Power Week

Université IBM i 2019

22 et 23 mai

IBM Client Center Paris

S19 – RPG fonctions avancées

Nathanaël Bonnet
Gaia
nathanael.bonnet@gaia.fr

Gaia

- Conseil et formation IBM i depuis 1995
  - Inter et intra entreprise
- Base de connaissance en ligne
  - https://know400.gaia.fr
- Centre de services
  - TMA / TME sur mesure
  - Outillage spécifique

GAIA

https://www.gaia.fr

https://twitter.com/GaiaFrance

COMMON FRANCE

Plan de la présentation

- Pointeurs
- Valeurs nulles
- CCSID
- Gestion des erreurs
- DATA-INTO
- ON-EXIT

Power Week

Université IBM i

22 et 23 mai 2019

Pointeurs

IBM

Pointeurs

- C’est une variable
  - Contenant une adresse, c’est-à-dire un emplacement dans la mémoire

```
 // Déclaration d'un pointeur
d ptr1            s                 *   inz(*null)
  dcl-s ptr1 pointer inz(*null) ;

  // Déclaration d'un pointeur initialisé sur l'adresse d'une variable
  dcl-ds enreg extname('CLIENT') qualified inz ;
d ptrEnreg        s                 *   inz( %addr( enreg ) )
  dcl-s ptrEnreg pointer inz( %addr( enreg ) ) ;
```

  - En RPG les pointeurs ne sont pas typés

Pointeurs

- Variable basée sur un pointeur
  - C'est une variable qui ne dispose pas de son propre espace mémoire pour stocker des valeurs
  - Elle va utiliser l'espace mémoire à l'adresse indiquée par le pointeur
    - Permet de coder/décoder l'espace mémoire indiqué par le pointeur en fonction du format de la variable
    - INZ interdit

```
dcl-s nom char(25) based( ptrEnreg ) ;

if nom = *blanks ;
   nom = 'Nouveau nom' ;
endif ;
```

Pointeurs - paramètres

- Vous utilisez déjà des pointeurs dans tous vos programmes/procédures !

```
// Variables
    d msg52           s             52a   inz

    c     *entry        plist
    c                   parm                    msgIn           52

      msg52 = 'Message reçu, long: ' + %char( %len( %trim( msgIn ) ) ) +
              ', début : ' + msgIn ;
      dsply msg52 ;

    c                   return
```

```
call param parm('données ici ...                                    ')
DSPLY  Message reçu, long: 52, début : données ici ...
```

Pointeurs - paramètres

- Débogage
  - STRDBG + CALL
  - ev %localvars

```
*IN(98) = '0'
*IN(99) = '0'
_QRNL_PRMCPY_MSGIN = SPP:CA94DAB602001646
_QRNL_PSTR_MSGIN = SPP:CA94DAB602001646
MSGIN = 'données ici ...                                                '
MSG52 = 'Message reçu, long: 52, début : données ici ...                '
```

```
> EVAL _QRNL_PRMCPY_MSGIN :c 50
  _QRNL_PRMCPY_MSGIN :C 50 =
            'données ici ...                                            '
```

- Pour les paramètres, le compilateur déclare pour vous des variables et des pointeurs
  - Lors d'un échange de paramètres, on ne copie pas les données, on indique à quelle endroit dans la mémoire elles se trouvent

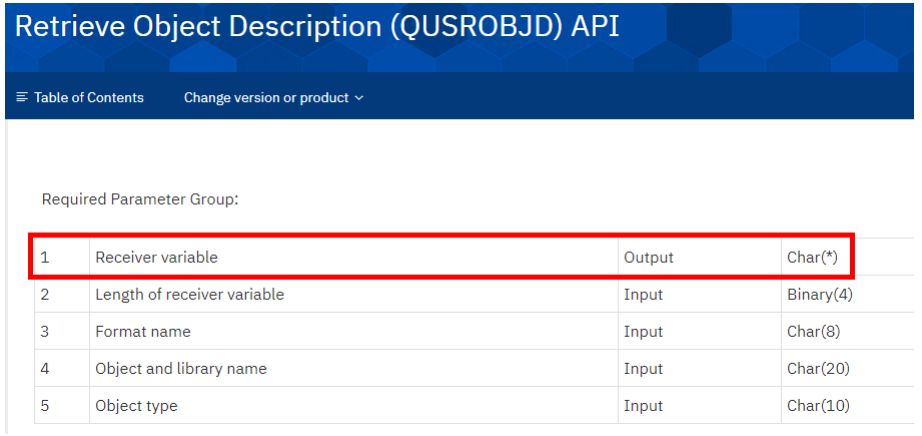
Pointeurs

- Fonctions disponibles

Fonction / Code	Description	Format libre
%alloc ALLOC(E)	Demande d'allocation de mémoire	Oui
%realloc REALLOC(E)	Réallocation de mémoire allouée par ALLOC	Oui
DEALLOC(E/N)	Libération de l'espace alloué par ALLOC	Oui
%addr	Obtenir l'adresse d'une variable	Oui
%str	Gestion des chaines à terminaison nulle (C)	Oui
+, -	Affectation d'une nouvelle adresse par ajout ou soustraction d'un décalage (en nombre d'octets)	Oui

Pointeurs

- Usages
  - Allocation dynamique de mémoire
  - Appel d'API
  - Appel de fonctions C
    - Bibliothèque standard
    - Axis pour les web services
    - Tri d'un tableau
    - …
  - Autres produits
    - CGIDEV2, HTTPAPI, XMLSERVICE, …



Retrieve Object Description (QUSROBJD) API

Table of Contents Change version or product

Required Parameter Group:

1	Receiver variable	Output	Char(*)
2	Length of receiver variable	Input	Binary(4)
3	Format name	Input	Char(8)
4	Object and library name	Input	Char(20)
5	Object type	Input	Char(10)

```
// Indiquer une valeur d'entête HTTTP pour Content-Type: application/json
propName = 'Content-Type' + NULL ;
propValue= 'application/json' + NULL ;
rc = axiscTransportSetProperty(tHandle:
                               AXISC_PROPERTY_HTTP_HEADER:
                               %addr(propName):
                               %addr(propValue));
```

Pointeurs – allocation dynamique

- Usage
  - Les variables ont une taille définie à la compilation
  - L'allocation dynamique : si l'on ne connait la taille qu'à l'exécution
    - Demande au système une quantité de mémoire
    - Utilisation
    - Libération de la mémoire (ne pas oublier, même en cas d'erreur …)
  - La mémoire est allouée au groupe d'activation

- Limites
  - STGMDL(*SNGLVL)
    - < 16 Mo (16.776.704 octets)
  - STGMDL(*TERASPACE)
    - < 4Go (4.294.967.295 octets)
  - **La mémoire obtenue n'est pas initialisée**

Pointeurs – allocation dynamique

```
dcl-s i           int(10) ;
dcl-s personnePtr pointer inz(*null) ;
dcl-s personneW   pointer inz(*null) ;
dcl-ds personne qualified based(personneW) ;
  nom       char(30) ;
  prenom    char(30) ;
  dateNaiss date ;
end-ds;

personnePtr = %alloc(50 * %size(personne)) ;    // Quel espace mémoire pour 50 personnes ?
if personnePtr = *null ;
  dsply 'Erreur allocation mémoire' ;
  return ;
endif ;

personneW = personnePtr ;                       // Utilisation de l'espace mémoire alloué
for i = 1 to 50 ;
  personne.nom = '...' ;
  personne.prenom = '...' ;
  personne.dateNaiss = %date() ;
  personneW += %size(personne) ;
endfor ;

dealloc personnePtr ;                           // Libération de la mémoire allouée
```

Pointeurs et APIs

- Exemple
  - https://github.com/FrenchIBMi/exemples/blob/master/api/dspexitpgm.rpgle

Pointeurs et APIs

```
// Example d'utilisation de l'API QusRetrieveExitInformation pour lister
// l'ensemble des points d'exit enregistrés

// Compilation :
// CRTBNDRPG PGM(NB/DSPEXITPGM)
//           SRCSTMF('/home/nb/QusRetrieveExitInformation/DSPEXITPGM.rpgle')
//           DBGVIEW(*SOURCE)

ctl-opt actgrp(*new) ;

// ----------------------------------------------------------------
// Définitions pour l'API
// ----------------------------------------------------------------

// EPI Error
dcl-ds ERRC0100 qualified inz ;
  BytPrv   int(10)   inz( %Size( ERRC0100 )) ;
  BytAvl   int(10)   inz ;
  MsgId    char(7)   inz ;
  reserved char(1)   inz ;
  MsgDta   char(256) inz ;
end-ds ;
```

Pointeurs et APIs

```
// procédure
dcl-pr QusRtvExitInfo extproc('QusRetrieveExitInformation') ;
  ContinuationHandle            char(16) const ;
  ReceiverVariable              likeds( EXTI0200 ) ;
  LengthOfReceiverVariable      int(10)  const ;
  FormatName                    char(8)  const ;
  ExitPointName                 char(20) const ;
  ExitPointFormatName           char(8)  const ;
  ExitProgramNumber             int(10)  const ;
  ExitProgramSelectionCriteria  likeds( ExitProgramSelectionCriteria ) const ;
  ErrorCode                     likeds( ERRC0100 ) ;
end-pr ;

// DS pour format EXTI0200
dcl-ds EXTI0200 qualified inz ;
  BytesReturned                         int(10) ;
  BytesAvailable                        int(10) ;
  ContinuationHandle                    char(16) ;
  OffsetToFirstExitProgramEntry         int(10) ;
  NumberOfExitProgramEntriesReturned    int(10) ;
  LengthOfExitProgramEntry              int(10) ;
  Entries                               char(65535) ;
end-ds;
```

Pointeurs et APIs

```
// Informations pour chaque programme d'exit :
dcl-s ptrEXTI0200Info pointer inz(*null) ;
dcl-ds EXTI0200Info qualified based( ptrEXTI0200Info ) ;
  OffsetToNextExitProgramEntry int(10) ;
  ExitPointName                char(20) ;
  ExitPointFormatName          char(8) ;
  RegisteredExitPoint          char(1) ;
  CompleteEntry                char(1) ;
  Reserved                     char(2) ;
  ExitProgramNumber            int(10) ;
  ExitProgramName              char(10) ;
  ExitProgramLibraryName       char(10) ;
  ExitProgramDataCCSID         int(10) ;
  OffsetToExitProgramData      int(10) ;
  LengthOfExitProgramData      int(10) ;
  Threadsafe                   char(1) ;
  MultithreadedJobAction       char(1) ;
  QMLTTHDACNSystemValue        char(1) ;
end-ds ;
```

Pointeurs et APIs

```
// Critères de sélection
dcl-ds ExitProgramSelectionCriteria qualified inz ;
  NumberOfSelectionCriteria         int(10) inz(0) ;
  dcl-ds criteria dim(10) inz ;
    SizeOfCriteriaEntry             int(10) ;
    ComparisonOperator              int(10) ;
    StartPositionInExitProgramData  int(10) ;
    LengthOfComparisonData          int(10) ;
    ComparisonData                  char(64) ;
  end-ds ;
end-ds;


// ----------------------------------------------------------------------------
// Variables programme
// ----------------------------------------------------------------------------
dcl-s ContinuationHandle char(16) inz ;        // pas assez de place pour tout récupérer
dcl-s continue           ind      inz(*on) ;   // appel suivant
dcl-s i                  int(10)  inz ;        // pour parcours des résultats
```

Pointeurs et APIs

```
// -------------------------------------------------------------------------
// Corps du programme
// -------------------------------------------------------------------------

affiche('Début du programme');

dow continue ;

  // Appel API
  QusRtvExitInfo( ContinuationHandle :
                  EXTI0200 :                        // ReceiverVariable
                  %size( EXTI0200 ) :               // LengthOfReceiverVariable
                  'EXTI0200' :                      // FormatName
                  '*REGISTERED' :                   // ExitPointName
                  '*ALL' :                          // ExitPointFormatName
                  -1 :                              // ExitProgramNumber (-1 = *ALL)
                  ExitProgramSelectionCriteria :    // Critères de sélection
                  ERRC0100 ) ;                      // Error Code

  // Appel OK ?
  if ERRC0100.BytAvl > 0 ;
    affiche('Erreur : ' + ERRC0100.MsgId ) ;
    return ;
  endif ;
```

Pointeurs et APIs

```
  // positionnement sur la première entrée
  ptrEXTI0200Info = %addr( EXTI0200 ) + EXTI0200.OffsetToFirstExitProgramEntry ;
  // Lister les points d'exit et leur programme
  for i = 1 to EXTI0200.NumberOfExitProgramEntriesReturned ;

    // Extraction des infos
    affiche( 'N°' + %char(i) + ' ' +
             EXTI0200Info.ExitPointName + '(' +
             EXTI0200Info.ExitPointFormatName + '), ' +
             %trim(EXTI0200Info.ExitProgramLibraryName) + '/' + %trim(EXTI0200Info.ExitProgramName) ) ;

    // entrée suivante
    if i < EXTI0200.NumberOfExitProgramEntriesReturned ;
      ptrEXTI0200Info = %addr( EXTI0200 ) + EXTI0200Info.OffsetToNextExitProgramEntry ;
    endif ;

  endfor ;

  // faut-il continuer ?
  continue = ( ContinuationHandle <> *blanks ) ;

enddo ;
```

Pointeurs et APIs

```
affiche('Fin du programme');
// ciao !
return ;



// Afficher à l'écran
// ----------------------------------
dcl-proc affiche ;
  dcl-pi *n ;
    p_msg varchar(512) const ;
  end-pi ;

  dcl-s msg char(52) ;

  msg = p_msg ;
  dsply msg ;
end-proc ;
```

Power Week

Université IBM i

22 et 23 mai 2019

Valeurs nulles

IBM

Valeurs nulles

- La gestion des valeurs nulles a toujours été « délicate » avec RPG
- Pourtant, nous manipulons de plus en plus de données pouvant être « nulles », principalement en provenance de la base de données
  - Jointures externes (LEFT JOIN, RIGHT JOIN)
  - Table SQL : par défaut, valeur nulle acceptée dans les colonnes
    - Nécessite NOT NULL sinon
  - Fichier physique : par défaut, valeur nulle non acceptée dans les zones
    - Nécessite ALLWNULL

Valeurs nulles

- Option de contrôles
  - ALWNULL(*NO | *INPUTONLY | *USRCTL)
    - *NO : il faudra se passer de valeur nulle (exception)
    - *INPUTONLY : permet la lecture d'un enregistrement contenant une valeur nulle
    - *USRCTL : celle qui nous intéresse
- Fonctions intégrées
  - %nullind
    - Tester ou indiquer la (non) nullité d'une valeur
- La PTF SI60418 (avril 2016) apporte
  - Capacité à déclarer des variables avec valeur nulle
  - Capacité à déclarer des DS contenant des zones avec valeur nulle (internes ou externes)

NULLIND - variables

```
**free
ctl-opt alwnull(*usrctl) ;

// variables locales
dcl-s noemp char(6) nullind ;

// mettre à null :
%nullind( noemp ) = *on ;

// Valoriser
%nullind( noemp ) = *off ;
noemp = '000012' ;

// tester la valeur
if not %nullind(noemp) and noemp <> '00010' ;
   dsply ( 'n° employé : ' + noemp ) ;
endif ;
```

NULLIND - tableaux

```
**free
ctl-opt alwnull(*usrctl) ;

// variables locales
dcl-s numemp char(6) dim(50) nullind  ;

// Mettre à null
%nullind( numemp(10) ) = *on ;

// valoriser
%nullind( numemp(10) ) = *off ;
numemp = '00010' ;

// tester la valeur
if not %nullind(numemp(10)) and numemp(10) <> '00010' ;
   dsply ( 'n° employé : ' + numemp(10) ) ;
endif ;
```

La nullité n'est pas prise en compte par SORTA

NULLIND – DS interne

```
ctl-opt alwnull(*usrctl) ;

dcl-ds client qualified ;
  id    char(10) nullind ;
  nom   char(25) ;
end-ds;

%nullind( client.id ) = *off ;
client.id = '123456' ;
client.nom = 'IBM' ;
```

NULLIND – DS externe

- Avec une DS externe par EXTNAME ou LIKEREC
  - Il est obligatoire de spécifier une DS de nullité

```
ctl-opt alwnull(*usrctl) ;

// fichier des employés
dcl-f employee disk keyed usage(*input) rename(employee:femp) ;
dcl-ds ds_employee extname( 'EMPLOYEE' : *INPUT )
       nullind(ds_emp_null) qualified  end-ds;
dcl-ds ds_emp_null extname( 'EMPLOYEE' : *INPUT : *NULL )
       qualified end-ds;

// lecture fichier
chain '000010' employee ds_employee ;
if %found( employee ) ;
   if not %nullind( ds_employee.JOB ) and
      ds_employee.JOB = 'MANAGER' ;
      // todo ...
   endif ;
endif ;
```

Power Week

Université IBM i

22 et 23 mai 2019

CCSID

IBM

CCSID

- Coded Character Set Identifier
  - C’est la codification d’un jeu de caractères
  - Permet d’encoder et de décoder les données alphanumériques par rapport à une localisation
- On le retrouve à plusieurs niveau
  - QCCSID
  - Job
  - Profil
  - Zone dans un fichier
  - Sur une variable, une constante !
- Quelques valeurs connues
  - 65535 = aucun encodage
  - 297/1147 = EBCDIC français sans/avec €
  - 37 = US
  - 1208/1200/13488= UTF-8 / UTF-16 / UCS-2

CCSID

- Depuis la 7.2, des facilités sont disponibles pour améliorer la gestion des CCSID
  - Mot-clé CCSID sur une déclaration

```
dcl-s var1 char(100);              // CCSID par défaut (1147)
dcl-s var2 char(100) ccsid(*utf8); // UTF-8 = 1208

/set ccsid(*char: *utf8)
dcl-s var3 varchar(100);
dcl-s var4 char(200);
/restore ccsid(*char)

var1 = 'Bonjour';          // EBCDIC 1147 par défaut
var2 = var1;               // Conversion implicite en UTF-8

var3 = 'Bonjour';          // Conversion (constante)
var4 = var3;               // Pas de conversion nécessaire
var1 = var3;               // Conversion vers EBCDIC


if var1 = var3 ;  // 1147 vs 1208
  dsply 'Identique' ;      // <<--
else ;
  dsply 'Différent' ;
endif ;
```

CCSID

- Nouveaux mots-clés pour CTL-OPT
  - CCSID
    - CCSID(*CHAR:valeur)
    - CCSID(*GRAPH:valeur)
    - CCSID(*UCS2:valeur)
      - Indique le CCSID par défaut pour caractères, graphiques et UCS-2
  - CCSIDCVT
    - *EXCP
      - Le programme RPG plante (exception) sur une perte de caractère dans une conversion
      - Messages : RNX0452 / RNQ0452
    - *LIST
      - Le compilateur liste l'ensemble des conversions automatiques

CCSID

- Exemple

```
**FREE

ctl-opt ccsidcvt(*excp: *list)
        ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);
```

  - Valeurs spéciales
    - *HEX (ou 65535) : pas de CCSID -> pas de conversion
    - *JOBRUN : CCSID par défaut du travail
    - *JOBRUNMIX : CCSID mixte relative au CCSID du travail (surtout si CCSID job 65535)
    - *UTF8 : UTF-8 (ou 1208)
    - Valeur : un CCSID EBCDIC ou ASCII
  - Sinon
    - *EXACT : CCSID de compilation et non celui d'exécution

CCSID

- Exemple

```
ctl-opt ccsidcvt(*excp: *list)
        ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);

dcl-s var1 char(50) ccsid(1147);
dcl-s var2 char(50) ccsid(297);

var1 = 'Je paie en €' ;
dsply var1 ;
var2 = 'Je paie en €' ;
dsply var2 ;
```

```
4 > call ccsid3
    DSPLY  Je paie en €
    Certains caractères n'ont pas été pris en compte lors de la conversion du
      CCSID(0) en CCSID(297).
```

CCSID

- La fonction intégrée %char admet également un second paramètre
  - CCSID
  - Permet de définir des conversions explicites

```
ctl-opt ccsidcvt(*excp: *list)
        ccsid(*CHAR: *UTF8) ccsid(*ucs2: *utf16);

dcl-s var1 char(15) ccsid(1147);
dcl-s var2 char(15) ccsid(*utf8);
dcl-s var3 char(15) ccsid(65535);
dcl-s var4 char(15) ccsid(819);

var1 = 'Je paie en EURO';
var2 = var1 ;
var3 = %char( var1 : 1208 ) ;
var4 = var1 ;
var4 = %char( var1 : 1208 ) ;
```

```
> EVAL var1 :x
     00000     D1854097 81898540 859540C5 E4D9D6..   - Je paie en EURO.
> EVAL var2 :x
     00000     4A652070 61696520 656E2045 55524F..   - ¤Á.o/ÑÁ.Á>.áíê!.
> EVAL var3 :x
     00000     4A652070 61696520 656E2045 55524F..   - ¤Á.o/ÑÁ.Á>.áíê!.
> EVAL var4 :x
     00000     4A652070 61696520 656E2045 55524F..   - ¤Á.o/ÑÁ.Á>.áíê!.
> EVAL var4 :x
     00000     4A652070 61696520 656E2045 55524F..   - ¤Á.o/ÑÁ.Á>.áíê!.
```

CCSID depuis des fichiers BD

- Table

```
create table NB.CLIENTWW (
name         char(30) ccsid 1208,
addr_fr      char(40) ccsid 1147,
addr_us      char(40) ccsid 37,
addr_ww      char(40) ccsid 1208
)
rcdfmt CLIENTWWF;
```

- Programme

```
ctl-opt ccsidcvt(*excp: *list) ;

dcl-f  clientww disk ;
dcl-ds client   extname('CLIENTWW':*input) ccsid(*exact) qualified end-ds ;

read clientww client ;
```

```
> EVAL client
  CLIENT.NAME = 'ä%ÑÁ>È???????????????????????'
  CLIENT.ADDR_FR = 'En France - cli3                    '
  CLIENT.ADDR_US = 'Aux US - cli3                       '
  CLIENT.ADDR_WW = ' Ñ%%ÁÍÊËÄ%Ñ?????????????????????????'
> EVAL client.name
  CLIENT.NAME = 'Client 3                    '
> EVAL client.addr_ww
  CLIENT.ADDR_WW = 'Ailleurs - cli3                     '
```

Conversion de données

- Historiquement, le RPG ne savait pas travailler avec de multiples CCSID dans un programme, mais DB2 si !
  - RPG convertissait toutes les données depuis les CCSID base de données vers/depuis CCSID du travail
- Maintenant, nos programmes RPG devraient travailler avec les CCSID de DB2
  - Mais RPG continue à convertir par défaut (compatibilité ascendante)

```
ctl-opt ccsidcvt(*excp: *list) openopt(*nocvtdata) ;

dcl-f clientww disk data(*nocvt);
```

- OPENOPT(*NOCVTDATA)
  - Pas de conversion pour l'ensemble des fichiers du programme
- DATA(*NOCVT)
  - Pas de conversion pour le fichier

Constantes, IFS

- Les littéraux sont encodés par défaut avec le CCSID du source
- Cela se voit en général dans des sources de l'IFS pour lesquels on est facilement en ASCII

```
dcl-s var char(3) ;
var = '€à@' ;
dsply '€à@' ;
dsply var ;
```

- En ajoutant

```
**free
ctl-opt ccsid(*exact) ;
```

```
4 > call ccsid7
    DSPLY  ?@à
    DSPLY  ?@à
4 > call ccsid7
    DSPLY  ?à@
    DSPLY  ?à@
```

- € n'existe pas en 850 (encodage du source IFS) => ?
- @ et à ne sont pas traduit convenablement, comme la plupart des caractères accentués

Constantes, IFS

- La PTF SI62605
  - Apporte le paramètre TGTCCSID sur CRTBNDRPG / CRTRPGMOD
    - *SRC
    - *JOB
    - Valeur : CCSID
  - Principalement prévu pour les sources en UNICODE

Power Week

Université IBM i

22 et 23 mai 2019

Gestion des erreurs

IBM

Erreurs

- Les erreurs peuvent provenir
  - De données
  - De logique
  - Des entrées/sorties
  - D'événements extérieurs (ENDJOB …)

- En RPG, les moyens de gérer les erreurs
  - Extension (e) sur certains codes opérations
  - BIF %status()
  - Code opération MONITOR
  - Sous-routine *pssr
  - Condition handler

Historiquement

- Anticipation de l'erreur

```
dcl-s dateCar char(10) inz('2017-05-17') ;
dcl-s date    date ;

test(ed) *iso dateCar ;
if not %error() ;
  date = %date( dateCar : *iso ) ;
else ;
  date = *hival ;
endif ;
```

- De façon plus générale
  - Peut-on anticiper toutes les erreurs ?
  - Peut-on produire le code qui anticipe toutes les erreurs sans bug ?
  - De toute façon il faudra également produire du code pour traiter ces erreurs

MONITOR

- Le code opération MONITOR fonctionne à l'inverse

```
dcl-s dateCar char(10) inz('2017-05-17') ;
dcl-s date    date ;

monitor ;
  date = %date( dateCar : *iso ) ;
on-error ;
  date = *hival ;
endmon ;

return ;
```

- On fait les actions et le bloc monitor intercepte les erreurs
- Un traitement d'erreurs est alors réalisé

Status et %status()

- RPG fournit un status pour indiquer quelle erreur s'est produite
  - Correspond à la zone des positions 11 à 15 dans la PSDS (Program Status Data Structure)
  - Liste des status
    - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasd/ptacode.htm#ptacode
    - https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_73/rzasd/stacod9.htm
- Permet de détecter
  - Une division par 0
  - Un dépassement de capacité
  - …

MONITOR et status

- Dans un bloc MONITOR … END-MONITOR
  - Possibilité de donner un traitement d'erreur distinct en fonction du status RPG

```
dcl-f  client disk usage(*input:*output) keyed ;
dcl-ds wclient likerec(CLIENTF) ;

monitor ;

  // Copie les 100 premiers clients
  setll *loval client ;
  read CLIENTF wclient ;
  dow not %eof(client) and
      wclient.NUMCLIENT < 100 ;
    wclient.NUMCLIENT += 100 ;
    wclient.DATECRT += %months(12);
    write CLIENTF wclient ;
    // client suivant
    read CLIENTF wclient ;
  enddo ;
```

```
// Clé en double
on-error 01021 ;
  dsply 'Clé en double' ;
// Date invalide, date overflow, date mapping error
on-error 00112 : 00113 : 00114 ;
  dsply 'Date invalide' ;
// Toute autre erreur programme
on-error *program ;
  dsply 'Erreur programme: données incorrectes ...' ;
// Toute autre erreur fichier
on-error *file ;
  dsply 'Erreur fichier: non trouvé, verrouillé ...' ;

endmon ;

return ;
```

MONITOR et status

```
// Status OK
dcl-c NO_ERROR          const(00000) ;
dcl-c RETURN_WITH_LR    const(00001) ;
dcl-c CONVERSION_RESULT const(00050) ;
// Status ERR - PGM
dcl-c DIDIVDE_BY_ZERO   const(00102) ;
dcl-c INVALID_DATETIME  const(00112) ;
dcl-c DATE_OVERFLOW     const(00113) ;
dcl-c DATE_MAPPING      const(00114) ;
// Status ERR - FILE
dcl-c DUP_KEY           const(01021) ;
dcl-c REF_CONSTRAINT    const(01022) ;
dcl-c TRIGGER_BEFORE    const(01023) ;
dcl-c TRIGGER_AFTER     const(01024) ;
```

```
on-error DUP_KEY ;
  dsply 'Clé en double' ;
on-error INVALID_DATETIME :
         DATE_OVERFLOW :
         DATE_MAPPING ;
  dsply 'Date invalide' ;
on-error *program ;
  dsply 'Erreur programme : données incorrectes
...' ;
on-error *file ;
  dsply 'Erreur sur fichier : non trouvé,
verrouillé ...' ;

endmon ;
```

Contraintes BD

- Cela ne permet pas de trouver le nom de la contrainte
  - Il est possible de retrouver ce dernier dans les messages CPF50*
- Extrait :

```
setMsgKey = *ALLx'00';
doW setMsgKey <> prevMsgKey;
  prevMsgKey = setMsgKey;
  receiveMsg( msgBack: %size(msgBack): 'RCVM0100': '*': 2: '*PRV'
            : setMsgKey: 0: '*SAME': APIError);
  If (msgBack.msgId = 'CPF502D' Or msgBack.msgId = 'CPF502E' Or
      msgBack.msgId = 'CPF502F' Or msgBack.msgId = 'CPF503A' Or
      msgBack.msgId = 'CPF503B');
    constraint = %subst(msgBack.msgData:177);
    monitor;
      msgId = %subst(constraint:%scan(' ':constraint)-7);
      return;
    on-Error;
      return;
    endMon;
  endIf;
  setMsgKey = msgBack.msgKey;
endDo;
```

- Plus simple en SQL embarqué !

Power Week

Université IBM i

22 et 23 mai 2019

DATA-INTO

IBM

Disponibilité

- Versions
  - 7.2, avec SI67184, SI67186, SI67182
  - 7.3, avec SI67185, SI67187, SI63445, SI67183
  - 7.4
- Par RDi depuis la 9.6.0.2

- Références
  - http://ibm.biz/data-into-rpg-opcode-ptfs
  - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasd/zzdatainto.htm
  - https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_73/rzasm/roaDataInto.htm

DATA-INTO

- Principe
  - Proche de XML-INTO, mais fonctionne avec n'importe quel type de données !
  - C'est à vous d'écrire le « parseur » capable de décoder les données …
    - DATA-INTO n'interprète pas les données
    - DATA-INTO appelle du code que vous fournissez qui décrypte les données
    - DATA-INTO gère l'alimentation de la DS résultat avec les données que vous avez traitées

- Appel
  - Simple
- Ecriture du parseur
  - Prise en charge de la mécanique DATA-INTO
  - Prise en charge du format de données, encodage …

DATA-INTO

- Syntaxe

```
DATA-INTO{(EH)} receiver %DATA(document {: options1 }) %PARSER(parser {:
options2 });
```

  - Le document est décodé en une fois

```
DATA-INTO{(EH)} %HANDLER(handlerProc : commArea)
%DATA(document {: options1 }) %PARSER(parser {: options2 });
```

  - Le document est décodé élément par élément
    - Utile pour les documents de taille importante

  - Avec
    - receiver : DS recevant le résultat
    - document : les données, en variable ou sur l'IFS
    - options1 et options2 : réglages du parseur
    - Le parseur peut être un programme ou une procédure d'un programme de service

DATA-INTO – exemple d'appel (IBM)

```
dcl-ds petInfo qualified inz;
   numPets int(10);
   dcl-ds pets dim(3);
      name varchar(15);
      type varchar(10);
      age int(3);
   end-ds;
   veterinarian varchar(15);
end-ds;

dcl-s msg char(52);
dcl-s i int(10);
dcl-s jsonString varchar(1000);

jsonString =
  '{'
 +   '"petInfo":'
 +   '{'
 +     '"a?b" : "a dash b",'
 +     '"pets":'
 +     '['
 +        '{"name":"Spot", "type":"dog", "age":3, "faveToy":"ball"},'
 +        '{"name":"Puff", "type":"cat", "age":7, "faveToy":"string"}'
 +     '],'
 +     '"veterinarian":"Dr Smith"'
 +   '}'
 + '}';
```

```
data-into petInfo
        %data(jsonString
            : 'case=any countprefix=num allowextra=yes')
        %parser('DATAINTOP');


for i = 1 to petInfo.numPets;
   msg = petInfo.pets(i).name + ' ' + petInfo.pets(i).type + ' '
       + 'age=' + %char(petInfo.pets(i).age);
   dsply msg;
endfor;
msg = 'Vet is ' + petInfo.veterinarian;
dsply msg;
return;
```

```
4 > call datainto
    DSPLY  Spot dog age=3
    DSPLY  Puff cat age=7
    DSPLY  Vet is Dr Smith
```

DATA-INTO – écriture du parseur

- Un exemple est fourni dans QOAR/SAMPLE

- Cinématique
  - DATA-INTO lit les données
    - Depuis fichier IFS ou variable
  - DATA-INTO appelle le programme/procédure « parseur »
    - Avec les paramètres, y compris options
  - Le parseur parcourt le document et appelle les procédures de DATA-INTO lui permettant de charger les données
    - Start of document (QrnDiStart)
    - End of document (QrnDiFinish)
    - Variable name (QrnDiReportName or QrnDiReportNameCCSID)
    - Variable value (QrnDiReportValue or QrnDiReportValueCCSID)
    - Start/End of DS (QrnDiStartStruct and QrnDiEndStruct)
    - Start/End of Array (QrnDiStartArray and QrnDiEndArray)
  - Une fois terminé, le programme continue

DATA-INTO – écriture du parseur

- Vous devez également
  - Gérer les erreurs
    - Report an error (QrnDiReportError)
    - Write a message to the "trace" log (QrnDiTrace)
    - Voir également la variable d'environnement QIBM_RPG_DATA_INTO_TRACE_PARSER
  - Gérer les CCSID
  - Nettoyer toutes les ressources utilisées par le parseur
    - ON-EXIT ?

- Pour vous aider, les déclarations pour les APIs QrnDi* sont fournies
  - QOAR/QRPGLESRC(QRNDTAINTO.RPGLE)

DATA-INTO – écriture du parseur

- Vous pouvez également trouver des parseurs
  - Pour les formats de données les plus courants : JSON, CSV …

- Par exemple
  - YAJL
    - https://www.scottklement.com/yajl/
    - https://www.scottklement.com/presentations/Working%20with%20JSON%20in%20RPG.pdf
    - https://www.scottklement.com/presentations/Nerds%20Guide%20to%20Data-Into%20in%20RPG.pdf

- Si vous voulez partager le votre
  - https://github.com/FrenchIBMi/exemples/

Power Week

Université IBM i

22 et 23 mai 2019

ON-EXIT

IBM

ON-EXIT

- Permet d'indiquer une section de code à exécuter à la fin d'une procédure
  - Que la fin soit normale ou non
- Nécessite des PTF
  - 7.2:
    - SI62949: RPG runtime
    - SI62955: TGTRLS(*CURRENT) compiler
    - Ou DB2 PTF group SF99702 Level 14
  - 7.3:
    - SI62950: RPG runtime
    - SI62957: TGTRLS(*CURRENT) compiler
    - SI62965: TGTRLS(*PRV) compiler
    - Ou DB2 PTF group SF99703 Level 3
- Prise en charge par RDi
  - Depuis 9.5.1

ON-EXIT

- Syntaxe
  - ON-EXIT {status}
    - status : indicateur de fin normale/anormale
    - Placé à la fin de la procédure, après les sous-routines
- La section de code après ON-EXIT s'exécute
  - Lors de la fin normale de la procédure / fonction
    - Dernière ligne de code ou return
  - Lors de la fin anormale de la procédure / fonction
    - Exception non gérée
    - Procédure annulée : arrêt du sous-système ou exception envoyée à une procédure plus haute dans la pile d'appel
- Usage
  - Permet de mettre du code de nettoyage (fichiers temporaires, allocation dynamique de mémoire …)
  - Le code sera exécuté même en cas d'exception
    - ON-EXIT propage l'exception (cf MONITOR … ON-ERROR … ENDMON)
  - Permet de modifier la valeur de retour de la fonction !

ON-EXIT

- Restrictions
  - Groupe d'activation autre que *DFTACGRP
  - Fichier local de type SPECIAL ou OPEN ACCESS
  - Sous-routine *PSSR (voir MONITOR)
  - DS OCCURS et tables: utiliser de DS DIM
  - Codes operations BEGSR, DUMP, EXSR, GOTO, RESET, TAG
  - Appel aux APIs CEEDOD, CEEGSI, and CEETSTA
  - ON-EXIT n'est pas autorisé dans
    - JNI
    - Une procédure avec cycle primaire

ON-EXIT

- Traitement de nettoyage

```
dcl-proc SaisiePersonnes ;

  dcl-S finAnormale ind ;
  dcl-S memDyn      pointer ;

  memDyn = %Alloc(300);
  memDyn = %ReAlloc(memDyn : 500);
  ...

On-Exit finAnormale ;

  If finAnormale ;
    // Traitement d'erreur, signalement
  EndIf;

  DeAlloc(n) memDyn;

end-proc;
```

ON-EXIT

- Modifier la valeur de retour

```
dcl-proc calcTarif ;

  dcl-pi *n zoned(5:2) ;
    limite zoned(5:2) const ;
    base   zoned(5:2) const ;
    promo  zoned(5:2) const ;
  end-pi;

  dcl-s finAnormale  ind;
  dcl-s tarif        zoned(5:2);

  tarif = base * ( 1.0 - promo / 100.0 ) ;
  return tarif;

on-exit finAnormale ;

  if finAnormale ;           // prendre le tarif de base
    return base ;
  endIf;

  if tarif < limite ;        // prendre le tarif limite
    return limite ;
  endIf;

end-proc;
```


MERCI