Power Week 2025

18 - 19 - 20 novembre 2025
IBM Innovation Studio Paris

S07 – Sécurité des développements

18 novembre 11:15 - 12:15

Nathanaël BONNET
Gaia-Volubis
nathanael.bonnet@gaia.fr

IBM

common FRANCE

Présentation

Nathanaël BONNET

IBM i depuis 1999

Expert IBM i

GAIA / VOLUBIS

Formation (débutant, perfectionnement)

Expertise IBM i

Centre de Services

Sommaire

- Bonnes pratiques de développement RPG/CL
- Bonnes pratiques de développement SQL
- Open Source
- Signature des objets

Introduction

- Nous parlons beaucoup de la sécurité, à raison : sécurité système, des droits, cryptage ...

- Quid des actions liés au développement, pour produire des programmes fiables ?

Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Bonnes pratiques de développement RPG/CL

Langages

- RPG/CL
  - Ce sont des langages compilés
    - C’est déjà une sécurité quant à la difficulté de modification toute ou partielle du code
  - Rappel : pas de sources sur la machine de production !
    - Enfin, on en reparle

Joblog

- RPG : les exceptions, y compris interceptés, les messages envoyés volontairement par SND-MSG (ou API)

```
dcl-s cpt1 int(10) inz(0);
dcl-s cpt2 int(10) inz(0);

for cpt1 = 1 to 10;
  cpt2 = cpt1 * 2 ;
endfor ;


monitor ;
  cpt1 = 10/cpt2;
  cpt2=0;
  cpt1 = 10/cpt2;
on-error *all ;
endmon ;

snd-msg *INFO 'Fin du programme';
return ;
```

```
4 > call demorpg
    Tentative de division par zéro pour opération en virgule fixe.
    Fin du programme
```

Joblog

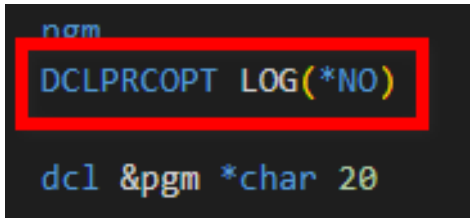
- Les commandes CL apparaissent dans la joblog en fonction de l'attribut LOGCLPGM du travail

```
CRTDTAARA DTAARA(SECURITI25/DTA1) TYPE(*CHAR) LEN(10) VALUE('Hop !') +
            TEXT('Data area 1')
MONMSG MSGID(CPF1023) +
       EXEC(CHGDTAARA DTAARA(SECURITI25/DTA1 *ALL) VALUE('Hop !'))

RTVSYSVAL SYSVAL(QSTRUPPGM) RTNVAR(&pgm)

SNDMSG MSG('Traitement OK. Programme de démarrage :' *bcat &pgm ) +
       TOUSR(*SYSOPR)
```

```
4 > call clcmd
      600 - CRTDTAARA DTAARA(SECURITI25/DTA1) TYPE(*CHAR) LEN(10) VALUE('Hop
     !') TEXT('Data area 1')
   La zone de données DTA1 existe déjà dans SECURITI25.
      800 - CHGDTAARA DTAARA(SECURITI25/DTA1 *ALL) VALUE('Hop !')
     1100 - RTVSYSVAL SYSVAL(QSTRUPPGM) RTNVAR(&PGM)
     1300 - SNDMSG MSG('Traitement OK. Programme de démarrage : QSTRUPPGM
     EXPLOIT') TOUSR(*SYSOPR)
   - RETURN        /* RETURN provoqué par la fin du programme CL */
4 > CHGJOB LOGCLPGM(*no)
4 > call clcmd
   La zone de données DTA1 existe déjà dans SECURITI25.
```

Joblog

- Pour ne pas impacter tout le job, on peut également ajouter une option de compilation :
  - Dans le source CLLE : DCLPRCOPT LOG(*NO) <<-- recommandé
  - Sur la commande de compilation : CRTBNDCL … LOG(*NO)




```
4 > CHGJOB LOGCLPGM(*yes)
4 > call clcmd
    La zone de données DTA1 existe déjà dans SECURITI25.
```

Extraction du source

- La commande

```
RTVCLSRC PGM(SECURITI25/CLCMD)
         SRCFILE(SECURITI25/QCLSRC)
         SRCMBR(EXTRACTE)
         RTVINCSRC(*YES)
```

- Permet d'extraire le source d'un CL

- Sur RTVCLSRC
  - *PUBLIC a *USE
- Droits nécessaires sur le programme
  - Opération et gestion

SECURITI25 > QCLSRC > EXTRACTE.CLLE > ...

```
1  /*****************************************************************/
2  /*                                                               */
3  /* 5770SS1 V7R6M0 250418 Sortie RTVCLSRC        01/08/25 16:01:42  */
4  /*                                                               */
5  /* Nom du programme . . . . . :   CLCMD            PN*/
6  /* Nom de la bibliothèque . . . . . :   SECURITI25       PL*/
7  /* Nom du module  . . . . . :   CLCMD            MN*/
8  /* Nom de la bibliothèque . . . . . :   *LIBL            ML*/
9  /* Fichier source d'origine . . . . . :   QCLSRC           SN*/
10 /* Nom de la bibliothèque . . . . . :   SECURITI25       SL*/
11 /* Membre source d'origine  . . . . . :   CLCMD            SM*/
12 /* Modification du fichier source                              */
13 /*   date/heure . . . . . :   01/08/25 15:54:30 SC*/
14 /* Texte  . . :                                            TX*/
15 /* Propriétaire . . . . . :   PLB8             OW*/
16 /* Extraire source incluse  . . . . . :   *YES             RI*/
17 /*                                                         ED*/
18 /*****************************************************************/
19     PGM
20     DCLPRCOPT LOG(*NO)
21     DCL VAR(&PGM) TYPE(*CHAR) LEN(20)
22     CRTDTAARA DTAARA(SECURITI25/DTA1) TYPE(*CHAR) LEN(10) VALUE('Hop-
23  !') TEXT('Data area 1')
24     MONMSG MSGID(CPF1023) EXEC(CHGDTAARA DTAARA(SECURITI25/DTA1 -
25 *ALL) VALUE('Hop !'))
26     RTVSYSVAL SYSVAL(QSTRUPPGM) RTNVAR(&PGM)
27     SNDMSG MSG('Traitement OK. Programme de démarrage :' *BCAT &PGM)-
28  TOUSR(*SYSOPR)
29     ENDPGM
```

Extraction du source

- Pour se prémunir
  - Dans le source : DCLPRCOPT LOG(*NO) ALWRTVSRC(*NO)
  - Sur la commande de compilation : CRTBNDCL … ALWRTVSRC(*NO)

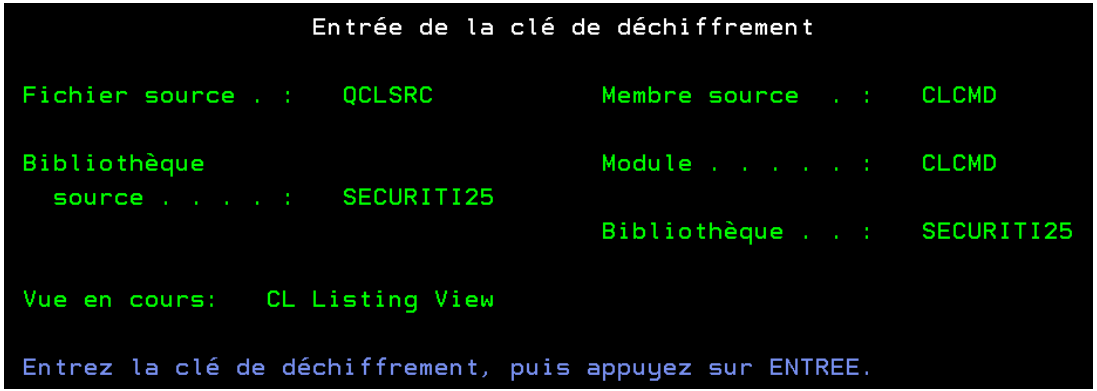
```
RTVCLSRC PGM(SECURITI25/CLCMD) SRCFILE(SECURITI25/QCLSRC) SRCMBR(EXTRACTE
2) RTVINCSRC(*YES)
Le module CLCMD dans CLCMD, *PGM, bibliothèque SECURITI25, ne contient
  pas le source CL.
```

Debug

- Nécessaire pour le développeur
  - Mais seulement le développeur
- Commande STRDBG/STRSRVJOB
  - Interdire la commande via les droits
  - Défaut : *PUBLIC *EXCLUDE, mais QPGMR et QPGMR_NC (7.6) à *USE
- Programmes
  - Pour avoir le droit de débuguer, il faut *CHANGE ou *USE + l'autorité spéciale *SERVICE
  - Pour exécuter *USE suffit

Debug

- Cryptage du source
  - DBGENCKEY('mot de passe') -> non modifiable sauf à recompiler le programme
- Si le source est absent de la machine, ou par F15 Vue listing :

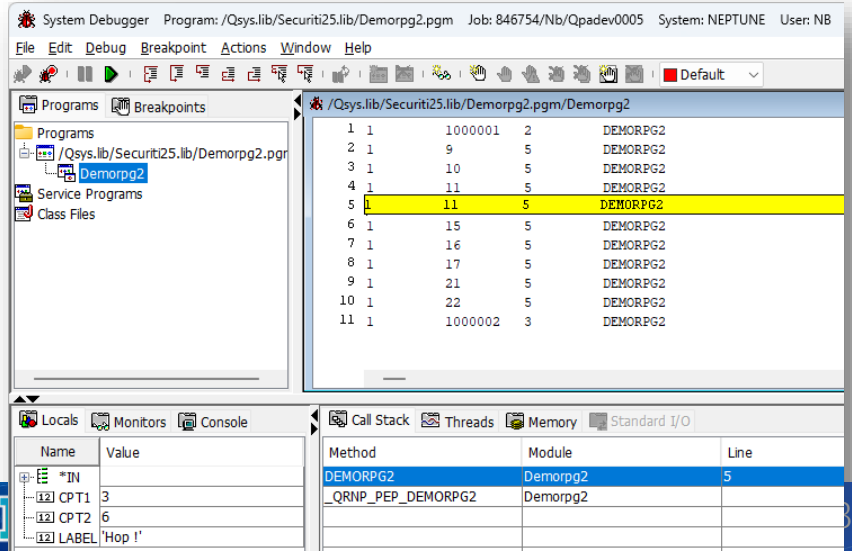


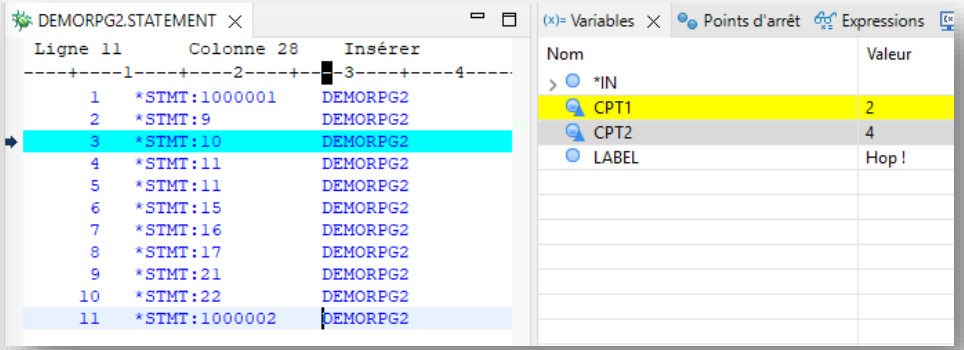
- Attention
  - RTVCLSRC fonctionne toujours !
  - Si ALWRTVSRC(*YES)

Debug

- DBGVIEW(*STMT)
  - Source non affiché
  - Mais, en fonction des débogueurs :
    - STRDBG + VSCode : programme non déboguable, variable non affichées
    - RDi + System Debugger : pas à pas possible, variables affichées et modifiables

```
                              Source d'un module

Programme:    DEMORPG2       Biblio:    SECURITI25     Module:    DEMORPG2

  (Source non disponible.)
```





Debug CL

- Quel que soit la valeur de ALWRTVSRC à la compilation, le débogage d'un CL est possible
- ALWRTVSRC(*NO)

```
                              Source d'un module

Programme:    CLCMD          Biblio:    SECURITI25     Module:    CLCMD

   (Source non disponible.)
```

- Si vous connaissez le nom des variables

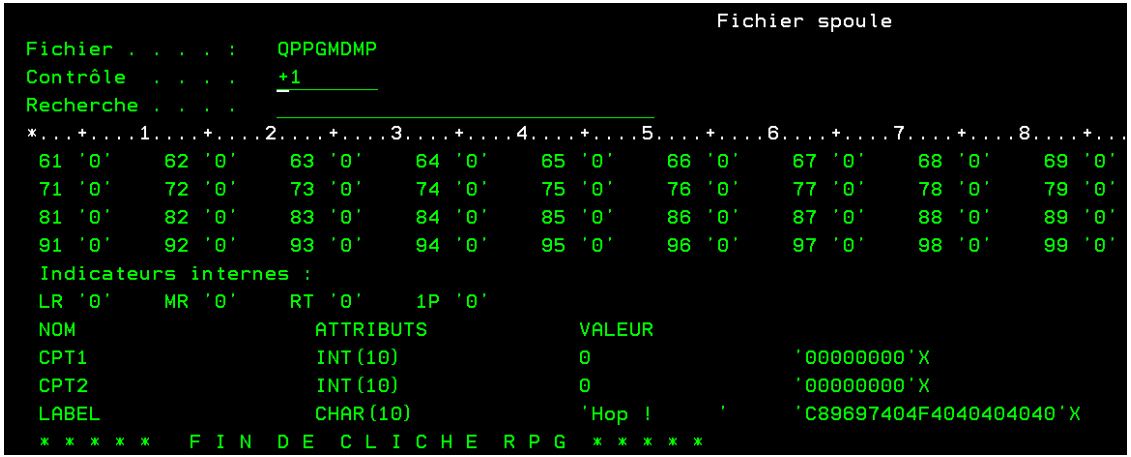
```
Débogage
F3=Arrêter programme
F11=Variable    F12=Reprendre
&PGM = 'QSTRUPPGM EXPLOIT    '
```

- Dans tous les cas, pour le CL

```
Débogage        EVAL %localvars
F3=Arrêter programme
F11=Variable    F12=Reprendre
Identifier does not exist.
```

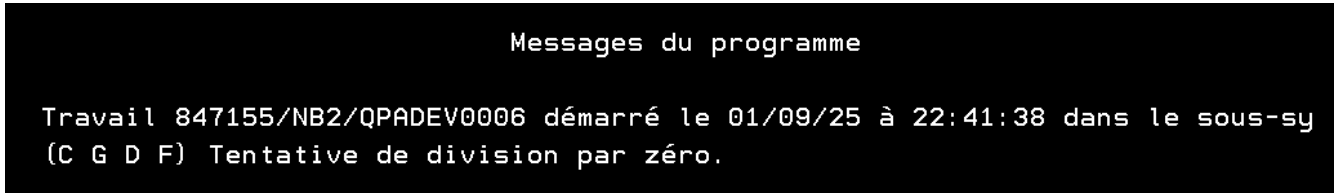
Dump

- Le dump permet à un utilisateur d'avoir accès à l'ensemble des variables d'un programmes
  - Génère un spoule QPPGMDMP dans QUSRSYS/QEZDEBUG
  - Accessible à un utilisateur (*PUBLIC *USE par défaut)
- Si DBGVIEW(*NONE) en option de compilation
  - Seulement program status data structure, file information data structures, et les indicateurs *IN
- Si DEBUG(*NO) en spécification de contrôle
  - Aucun dump produit
  - On peut forcer avec DUMP(A)



Dump

- Dump en cas de plantage
  - Impossible d’éviter le dump !



  - Astuce : appeler des programmes sans paramètres, ou avec des valeurs extrêmes -> cela a toutes les chances d’arriver à forcer un dump, et vous donner des indications sur le programme

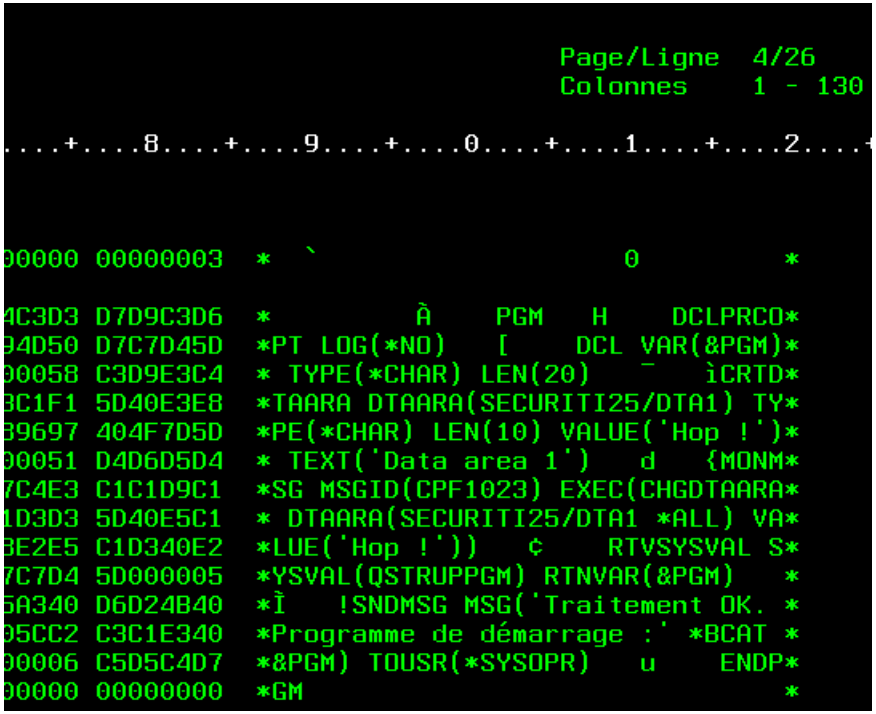
Dump Objet

- Permet d'accéder au source

```
DMPOBJ OBJ(SECURITI25/CLCMD)
       OBJTYPE(*PGM)
```

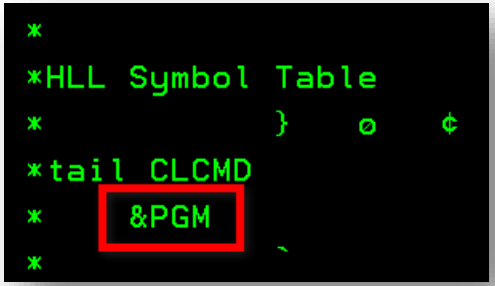
- Sauf si compilé avec
  - DBGENCKEY('mot de passe')

- DMPOBJ
  - Défaut : *PUBLIC *EXCLUDE
- Droits nécessaires sur le programme
  - Soit *ALLOBJ sur le profil
  - Soit : opération (*OBJOPR) sur le programme + exécute (*EXECUTE) sur la bibliothèque

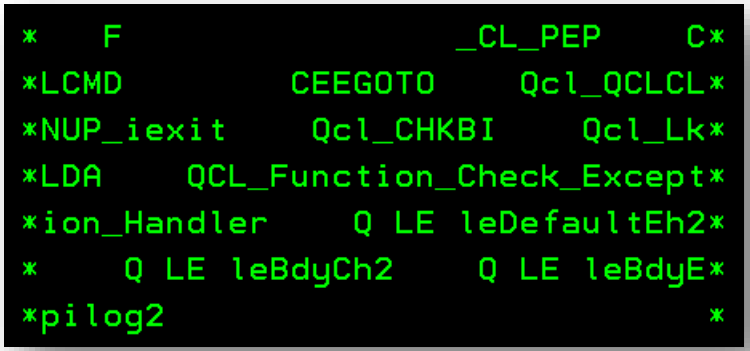


Dump Objet

- Mais on peut tout de même accéder à d'autres informations
  - Noms des variables
    - Utilisable en débug même sans affichage du source



  - Informations de liage
    - Les procédures importées/exportées



Injection de code SQL

- Eviter l'interface permettant d'exécuter n'importe quelle instruction SQL
  - Par facilité on en trouve souvent (tout le temps !)
  - Sous différente forme : proc stock, web service ...

- Dans les programmes
  - La encore, la compilation des programmes (vs interprétation) nous offre de facto une protection
  - Mais, avec les SQL dynamique, on a tout de même des capacités d'agir, principalement avec les services SQL

Injection de code SQL

- Exemple de programme classique
  - Ici simplifié à l'extrême pour montrer la mécanique
  - On prend une valeur en paramètre pour recherche dans un fichier
  - Très utilisé dans les recherches multicritères ou le nombre de combinaisons est important

```
5  dcl-ds emp qualified inz ;
6      firstnam    char(20) ;
7      lastname    char(25) ;
8  end-ds ;
9  dcl-s sqlStmt varchar(1024) inz ;
10
11 dcl-pi *n ;
12   empno      char(200) const ;
13 end-pi ;
14
15 sqlStmt = 'select FIRSTNME, LASTNAME from employee where empno = ''' + empno + '''' ;
16 exec sql prepare s1 from :sqlStmt ;
17 exec sql declare c1 cursor for s1 ;
18 exec sql open c1 ;
19 exec sql fetch next from c1 into :emp ;
20 exec sql close c1 ;
21
22 snd-msg *INFO ('Employee Name: ' + %trim(emp.firstnam) + ' ' + %trim(emp.lastname)) ;
23
```

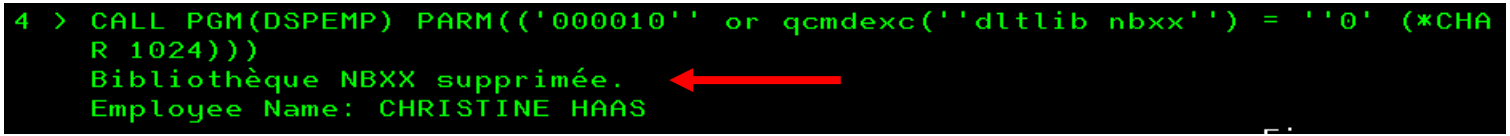
Injection de code SQL

- Appel

```
CALL PGM(DSPEMP)
     PARM(('000010' (*CHAR 1024)))
```

- Mais

```
CALL PGM(DSPEMP)
     PARM(('000010'' or qcmdexc(''dltlib nbxx'') = ''0' (*CHAR 1024)))
```



Injection de code SQL

- Conseils
  - Utiliser des marqueurs de paramètres au lieu de SQL dynamique
  - En cas de SQL dynamique, ajouter quelques contrôles de base :
    - Longueur de la donnée, contrôle de surface, recherche de mot-clé (QCMDEXC), etc ...

```
 9   dcl-s sqlStmt varchar(1024) inz ;
10
11   dcl-pi *n ;
12     empno      char(200) const ;
13   end-pi ;
14
15
16
17   sqlStmt = 'select FIRSTNME, LASTNAME from employee where empno = ?' ;
18   exec sql prepare s1 from :sqlStmt ;
19   exec sql declare c1 cursor for s1 ;
20   exec sql open c1 using :empno;
21   exec sql fetch next from c1 into :emp ;
22   exec sql close c1 ;
23
24   snd-msg *INFO ('Employee Name: ' + %trim(emp.firstnam) + ' ' + %trim(emp.lastname)) ;
25
```

Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Bonnes pratiques de développement SQL

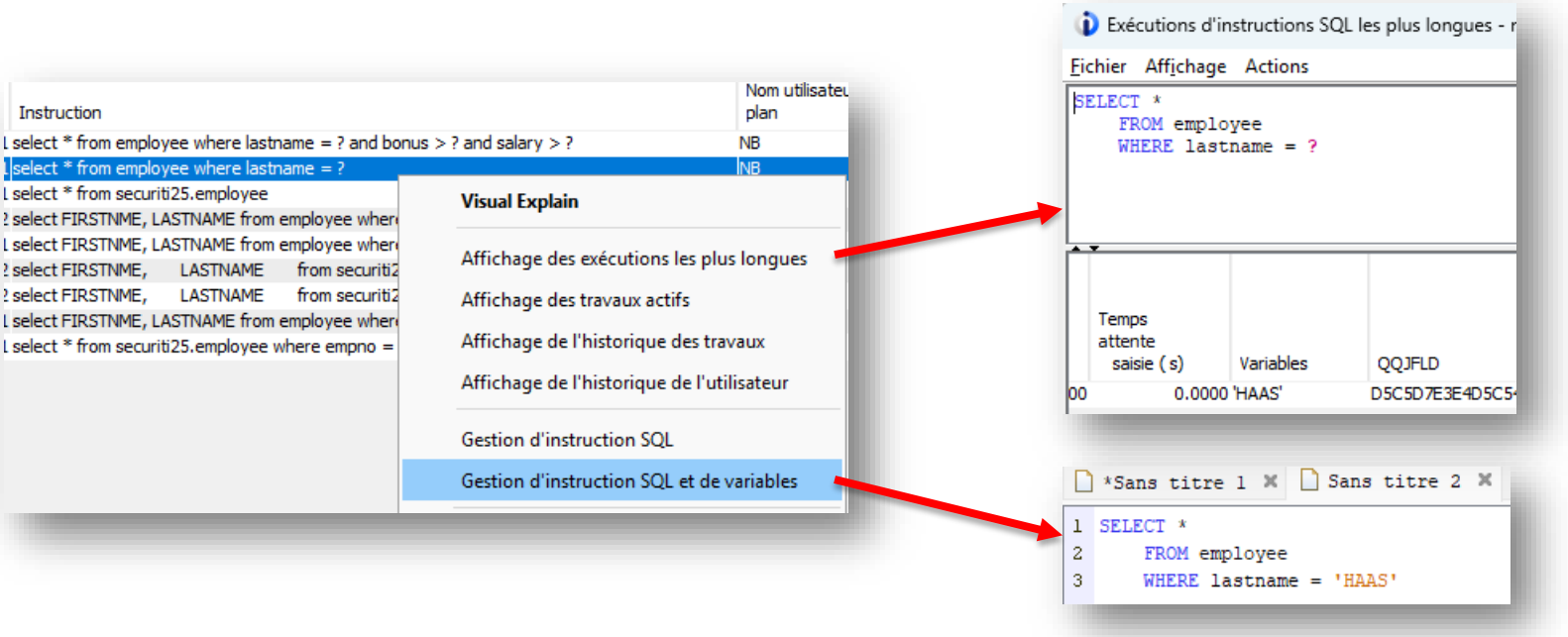
Plan Cache

- Outil indispensable au fonctionnement de DB2 et à son tunning

- On peut dumper le plan cache SQL par la procédure DUMP_PLAN_CACHE
  - Très bavard
  - *JOBCTL ou la fonction d’usage QIBM_DB_SQLADM suffisent pour le générer

Plan Cache

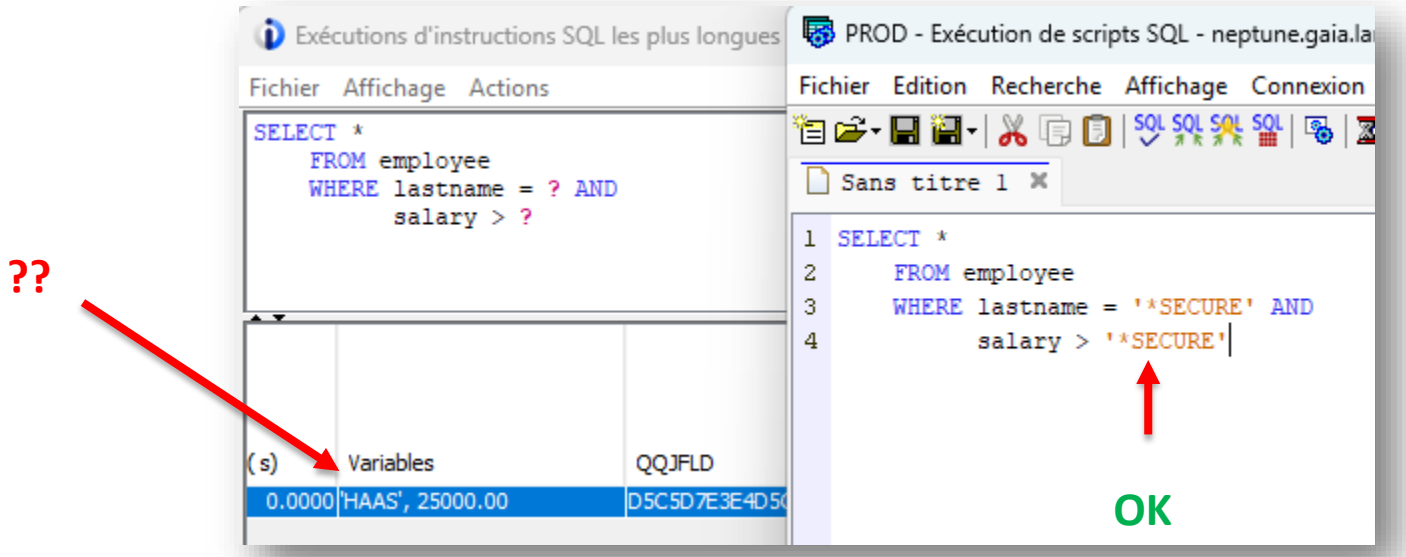
- Mais les plans contiennent également de la donnée : les variables hôtes



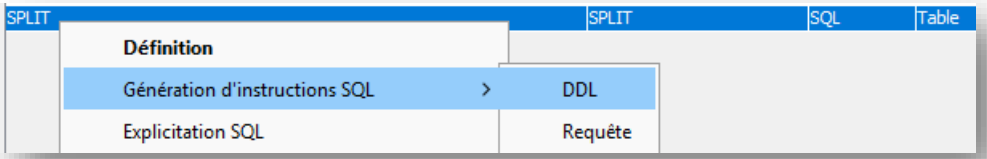
Plan Cache

- Il est possible de cacher certaines valeurs
  - N'apparait plus dans les moniteurs et le plan cache

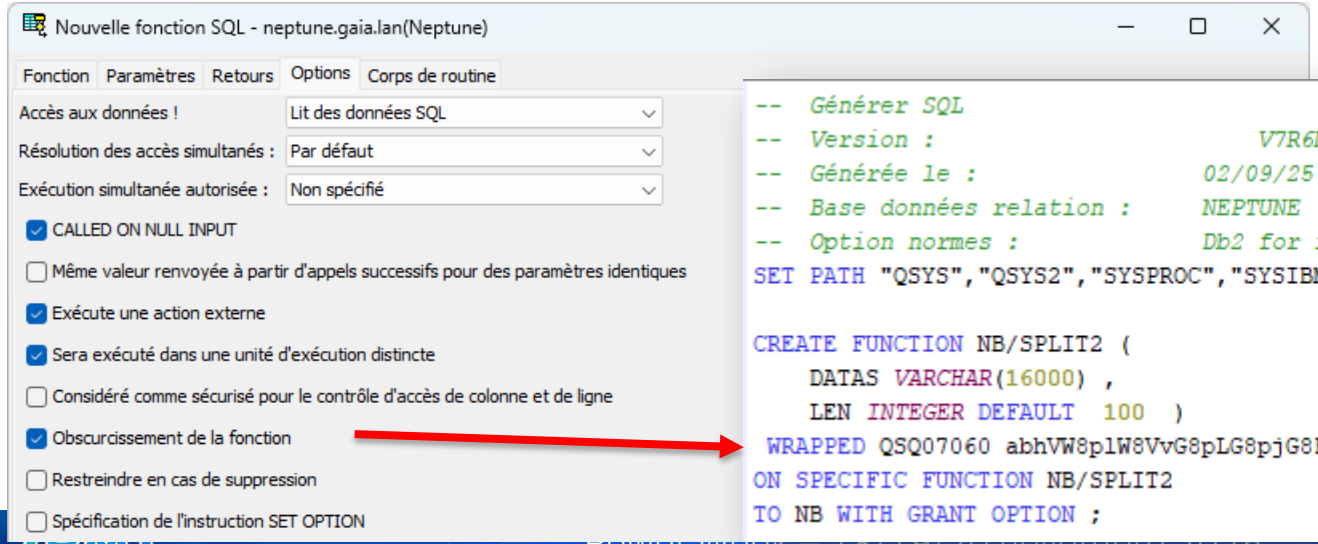
```
CALL SYSPROC.SET_COLUMN_ATTRIBUTE('SECURITI25', 'EMPLOYEE', 'SALARY', 'SECURE YES');
```



Génération de code



- Vous pouvez régénérer le source des objets SQL
  - Y compris les programmes : procédures, fonctions (scalaire/table), triggers
- Pour l'empêcher : obfuscation
  - La génération est toujours possible, mais



Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Open source

Open Source

- Totalement dépendant des technologies utilisées

- Cependant
  - Pour IBM i natif, nous avons de nombreuses modalités
    - Journaux d'audit
    - Points d'exit
  - Pour l'Open Source, dans PASE
    - Aucun de ces moyens n'est efficace

Power Week

18 -19 - 20 novembre 2025

IBM

common FRANCE

IBM

Signature des objets

Qu’est-ce ?

- Le mécanisme de signature permet :
  - D’authentifier le propriétaire (signataire)
  - Garantir la non modification de l’objet
  - Cf https://fr.wikipedia.org/wiki/Signature_num%C3%A9rique

- Quel usage :
  - Garantir l’authenticité des programmes livrés
  - Par un éditeur
  - Entre vos systèmes de développement / production

- Quels objets ?
  - *PGM, *SRVPGM, *MODULE, *SQLPKG, *FILE (SAVF) et *CMD

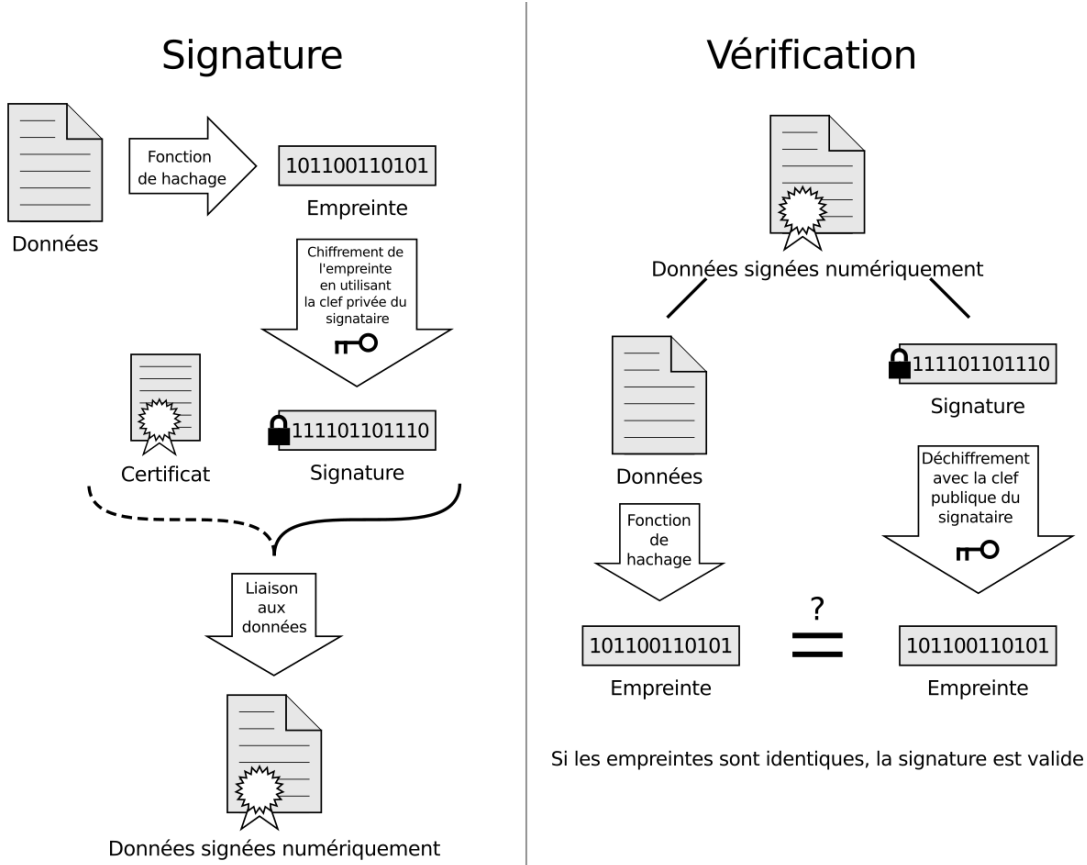
- Compatibilité : V5R1+

Mécanique – via DCM

- Machine source
  - Création magasin *OBJECTSIGNING
  - Création/import d'un certificat
  - Création d'une définition d'application de signature
  - Signer des objets

- Machine cible
  - Création magasin *SIGNATUREVERIFICATION
  - Importation de la chaine de certification
  - Vérifier des objets

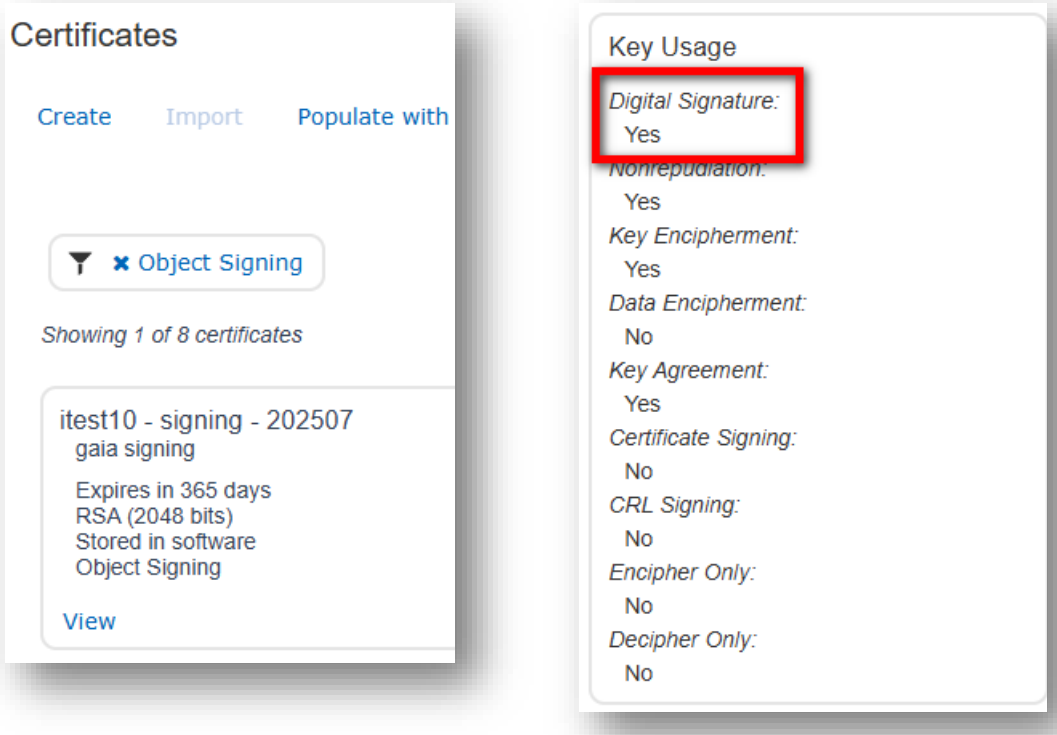
Qu’est-ce ?


Signature
Fonction de hachage
101100110101
Empreinte
Données
Chiffrement de l'empreinte en utilisant la clef privée du signataire
Certificat
111101101110
Signature
Liaison aux données
Données signées numériquement
Vérification
Données signées numériquement
Données
Fonction de hachage
101100110101
Empreinte
?
111101101110
Signature
Déchiffrement avec la clef publique du signataire
101100110101
Empreinte
Si les empreintes sont identiques, la signature est valide

Magasin dédié

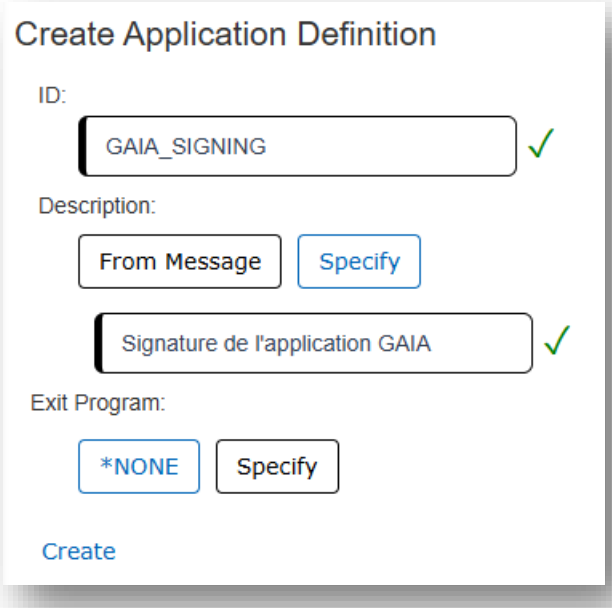
Certificat pour signature

- Remarquer l’usage



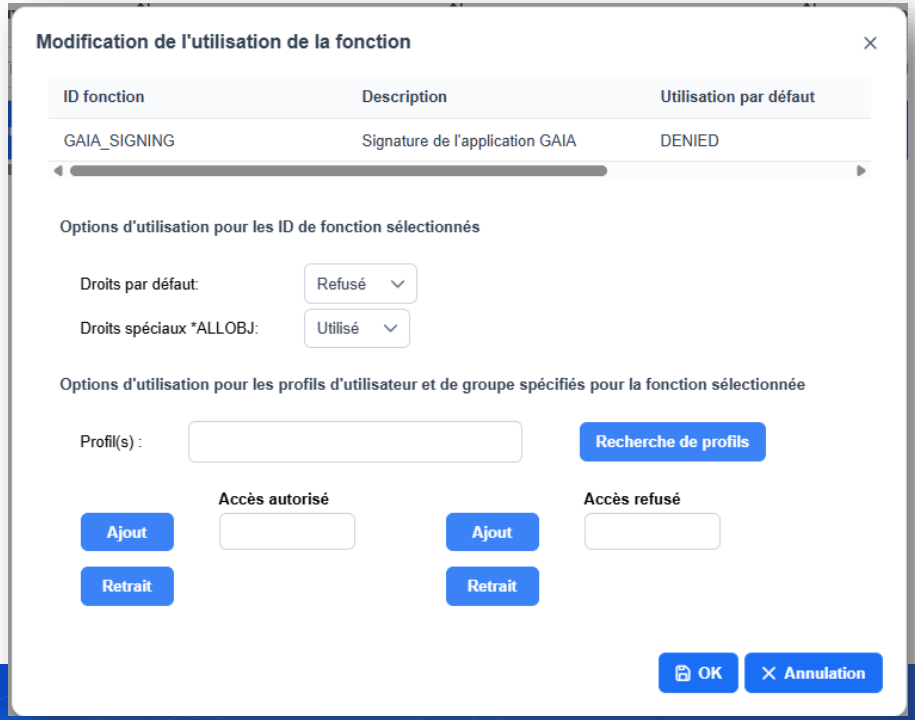
Application pour signature

- Il faut encore créer une application avant de pouvoir signer un objet
  - Description : informatif. Peut provenir d'un message (fichier de messages)
  - Exit Program : appelé par DCM dans les cas suivants
    - L'application est supprimée ou modifiée
    - Changement de certificat assigné
    - La liste des CA de confiance est modifiée



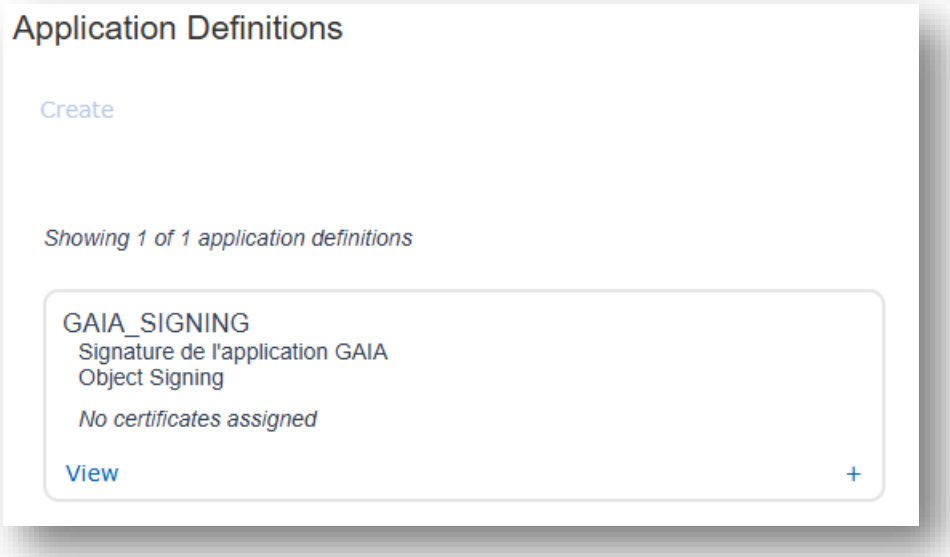
Application pour signature

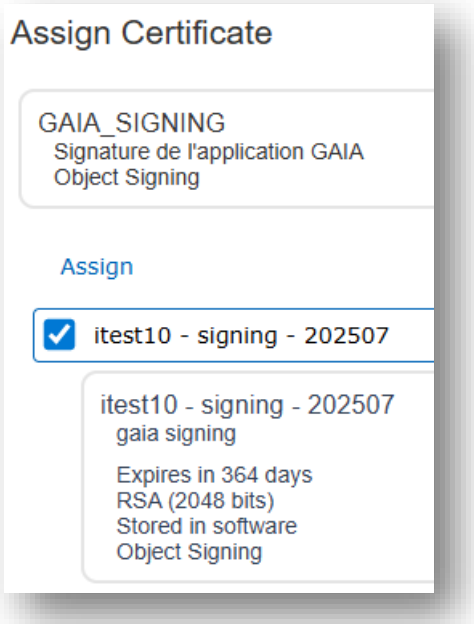
- La création de l’application provoque la création d’une fonction d’usage du nom de l’application :



Application pour signature

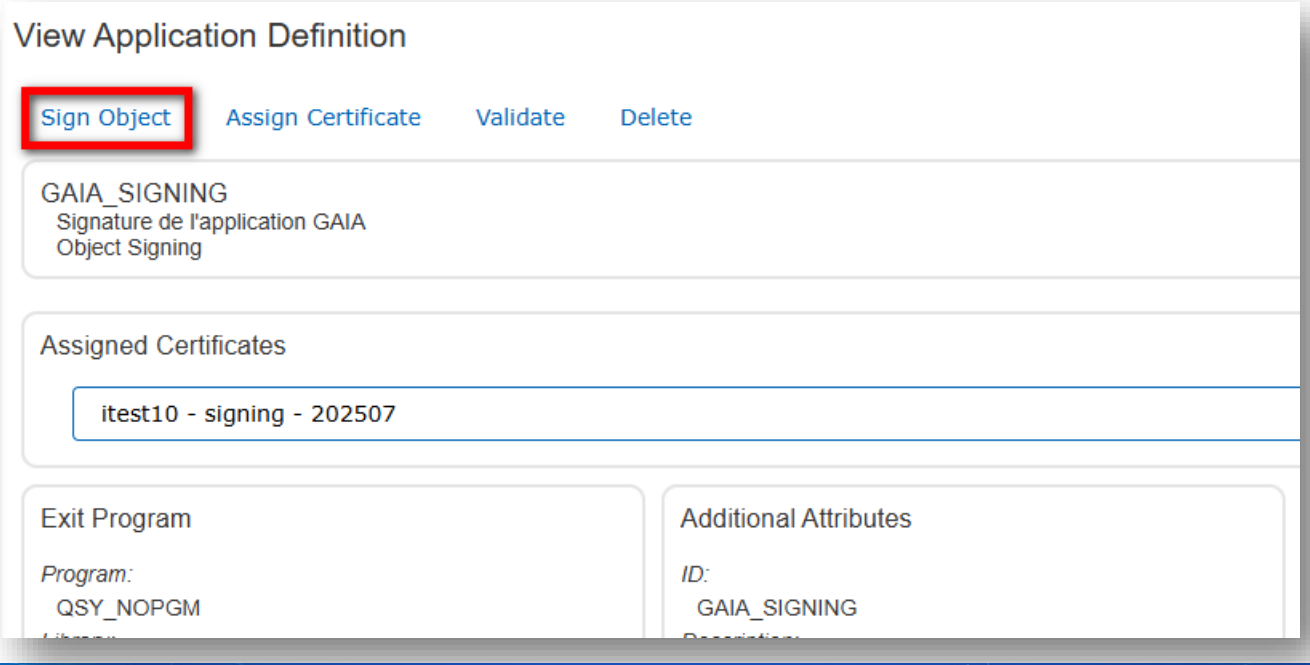
- On assigne le certificat à l'application





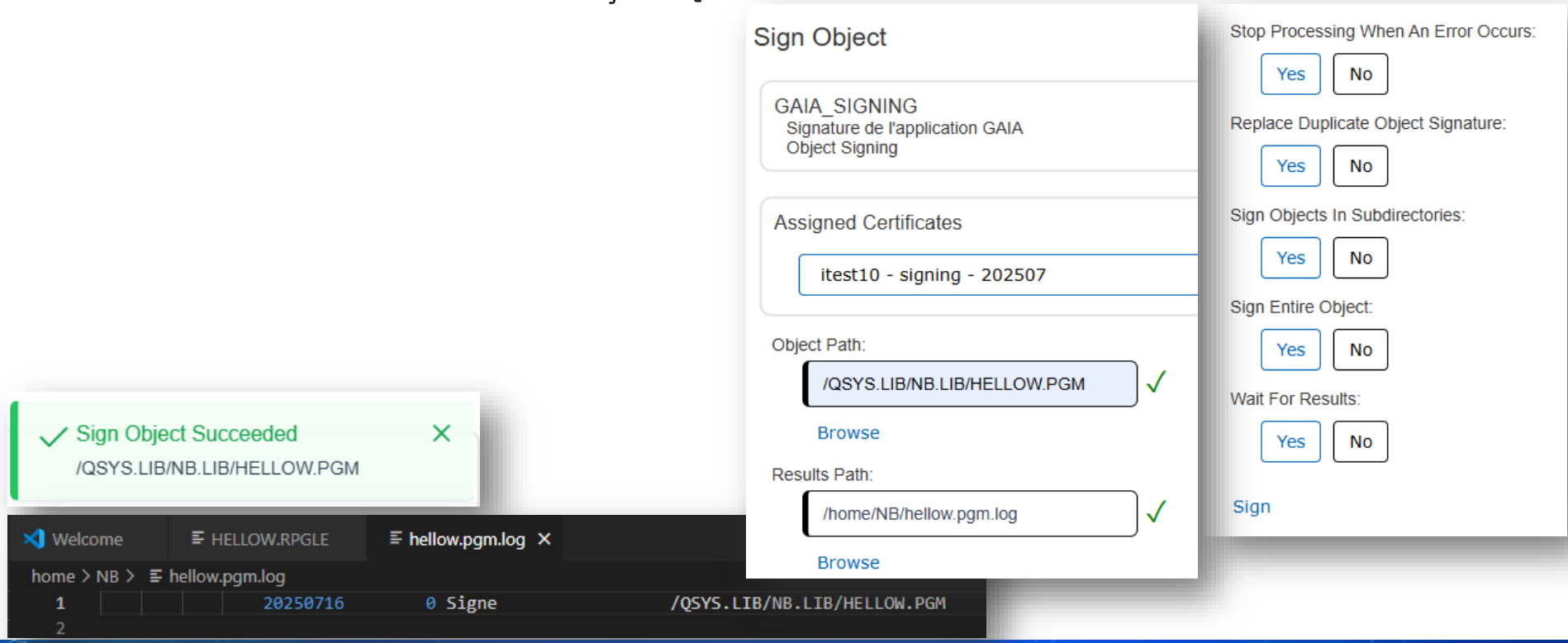
Signature d’un objet

- Maintenant, il est possible de signer des objets, **depuis la définition de l’application**



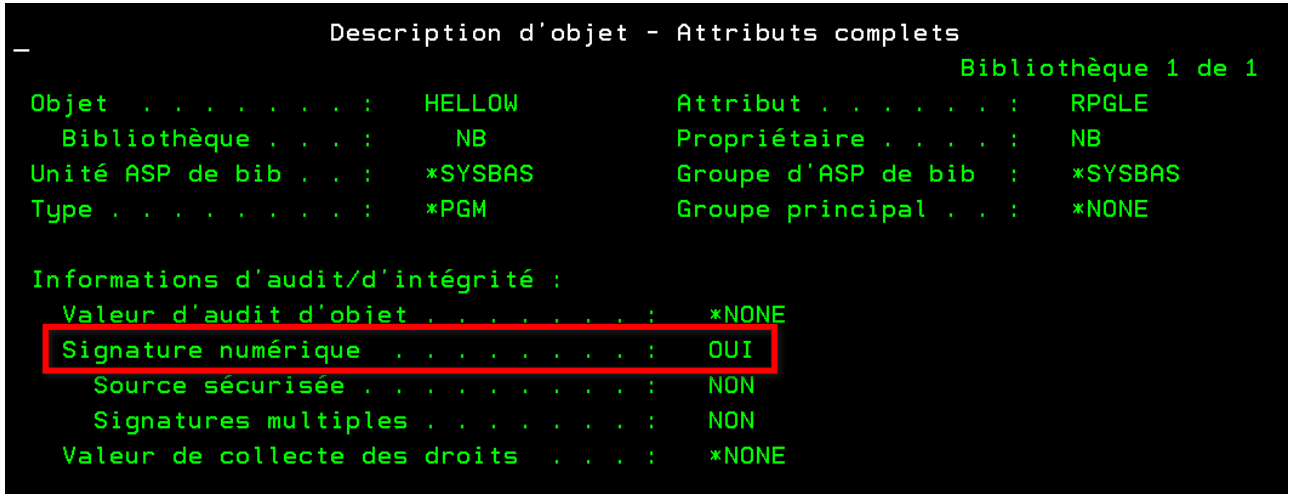
Signature d’un objet

- Wait for result : soumet un job QOBJSGNBAT



Signature d’un objet

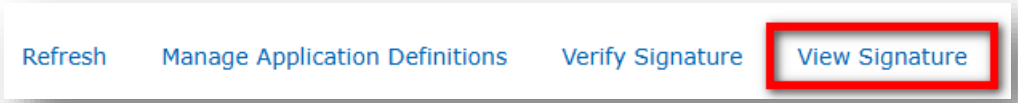
- DSPOBJD

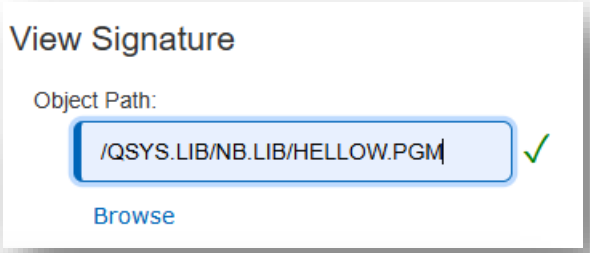


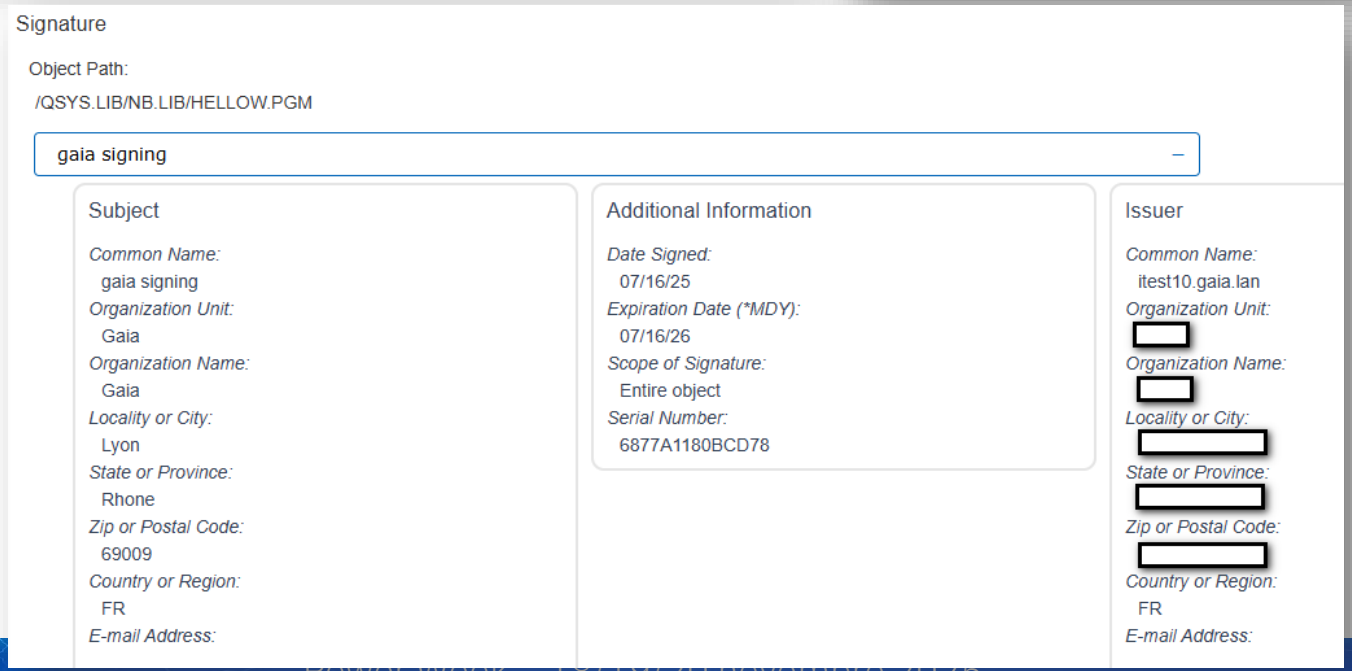
- Par SQL

```
SELECT OBJECT_SIGNED
  FROM TABLE (qsys2.object_statistics('NB', '*PGM', 'HELLOW'))
```

Affichage de la signature







Validation de la signature

- Scénario :
  - Transfert de l'objet sur un autre système
  - Restauration
  - Contrôle de la signature
- Restauration

```
RSTOBJ OBJ(HELLOW)
       SAVLIB(NB)
       DEV(*SAVF)
       OBJTYPE(*PGM)
       SAVF(NB/HELLOW)
       RSTLIB(NB)
       OUTPUT(*PRINT)
```

```
1 objet(s) restauré(s) de NB dans NB.
```

```
                    Description d'objet - Attributs complets

Objet  . . . . . :   HELLOW          Attribut . . . . .
  Bibliothèque . . :   NB            Propriétaire . . .
Unité ASP de bib . . :  *SYSBAS      Groupe d'ASP de bi
Type . . . . . :     *PGM            Groupe principal .

Informations d'audit/d'intégrité :
  Valeur d'audit d'objet . . . . . :   *NONE
  Signature numérique  . . . . . :     OUI
    Source sécurisée . . . . . :       NON
    Signatures multiples . . . . . :   NON
  Valeur de collecte des droits  . . . :  *NONE
```

Validation de la signature

- Cf la valeur système QVFYOBJRST, de 1 à 5
- Il est utilisé pour contrôler la restauration des objets signés numériquement.
  - 1 et 2
    - Restauration de tous les objets
  - 3 (défaut)
    - Autorise la restauration d'objets non signés
    - Autorise la restauration d'objets signés :
      - Si la signature est validée (le certificat est présent et valide)
      - Si le certificat est absent (le système considère que la signature n'est pas vérifiable et considère l'objet comme non signé)
    - N'autorise pas les objets signés pour lesquels la signature est invalide (le certificat est présent mais la valeur de signature obtenue n'est pas celle attendue)
  - 4 et 5
    - Ne restaure QUE des objets signés et valides

Validation de la signature - CHKOBJITG

- Par la commande

```
CHKOBJITG OBJ('/QSYS.LIB/NB.LIB/HELLOW.PGM')
          OUTFILE(NB/VFYSGN)
          CHKSIG(*ALL)
```

```
La commande s'est exécutée mais des violations ont été détectées.
```

AIDCEN	AIDDAT	AIDTIM	AISYST	AIOIND	AINAME	AILIB	AITYPE	AIOWNR	AIVIOL	AITRUN	AICCSD	AICTID	AILNID	AIRES1
1	071625	160427	NEPTUNE	1			*PGM	QSECOFR	NOSIG	0	1147	FR	FRA	

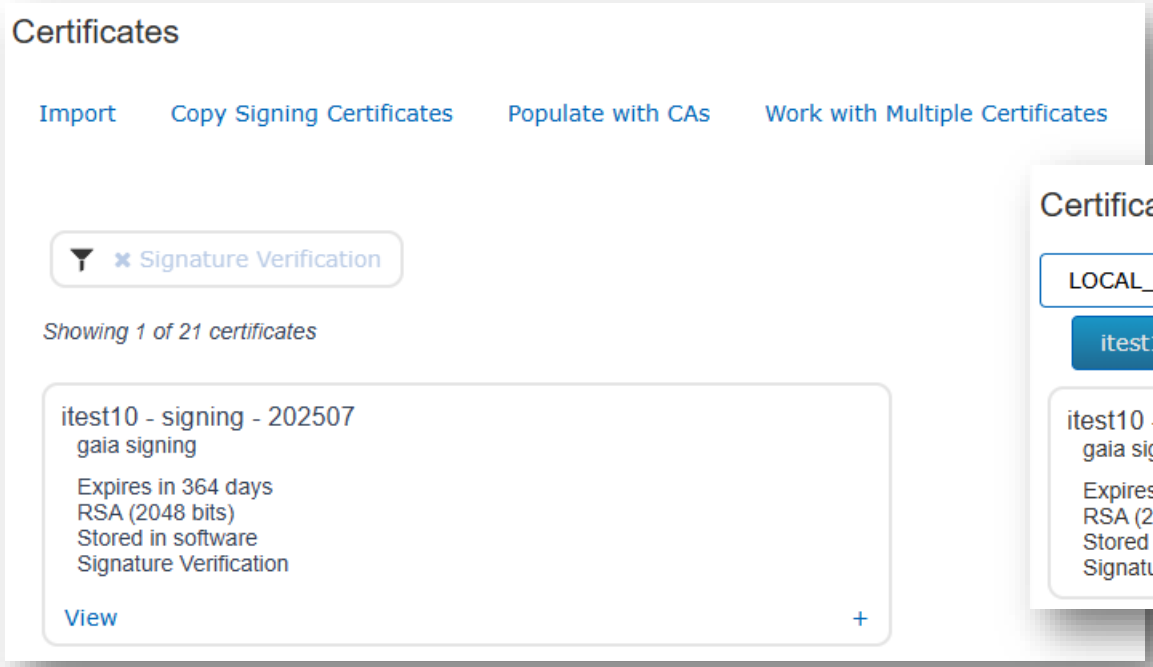
AIPBYT	AIFLID	AIRES2	AIPATH	AILASP	AILASN	AIOASP	AIOASN
27	□¿6MG□dn		/QSYS.LIB/NB.LIB/HELLOW.PGM	*SYSBAS	0	*SYSBAS	0

The types of violations that can occur are:

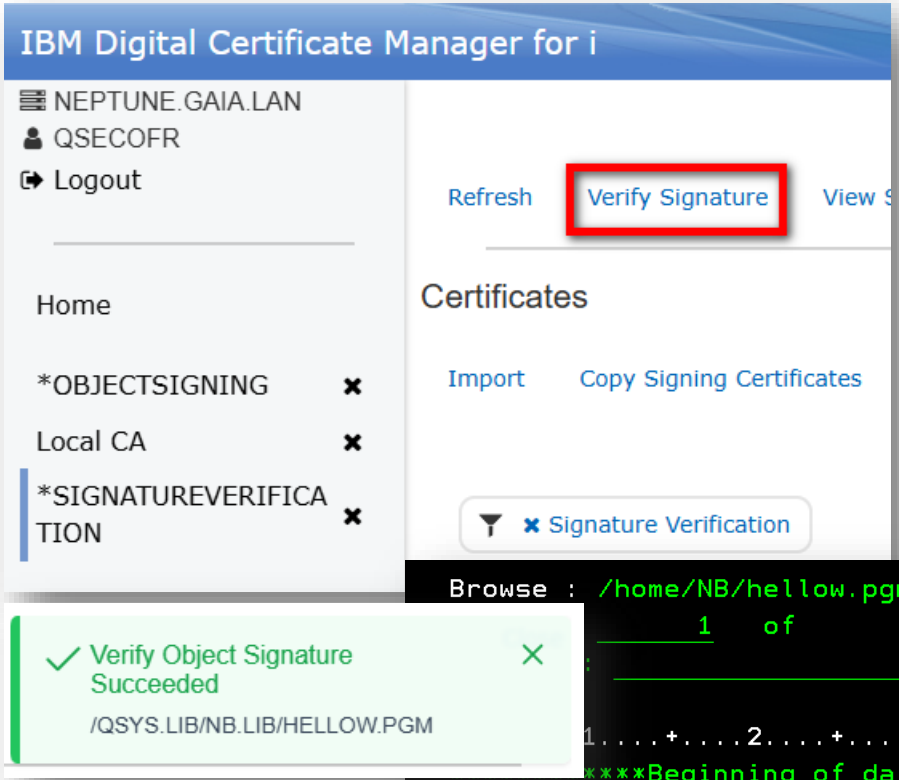
- NOSIG - The object can be signed but does not have a digital signature.
- NOTCHECKED - The object cannot be checked, it is in debug mode, saved with storage freed, or compressed.
- NOTTRANS - The object has not been converted to RISC format.

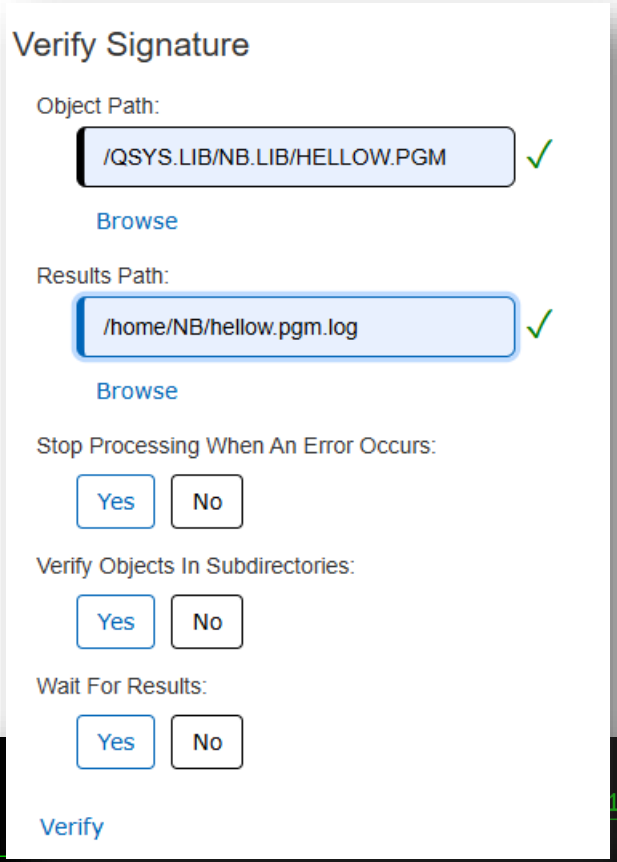
Validation de la signature – Installation certificat

- Sur le système cible, importer le certificat et son autorité dans le magasin *SIGNATUREVERIFICATION



Validation de la signature - DCM





CCSID 1200 !

```
Browse :  /home/NB/hellow.pgm.log
          1   of   3 by  18
:
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+.
****Beginning of data**************
CPFB72A         20250716        1 Vérif                 /QSYS.LIB/NB.LIB/HELLOW.PGM
                20250716        1 Vérif                 /QSYS.LIB/NB.LIB/HELLOW.PGM
```

Automatisation

- Objectif
  - Automatiser aussi bien les signatures que les contrôles

- Avec les API correspondantes
  - Sign Object (QYDOSGNO, QydoSignObject) : https://www.ibm.com/docs/fr/i/7.6.0?topic=ssw_ibm_i_76/apis/qydosgno.html
  - Verify Object (QYDOVFYO, QydoVerifyObject) : https://www.ibm.com/docs/fr/i/7.6.0?topic=ssw_ibm_i_76/apis/qydovfyo.htm

- Exemple implémentation signature/contrôle
  - https://github.com/nathanaelGaia/events/blob/main/Securiti.2025/chksgn.sqlrpgle

Automatisation

SECURITI25 > QRPGLESRC > CHKSGN.SQLRPGLE > ...

```
13
14  // Structure pour appel API QydoVerifyObject
15  dcl-ds MultipleObjectsCharacteristics_t template qualified ;
16    Subdirectories                  char(1)   inz('0') ;
17    StopOfFirstError                char(1)   inz('0') ;
18    Reserved                        char(6)   inz(*allx'00') ;
19    OffsetToResultsFilePathName     int(10)   inz ;
20    LengthOfResultsFilePathName     int(10)   inz ;
21    FormatOfResultsFilePathName     char(8)   inz('OBJN0100') ;
22    FormatOfContentsOfResultsFile   char(8)   inz('RSLT0100') ;
23    ResultsFilePathName             char(128) inz ;
24  end-ds ;
25
26  // Prototype API QydoVerifyObject
27  dcl-pr QydoVerifyObject extproc('QydoVerifyObject') ;
28    ObjectPathName                          char(100) const ;
29    LengthOfObjectPathName                  int(10)   const ;
30    FormatOfObjectPathName                  char(8)   const ;
31    MultipleObjectsCharacteristics          likeds(MultipleObjectsCharacteristics_t) const ;
32    LengthOfMultipleObjectsCharacteristics  int(10)   const ;
33    ErrorCode                               likeds(ERRC0100_t) ;
34  end-pr ;
35
```

Et plus …

- Renouvellement certificat
- Multi-certificats
- Gestion fine des droits signataires via la fonction d’usage

Conclusion

- Pas de solution magique
- Mais de multiples moyens de trouver des informations de façon plus ou moins détournées

- Des options de compilation à intégrer dans vos défauts !


MERCI