontes de Arquivos, telas e relatórios
  QCLLESRC - Fontes de CL ILE
  QUIMSRC - Fontes de painéis UIM
  QCPYSRC - Fontes de /Copy e /Include
  QSRVSRC - Fontes de Programas de Serviços e "Bind Language"

- Todos os projetos devem ter um conjunto de 3 (três) bibliotecas: uma para Fontes, uma para arquivos de dados e uma para objetos. Os nomes dessas bibliotecas devem ser definidos de acordo com a nomenclatura definida no ponto 2.2 mais adiante neste documento. Quando o projeto não tiverem fontes e objetos suficientes, o líder do projeto pode optar por utilizar uma única biblioteca para fontes, arquivos e objetos.

- As compilações interativas devem ser evitadas. Todas as compilações devem ser executadas por lotes.

- Todos os programas: Interativos ou lote, devem respeitar a lógica do modelo do tipo apropriado.

- As instruções nos programas ILE RPG e CLLE devem ser escritas em um padrão Maiúsculas/Minúsculas (misto) (ver "nomes das variáveis" descritas abaixo).

- As instruções aninhadas devem ter recuo de 3 (três) caracteres (os três pontos indicam o número de caracteres em branco e são apenas para fins de exibição).
  Exemplo:

```
//->01 Se Exit
If *InKg;
...Limpar $Nível;
...// ====
...LeaveSr
...// ====
EndIf;
//->01 EndIf
```

- Tente alinhar instruções de atribuição pelo sinal de igual (=).
  Exemplo:

```
Revenue               = RpfIng;
EstabilidadLaboral    = RpfELb;
Debt                  = RpfPEn;
RelacionCuotaIngresos = RpfPCI;
ValorMinimoGarantia   = RpfVMG;
```

- Use uma linha em branco entre as declarações para melhorar a legibilidade quando seja realmente necessário.
  Exemplo:

```
// Busca gama de Valores para cada campo

KValorHasta = Age;

// Taxa prazo
KKey = kRateTerm;
Setll KCApclsXFIR RApclsXFIR;
Reade KPApclsXFIR RApclsXFIR;
//->01 Se não for encontrado
If Not %Eof();
```

- Use uma linha para cada expressão lógica, terminando a linha com o operador lógico para melhorar a leitura.
  Exemplo:

```
//->01 Se não foi carregado primeiro ou último nome
If FirstName = *Blanks And
   LastName  = *Blanks;
   Error;
EndIf;
//->01 EndIf
```

- Para chamadas para programas ou procedimentos utilizando protótipos, definir todos os parâmetros, se couberem, em uma única linha, caso contrário, colocá-los cada um em linhas separadas.
  Exemplo:

```
// Pesquisa de dados do usuário
CALLP RtvCusCun(CusCun :
                @Cusna1:
                @Cusna2:
                @Cusna3:
                @Cusna4);

// Pesquisa Dados da Conta
CALLP RtvAccRcd(Conta : PtrAcMst)
```

2.2. NOMENCLATURA

Toda esta nomenclatura é para desenvolvimentos de i/400 Consulting. Qualquer solicitação do cliente deve ser respeitada.

- Evite o uso de símbolos **#** e **Ñ** em nomes, usá-los pode gerar um problema de conversão.

- Você deve usar as seguintes siglas em nomes de arquivo, variáveis, rotinas, etc. (lista não completa).

```
ACC: Ação
MOV: Movimento
BUS: Pesquisa
MTO: Montante
CAN: Quantidade
NEW: Nova (Atual)
CHG: Mudança
NOM: Nome
CLI: Cliente
NUM: Número
CPY: Cópia
OLD: Old (Anterior)
CRT: Criar
ORI: Original/Origem
CTA: Conta
RPC: Processo
DEA: desincorporar (Off)
REA: Restabeleça (Reativar)
DET: Detalhe
REG: Registrar
DIR: Direção
SND: Enviar
ENC: Cabeceira
SOL: Pedido
ERR: Erro
STT: Subtotal
GRA: Grave (registre)
TIP: Tipo
HIS: História
TOT: Total
INQ: Pesquisa (Consulta)
TRN: Transação
LOD: Carga
UPD: Atualização
MAX: Máximo
VAL: valor ou Validar (dependendo do uso)
MIN: Mínimo
VTA: Venda
```

- Ao criar uma abreviatura, esta deve permitir identificar o seu conteúdo.
  Exemplo: PMV – Preço Máximo de Venda.

- Para todos os nomes:

DESCRIÇÃO	INSTRUÇÃO
Perfil do Usuário	1. **Perfil do Usuário** deve ser identificado pela primeira letra do primeiro nome seguido pelo sobrenome do usuário.

Bibliotecas	**1.** Cada programador deve ter uma biblioteca nomeada de acordo com o seu perfil de usuário. **2.** Bibliotecas de projeto devem ser nomeadas como descrito: **PPPX*LIB*** Onde: PPP :Id projeto. X : O conteúdo da biblioteca F = Arquivos de Dados O = Objects S = Fontes LIB : Constante **3.** Para clientes com múltiplos projetos, uma biblioteca comum será criada para armazenar todas os fontes e objetos comuns e deve ser nomeada como segue: • ***CMM*CCC** Onde: CMM : Constante CCC : Código do Cliente usado no Sistema de Controle de Projetos (CTP).
Arquivos de Dados	**1.** Todos os arquivos de dados devem ser nomeados de acordo com a seguinte nomenclatura: • **Arquivos físicos: PPPXXXXX** Onde: PPP : Id Project. XXXXX : Mnemônico identificando seu conteúdo. • **Arquivos lógicos e índices: PPPXXX*L*Y** Onde: PPP : Id Project. XXX : Mnemônico identificando o arquivo físico associado. L : Constante. Y : Identificador adicional (1 a 9 e A a Z). **2.** Todos os Mnemônicos devem permitir identificar o conteúdo do arquivo. Exemplo: Client: Cliente, MOVLG: Movimentos Log, etc. **3.** Todos os nomes de campo em arquivos devem ser nomeados com um **mínimo** de 6 caracteres e respeitar as abreviaturas listadas acima. **4.** Variáveis com mesmo conteúdo devem ter o mesmo nome em todos os arquivos. Por exemplo: Nome do Cliente deve ser nomeado CLINAM em todos os arquivos e será responsabilidade do programador para controlar essa duplicidade nos programas.

Programas	**1.** Todos os programas devem respeitar a seguinte nomenclatura (exceto por solicitação do cliente): **PPPYYY:** Onde: PPP : ID de projeto YYY : Número consecutivo em intervalos (com incrementos de 5) *001-399* - Entrada de Dados e Manutenção de Cadastros *400-499* - Relatórios *500-699* - Tabelas de Manutenção *700-799* - Processos diários *800-899* – Processos Mensais e Anuais *900-999* - Processo de Monitores 900-999 - Utilitários
Programas de Serviços	**1.** Nomes de programas de serviço terão a seguinte nomenclatura: **PPP*SRV*** Onde: PPP : ID de projeto SRV : Constante **2.** Programas de serviço genéricos (não específicos de um projeto) devem ser nomeados de acordo com a suas funcionalidades. Exemplo: CMD - Gerencia funções relacionados com Comandos USRLST - Gerencia funções relacionadas com "Listas de Usuários"
Arquivos de Tela	**1.** Arquivos de tela devem ser nomeados de acordo com: **PPPPPP*FM*:** Onde: PPPPPP : Nome do Programa, onde o arquivo de tela é usado. FM : Constante **2.** Registros de arquivos de tela usarão a seguinte nomenclatura: • Registro de cabeçalho: **ENCAB -** Se existir apenas um registro de cabeçalho no arquivo ou uma combinação de mnemônicos para identificar seu conteúdo. (Exemplo: CLIHDR: Cabeçalho de Cliente). • Registro de Mensagem: **MESSAGE** • Registro de identificação: **IDENT** (Por exemplo, Identificação do número de pedido). • Subarquivos: **SFLxxx** - Onde xxx é um mnemônico que identifica seu conteúdo

	- **CTLxxx** (Controle **SFLxxx**). • Registro de Detalhe: **DETAIL** - Se existir apenas um registro de detalhe ou uma combinação de mnemônicos identificando seu conteúdo. **3.** Arquivos de tela usando a função de ajuda de conteúdo (F10/F4) devem ter associados os respectivos /Copy gerados pelo utilitário específico. O nome dos /Copy serão: **PPPPPP*SC***: Onde: PPPPPP : Nome do Programa onde o arquivo de tela associado é usado SC : Constante
Arquivos de impressora	**1.** Se houver apenas um arquivo de impressora em um programa, terá a seguinte nomenclatura: **PPPPPP*PR*** Onde: PPPPPP : Nome do Programa onde o arquivo de impressora é usado. PR : Constante **2.** Se um programa utiliza mais de um arquivo de impressora a nomenclatura a ser utilizada é: **PPPPPP*P*X** Onde: PPPPPP : Nome do programa onde o arquivo de impressora é usado. P : Constante X : Letra de A a Z para identificar cada arquivo de impressora no programa **3.** Nomes de registros em arquivos de impressora devem respeitar a seguinte nomenclatura: • Registro de cabeçalho **: HEADER** • Registro de detalhe **: DETAIL** • Registro de Total **: TOTAL** Se mais de um registro do mesmo tipo (cabeçalho, detalhe ou total) for necessário, um nome mnemônico que identifique o seu conteúdo deve ser usado.
Fontes /Copy	**1.** Sempre que possível, fontes /Copy devem ser usados para evitar a duplicidade de código e melhorar o tempo de manutenção. **2.** Todos os "arquivos flat" (sem as descrições externas ou do IFS) terão suas estruturas descritas em /Copy. Cada programa que faça referencia a esses arquivos deve usar o fonte /Copy apropriado.

	3. Os fontes /Copy abaixo listados serão criados em quase todos os projetos: • **PPP*PARM***: Área de Dados para Parâmetros Gerais Onde: PPP : ID de projeto PARM : Constante **4.** Protótipos de Procedimentos serão armazenados em fontes /Copy baseados nos módulos com a seguinte nomenclatura: **MMMMMMMMM*H*** Onde: MMMMMMMMM : Nome do Módulo H : Constante Exemplo: CvtH - Protótipos de Funções de conversão. **6.** Qualquer outro fonte /Copy será criado usando com um nome mnemônico que identifique seu conteúdo.
Variáveis do programa	**1.** Em um programa ILE RPG, as definições devem ter a seguinte ordem: 1. Protótipos 2. Arrays e Tabelas 3. Estruturas de Dados 4. Campos autônomos alfanuméricos 5. Campos autônomos numéricos 6. Ponteiros 7. Constantes **2.** Variáveis e Instruções em programas ILE RPG devem usar letras maiúsculas em primeiro lugar e qualquer outra letra significativa. Exemplo: **B**nk**E**rr : Erro Banco **C**li**N**am : Nome Do Cliente **V**al**D**ig**N**br : Número de dígitos válidos, etc. **É importante lembrar que os nomes em RPG ILE não estão limitados a 6 caracteres. É uma boa prática a utilização de nomes extensos para identificar campos.** **No exemplo anterior, será melhor usar ClientName, BankError e ValidDigitsNumber.** **3.** Variáveis de arquivo devem estar todas em maiúsculas para diferenciar de variáveis internas. Exemplo: NomCli = CUSNA1 **4.** Variáveis criadas dentro do programa devem respeitar a seguinte

	nomenclatura: **Prefixos:** **$** : Variáveis de trabalho imediato: Usada em um ou duas instruções consecutivas e podem ser usados em qualquer parte do programa (resultados de multiplicar, concatenação, divisão, etc.) (Exemplo $WK152 - Variável de trabalho com 15 dígitos e 2 decimais). **W** : Variáveis de trabalho mais duráveis tais como acumuladores, resultados que devem ser guardados para uma atualização, etc. **@** : Variáveis de Parâmetro - Instrução PARM, Campo de Interface de Procedimento (PI) ou qualquer variável usada exclusivamente como um parâmetro recebido pelo programa mesmo ou para chamar a um programa ou procedimento. **K** : As variáveis-chave para acessar os arquivos. **k** : Constantes Nomeadas **Arr** : Arrays **Ind** : Indicadores (Variáveis Tipo N ou definida como caractere, mas usada como um indicador (true / false)) **Pgm** : Constantes nomeadas para programas chamados pelo próprio programa **Ds** : Estruturas de Dados. Pode preceder um nome de arquivo para uma estrutura de dados definida externamente **5.** Qualquer variável precisando ser renomeada para evitar duplicidade deve ter um prefixo (minúscula) de acordo com: **s:** Para uma variável de subarquivo. Por exemplo **CLINAM** se torna **sCliNam**. **r:** Para um arquivo de registro de exibição ou impressora. Exemplo **IDENT** se torna **rIdent**. Qualquer outro prefixo pode ser utilizado para mudar o nome de variáveis usando este método de prefixo em minúscula.
Indicadores	**1.** Os indicadores devem ser utilizados eficientemente evitando um número elevado. **2.** Devem ser usados de acordo com os seguintes intervalos: ***01 24*** : Condicionamento de teclas de Função F1 a F24

	25 a 26 : Condicionamento de teclas PageDown/PageUp 25 : PageDown 26 : PageUp 27 : F4 ou F10 (ajuda de conteúdo) indicador de resultado 28 : Indicador de resultado PageDown 29 : Indicador de resultado PageUp ***30 a 49*** : Posicionamento de Cursor em arquivos de tela 51 : SFLNXTCHG ***60 a 79*** : Condicionamento de Campos de arquivos de tela e de impressora: 60 : MDT 61 : Proteção de Campos de Chave ***80 a 87*** : Controle de Subarquivos (em pares 80/81, 82/83, 84/85, 86/87) 80 : SFLDSPCTL N80 : SFLCLR/SFLINZ 81 : SFLDSP+SFLEND ***88 A 89*** : Overflow (PRTF) ***90 a 99*** Erros: 97 : Modificação de arquivo ou registro (Palavra-chave CHANGE) 99 : Condicionamento de Campo de Mensagem
Sub-rotinas e procedimentos	**1.** Sub-rotinas e procedimentos devem ser nomeados usando o método de Maiúsculas e Minúsculas. Os nomes devem ser significativos de acordo com a funcionalidade de sub-rotina. Exemplo: **LodSfl** **:** Carga Subarquivo **Valdet** : Validar Detalhe, etc. **2.** Nomes mais usados em programas de manutenção são: • **InqRecord : Pesquisa de registro** • **DeaRecord : Desativação de registro** • **CrtRecord : Criar registo** • **ReaRecord : Reativar registro** • **UpdRecord : Atualização de registro** • **UpdFiles : Atualização de arquivos** • **PrcF10 : Processo da tecla F10 (F4)** **3.** A instrução EndSr deve ter um comentário na mesma linha com o nome da sub-rotina. Exemplo: `EndSr; // LodRecord`
Lista de Parâmetros	**1.** Em "Free", apenas Protótipos (PI e PR) podem ser usado para definir parâmetros.

	2. A definição "PR" (Protótipo) devem ser armazenada em um fonte /Copy. **3.** A definição "PR" deve ser nomeada de acordo com a sua funcionalidade (e para programas, não necessariamente deve coincidir com o Nome do Programa). Exemplo: `D AccountList Pr ExtPgm ('NNN105')`
Lista de Chaves	**1.** Todas as listas de chaves (instrução KList) devem ser nomeadas como segue: Prefixo + Nome do arquivo + Sufixo. Os possíveis Prefixos são: **KC**: Chave Completa (Todos os campos chave estão na lista) **KP**: Chave parcial (Só x primeiros campos-chave estão na lista) Sufixos são opcionais e devem ser usados para identificar a sua utilização. Exemplos: KCSrbCbn : Chave completa para arquivo SRBCBN. KPRinLot : Chave parcial para RINLOT(Apenas uma chave parcial definido no programa). KPRinLotOri : Chave parcial para RINLOT com os dados da chave original (mais de uma chave parcial definida para o arquivo no programa).
Arquivos de Mensagens	**1.** Arquivos de mensagem serão criados respeitando a seguinte nomenclatura: **PPP*MSGF*** Onde: PPP: ID de projeto MSGF: Constante **2.** IDs de mensagens usarão: **PPPYYYY:**Emitido por programas Onde: PPP : Projeto YYYY : Consecutivo por intervalos: *0001 – 0999* - Mensagens genéricas (Data inválida, Campo Obrigatório, etc.) *1000 – 2999* - Mensagens específicas utilizadas em programas de validações. *9000 – 9999* - Mensagens CL (LE) *9898* - Mensagem genérica - contém uma única variável de 256 caracteres. ***VAL*YYYY:** Emitidas pelo Sistema Operativo (validações SDA) Onde:

	VAL : Constante YYYY : Consecutivo por intervalos: *0001 - 9999* - Mensagens de genéricos ***SCR*YYYY:** Mensagens de Títulos de telas Onde: SCR : Constante YYYY : Consecutivo por intervalos *0001 - 9999* – Títulos de telas
Grupos de painéis	**1.** Painéis de Grupos de Ajuda respeitarão a seguinte nomenclatura: **PPP*HLP*:** Painel de Grupo Principal Onde: PPP : ID de projeto HLP : Constante **PPPPPP*HLP*:** Painel de Grupo específico a Arquivo de Tela Onde: PPPPPP : Programa (Arquivo de Tela) HLP : Constante **2.** Cada Rótulo de Ajuda deve usar a seguinte nomenclatura: **XXXXXX/Y:** Rótulo de texto Ajuda Onde: XXXXXX : Nome do campo para o qual a ajuda está associada / : Constante Y : Indica se o texto é para um campo de SAÍDA (**O**) ou de ENTRADA (**I**) Exemplo: CLINAM/O: Este rótulo será associado ao campo CLINAM que é definido como de SAÍDA unicamente.
Menus UIM	**1.** Menus do tipo UIM devem usar a seguinte nomenclatura: **PPP*MENU*** Onde: PPP : ID de projeto MENU : Constante
Diretórios Bind	**1.** Diretórios "bind" devem ser nomeados de acordo com: **PPP*BNDDIR*** Onde: PPP : ID de projeto BNDDIR : Constante
"Bind language"	**1.** Cada Programa de Serviço deve ter seu fonte "Bind Language"

	associado. **2.** Estes fontes devem ser nomeados de acordo com a seguinte nomenclatura: **PPPPPPP*BND*** Onde: PPPPPPP : Nome do Programa de Serviço BND : Constante **3.** Quando o nome de Programa de Serviço tiver mais de 7 caracteres, os caracteres menos significativos devem ser removidos para respeitar a parte constante.
Programas gatilho	**1.** Programas gatilho usarão a seguinte nomenclatura: **PPP*T*XFFFFF** Onde: PPP : ID de projeto T : Constante X : Tipo de Gatilho H = Geração de Histórico (log) D = Eliminações I = Inserções M = Atualizações FFFFF : Arquivo ao qual o gatilho está amarrado

2.3 -. PROGRAMAÇÃO EFICIENTE

Comentários:

A documentação de um programa é essencial para uma programação eficiente já que permite ter uma melhor compreensão da lógica de um programa e menor tempo para executar a manutenção.

Entre as melhores práticas para criar comentários em um programa devemos:

1. Evitar o uso de comentários na mesma linha de declaração. Coloque-o antes das instruções em uma linha separada.
2. Adicione comentários para esclarecer o código não para refletir o código
   - Evitar

```
//->01 If Var = 5
If Var = 5;
```

   - Use algo mais explicativo

```
//->01 If variável Nível é New
If Var = kNew; (Aqui a Constante nomeada kNew tem valor 5)
```

3. Explicar extensivamente todo o processo que possa ser difícil de ser entendido por outra pessoa.

4. Documente passos simples com comentários simples.

5. Documente ***TODAS*** as instruções condicionais/bloco (If, Else, For, Select/When, etc.)

6. Coloque explicitamente entre comentários todas as instruções que impliquem em uma mudança de fluxo de lógica (Leave, LeavSr, Return, ExSr, etc.).

```
//*****
LeaveSr;
//*****

//** ======
ExSr PrcF10;
//** ======

//====
Return;
//====
```

Modelos:

1. Dependendo do tipo de programa (interativo ou em lotes), existe um programa "modelo" que pode ser usado como uma base para todos os programas a serem desenvolvidos e devem ser usados sempre que possível. Esses modelos são o resultado de anos de aperfeiçoamento das técnicas de programação que provaram ser eficientes em tempo de desenvolvimento e manutenção.

Cláusulas /Copy e /Include:

1. Esta função permite copiar fontes de origem externa em um programa em tempo de compilação. As principais vantagens da utilização são a de evitar a duplicação de código e a normalização.

2. É amplamente utilizado em:
   - Definições de Protótipos
   - Descrições de "Arquivos Flat"
   - Definições de Estruturas de Áreas de Dados (DtaAta)
   - Listas de Parâmetros (PList)
   - Descrições de Estruturas de Entradas de Filas de Dados (Data Queues)
   - Descrições de Estruturas de Dados Especiais (SDS, "Feedback", LDA, etc.)
   - Descrições de Estruturas de Dados (Parâmetros, dados comum, etc.)

Descrições de Variáveis:

1. Todas as variáveis, sem exceção, devem ser definidas na Especificação de Definição (D).

2. Sempre que possível, use a palavra-chave *Like* para definir uma variável baseada em outra, definida em um arquivo (devem estar relacionadas). Por exemplo, variáveis de trabalho para totais, contagem, variáveis temporárias, variáveis-chave do arquivo, etc.

3. Ao definir uma variável, defina explicitamente o seu tipo.

```
D $TotalAmount S 15P 2 // P é importante para a leitura
                       mesmo que seja implícito para
                       a definição de um campo packed
```

4. Sempre que possível, use variáveis Integer (I) (3,0, 5,0, 10,0 ou 20,0) ou packed. Utilizam menos memória e são tratados melhor pelo sistema operacional.

5. Use auto-inicialização de variáveis (palavra-chave "INZ") sempre que possível. Isso evitará instruções adicionais.

6. Os nomes das variáveis devem começar na posição 8 (7 deve estar em branco)

7. Cada nome de subcampo deve começar na posição 8 + n, onde n é o nível do subcampo
   Exemplo:

```
D DsEjemplo Ds          (Um único branco entre D e o nome da
                        DS)
D  Subcampo     8S 0    (dois espaços em branco (primeiro
                        nível subcampo))
D   SubSubCampo 4S 0    Overlay (SubCampo)
                        (Três espaços em branco (segundo nível
                        subcampo))
```

8. As seguintes variáveis listadas são "standard" e são/devem ser utilizadas por quase todos os programas interativos e devem ser respeitadas.

$LEVEL	I (5,0)	: Nível de Tela
MSGID	A (4)	: Código de Mensagem (variante WMSGID)
RRN	S (4,0)	: Número de Registro Relativo no subarquivo (variantes RRN1, RRN2, etc.)
SOPC	A (2)	: Campo de Seleção em subarquivo (variantes SOPC1, SOPC2, etc.)

Constantes:

1. Utilize apenas constantes nomeadas. Isso permite uma melhor legibilidade e fácil manutenção.
   Exemplo:
   `Name = "*";` (Isso não diz muito)

   `Name = kVariableName;` (kVariableName é uma constante nomeada com valor '*'. O código é um pouco mais extenso, mas mais fácil de entender)

2. Não use a palavra-chave CONST nas definições de constantes nomeadas.

3. Utilize Constantes Especiais **Blanks*, **Zeros*, **All'x'* (e outros) para inicializar ou comparar campos.

4. Use a instrução *Clear* para inicializar os campos com seu valor de base.

5. Use somente as constantes especiais **On* e **Off* para ligar, desligar e testar indicadores e variáveis lógicas.

Indicadores:

1. Se não for possível renomeá-los, use apenas as variáveis *Inxx e o array *In(x) para referir-se aos indicadores.

2. Ao inicializar múltiplos indicadores consecutivos, utilize %SubArr.
   Exemplo:
   ```
   % SubArr(*In : 30 : 10) = *Off;
   ```
   Esta única declaração apaga o conjunto de indicadores 30 a 39.

3. Use indicadores quando realmente seja necessário. Se possível, use uma variável de trabalho (tipo N) como indicador.
   Exemplo:
   A variável IndNewClient pode conter *On se é um novo cliente e *Off em caso contrário.

4. Use expressões lógicas para atribuir valores a um indicador ou variável tipo lógica
   Exemplo:
   ```
   *IN99 = (MsgId <> *Blanks);
   ```

5. Use variáveis de indicador diretamente pelo seu valor e evite a comparação
   Exemplo:
   ```
   If *In97             // (Em vez de *In97 = *On)
   If Not *In60         // (Em vez de *In60 = *Off)
   ```

Lista de Chaves (KList):

1. Use Klists diferentes, conforme seja necessário. Evite definir uma Klist genérica. Isso forçará a fazer um ou varios Eval antes de acessar os arquivos.

Instruções estruturadas:

1. Utilize Instruções Estruturadas de RPG (If, Select, Do, For, etc.) extensivamente. É fácil de manter e ler.

2. NÃO use a instrução END (genérica). Utilize apenas instruções as significativas (EndIf, EndSl, EndDo, etc.)

Documentando Instruções Estruturadas:

1. Para fazer uma documentação simples de Instruções Estruturadas (If, When, Do, etc.), utilizaremos os comentários com um formato específico. Isso nos indicará o nível de cada bloco e permitiár uma fácil leitura e manutenção.

Um comentário deve ser usado ANTES do bloco começar e depois de cada instrução de bloco (Else, Other, etc.).

2. Vamos usar o formato **//->xx** para indicar o nível de cada ninho, começando na mesma coluna da instrução.
   xx representa o nível e deve conter sempre dois caracteres (01, 02, etc.).
   Exemplos:

```
//->01 If deve chamar a Calculadora (Início do primeiro nível)
If *InKE;
   //->02 If NO existe número associado (Início do segundo nível)
   If IlsNso = * Zeros;
      MsgId = ErrAssociateNumMissing;
      // =====
      LEAVESR;
      // =====
   EndIf;
   //->02 EndIf (Fim do segundo nível)
EndIf;
//->01 EndIf (Fim do primeiro nível)
```

Goto/Tag:

1. As instruções GOTO e TAG não podem ser usadas em "free" pelo que será necessário projetar o programa para usar DO, DOW, DOU, EXSR, EVAL (usando um procedimento), LEAVESR, RETURN ou LEAVE segundo as necessidades.

Leitura de Arquivos, bloqueio de registros:

1. Evite bloqueios desnecessários a registros de dados. Use a opção "No Lock" (N) nas instruções de acesso a arquivos (Read(N), Chain(N), etc.) na primeira vez que ler um registro e faça-lo novamente, sem a opção, antes de atualizá-lo.

2. Evite a transferência de dados desnecessários do disco para a memória (programa). Ao fazer apenas uma validação de chave, onde não são necessários os dados do arquivo, use a instrução SETLL para verificar a existência da chave.
   Os dados não serão lidos e não serão transferidos a partir do disco para a memória.
   Exemplo:

```
// Valida chave no arquivo
Setll KCChave Arquivo;
//->01 Se não encontra a chave
If Not %Equal();
   (gerenciar não existência)
EndIf;
//->01 EndIf
```

3. Evite usar as instruções READPE e READP se muitos registros são lidos de uma só vez. Consume muitos recursos.

4. Para fazer um loop para ler um arquivo, evite a instrução ReadE se espera ler muitos registros de uma vez só. Em vez disso, use Read e controle o grupo (chave) através de comparações no loop.

5. Ao fazer referência a um arquivo em um programa (Read, Update, Chain, etc.), faça-lo usando o Nome do Registro de Formato e não o Nome do Arquivo (quando permitido).

6. Use as funções internas EOF%(), %FOUND(%) e %EQUAL() para validar o acesso aos arquivos. NÃO USAR INDICADORES.

7. Nas funções internas anteriores, não use o nome de arquivo como parâmetro quando o arquivo foi o último em ser acessado. Utilize apenas um nome de arquivo se você quiser se referir a um resultado de acesso a arquivo em outra parte do fluxo e/ou possa causar problemas com a leitura ou uma validação adicional.

Arquivos de Mensagem:

1. Use arquivos de mensagens extensivamente. Manter esses objetos é muito simples e rápido e permite padronização nas mensagens.

2. Use mensagens em títulos em arquivos de tela quando este possa ser usado por mais de um programa (basicamente CLPs). Isso permitirá que um arquivo de tela possa ser utilizado por vários programas.

Estruturas de Dados - Múltiplas ocorrências / Array:

1. Sempre que possível e o programa mantenha um tamanho razoável na memória, use Estruturas de Dados de múltiplas ocorrências (ou array) para manter os dados na memória, evitando acessos ao disco.

2. Elas podem ser usadas para armazenar duas versões de um conjunto de variáveis. Por exemplo, as variáveis no arquivo de tela podem ser uma ocorrência de uma DS e as variáveis de no arquivo em disco outra ocorrência.

Sub-rotinas/Procedimentos:

1. Use Procedimentos em vez sub-rotinas nos programas.

2. Um programa estruturado não é um programa com sub-rotinas ou procedimentos. Use sub-rotinas/procedimentos quando seja realmente necessário ou quando seja chamada mais de uma vez dentro do programa.

3. * INZSR - Use esta sub-rotina especial para inicializar campos, fazer validações nos parâmetros passados, etc. Só é executada automaticamente na primeira vez que o programa é chamado ou usando a instrução EXSR.

4. Definir a sub-rotina *INZSR pouco antes última em um programa. (A última deve ser *PSSR)

5. Definir sub-rotinas na mesma ordem que elas são referenciados.

6. *PSSR - Use esta rotina especial para assumir o controle de erros inesperados do programa.