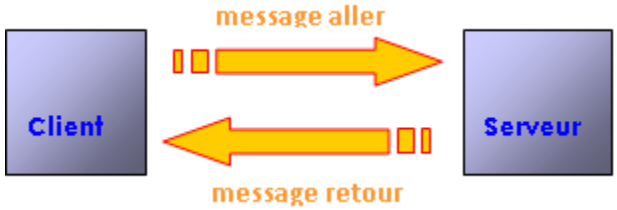
Optimisation Client-Serveur

Principe d'échange client/serveur

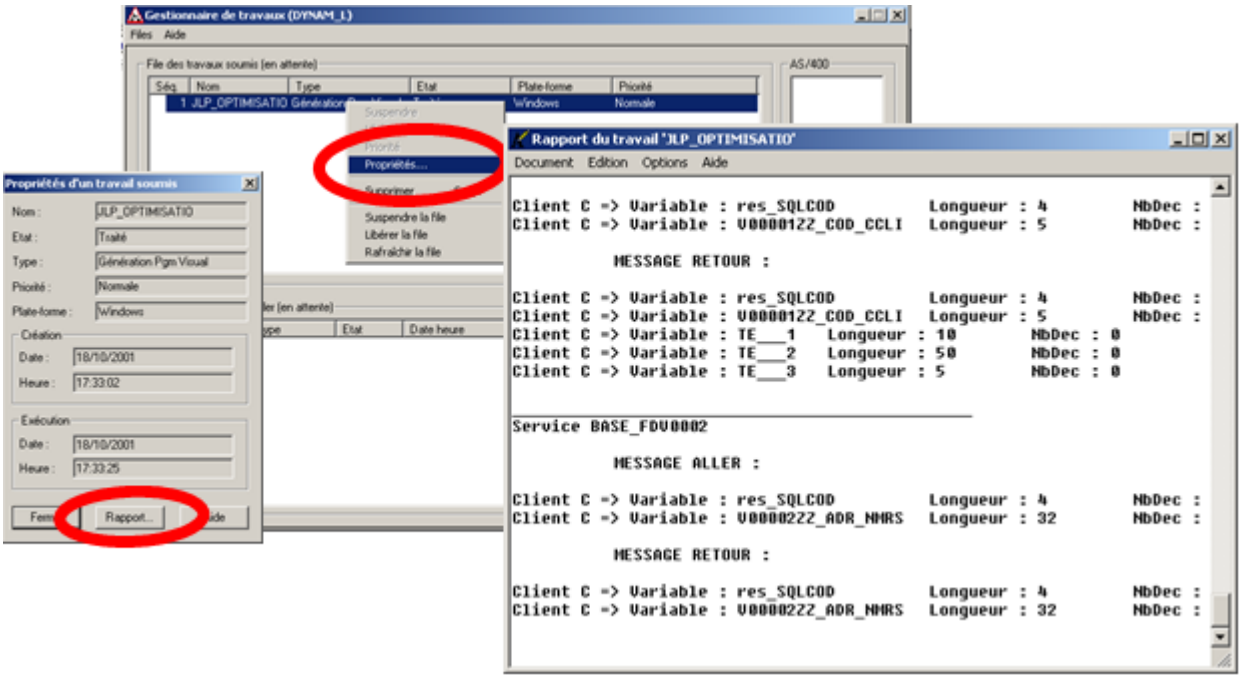


Lors d'un accès à la partie serveur d'un programme Visual Adélia, deux trames de données sont échangées :

- La première transporte du client vers le serveur les données nécessaires à la partie du traitement du côté serveur,
- La seconde ramène du côté client les données traitées par la partie serveur.

Constitution des trames

On peut observer la constitution de chaque trame dans la partie de **vérification** du gestionnaire de travaux :



Règles d'optimisation

Optimiser un programme client/serveur Visual Adélia, c'est respecter les deux règles suivantes :

- **Minimiser les allers-retours**.

> ⚠ En effet, plus le nombre d'allers-retours sera important, plus le système sera ralenti par le temps passé à échanger des données sur le réseau reliant la partie cliente à la partie serveur.

- **Minimiser la taille des messages échangés**.

> ⚠ Il faudra veiller à ne pas transporter vers une partie serveur ou cliente des données inutiles.
> On veillera par exemple à vider une liste graphique sur la partie cliente avant de la remplir côté serveur, plutôt que de l'envoyer pleine au serveur afin qu'elle y soit vidée avant son remplissage.

Exemples d'optimisation

Exemple 1 : Gestion des boucles avec traitements serveurs

Option Lente

```
* --- Déclarations
NUM_E(5,0) I

* --- Programme
I = 0
TANT_QUE I < 1000

  PE_COD_MATRICUL = I
  PE_NOM = 'Agent ' // I

  CREER_SQL PERSONNEL

  I = I + 1
REFAIRE
```

Option Rapide

```
* --- Déclarations
NUM_E(5,0) I

* --- Programme
I = 0
TANT_QUE I < 1000

  PE_COD_MATRICUL = I
  PE_NOM = 'Agent ' // I

  CREER_SQL PERSONNEL

  I = I + 1
REFAIRE
```

Optimisation

Ne pas « découper » une boucle de traitements serveurs

Afin de **minimiser le nombre d'allers-retours** entre partie cliente et partie serveur, il est conseillé de traiter la totalité de la boucle en serveur.
Bien évidemment cette recommandation dépend du volume de données à traiter par la boucle. Dans le cas de volumes de données importants, il faut mettre en œuvre une gestion par **paquets** en utilisant un pas. (Voir l'exemple N°7)

Exemple 2 : VIDER_LST

Option Lente

```
* --- Déclarations
CHARGEMENT CHRGT_PERSO -
LST_PERSO:LISTE PERSONNEL -
*COND(PE_NOM > :ZZ_A_PARTIR) -
*TRI(PE_NOM)

* --- Programme
VIDER_LST LST_PERSO: LISTE
CHARGT_LST CHRGT_PERSO
  INSERER_ELT LST_PERSO:LISTE
FIN_CHRGT_LST
```

Option Rapide

```
* --- Déclarations
CHARGEMENT CHRGT_PERSO -
LST_PERSO:LISTE PERSONNEL -
*COND(PE_NOM > :ZZ_A_PARTIR) -
*TRI(PE_NOM)

* --- Programme
VIDER_LST LST_PERSO: LISTE
CHARGT_LST CHRGT_PERSO
  INSERER_ELT LST_PERSO:LISTE
FIN_CHRGT_LST
```

Toujours vider une liste graphique côté client

Afin de **minimiser la taille de la trame de données échangée** entre partie cliente et partie serveur, on veillera à vider une liste graphique en partie cliente avant de la remplir côté serveur. Ceci afin de ne pas l'envoyer pleine au serveur afin qu'elle y soit vidée avant son remplissage.
De même, lorsqu'une liste mémoire est remplie côté client pour être traitée côté serveur, on veillera à vider la liste mémoire coté serveur avant le retour côté client.

Exemple 3 : Traitements clients et serveurs

Option Lente

```
* --- Programme
VIDER_LST LST_VILLES:LISTE
CHARGT_LST CHRGT_VILLE
  INSERER_ELT LST_VILLES:LISTE
FIN_CHRGT_LST

VIDER_LST LST_PERSO:LISTE
CHARGT_LST CHRGT_PERSO
  INSERER_ELT LST_PERSO:LISTE
FIN_CHARG_LST
```

Option Rapide

```
* --- Programme

VIDER_LST LST_VILLES:LISTE
VIDER_LST LST_PERSO:LISTE

CHARGT_LST CHRGT_VILLE
  INSERER_ELT LST_VILLES:LISTE
FIN_CHRGT_LST

CHARGT_LST CHRGT_PERSO
  INSERER_ELT LST_PERSO:LISTE
FIN_CHARG_LST
```

Optimisation

Regrouper les traitements clients et serveurs

Afin de minimiser le nombre d'allers-retours client/serveur, il faut, autant que possible regrouper les traitements clients et les traitements serveurs.
Bien évidemment, là aussi, cette recommandation dépend du volume de données à traiter.

Exemple 4 : Gestion listes et boucle de traitements serveurs

Non optimisé

```
* --- Programme
  LECTURE_LST LST_PERSO:LISTE *SELECT
   SUPPRIMER_SQL PERSONNEL *COND(PE_MATRICUL = :ZZ_MATRICUL)
  FIN_LECTURE_LST
```

Optimisé

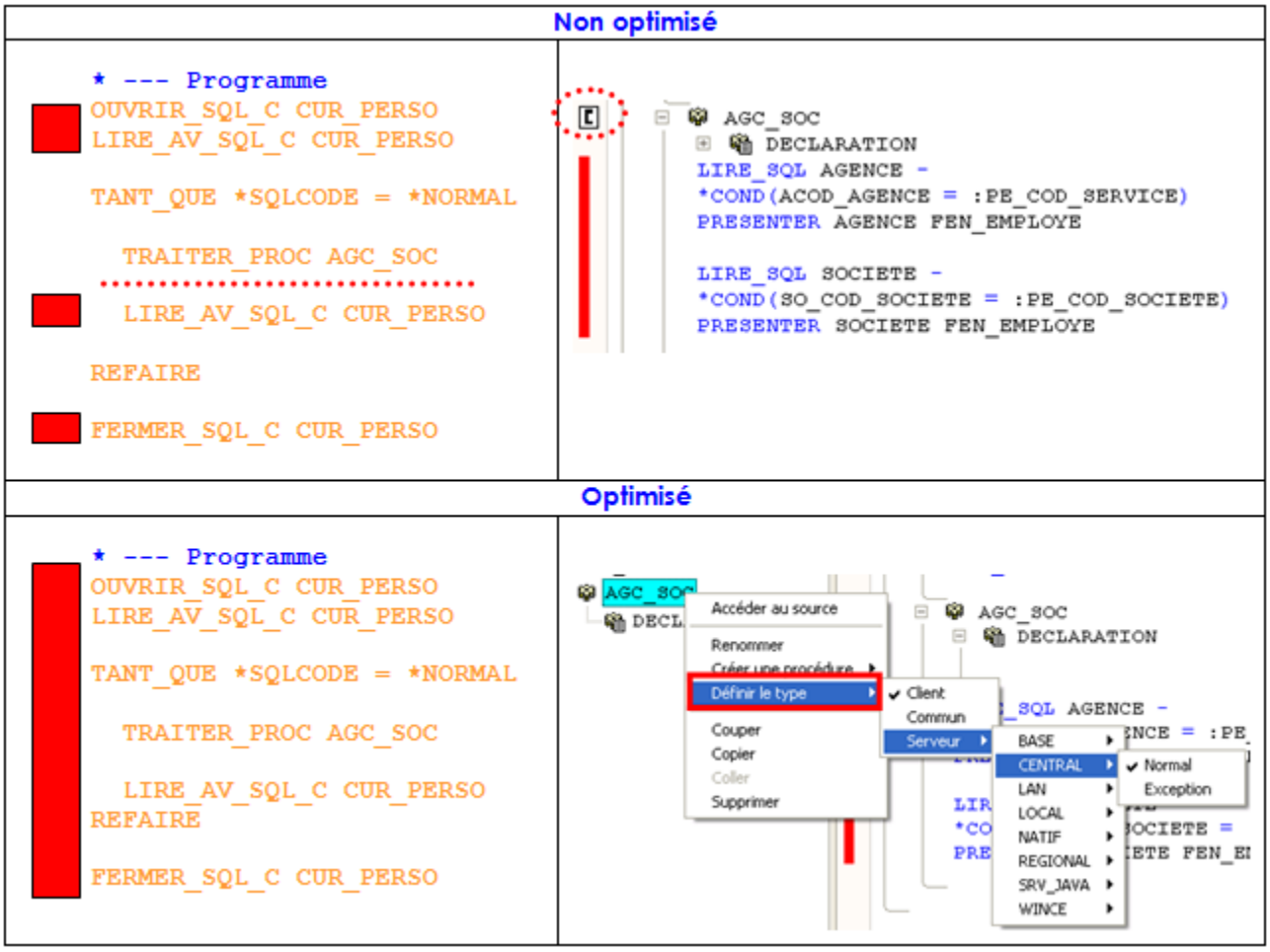
```
* --- Déclaration
  LISTE LSM_TEMP ZZ_CODE_MATRICULE

* --- Programme
  LECTURE_LST LST_PERSO:LISTE *SELECT
   INSERER_ELT LSM_TEMP
  FIN_LECTURE_LST

  LECTURE_LST LSM_TEMP
   SUPPRIMER_SQL PERSONNEL *COND(PE_MATRICUL = :ZZ_MATRICUL)
  FIN_LECTURE_LST
  VIDER_LST LSM_TEMP
```

Optimisation

Utilisation d'une liste mémoire temporaire

Cette optimisation est à mettre en œuvre dans le cas où la liste est en multi-sélection.
Les listes mémoires ne doivent comporter que les colonnes utiles aux traitements. Eviter les *REF_MLD et *REF_L systématique. Dans cet exemple, seul le code matricule est nécessaire.
Conformément aux recommandations de l'exemple N°2, la liste mémoire doit être vidée en serveur.

Exemple 5 : Gestion procédure



Optimisation

Attention au type client ou serveur d'une procédure afin de minimiser les allers-retours client/serveur

Lorsqu'une procédure est créée, elle est, par défaut, de **type Client**. Si la totalité de ses traitements peut être effectuée en serveur, il faut changer son type en **Serveur**.
Une procédure qui ne fait pas d'accès base de données peut aussi avoir pour type **Commun** afin d'être appelée, selon les besoins, soit en **Client** soit en **Serveur**.

Exemple 6 : Visual Batch ou SAdelia

Non optimisé

```
* --- Programme
OUVRIR_SQL_C CUR_PERSO
LIRE_AV_SQL_C CUR_PERSO

TANT_QUE *SQLCODE = *NORMAL

  APPELER VA_AGC_SOC -
  PSRV PAGC PSOC

  LIRE_AV_SQL_C CUR_PERSO

REFAIRE

FERMER_SQL_C CUR_PERSO
```

```
DECL PGM
  REF(ALIB_AGENCE)     PLIB_AGC
  REF(SO_NOM_SOCIETE) PLIB_SOC
  REF(ACOD_AGENCE)     PCOD_AGC
  PARAM PCOD_AGC PLIB_AGC PLIB_SOC
INIT PGM
  LIRE_SQL AGENCE -
  *COND(ACOD_AGENCE = :PCOD_AGC)
  SI *SQLCODE = *NORMAL
     PLIB_AGC = ALIB_AGENCE
     LIRE_SQL SOCIETE -
     *COND(SO_COD_SOCIETE = :ACOD_SOCIETE)
     SI *SQLCODE = *NORMAL
        PLIB_SOC =  SO_NOM_SOCIETE
     FIN
  FIN
```

Optimisé

```
* --- Programme
OUVRIR_SQL_C CUR_PERSO
LIRE_AV_SQL_C CUR_PERSO

TANT_QUE *SQLCODE = *NORMAL

  APPELER SA_AGC_SOC -
  PSRV PAGC PSOC

  LIRE_AV_SQL_C CUR_PERSO
REFAIRE

FERMER_SQL_C CUR_PERSO
```

```
DECL PGM
  REF(ALIB_AGENCE)     PLIB_AGC
  REF(SO_NOM_SOCIETE) PLIB_SOC
  REF(ACOD_AGENCE)     PCOD_AGC
  PARAM PCOD_AGC PLIB_AGC PLIB_SOC
INIT PGM
  LIRE_SQL AGENCE -
  *COND(ACOD_AGENCE = :PCOD_AGC)
  SI *SQLCODE = *NORMAL
     PLIB_AGC = ALIB_AGENCE
     LIRE_SQL SOCIETE -
     *COND(SO_COD_SOCIETE = :ACOD_SOCIETE)
     SI *SQLCODE = *NORMAL
        PLIB_SOC =  SO_NOM_SOCIETE
     FIN
  FIN
```

Optimisation

Attention au type de programme Batch : Visual Adelia batch ou SAdelia batch afin de minimiser les allers-retours client/serveur

Si la totalité des traitements d'un programme batch peut être effectuée en serveur, il faut créer un programme de type **SAdelia** et non pas un programme **Visual Adelia batch**.
En effet, ce dernier, comporte une partie **Client** (donc un fichier **.Exe** et un fichier .DLL), et doit donc être appelé en **Client**.
Alors que le **SAdelia** est entièrement généré sur le serveur. Il ne comporte donc pas de partie **Client** et est appelé en **Serveur**.

Exemple 7 : Traitement par « paquets » d'enregistrements

Non optimisé

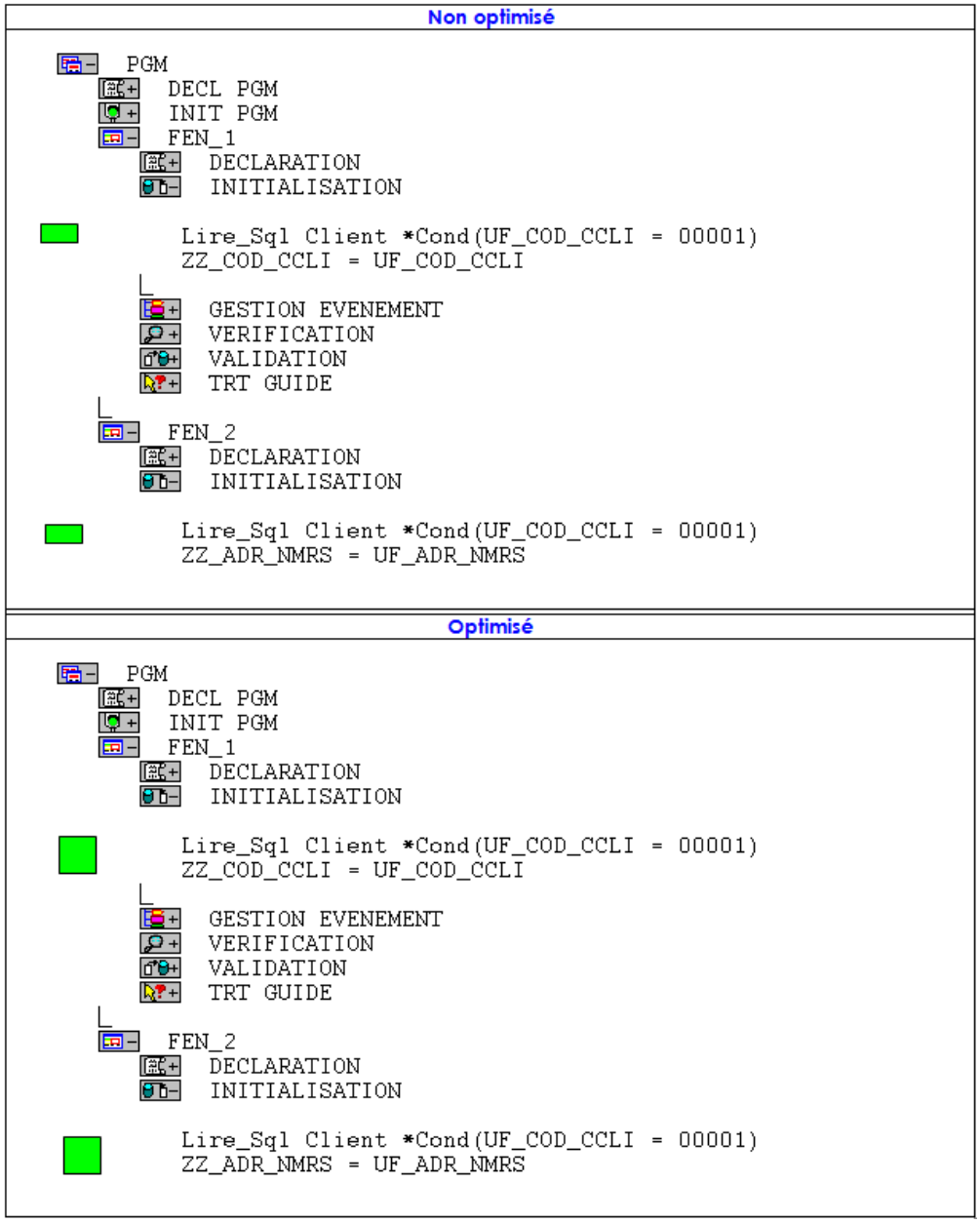
```
* --- Déclaration
LISTE LSM_TEMP *REF_MLD(ARCHIV_PERSO)

* --- Programme
VIDER_LST LSM_TEMP

OUVRIR_SQL_C CUR_PERSO
LIRE_AV_SQL_C CUR_PERSO
TANT_QUE *SQLCODE = *NORMAL
    colonnes LSM_TEMP = zones PERSONNEL
    INSERER_ELT LSM_TEMP
    LIRE_AV_SQL_C CUR_PERSO
  REFAIRE
FERMER_SQL_C CUR_PERSO

LECTURE_LST LSM_TEMP
  CREER_SQL ARCHIV_PERSO
FIN_LECTURE_LST
```

Optimisé

```
* --- Déclaration
LISTE LSM_TEMP *REF_MLD(ARCHIV_PERSO)

* --- Programme
OUVRIR_SQL_C CUR_PERSO
LIRE_AV_SQL_C CUR_PERSO
WW_SQLCODE = *SQLCODE

TANT_QUE WW_SQLCODE = *NORMAL
  WW_PAS = 1

  TANT_QUE WW_SQLCODE = *NORMAL ET WW_PAS <= WW_BORNE_SUPP
    WW_PAS = WW_PAS + 1
    colonnes LSM_TEMP = zones PERSONNEL
    INSERER_ELT LSM_TEMP
    LIRE_AV_SQL_C CUR_PERSO
    WW_SQLCODE = *SQLCODE
  REFAIRE

  LECTURE_LST LSM_TEMP
    CREER_SQL ARCHIV_PERSO
  FIN_LECTURE_LST
  VIDER_LST LSM_TEMP

REFAIRE

FERMER_SQL_C CUR_PERSO
```

Afin d'améliorer les performances sur les traitements de volume de données important, **Il est recommandé de charger les enregistrements de base de données par « pas »** dans une liste mémoire sur le serveur.Le **pas** est défini en fonction du volume supposé des fichiers traités et en fonction de la somme des longueurs des zones à prendre en compte.
Si la lecture de la liste mémoire doit se faire par la suite sur un autre serveur (pour une lecture ou une création base de données par exemple) l'instruction **VIDER_LST** sur cette liste doit se faire après la gestion de la liste en partie serveur, comme cela est le cas dans notre exemple.

Exemple 8 : PRESENTER/RAPATRIER



Optimisation

Utiliser autant que possible les zones fichiers dans les parties **serveur**

Particulièrement en ce qui concerne les instructions **PRESENTER** ou **RAPATRIER**.

Sinon, à chaque lecture toutes les zones fichiers utilisées dans le programme seront retournées en **Client** depuis la partie **Serveur**.

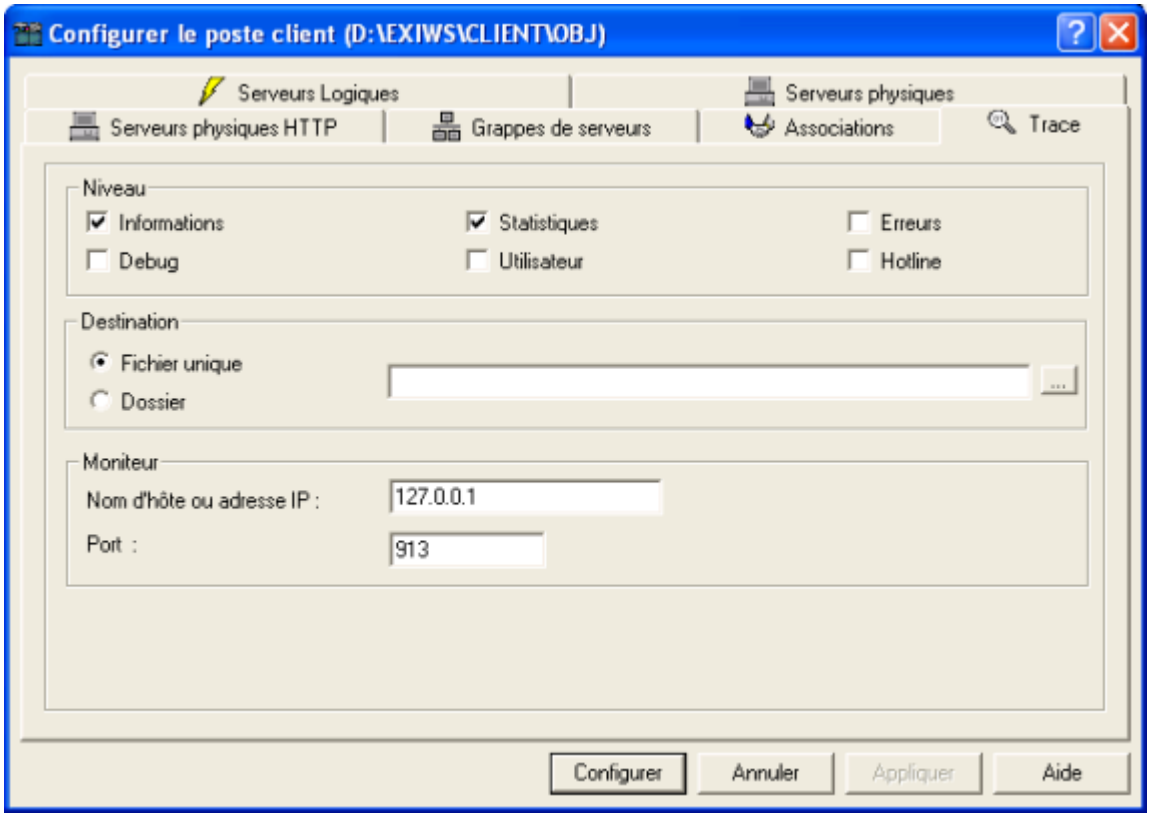
Trace

L'outil de trace permet d'observer les trames qui circulent entre la partie cliente et la partie serveur d'un programme Visual Adélia.
Pour l'utiliser, on suivra les trois étapes suivantes :

1. Configuration du poste client,
2. Lancement du programme **espion**,
3. Lancement du programme à observer.

Configuration du poste client

Configurer le poste client consiste à indiquer vers quelle machine seront envoyées les traces d'échange. Dans l'exemple suivant, les échanges seront envoyés vers la machine cliente elle-même.



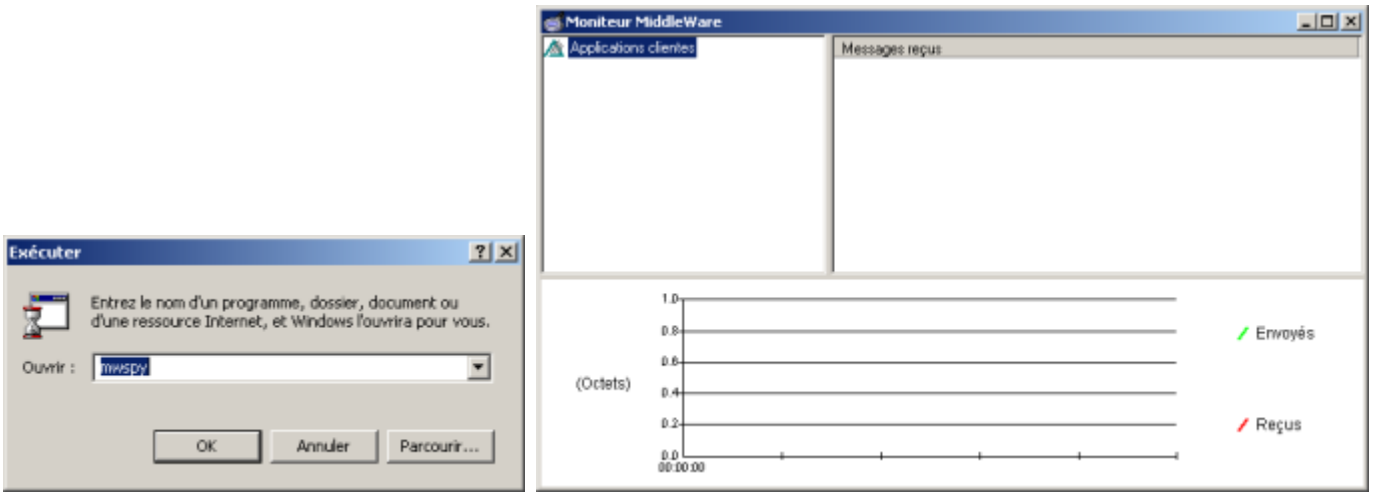
concernant la section suivante, l'outil mwspy.exe n'est plus distribué depuis la V13 d'Adélia Studio.

Lancement du programme "espion"

Le programme moniteur **espion** (**mwspy.exe**) peut être lancé depuis la boîte de démarrage de Windows ou en double-cliquant sur l'exécutable qui se trouve dans le répertoire du runtime Adélia (habituellement **c:\Program Files\Adelia Studio**).

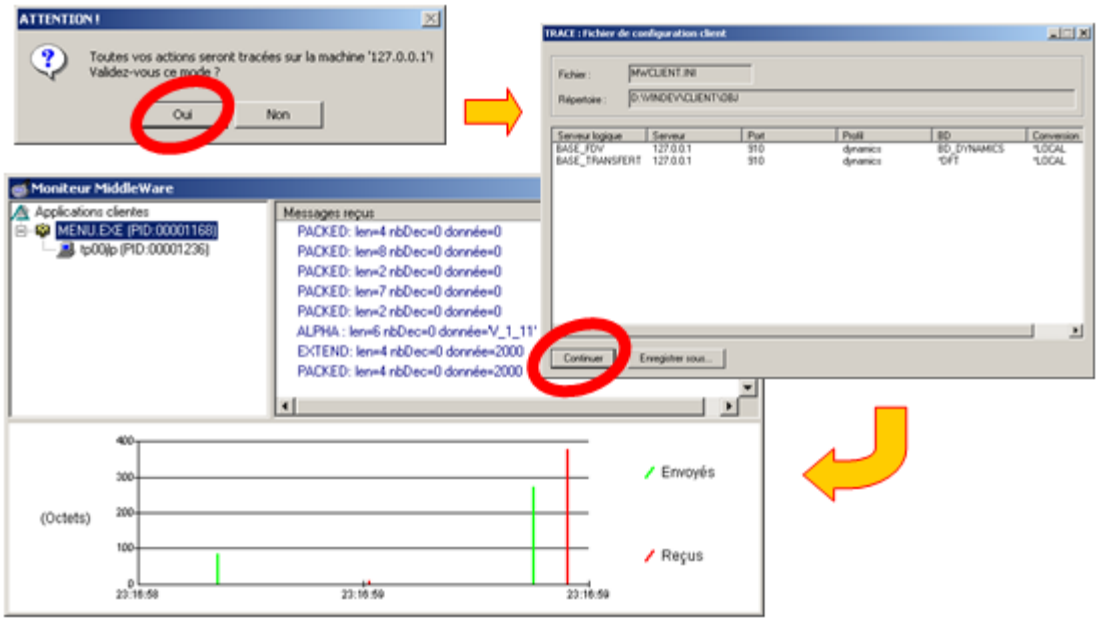
Une fois lancé, le moniteur de trace affiche une fenêtre comportant trois parties :

- La partie supérieure gauche montre les **processus démarrés**,
- La partie supérieure droite montre le contenu des **trames échangées**,
- La partie inférieure montre les **volumes de données échangées** en fonction du temps.



Lancement du programme à observer

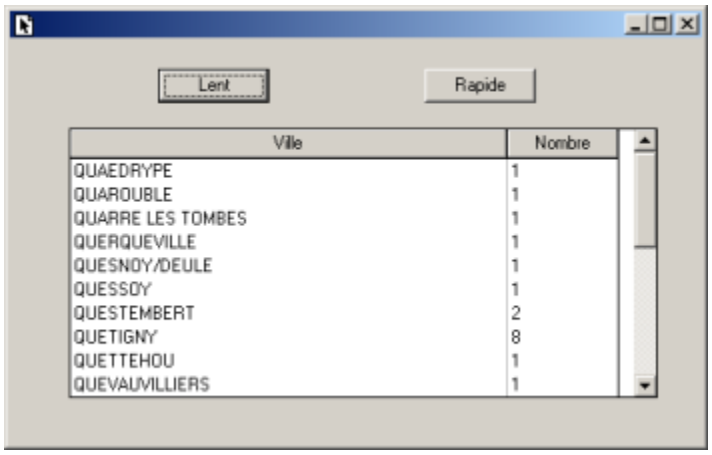
Une fois la configuration cliente modifiée et le programme moniteur lancé, on peut exécuter le programme à observer.



Les données envoyées vers le serveur sont représentées par des **traces vertes/claires** et celles reçues du serveur par des **traces rouges/sombres**.

Exemples de Trace

Soit un programme affichant la liste des villes d'un fichier du personnel en calculant par le biais d'une rupture sur curseur SQL le nombre de personnes par ville :

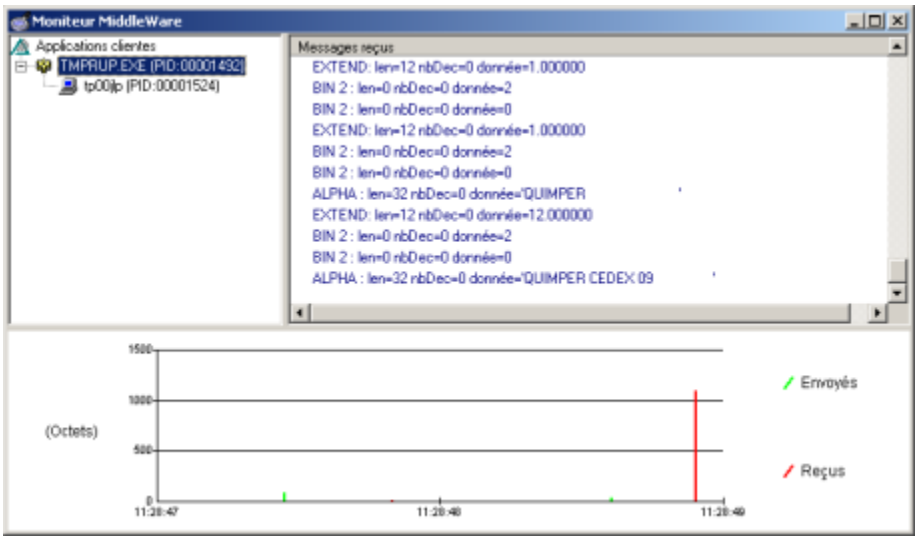


Selon que la rupture sera de type cliente ou de type serveur, le parcours du fichier du personnel sera lent (autant d'allers-retours que d'enregistrements), ou bien rapide (un seul aller-retour au cours duquel le serveur effectue tout le parcours du fichier).

Traitement lent : rupture de type cliente

Sur l'image précédente, on voit dans la partie basse le nombre important d'allers-retours client/serveur.
En effet, la rupture étant de type cliente, la boucle de lecture du curseur est donc « découpée » afin de gérer la rupture en client.

Traitement rapide : rupture de type serveur



Sur l'image précédente, on voit dans la partie basse qu'on a cette fois un seul message envoyé et un seul message reçu.
En effet, la gestion de la rupture se fait en serveur dans la boucle de lecture du curseur. La totalité des éléments à charger dans la liste graphique est renvoyée en client en une seule fois en fin de traitements serveur.

Profileur pour applications Visual Adélia

Le but de cet utilitaire est d'analyser les performances d'une application Visual Adélia.
Suite à l'exécution de l'application, il permet de connaître les programmes (avec comme niveau de détail les paragraphes et les parties serveur) exécutés effectivement (avec leur nombre d'appels) et les temps d'exécution passés dans ces programmes (au niveau global du programme ou au niveau de ses paragraphes et parties serveur).

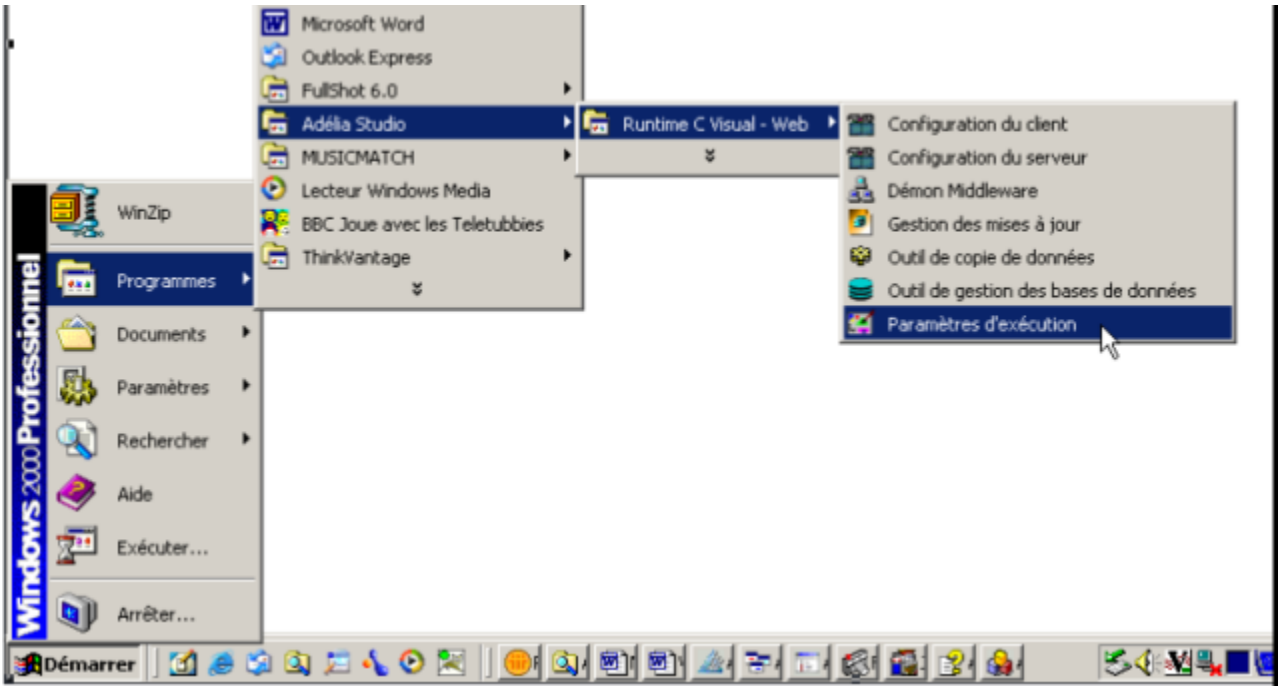
Le Profileur Visual Adélia permet d'obtenir des traces de l'exécution des applications client/serveur Visual Adélia.
Il supporte :

- les parties clientes Windows et Java ;
- les parties serveur Windows, Linux, Unix (AIX et Solaris), AS/400 et Java.

Information utiles

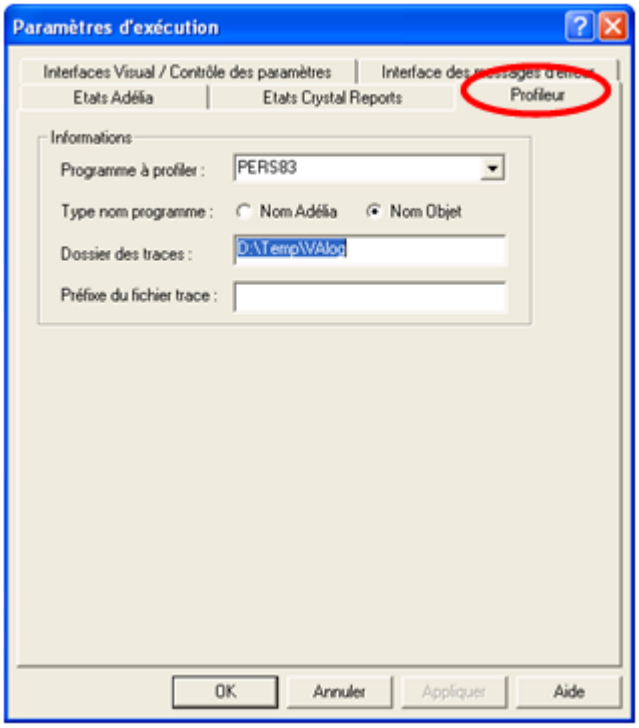
Les parties serveur d'un programme Visual batch appelées depuis une partie serveur ne sont pas tracées.
Dans le cas de l'exécution non modale d'une fenêtre, certains paragraphes peuvent ne pas être rattachés à la bonne fenêtre.
Il est possible de tracer les programmes Visual Adélia Batch, Interactif et Serveur.Configuration du profileur :Les parties serveur d'un programme Visual batch appelées depuis une partie serveur ne sont pas tracées.
Dans le cas de l'exécution non modale d'une fenêtre, certains paragraphes peuvent ne pas être rattachés à la bonne fenêtre.
Il est possible de tracer les programmes Visual Adélia Batch, Interactif et Serveur.

- Option **Paramètres d'exécution** du **Runtime Visual-Web**



- Onglet **Profileur**

Sélection du programme, ou bien de tous les programmes, à **profiler :**

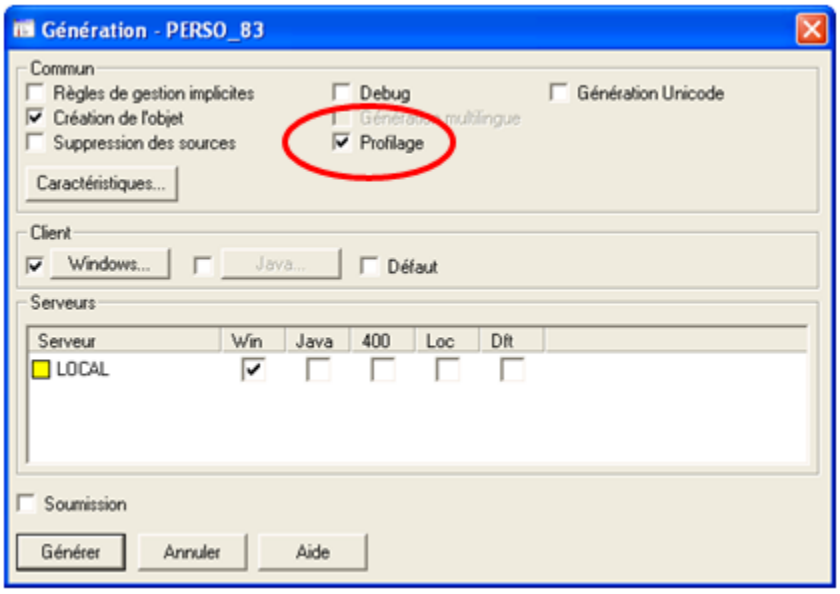


- Lors de l'exécution du programme PERSO83 les traces sont générées dans le **Dossier des traces**, dans un fichier de traces ayant la forme suivante : *(Préfixe ou nom programme)-date-heure.**prf***

Remarque : Le profilage d'un programme Visual Adélia n'est possible que si ce dernier a été compilé en mode **profilage**.
Pour cela, lors de la compilation du programme, cochez l'option **Profilage** située :

- dans la boîte de dialogue des **caractéristiques de compilation du programme**,

- ou encore, dans la boîte de dialogue des attributs d'environnement, onglet **Génération L3G - Général**, section **Options de génération**.



Utilisation du profileur :

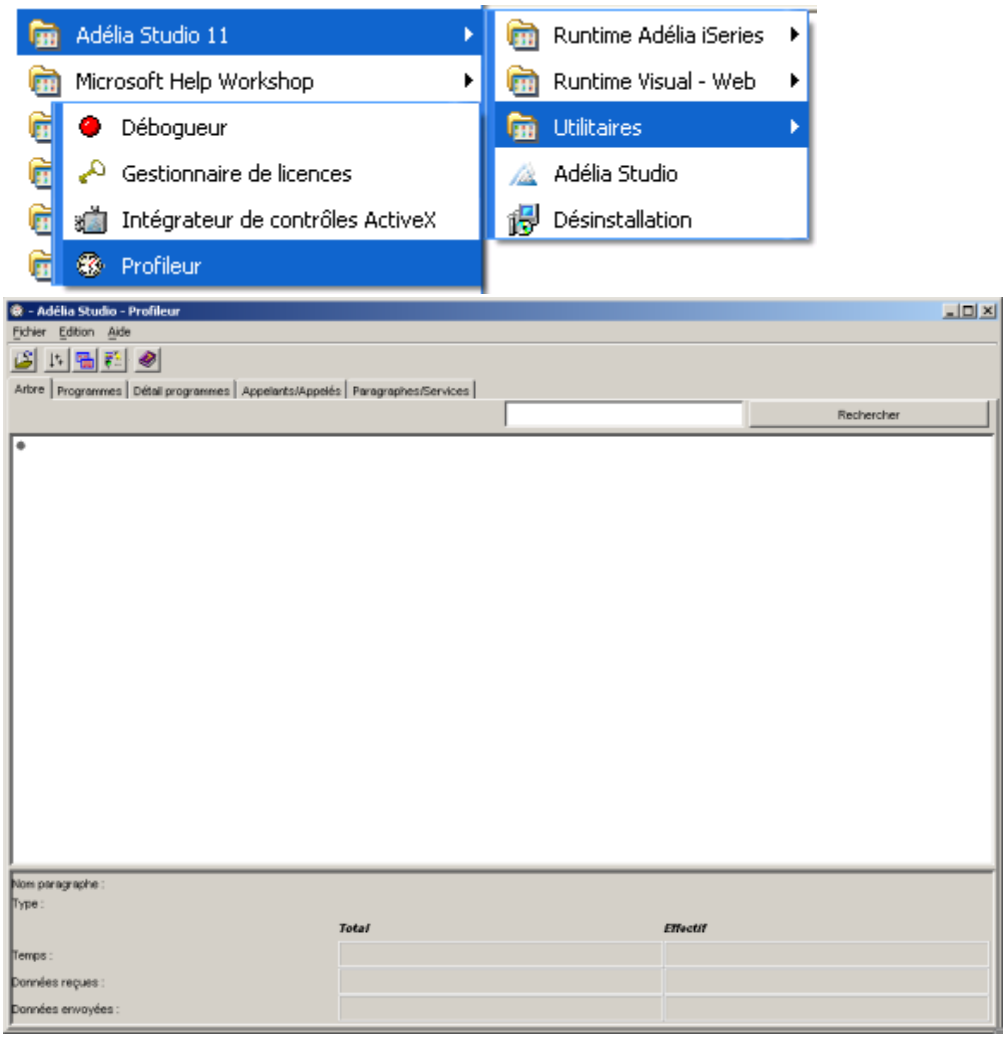
- Exécution du programme

Lorsque le paramétrage du Profileur est effectué, le programme doit être lancé sur le poste client et être exécuté.
Le fichier des traces est alors mis à jour.

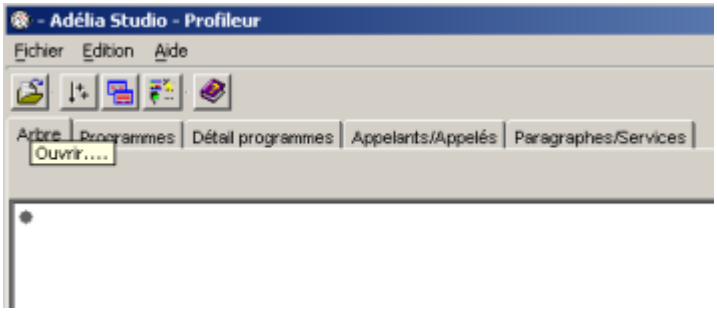
- Visualisation des traces

Après exécution du programme, lancer l'interface du Profileur.

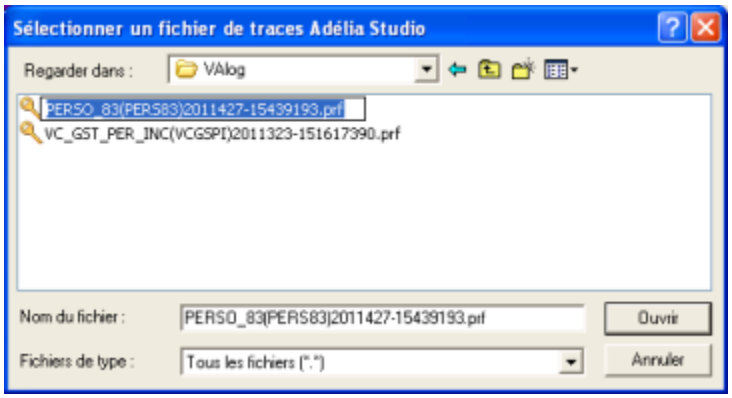
- Option **Profileur** du menu **Utilitaires** d'Adélia Studio



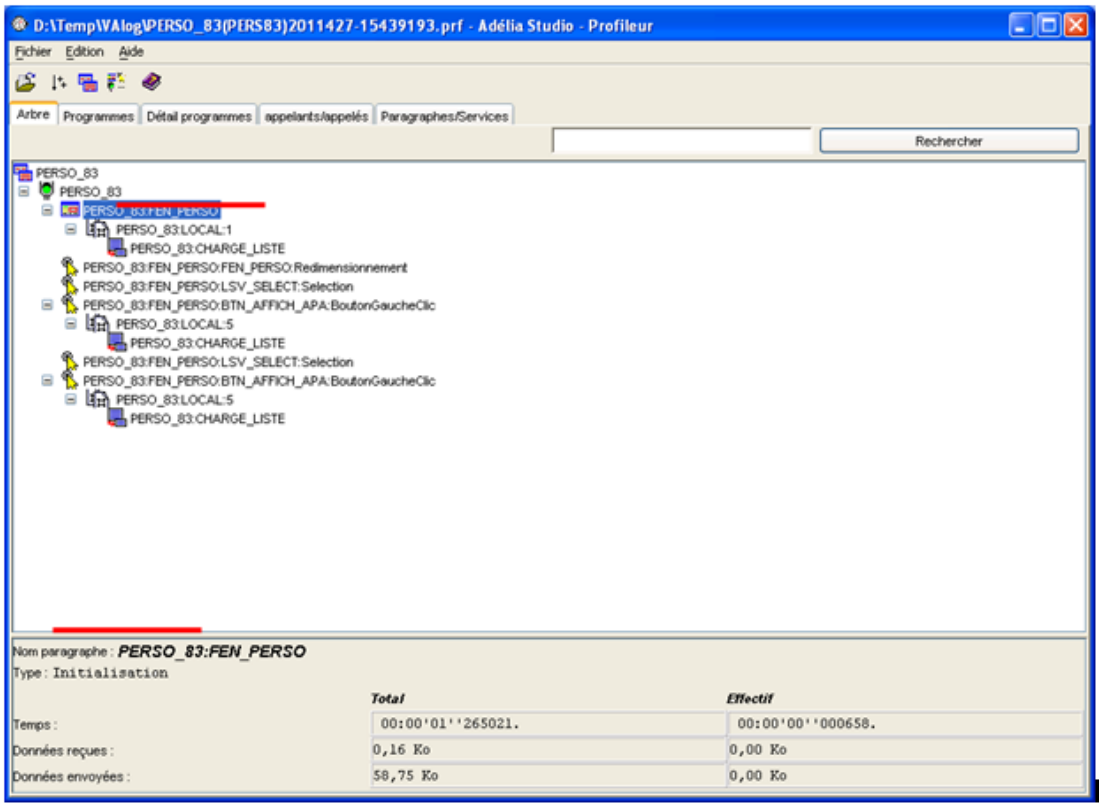
- Option **Ouvrir**... du menu **Fichier** ou icône **folder**



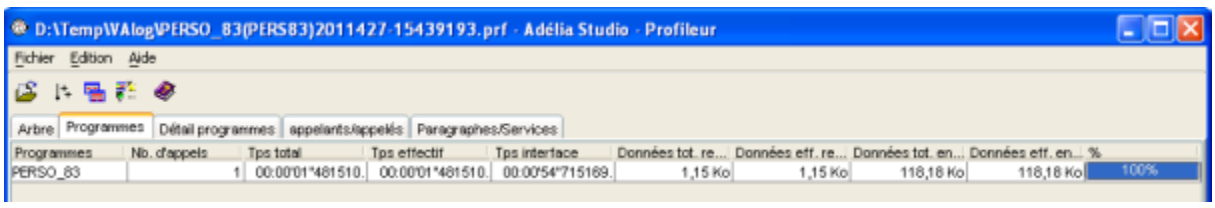
- Recherche du fichier « **.prf** » des traces dans le répertoire paramétré dans le profileur : **VAlog**



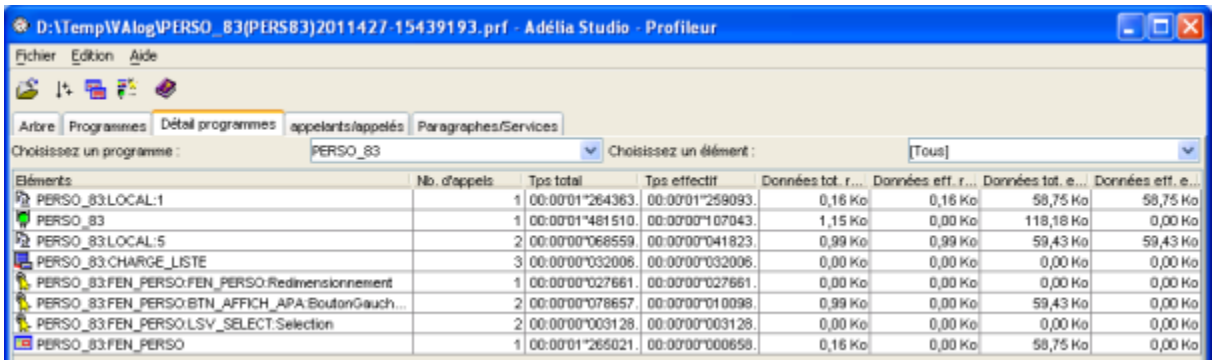
- Onglet **Arbre** : affiche une vue arborescente de l'exécution du programme en mode profilage.



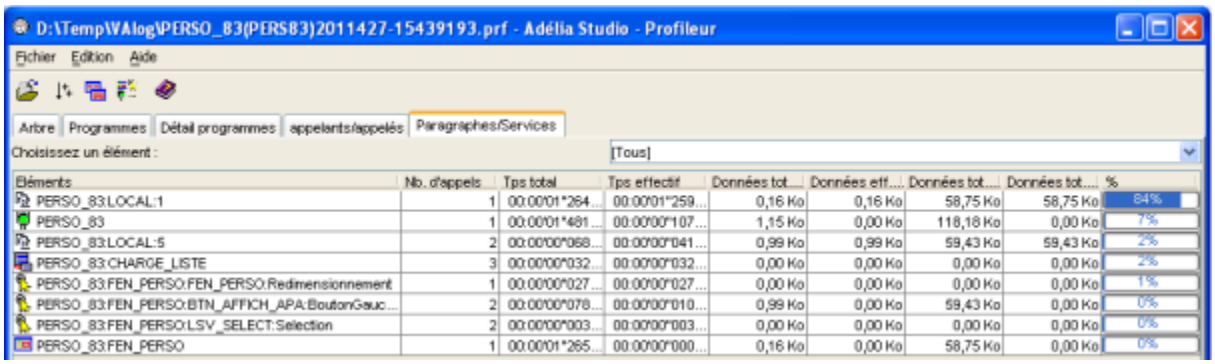
- Onglet **Programmes** : temps d'exécution et données reçues et envoyées



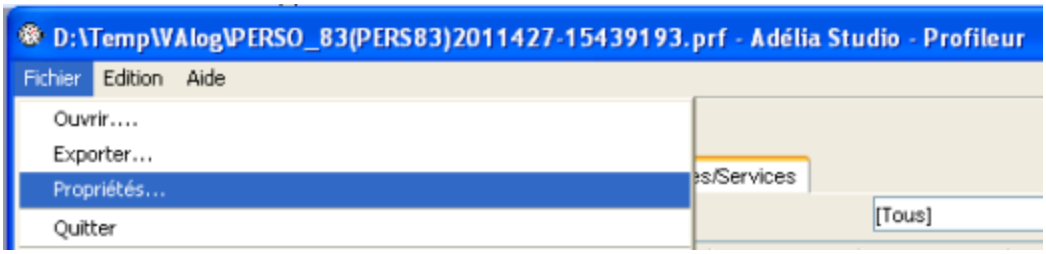
- Onglet **Détail programmes** :

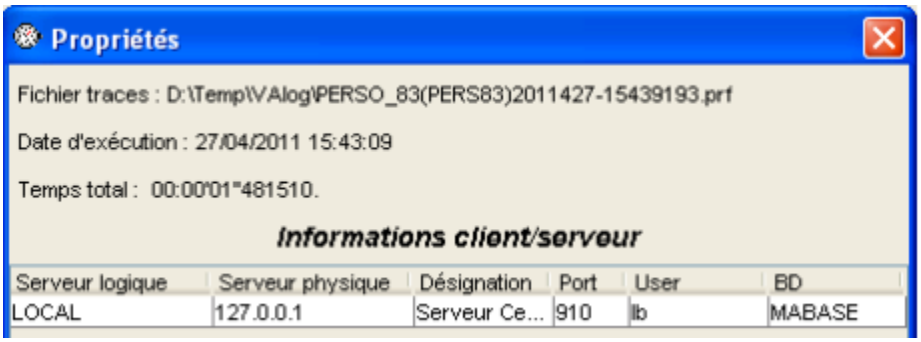


- Onglet **Paragraphes/Services** : paragraphes et services exécutés



- Option **Propriétés**... du menu **Fichier** :





- **Serveurs logiques** utilisés, machine physique et base de données accédée.

SQL

Index

Utiliser SQL dans le cadre de programmes client/serveur Visual Adelia Studio permet de faire les lectures de bases de données sur le fichier physique plutôt que sur les fichiers logiques (ou index) comme cela est le cas en natif AS/400.
Cependant, lors d'une lecture sur un fichier ne possédant **pas d'index**, le moteur SQL va lui-même indexer le fichier afin de générer un ordre d'accès qui corresponde à l'accès demandé. On peut comprendre que cette **génération d'index** puisse être très **pénalisante** suivant la taille des tables ou fichiers accédés.
Si par contre le moteur SQL trouve un index correspondant à l'accès demandé par le programme, il va l'utiliser.
Aussi, il faudra toujours veiller à générer les index correspondant aux accès SQL les plus fréquents effectués dans les programmes afin de rendre les accès aux bases de données les plus rapides possibles.

Conditions

Définir si possible des **conditions** en indiquant les zones correspondants aux index existants dans le paramètre ***COND**.

Exemple : LIRE_SQL NomEntité *COL(NomPpté1, NomPpté2, …) ***COND**(NomPpté2= :w_ NomPpté2)
Avec NomPpté2, zone clé d'un index sur l'entité NomEntité.

*Col et *Opt

Nombre de colonnes

Minimiser le **nombre de colonnes lues** en privilégiant l'emploi du paramètre ***COL**.

Exemple : LIRE_SQL NomEntité ***COL**(NomPpté1, NomPpté2, …) *COND(...)

Optimisation de curseur

Dans les **curseurs**, utiliser la clause ***OPT** dans la définition pour améliorer les performances de lecture sur les **n** premiers enregistrements indiqués sur cette clause.
Exemple **:** **CURSEUR** CUR_BASE SQL_BASE ***COL(B1, B2) *COND(B > 50)** ***OPT**(100)

PRTSQLINF

La commande AS/400 **PRTSQLINF** permet d'analyser un programme effectuant des accès SQL. Elle génère un spoule dans lequel elle indique, pour chaque accès SQL, le fichier index qu'elle compte utiliser et le temps moyen d'accès à l'information demandée.

> **Attention** cependant, car ces informations sont générées au moment de la compilation du programme, en fonction de la taille des données rencontrées dans les fichiers de données en ligne au moment de la compilation. On devra donc veiller, pour que cette information soit pertinente, à ce que ces fichiers aient une taille la plus proche possible de celles des fichiers d'exploitation.

Gestion des listes

Objectif

Gérer au mieux le remplissage des listes graphiques à partir des zones d'une ou plusieurs tables de la base de données en tenant compte du volume des données à charger et des critères de recherche.

Critères de choix

1. Rechercher le volume potentiel des données susceptibles d'être affichées dans la liste graphique. Le résultat de cette recherche déterminera si le remplissage sera **statique** ou **dynamique**.

Ce volume doit être estimé en calculant la somme des longueurs des zones qui doivent être affichées dans la liste graphique, multipliée par le nombre d'occurrences de la liste susceptibles de passer du serveur au client.

Remplissage statique

Le volume potentiel de données à charger est relativement faible (normalement inférieur à 1000 occurrences pour une longueur totale d'occurrence inférieure à 100 caractères, soit **moins de 100 ko**), le chargement pourra se faire en une seule fois.

- Si les critères sont simples, utiliser l'instruction **CHARGEMENT** sans indiquer de pas.
- Si les critères sont élaborés, utiliser un curseur explicite en passant éventuellement par une liste mémoire intermédiaire pour gérer au mieux le transfert des données entre les parties serveur et les parties clientes.
  Dans ce cas pour optimiser la gestion de l'affichage de la liste graphique il faut utiliser la méthode **RAFRAICHIR_LISTE**.

Remplissage dynamique

Le volume potentiel de données est important (**supérieur à 100 ko**), le chargement se fera en plusieurs fois en définissant un pas de pagination, même remarque que ci-dessus pour les critères.
Veiller à ce que l'événement **PaginationDynamiqueAvant** soit défini sur la liste graphique.