Chapitre 8
Le langage RPG et le BATCH

1. Les principes

Le langage RPG d'origine (il a beaucoup évolué depuis), est basé sur un mode de programmation par cartes perforées. IBM a commencé par faire des cartes perforées pour les recensements. Pendant une longue période, les programmes étaient écrits sur du papier, puis ensuite on les traduisait sur des cartes perforées, et enfin on compilait le programme en empilant les cartes, de préférence dans le bon ordre, dans un lecteur. Quand on écrit un programme en RPG, on hérite de ces cartes, chaque ligne de programme est d'ailleurs appelée carte. On commence tout programme par une carte H (header, pour les instructions de compilation et autres), puis viennent les cartes F (pour la déclaration des fichiers) et les cartes C (la partie programme). Depuis le RPG ILE il y a aussi les cartes D (déclaration des variables).

Compte tenu de cette façon de programmer, on comprend mieux les contraintes du langage. Les fonctions, les facteurs, les résultats ou les indicateurs doivent être positionnés à un endroit précis. Pour faciliter le travail du programmeur il y a aussi les indicateurs (booléens) *IN01 à *IN99 qui sont implicites à tout programme RPG. Pour utiliser un indicateur, il suffit de lui affecter une valeur "0" ou "1", à la suite de quoi on pourra tester cette valeur et faire ce qui est nécessaire. Il est parfois assez difficile de comprendre comment va réagir telle ligne de code quand elle commence par : N17 22N54. Parce que si on parle de *IN17, dans la ligne il se nomme 17, et, dans cette ligne si l'indicateur 17 est éteint (SETOFF), l'indicateur 22 allumé (SETON) et l'indicateur *IN54 = '0' (SETOFF ou MOVE '0' *IN54), l'instruction qui suit sera exécutée. Pour la compacité, on peut difficilement faire mieux, pour la compréhension c'est moins évident. Rassurez-vous, on programme de moins en moins de cette façon, mais comme sur AS/400 on finit toujours par tomber sur un programme qu'il faut impérativement modifier et qui date de 1992, il vaut mieux avoir une idée sur la question.

Avant le RPG ILE, la déclaration des variables se faisait au fil de l'eau, il suffisait de définir le type de la variable à un moment quelconque du programme. On pouvait aussi définir la variable à chaque utilisation dans le programme, en évitant de changer de type en cours de route. Désormais, on définit les variables dans les cartes D en début de programme, mais rien n'interdit de remettre la définition à chaque utilisation, cela ne changera strictement rien au programme compilé (pas un octet d'écart), mais cela permet de savoir quel est le type de la variable à chaque ligne du programme.

Le RPG ILE a également permis de rendre les expressions plus humaines, un peu plus proches des langages utilisés en micro. Mais cela reste toutefois assez rigide. D'un autre côté, les programmes sont relativement courts, à partir de 200 ou 300 lignes de code on fait des choses assez performantes. Quant aux programmes qui dépassent les 10000 lignes, ils sont assez rares et en tout cas quasiment impossibles à lire.

Le GOTO est assez couramment utilisé en RPG. On peut toujours s'en passer, mais il y a des cas où la tentation est forte d'aller directement un peu plus loin dans le cours du programme sans tester en cours de route.

Pour les boucles, le RPG ILE connaît le même type de boucles que dans les autres langages. Dans les temps historiques, le FOR n'existait pas, on trouvera donc le plus souvent :

```
0080.00 C     I              DOWLE     10
0081.00 C     I              ADD       1              I
0082.00 C                    EXSR      TRTXXX
0083.00 C                    ENDDO
```

Le ENDDO n'est là que pour la lisibilité, un END marche aussi bien.

Avec le RPG ILE on peut utiliser l'instruction FOR I = 1 TO 10 avec un ENDFOR en fin de boucle, la différence tient à la lisibilité et aux habitudes acquises dans d'autres langages. Au niveau des performances il ne doit pas y avoir de différences notables.

Le IF décline aussi diverses possibilités IFEQ, IFLE, IFGT associé au ELSE et au ENDIF (même remarque que pour le ENDDO).

On a également à sa disposition le SELECT ENDSL et le CAS ENDCS. Le premier permet d'exécuter une séquence d'instructions en fonction de la valeur d'une variable, le deuxième permet de se débrancher sur une procédure de la même manière.

Le BEGSR est l'instruction qui définit le début d'une procédure, qui se termine par ENDSR. La règle la plus commune est la suivante. Si un ensemble d'instructions est utilisé plusieurs fois dans un programme, il faut toujours créer une procédure. Ensuite, tout est question de bon sens. Si le corps du programme est relativement simple et court, inutile de créer des tas de procédures qui ne seront utilisées qu'une fois.

L'appel d'une procédure est fait par l'instruction EXSR.

Par contre dans le cas d'un traitement assez complexe, il vaut mieux morceler le programme en parties homogènes.

```
0010.00 C                    EXSR      INIT
0020.00 C                    EXSR      PARAM
0030.00 C                    EXSR      BOUCLE
0040.00 C                    EXSR      FINPGM
```

Tout est question de mesure.

Les variables étaient à l'origine toutes globales, depuis le RPG ILE on peut aussi avoir des variables locales dans les procédures. À dire vrai, à part compliquer la partie déclaration, l'utilité de ce type de variable est assez restreinte et l'impact en matière de performances négligeable. Autre petit détail non négligeable, toutes les variables sont initialisées à blanc ou à zéro selon le type, sauf les DS (zone Donnée Structurée) qui ne sont pas toujours bien nettes au départ. On peut aussi affecter une valeur lors de la déclaration d'une variable, du moins en RPG ILE, et déclarer en constante.

Bien entendu on peut également appeler un autre programme à partir d'un programme (CALL MONPMG) qui lui-même peut en appeler un autre et ainsi de suite. Il faudra faire attention toutefois à ce que dans l'imbrication des programmes on ne se retrouve pas à appeler le programme initial. Si un programme s'appelle lui-même cela ne durera pas très longtemps, au premier appel l'AS/400 vous signalera l'erreur. Par contre, si l'appel est indirect, c'est un peu plus long et il faudra quelque 32766 boucles avant que l'AS/400 s'aperçoive de quelque chose, mais il le verra et vous rappellera à l'ordre à peu près de la même façon que dans le cas précédent.

Depuis le RPG ILE on peut aussi utiliser l'instruction CALLP. C'est un peu plus compliqué car il faut créer un programme de service, un répertoire de liage, y mettre tous les programmes à utiliser, compiler le programme utilisant ces programmes avec les options BNDSRVPGM et BNDDIR correctement renseignées, dans le source du programme déclarer la procédure comme un prototype… et faire de nombreuses tentatives pour que tout fonctionne. Si les programmes ne sont écrits qu'en RPG, le gain en matière de performances est quasiment nul, sauf si les programmes ne se terminent pas par l'instruction RETURN. Le seul intérêt est que l'on peut utiliser directement les API de l'AS/400 dans ces programmes de service, ou alors créer ses propres API écrites en C par exemple. Dans ce cas il est probable que le temps d'exécution puisse être divisé par 100, mais le temps de mise au point risque lui d'être multiplié par le même facteur. Cela est intéressant si le traitement standard dure 1 heure, et, là encore, tout dépendra du contexte et de la capacité de chacun de lire, en anglais, la documentation IBM.

L'autre méthode consiste à terminer le programme appelé par le mot-clé RETURN avant l'instruction SETON LR. Lors des appels suivants, dans le même contexte, le programme sera déjà monté en mémoire et tous les contrôles d'ouverture des fichiers ne seront pas réexécutés. C'est ce qui permet de gagner beaucoup de temps lors de l'exécution car, dans certains cas, le préalable AS/400 avant le traitement proprement dit est nettement plus important que la durée du traitement en lui-même.

Mais, comme rien n'est jamais parfait, les variables garderont leur valeur antérieure (penser à les réinitialiser si besoin est), les fichiers resteront positionnés là où on les a quittés, il faut donc être un peu plus attentif lors de la conception d'un tel programme.

En fait, on se rend vraiment compte de la nécessité de faire un choix ou un autre lors du passage en production quand, après quelque temps d'utilisation, le volume des données devient assez important, et les temps de réponse rédhibitoires.

Le BATCH est utilisé principalement de deux façons, la création d'un fichier spoule pour imprimer un document et un traitement de mise à jour ou d'extraction de fichiers. Par définition un programme BATCH n'a pas d'écran et s'exécute dans un sous-système autre que l'interactif. Toutefois, rien n'interdit d'exécuter un programme BATCH dans QINTER, si ce n'est que l'on risque de pénaliser les autres utilisateurs.

Prenons un exemple simple. Il faut éditer, à partir du fichier du personnel, la liste de tous les employés qui habitent dans la même commune. Si le fichier comporte un millier de noms répartis dans une cinquantaine de villes et que l'on peut placer 10 fiches par page, le fichier spoule résultant fera deux à quatre pages en moyenne et le temps moyen d'exécution sera de l'ordre de la seconde. On peut donc faire cette opération en interactif sans que cela pose problème. Maintenant, si le fichier comporte 10 000 noms, répartis sur une dizaine de villes et qu'il est nécessaire d'imprimer beaucoup de détails, il est fort probable que l'on obtiendra un spoule de 100 pages ou plus, le temps de traitement sera plutôt de l'ordre de 10 à 20 secondes durant lesquelles l'utilisateur attendra, plus ou moins patiemment, que l'édition soit terminée avant de pouvoir faire autre chose. Dans ce dernier cas, il est préférable de soumettre le JOB. L'utilisateur reprend la main tout de suite et 30 secondes plus tard son document s'imprimera.

De la même façon, une mise à jour simple dans un fichier ne mérite pas d'être soumise, en revanche si le traitement est long il faut impérativement le soumettre.

Dans tous les cas, il vaut mieux prévoir que le JOB soit soumis, quitte à l'exécuter en interactif si nécessaire, c'est toujours plus facile à faire dans ce sens-là qu'à l'inverse. L'AS/400 propose dans certaines commandes les deux options. Taper CLRLIB, faire [F4] et choisir l'une ou l'autre option. Si la bibliothèque contient moins de 50 objets on peut le faire en interactif, mais s'il y a 1 000 ou 2 000 objets il vaut mieux prévoir une pause café avant de reprendre la main.

Avant de passer aux travaux pratiques, passons en revue les DS (*Data Structures*), utilisées dans la plupart des programmes.

Ces DS permettent de récupérer bon nombre d'informations sur l'état général d'un fichier, du programme en cours, d'un écran ou d'un spoule.

1.1 DS d'information du programme

Cette DS est chargée automatiquement et sera présente dans chaque programme utilisé. À mesure de l'exécution du programme, certaines variables seront actualisées. Le nom du programme et l'utilisateur ne devraient pas changer en cours de route, par contre le numéro de séquence dans le source évoluera tout au long de l'exécution. Dans la pratique on ne se sert pas beaucoup de la plupart de ces variables, mais comme il y a toujours le cas où cela devient nécessaire, il vaut mieux les inscrire systématiquement en début de chaque nouveau programme, d'autant qu'après en avoir créé une on se contentera de la recopier à chaque fois. Cela ne prend pas beaucoup plus de temps, mais cela peut en faire gagner énormément par la suite.

Ce qui est vrai pour cette DS, l'est également pour les suivantes.

Le type indique s'il s'agit d'alphanumérique ou de numérique, D et F donnent la position de début et de fin dans la DS.