i/400 Consulting

Always Evolving

TECHNICAL SPECIFICATIONS FOR PROGRAMMING (In RPG ILE/free)

CONTENT

OBJECTIVE

SCOPE

CHAPTER I: GENERAL RULES FOR SYSTEMS DEVELOPMENT IN RPG ILE (Free)

CHAPTER II: TECHNICAL SPECIFICATIONS.

2.1. FOR SYSTEM DEVELOPMENT
2.2. FOR NOMENCLATURE TO BE USED
2.3. FOR EFFICIENT PROGRAMMING

OBJECTIVE

This document aims to establish standards and technical specifications to be followed by employees of i/400 Consulting in RPG ILE language in "Free" format.

SCOPE

Maintain a standard for systems' development, nomenclature to be used and programming techniques, resulting in:

- A faster system development
- ease of maintenance of programs and
- ease programs reading.

CHAPTER I: GENERAL RULES FOR SYSTEMS DEVELOPMENT IN RPG ILE "FREE".

1. Program's appearance is very important and the writing rules must be respected. Therefore it's necessary to follow the instructions given here for systems' development.

2. Each employee is responsible for respecting libraries assigned to the project, according to specifications described in chapter II on this document.

3. All finished program must be submitted to project leader who will be responsible for additional testing and approval.

4. Project leader is responsible for maintaining control over files and objects in libraries, so other participants must request him any replacement, creation or suppression.

5. An interactive program will be considered finished when corresponding help panels (in UIM) are ready and tested (when applicable).

6. All developed program must be documented with their respective "technical description datasheet" which must be stored in the corresponding project's folder.

7. All developed screens must comply with the **IBM SAA Standards**, (unless client requested otherwise).

8. Pass to Production must respect instructions given in the "Pass to Production format".

CHAPTER II: TECHNICAL SPECIFICATIONS.

2.1. SYSTEM DEVELOPMENT

- Each Project will have in its source library all standard source files used by IBM plus some specific of i/400 Consulting:

QRPGLESRC	- RPG ILE Sources
QCMDSRC	- Commands Sources
QDDSSRC	- Files, screens and reports Sources
QCLLESRC	- CL ILE Sources
QUIMSRC	- UIM panels Sources
QCPYSRC	- /Copy and /INCLUDE Sources
QSRVSRC	- Service Programs and Bind Language Sources

- All projects must have a set of three (3) libraries: One for Sources, one for Data Files and one for Objects. The names of these libraries should be defined according to the nomenclature defined in section 2.2 further in this document. When the project does not have enough sources and objects, the project leader may choose to use a single library for sources, files and objects.

- Interactive Compilations MUST be avoided. All compilations must be submitted in BATCH.

- All programs: Interactive or batch, must respect the logic of the appropriate type template.

- Instructions in the RPG ILE and CLLE programs must be written in a Upper/Low Case (mixed) format (see "variable names" detailed below).

- Nested statements should be indented three (3) characters (the three dots indicate the number of blank characters and are for display purposes only).
  Example:

```
//->01 If Exit
If *InKg;
...Clear $Level;
...//====
...LeaveSr
...//====
...EndIf;
//->01 EndIf
```

- Try to aligning assignment statements by the equal sign (=).
  Example:

```
@Revenue                = RpfIng;
@EstabilidadLaboral     = RpfELb;
```

```
@Debt                  = RpfPEn;
@RelacionCuotaIngresos = RpfPCI;
@ValorMinimoGarantia   = RpfVMG;
```

- Use a blank line between statements to improve readability when really necessary.
  Example:

```
//Search Range Values for each field

KValorHasta = @Age;

// Rate term
KKey = kRateTerm;
Setll KCApclsXFIR RApclsXFIR;
ReadE KPApclsXFIR RApclsXFIR;
//->01 If Not Found
If Not %Eof();
```

- Use one line for each logical expression finishing line by the logical operator to improve reading.
  Example:

```
//->01 If not loaded first or last name
If FirstName = *Blanks And
   LastName  = *Blanks;
   Error;
EndIf;
//->01 EndIf
```

- For calls to programs or procedures using prototypes, set ALL parameters, if they fit, on a single line, otherwise, place each one on separate lines.
  Example:

```
// Search User Data
CallP RtvCusCun(CusCun  :
                @Cusna1 :
                @Cusna2 :
                @Cusna3 :
                @Cusna4 );

/ / Search Account Data
CallP RtvAccRcd(Account : PtrAcMst)
```

2.2 - . NOMENCLATURE

All nomenclature is for i/400 Consulting developments. Any Client request must be respected.

- Avoid using symbols # and Ñ in names, using them can become conversion issues.

- You must use the following abbreviations in File Names, Variables, Routines, etc. (not exhaustive list).

```
ACC : Action
MOV : Movement
BUS : Search
MTO : Amount
CAN : Quantity
NEW : New (Actual)
CHG : Change
NOM : Name
CLI : Client
NUM : Number
CPY : Copy
OLD : Old (Previous)
CRT : Create
ORI : Original/Origin
CTA : Account
PRC : Process
DEA : desincorporar (Off)
REA : Reinstate (Reactivate )
DET : Detail
REG : Register
DIR : Direction
SND : Send
ENC : Header
SOL : Request
ERR : Error
STT : Subtotal
GRA : Record
TIP : Type
HIS : History
TOT : Total
INQ : Inquiry
TRN : Transaction
LOD : Load
UPD : Update
MAX : Maximum
VAL : Value or Validate (depending on use)
MIN : Minimum
VTA : Sale
```

- When creating an abbreviation, it must allow identify its content.
  Example : MSP - Maximum Sales Price .

- For all names:

DESCRIPTION	INSTRUCTION
User Profiles	**1.** **User Profiles** must be identified by the first letter of first name followed by user's last name.
Libraries	**1.** Each programmer must have a library named according to his User Profile.

	2. Project Libraries must be named as described: • **PPPX*LIB*** Where: PPP : Project Id. X : Library content F = Data Files O = Objects S = Sources LIB : Constant **3.** For clients with multiple projects, a common library will be created to store all common sources and objects and must be named as follow: • ***CMM*CCC** Where: CMM : Constant CCC : Client Code used on Project Control System (CTP).
Data Files	**1.** All data files must be named according the following nomenclature: • **Physical Files: PPPXXXXX** Where: PPP : Project Id. XXXXX : Mnemonic identifying it's content. • **Logical Files and Indexes: PPPXXXLY** Where: PPP : Project Id. XXX : Mnemonic identifying the associated physical file. L : Constant. Y : Additional identifier (1 to 9 and A to Z). **2.** All Mnemonics must identify file's content. Example**: CUST**: Customer, **MOVLG**: Movements Log, etc. **3.** All Field Names in Files must be named with a **minimum** of de 6 characters and respect abbreviations listed above. **4.** Variables with same content must have the same name across all files. For example Client Name must be named **CLINAM** in all Files and will be programmer responsibility to manage duplicity in programs.
Programs	**1.** All Programs must respect the following nomenclature (except by client request): **PPPYYY:** Where: PPP : Project Id YYY : Consecutive number in a range (with increments by 5)

	001 - 399 – Data Entry and Master Files Maintenance *400 - 499* - Reports *500 - 699* – Tables Maintenance *700 - 799* – Daily process *800 - 899* – Monthly and Annual process *900 - 999* – Monitors process 900 - 999 – Utilities
Service Programs	**1.** Service programs names will use the following nomenclature: **PPP*SRV*** Where: PPP : Project Id SRV : Constant **2.** Generic Service Programs (not project specific) must be named according its functionality. Example: CMD – Manages Commands related functions USRLST – Manages "User Lists" related functions
Display Files	**1.** Display Files must be named according to: **PPPPPP*FM***: Where: PPPPPP : Program Name where Display File is used FM : Constant **2.** Display Files Records will use the following nomenclature: • Header Record: **HEADER –** If only one header record in the file or a combination of mnemonics to identify its content. (Example: **CLIHDR**: Client Header). • Message Record: **MESSAGE** • Id Record: **IDENT** (for example, Id Number request). • Subfiles: **SFLxxx** - where xxx is a mnemonic identifying it's content - **CTLxxx** (SFLxxx Control). • Detail Record: **DETAIL** - If just one detail record or a combination of mnemonics identifying its content.

	3. Display Files using Content Help function (F10/F4) must have the associated /COPY generated by the specific utility. /Copy name will be: **PPPPPPSC:** Where: PPPPPP : Program Name where associated Display File is used SC : Constant
Printer Files	**1.** If there is just ONE Printer File in a program it will use the following nomenclature: **PPPPPP*PR*** Where: PPPPPP : Program Name where printer file is used. PR : Constant **2.** If a program uses more than one Printer File the nomenclature to be used is: **PPPPPP*PX*** Where: PPPPPP : Program name where printer file is used. P : Constant X : Letter from **A** to **Z** to identify each printer file in program **3.** Record names in Printer Files must respect the following nomenclature: • **Header record : HEADER** • **Detail record : DETAIL** • **Total record : TOTAL** If more than one record kind (header, detail or total) is necessary, a mnemonic name identifying its content must be used.
/Copy members.	**1.** Whenever possible, /COPY members must be used to avoid code duplicity and improve maintenance time. **2.** All "flat files" (without external descriptions or on IFS) will have their structure described in /Copy members. Each program referencing those files must use the appropriate /Copy member. **3.** The below listed /Copy members will be created on almost all projects: • **PPP*PARM***: Data Area for General Parameters Where: **PPP** : Project Id PARM : Constant **4.** Procedures Prototypes will be stored in /Copy members in a Module basis with the following nomenclature: **MMMMMMMMM*H*** Where:

	MMMMMMMMM : Module Name H : Constant Example: CvtH – Conversion functions Prototypes. **6.** Any other member will be created using a mnemonic name identifying its content.
Program Variables	**1.** In an **RPG ILE** program, definitions must have the following order: 1. Prototypes 2. Arrays and Tables 3. Data Structures 4. Alphanumeric standalone fields 5. Numeric standalone fields 6. Pointers 7. Constants **2.** Variables and Instructions in RPG ILE programs must use capital letters on first and any other significant letter. Example: **B**nk**E**rr : Bank Error **C**li**N**am : Client Name **V**al**D**ig**N**br : Valid Digits Number, etc. **It's important to remember that in RPG ILE names are NOT limited to 6 characters. It's a good practice to use extended names to identify fields.** **In previous example it will be better to use ClientName, BankError and ValidDigitsNumber.** **3.** File's variables must be all in UPPER case to differentiate from internal variables. Example: NomCli = CUSNA1 **4.** Variables created in the program must respect the following nomenclature: **Prefixes:** **$** : Immediate Work Variables: Used in one or two consecutives instructions and can be used anywhere in the program (results of multiply, concatenation, division, etc.) (Example $WK152 - Work variable with 15 digits and 2 decimals). **W** : More durable Work variables such as accumulators, results to be saved for an update, etc. **@** : Parameter Variables - PARM instruction, Procedure Interface (PI)

	field or any variable used exclusively as a parameter received by the program itself or to call a Program or Procedure. **K** : Key Variables to access files. **k** : Named Constants **Arr** : Arrays **Ind** : Indicators (type N variables or defined as character but used as an indicator (true/false)) **Pgm** : Named Constants for Programs called by the program itself **Ds** : Data Structures. Can precede file name for an externally defined Data Structure **5.** Any variable needing be renamed to avoid duplicity must have a prefix (low case) according to: **s:** For a SubFile variable. For example **CLINAM** becomes **sCliNam**. **r:** For a Display or Printer Record file. Example **IDENT** becomes **rIdent**. Any other prefix can be used to rename variables using this low case prefix method.
Indicators	**1.** Indicators must be used efficiently avoiding a high number of it. **2.** Must be used by the following ranges: ***01 a 24*** : Function F1 a F24 key conditioning ***25 a 26*** : PageDown/pageUp keys conditioning 25 : PageDown 26 : pageUp 27 : F4 or F10 (content help) result indicator 28 : PageDown result indicator 29 : PageUp result indicator ***30 a 49*** : Cursor Positioning in Display Files 51 : SFLNXTCHG ***60 a 79*** : Field conditioning in Display and Printer Files: 60 : MDT 61 : Key Fields Protection ***80 a 87*** : Subfiles Control (in pairs 80/81, 82/83, 84/85, 86/87) 80 : SFLDSPCTL N80 : SFLCLR/SFLINZ 81 : SFLDSP + SFLEND

	88 a 89 : Overflow (PRTF) ***90 a 99*** : Errors 97 : Record or File Change (CHANGE keyword) 99 : Message field conditioning
Subroutines and Procedures	**1.** Subroutines and Procedures must be named using the Upper/Low Case method. Names must be significant according to subroutine functionality. Example: **LodSfl** : Load Subfile **ValDet** : Validate Detail, etc. **2.** Most used names in maintenance programs are: • **InqRecord : Inquiry record** • **DeaRecord : Deactivate record** • **CrtRecord : Create record** • **ReaRecord : Reactivate record** • **UpdRecord : Update record** • **UpdFiles : Update Files** • **PrcF10 : Process F10(F4) Key** **3.** The EndSr instruction must have a comment in the same line with the Subroutine's name. Example: `EndSr; // LodRecord`
Parameter List	**1.** In Free, only Prototypes (PI and PR) wil be used to define parameters. **2.** The "PR" definition (Prototype) must be stored in a /Copy member. **3.** The PR must be named according its functionality (and not necessarily must match Program Name for programs). Example: `D AccountList Pr ExtPgm('NNN105')`
Key List	**1.** All Key Lists (KList instruction) must be named as follow: Prefix + File Name + Sufix. Posibles prefixes are: **KC**: Complete Key (All key fields are in the list) **KP**: Partial Key (Only x first key fields are in the list) Sufix are optional and must be used to identify its use. Examples: KCSrbCbn : Complete Key for file SRBCBN. KPRinLot : Partial Key for file RINLOT (Just one partial key defined in program). KPRinLotOri : Partial key for file RINLOT with original key

	data (more than one partial key defined for the file in the program).
Message Files	**1.** Message Files will be created respecting the following nomenclature: **PPP*MSGF*** Where: PPP: Project Id MSGF: Constant **2.** Message ids will use: **PPPYYYY:** Issued by programs Where: PPP : Project YYYY : Consecutive by ranges: *0001 - 0999* – Generic Messages (Invalid Date, Field Required, etc.) *1000 - 2999* – Specific Messages used on programs validations. *9000 - 9999* – CL(LE) messages *9898* - Generic Message – Contains an unique 256 characters variable. ***VAL*YYYY:** Issued by OS (SDA validations) Where: VAL : Constant YYYY : Consecutive by ranges: *0001 - 9999* - Generic Messages ***SCR*YYYY:** Title Screen Messages Where: SCR : Constant YYYY : Consecutive by renages *0001 - 9999* – Screen Titles
Panel Groups	**1.** Help Panel Groups will respect the following nomenclature: **PPP*HLP*:** Main Panel Group Where: PPP : Project Id HLP : Constant **PPPPPP*HLP*:** Display File Specific Panel Group Where: PPPPPP : Program (Display File) HLP : Constant **2.** Echa Help Label must use the following nomenclature:

	XXXXXX/Y: Help text label Where: XXXXXX : Field's Name for which help is associated / : Constant Y : Indicates if text is for an OUTPUT(**O**) or INPUT(**I**) field Example: CLINAM/O: These label will be associated to CLINAM field which is defined as OUTPUT only.
UIM Menus	**1.** UIM type Menus must use the following nomenclature: **PPP*MENU*** Where: PPP : Project Id MENU : Constant
Bind Directories	**1.** Bidn Directories wil be named according to: **PPP*BNDDIR*** Where: PPP : Project Id BNDDIR : Constant
Bind Language	**1.** Each Service Program must have its associated Bind Language member. **2.** These members must be named according to the following nomenclature: **PPPPPPP*BND*** Where: PPPPPPP : Service Program Name BND : Constant **3.** When Service Program's Name has more than 7 characters, less significant characters must be removed to respect the constant part.
Trigger Programs	**1.** Trigger Programs will use the following nomenclature: **PPP*T*XFFFFF** Where: PPP : Project Id T : Constant X : Trigger Type H = Log generation D = Deletions I = Adds M = Updates FFFFF : File which trigger is attached to

2.3 - . EFFICIENT PROGRAMMING

Comments:

Documenting a program is essential for an efficient programming since it allows a better understanding program's logic and less time to perform maintenance.

Among the best practices to create comments in a program we should:

1. Avoid using comments on the same statement line. Place it before instructions in a separate line.
2. Add comments to clarify the code not to reflect the code
   - Avoid

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

   - Use something more explanatory

```
//->01 If Level variable is New
If Var = kNew; (here named constant kNew has value 5)
```

3. Explain extensively all process that can be difficult to be understood by someone else.

4. Document simple steps with simple comments.

5. Document all conditional/block instructions (If, Else, When, For, Select/When, etc.)

6. Place all instructions implying in a logic flow change (Leave, LeavSr, Return, ExSr, etc.) explicitly between comments.

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

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

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

Templates:

1. Depending on program's type (interactive or batch) there is a "template" program that can be used as a base for all programs to be developed and should be used whenever possible. These templates are the result of years of improving programming techniques that proved to be efficient in development time and maintenance.

Clauses /Copy and /Include:

1. This function allows you to copy external source members in your program at compilation time. The main advantage of using it is to avoid code duplication and standardization.

2. It is extensively used in:
   - Prototypes Definitions
   - Flat Files Descriptions
   - Data Areas Structures Descriptions
   - Parameters lists
   - Data Queues Entries Descriptions
   - Special Data Structures Descriptions (SDS, Feedback, LDA, etc.)
   - Data Structures Descriptions (Parameters, Common Data, etc.)

Variables Definitions:

1. All Variables without exception must be defined in Definition Specification (D).

2. Whenever possible, use the keyword *Like* to define a variable based on an another one defined in a file (they must be related). For example, work variables for totals, counting, temporary variables, File key variables, etc.

3. When defining a variable, explicitly set its type.

```
D $TotalAmount   S    15P 2 // P is important for reading
                             even if it's implicit for
                             defining a packed field
```

4. Whenever possible, use Integer variables (I) (3,0, 5,0, 10,0 or 20,0) or packed. They use less memory and are best handled by OS.

5. Use auto-initialization on variables (keyword "INZ") whenever possible. This will avoid additional instructions.

6. Variable names must start at position 8 (7 must be blank)

7. Each subfield name must start at position 8 + **n** where **n** is the subfield level
   Example:

```
D DsEjemplo     Ds            (A single blank between D and
                              the DS name)
D  Subfield               8S 0 (Two blanks (first level
                              subfield))
D   SubSubCampo           4S 0 Overlay(Subfield)
                              (Three blanks(second level
                              subfield))
```

8. The following listed variables are "standard" and are/must be used by nearly all interactive programs and must be respected.
   $LEVEL I (5,0) : Screen Level

MSGID	A (4)	: Message Code (variant WMSGID)
RRN	S (4,0)	: Relative Record Number on SubFile(variants RRN1, RRN2, etc.)
SOPC	A (2)	: Selection Field in SubFile (variants SOPC1, SOPC2, etc.)

Constants:

1. Use only Named Constants. It allows better readability and easy maintenance.
   Example:
   `Name = '*';` (that doesn't say much)

   ```
   Name = kVariableName;
   ```
   (kVariableName is a named constant with value '*'. The code is a little bit more extensive but easier to understand)

2. Do NOT use the CONST keyword in the Named Constant definition.

3. Use Special Constants *Blanks, *Zeros, *All'x' (and others) to initialize or compare fields.

4. Use the Clear instruction to initialize fields with its basic value.

5. Use only *On and *Off Special Constants to turn on, off and test indicators and logical variables.

Indicators:

1. If you can't rename them, use only *Inxx variables and *In(x) array to refer to indicators.

2. When initializing multiple consecutive indicators, use %SubArr.
   Example:
   ```
   %SubArr(*In : 30 : 10 ) = *Off;
   ```
   This single statement set indicators 30 to 39 off.

3. Use indicators when really necessary. If possible, use a work variable (N type) as indicator.
   Example:
   Variable IndNewClient may contain *On whether it is a new client and *Off otherwise.

4. Use logical expressions to assign values to an indicator or logical type variable
   Example:
   ```
   *IN99 = (MsgId <> *Blanks);
   ```

5. Use indicator variables for their value directly and avoid the comparison
   Example:
   ```
   If *In97            // (instead of *In97 = *On)
   If Not *In60        // (instead of *In60 = *Off)
   ```

Keys Lists (Klist):

1. Use different Klists as needed. Avoid defining a generic Klist. It will force to do an Eval before accessing files.

Structured Instructions:

1. Use Structured RPG Instructions (If, Select, Do, For, etc.) extensively. It's easy to maintain and reading.

2. DO NOT use simple (generic) END instruction. Use only complete meaningful instructions (EndIf, EndSl, EndDo, etc.)

Documenting Structured Instructions:

1. To make a simple documentation of Structured Instructions (If, When, Do, etc.) we will use comments with a specific format. It will indicate the block's level and will allow easy reading and maintenance.
   A comment must be used BEFORE block begins and after each block instruction (Else, Other, etc.).

2. We will use the format **//->xx** to indicate the level of each nest, beginning at the same column of instruction.
   xx represents the level and must always contains 2 characters (01, 02, etc.).
   Examples:

```
//->01 If must call the Calculator (Begin First level)
If *InKE;
   //->02 If NO Requested Associated Number (Begin Second level)
   If IlsNso = *Zeros;
      MsgId = ErrAssociateNumMissing;
      //=====
      LEAVESR;
      //=====
   EndIf;
   //->02 EndIf (End Second level)
EndIf;
//->01 EndIf (End First level)
```

Goto/Tag:

1. The GoTo and Tag instructions can NOT be used on "Free" so you must design the program to use DO, DOW, DOU, EXSR, EVAL (using a procedure), LEAVESR, RETURN or LEAVE accordingly.

Reading Files, Locking Records:

1. Avoid unnecessary locks on data records. Use the "No Lock" Option (N) in file access instructions (Read(N), Chain(N), etc.) the first time you read a record and do it again just before Updating file.

2. Avoid unnecessary data transfer from disk to memory (program). When making just a key validation where no data from file is needed, use the SETLL instruction to verify key existence. No data will be read and no data will be transferred from disk to memory.
   Example:
```
// Validates file key
Setll Key File;
//->01 If No key on file
If NOT %Equal();
   (manage non existence)
EndIf;
//->01 EndIf
```

3. Avoid using the instructions READPE and READP if many records are read at once. Resource-intensive.

4. To make a loop to read a file, avoid instruction ReadE if expecting many records at once to be read. Instead use Read and control group (key) through comparisons in the loop.

5. When making reference to a file in a program (Read, Update, Chain, etc. .), do it by using Record Names and not File Names (where allowed).

6. Use the %Eof(), %Found() and %Equal() bifs to validated file access. DO NOT USE INDICATORS.

7. On previous bifs, do not use the file name as parameter when the file was the last accessed. Only use file name if you want to refer to a file access result in another part of the flow and/or could cause problems with reading or further validation.

Message Files:

1. Use message files extensively. Maintaining these objects is very simple and quick and allows standardization in messages.

2. Use messages in titles on Display Files when it can be used by more than one program (basically CLPs). It will allow using one display file for many programs.

Multiple-occurrences / Array Data Structures:

1. Whenever possible and the program keeps a reasonable size in memory, use multiple-occurrences (or array) Data Structures to keep data in memory, avoiding disk accesses.

2. They can be used to store two versions of a set of variables. For example, variables in the Display File may be an occurrence of a DS and Disk File Variables another occurrence.

Subroutines/Procedures:

1. Use Procedures instead Subroutines in programs.

2. A structured program is not a program with subroutines or procedures. Use subroutines when really necessary or when it will be called more than once within the program.

3. ***INZSR** - Use this special subroutine to initialize fields, do validations on passed parameters, etc. Only runs automatically the first time the program is called or using EXSR instruction.

4. Define *INZSR Subroutine just before last in a program. (The last should be *PSSR)

5. Define subroutines in the same order as they are referenced.

6. ***PSSR** - Use this special routine to take control of unexpected program errors.