Les composants 6.7 (I)

Par
Yves CASQUEL, Support technique 4D S.A.
Note technique 4D-200009-25-FR
Version 1

Date 1 Septembre 2000

Résumé

Cette note technique permet de se familiariser avec les composants : nouveaux “modules”de la version 6.7 de 4ème Dimenson. Une deuxième partie de cette note détaillera en profondeur les autres subtilités de ce nouveau concept.

4D Notes techniques

Introduction

Cette note techique a pour but de vous faire découvrir une des richesses de la version 6.7 de 4D et de 4D Insider : les composants.

Dans vette première partie, nous allons décrire les étapes de réalisation d'un composant simple.

Cette note est construite suivant ce plan :

I - Définition d'un composant

Un composant est un ensemble d'objets 4D rassemblés dans différents dossiers qui ont pour particularité de posséder un droit d'accès.

Un composant peut contenir des tables qui seront insérées dans la base de destination. Ces tables feront parties intégrantes de la structure. Elles peuvent être considérées comme une partie de structure indépendante pour le développeur.

L'insertion de table dans un composant n'est pas obligatoire, il faut alors considérer celui-ci comme une bibliothèque proposant un certain nombre de services.

Dans tous les cas, le composant fait partie intégrante d'une structure 4D.

Ces objets sont rassemblés à l'aide de 4D Insider.

Il n'y a pas de différence de terminologie entre le composant que vous allez développer et le composant terminé qui sera intégré dans une base dite de destination.

IMPORTANT : il faudra donc parler de composant en cours de développement et de composant intégré (celui qui est intégré dans une base).
Le composant en cours de développement est stocké dans la structure originelle au même titre que les groupes.
Il n'y a pas de limite au nombre de composants.

FONDAMENTAL : un composant doit être autonome. Pour de plus amples renseignements, vous référer à la documentation de 4D Insider.

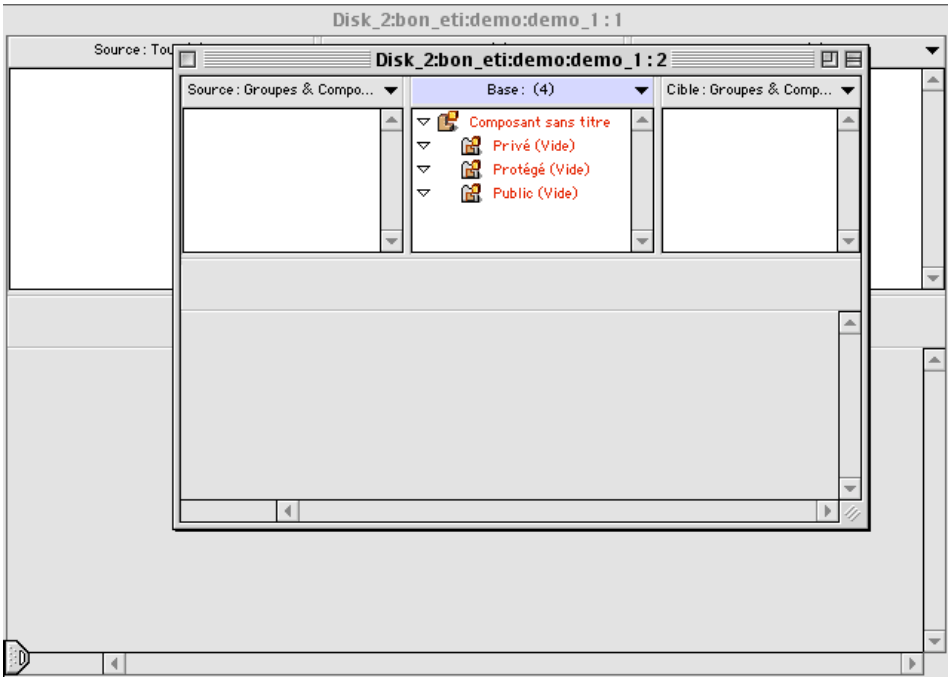
II - Organisation d'un composant 4D Insider

1) Pour le développeur du composant :

Un composant est organisé sous la forme de trois dossiers (voir ci-dessous) qui accueillent des objets 4D. Tous ces objets sont bien entendus manipulables sans aucune restriction. Le classement des objets dans les dossiers suivants permet simplement de restreindre leurs utilisations au client qui intégrera le composant :

Dossier protégé
Dossier privé
Dossier public

Ci-dessous la composition d'un composant dans 4D Insider :



2) Pour le client utilisateur du composant.

Dossier protégé.

Ce dossier contient la liste des objets que le développeur du composant permet d'utiliser. Ces objets ne sont visibles que par leur nom et ne peuvent, en aucun cas, êtres modifiés, supprimés ou renommés. Ce sont donc ces objets qui vont former les points d'entrée du composant que nous verrons au paragraphe XI-3.

Dossier public.

Ce dossier contient la liste des objets que le développeur du composant permet d'utiliser et de modifier (par exemple une méthode ou un formulaire). Ces objets sont visibles par leur nom mais aussi par leur contenu. Cependant ils ne peuvent, en aucun cas, êtres supprimés ou renommés.

III - Les objets acceptés dans un composant

Il s'agit des déplaçables utilisés par 4D Insider :

Tables et champs
Trigger
Formulaires
Méthodes formulaires, objets et projets
Menu et barre de menu
Info bulle
Ressources STR# et PICT
Images de la bibliothèque

Feuilles de style
Enumérations

Les variables, les ensembles, les sélections et les sémaphores pouvant aussi faire partie du composant.

Les objets interdits sont les suivants :

- Toutes les méthodes base.
- Les plug-ins. Ceci ne veut pas dire qu'il est interdit d'utiliser les plug-ins (commandes) mais, attention, ils devront être présent dans le dossier Win4DX/Mac4DX situé à côté de la base de destination ou dans le dossier 4D du système.
- Les groupes.
- Les composants.

Remarque : il n'est pas possible d'utiliser un pointeur vers un objet qui est en dehors d'un composant.

Attention : les tables et les champs insérés dans un composant ne peuvent pas avoir le statut "privé".

IV - Réalisation d'un composant simple

Création
Mise en forme
Génération
Intégration
Suppression

1) Création

Le composant se génère à partir de 4D Insider. Deux solutions vous sont proposées :

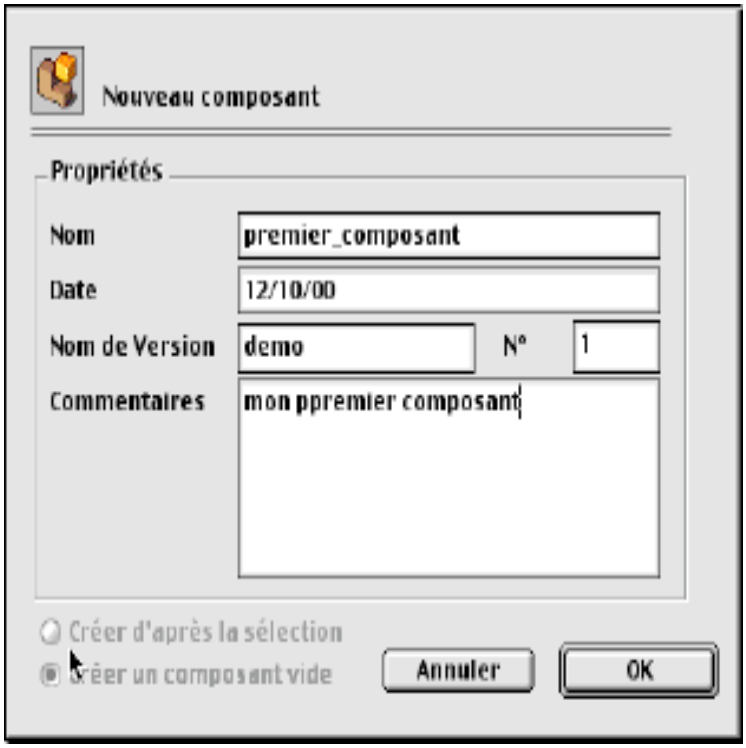
Créer le composant d'après la sélection.
Créer un composant vide.

Le fait de choisir l'option de création ne veut pas dire que le composant est créé physiquement sur le disque. Cette action a pour effet de construire un ensemble de dossiers sous la forme de ressources à l'intérieur de votre structure. Ce mécanisme sera présenté dans quelques instants.

Attention : tous changements à l'intérieur d'un composant par 4D Insider sera répercuté sur la structure de la base. Ceci implique la sauvegarde de la structure de la base si vous voulez sauvegarder votre composant dans un état particulier (par exemple si vous avez changé une méthode de dossier). Il n'est pas possible de sauvegarder un composant séparément.

Dans cette démonstration, nous allons créer un composant vide puis lui intégrer des objets.

Ci-dessous le dialogue de création d'un composant :



Note : le curseur montre les boutons qui permettent de créer un composant vierge ou à partir d'une sélection.

2) Mise en forme (assemblage à l'aide de 4D Insider)

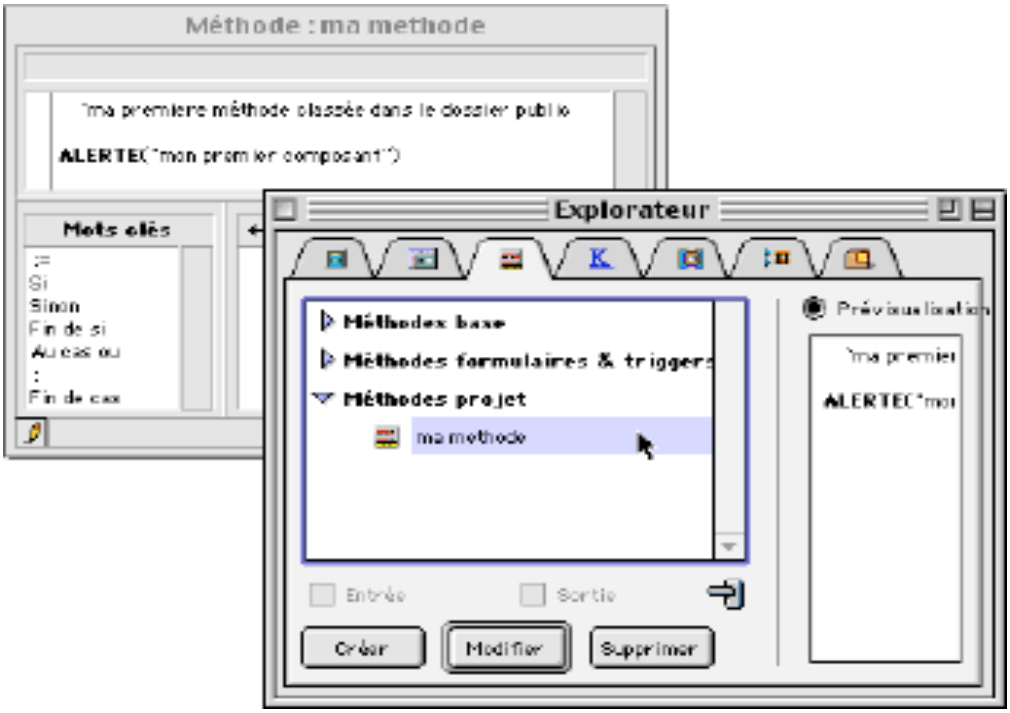
Ceci implique que les méthodes qui vont faire partie du composant aient été créées et que leur enchaînement soit fonctionnel. Il faut donc dans cette étape que vous ayez défini la répartition de vos objets ainsi que la manière dont va être amené le futur client à utiliser votre composant. C'est-à-dire qu'il faut avoir défini les points d'entrée que nous étudierons par la suite.

Dans la démonstration illustrée ci-dessous, notre composant est constitué d'une méthode dont voici le code :

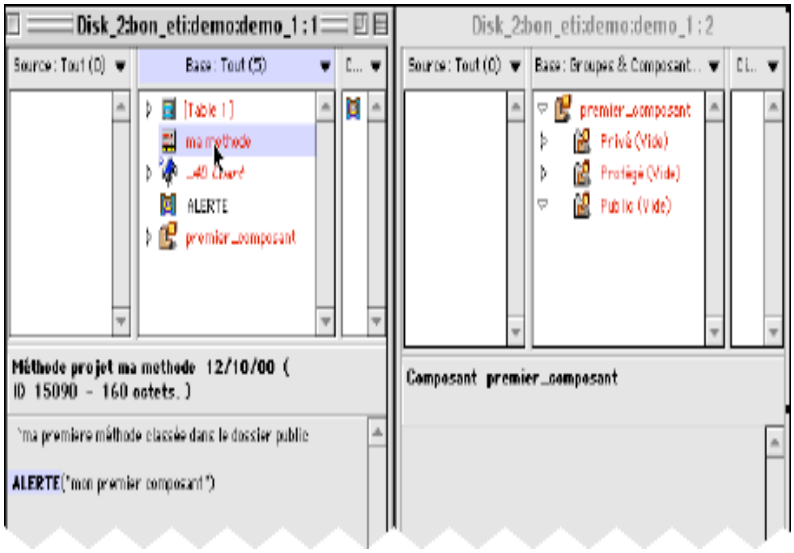
```
Méthode : ma méthode

ALERTE (" mon premier composant ")
```

Les trois figures ci-dessous montre l'intégration d'objets (dans le cas présent l'objet "ma méthode") dans votre composant.

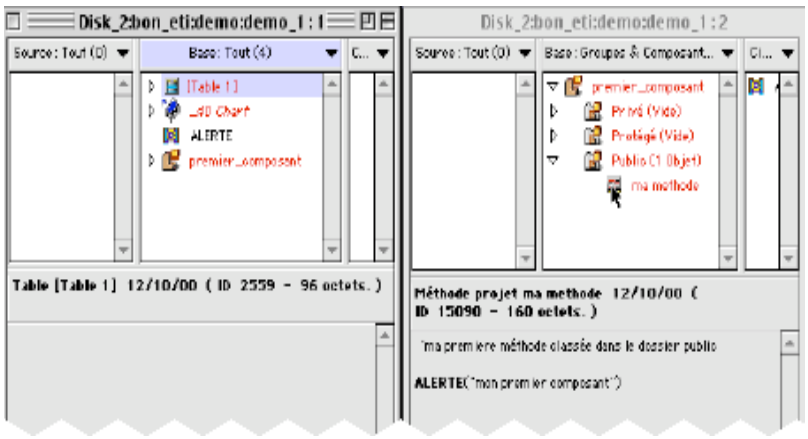
1) La méthode dans 4D :



2) Le composant vide :



3) Le composant avec l'objet " ma méthode " intégré :

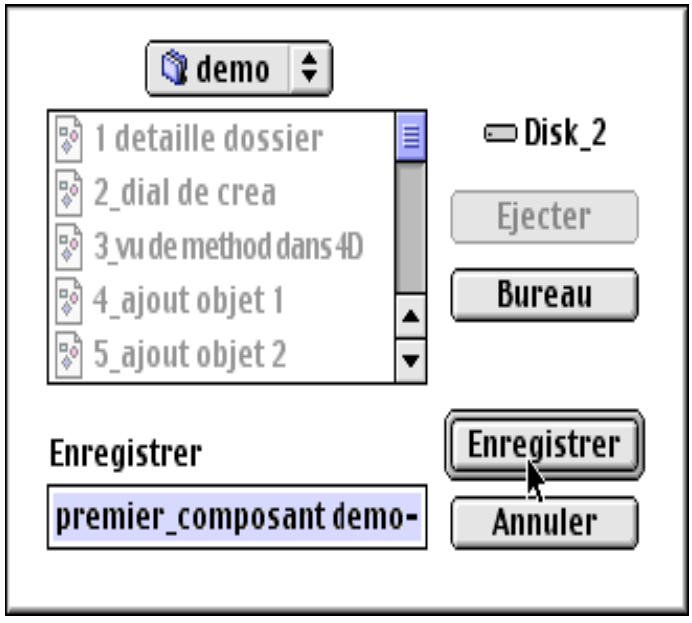


Comme vous pouvez le constater, la méthode " ma méthode " fait partie du composant. Ceci ne veut pas dire qu'elle ne fait plus partie de votre structure. Elle est simplement déplacée vers le composant qui fait partie intégrante de votre structure.

3) Génération

La génération proprement dite du composant, en cours de développement ne pose aucun problème. Dans ce cas simple l'étape de vérification du composant s'effectuera lors de son intégration dans la base de destination.

Voici ce que vous obtenez si vous demandez " générer un composant " :

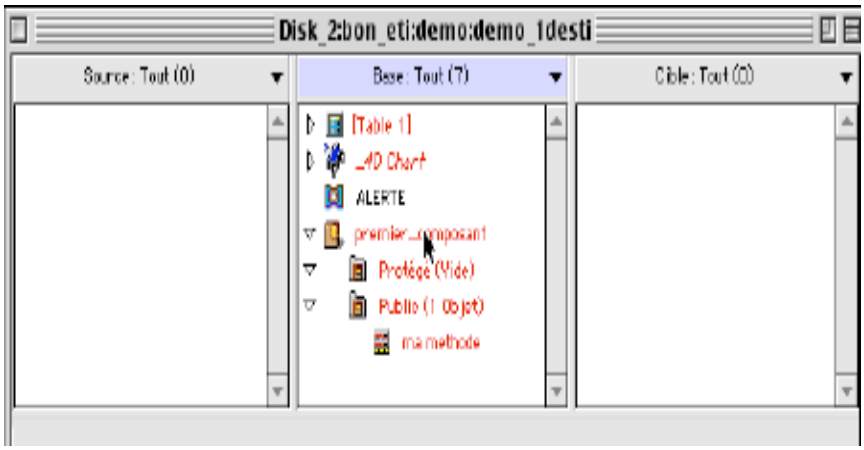


4) Intégration

Dans notre démonstration, notre composant est très simple et l'intégration ne va poser aucun problème. Nous détaillerons plus loin, tous les problèmes en rapport avec l'intégration de composants compliqués qui peuvent générer des erreurs.

Voici ce que vous obtenez si vous demandez " installer/mettre à jour" :

Vous pouvez voir sur l'image ci-dessous, le composant intégré avec ces deux dossier visibles.



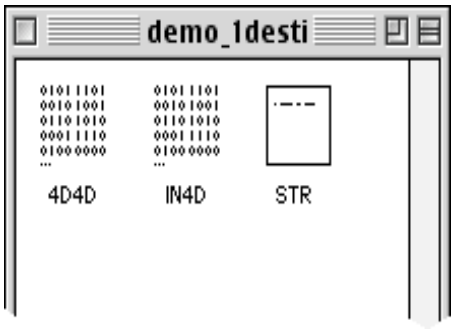
5) Suppression

Dans le cas présent, le composant peut être supprimé sans qu'aucun objet venant de celui-ci ne reste dans la structure où il a été installé.
Cependant il existe d'autres cas ou certains objets du composant restent à l'intérieur de la structure de destination. Ce point sera détaillé ultérieurement.

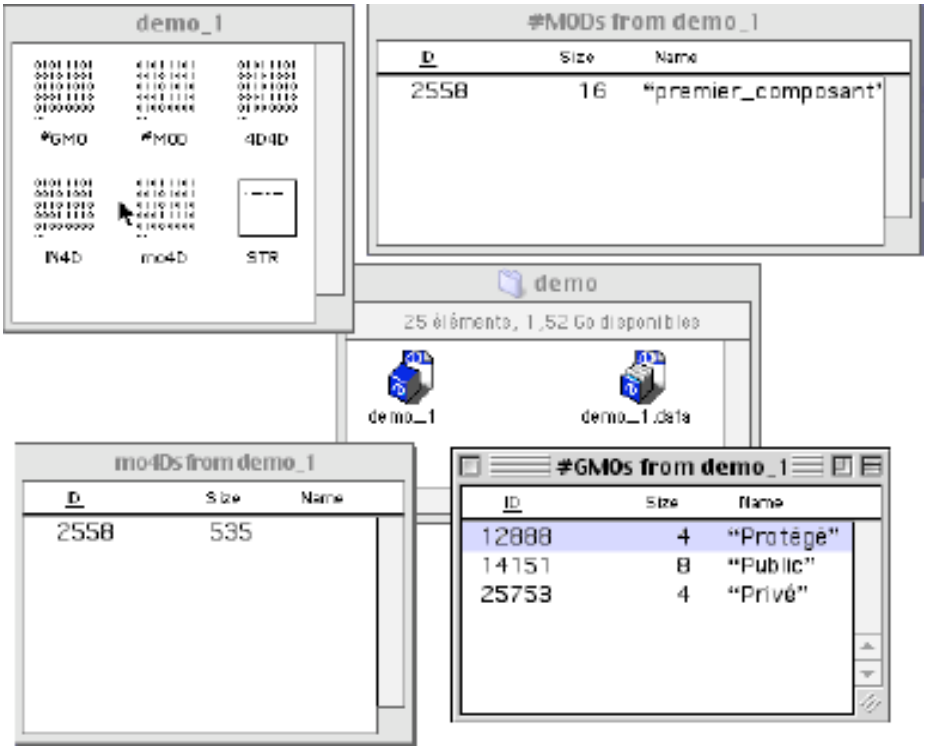
V - Ressources des composants

L'image ci-dessous représente les ressources d'une base qui ne contient pas de composants

(vous obtenez ces informations à l'aide d'un éditeur de ressources).



L 'image ci-dessous représente les ressources d'une base qui contient un composant.
(vous obtenez ces informations à l'aide d'un éditeur de ressources).



Vous pouvez constater qu'il y a une différence du nombre de ressources et qu'il est interdit de les modifier sous peine disfonctionnement du composant.

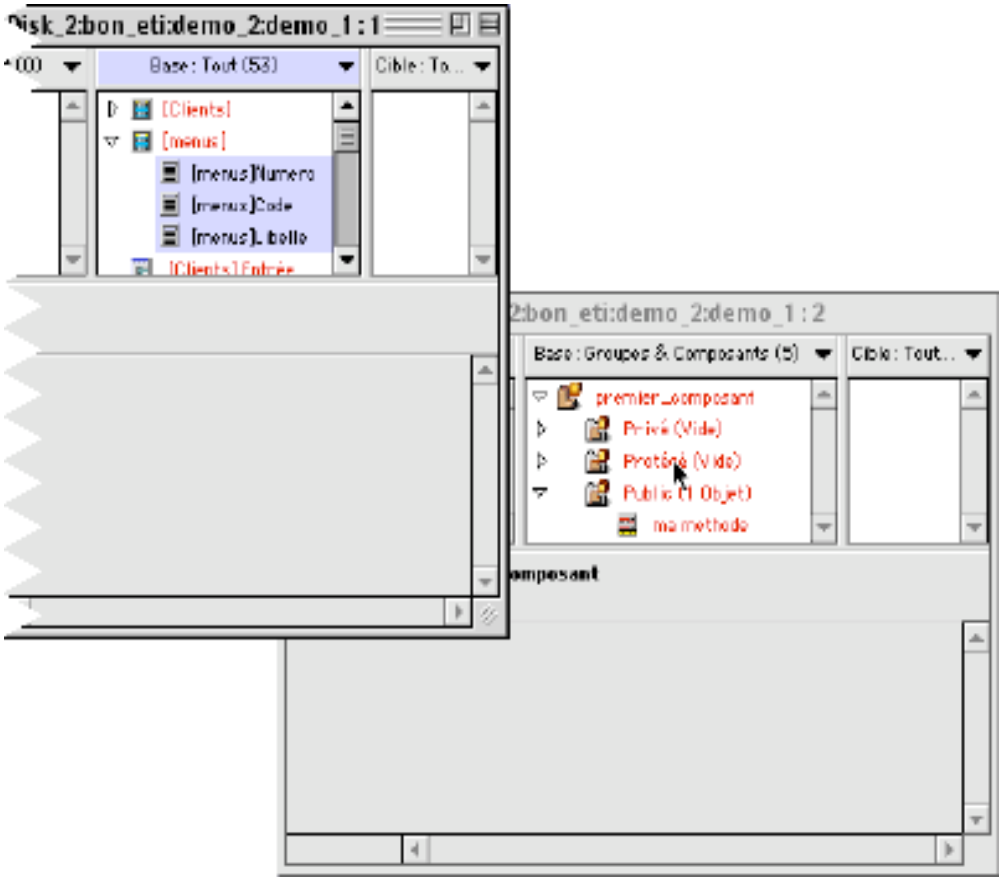
Note : la destruction des ressources de la structure n'implique pas la destruction des objets associés dans la structure. Il ne faut donc pas utiliser le principe de destruction des ressources avec un éditeur de ressources pour supprimer un composant.

VI - Association de tables dans un composant en cours de développement

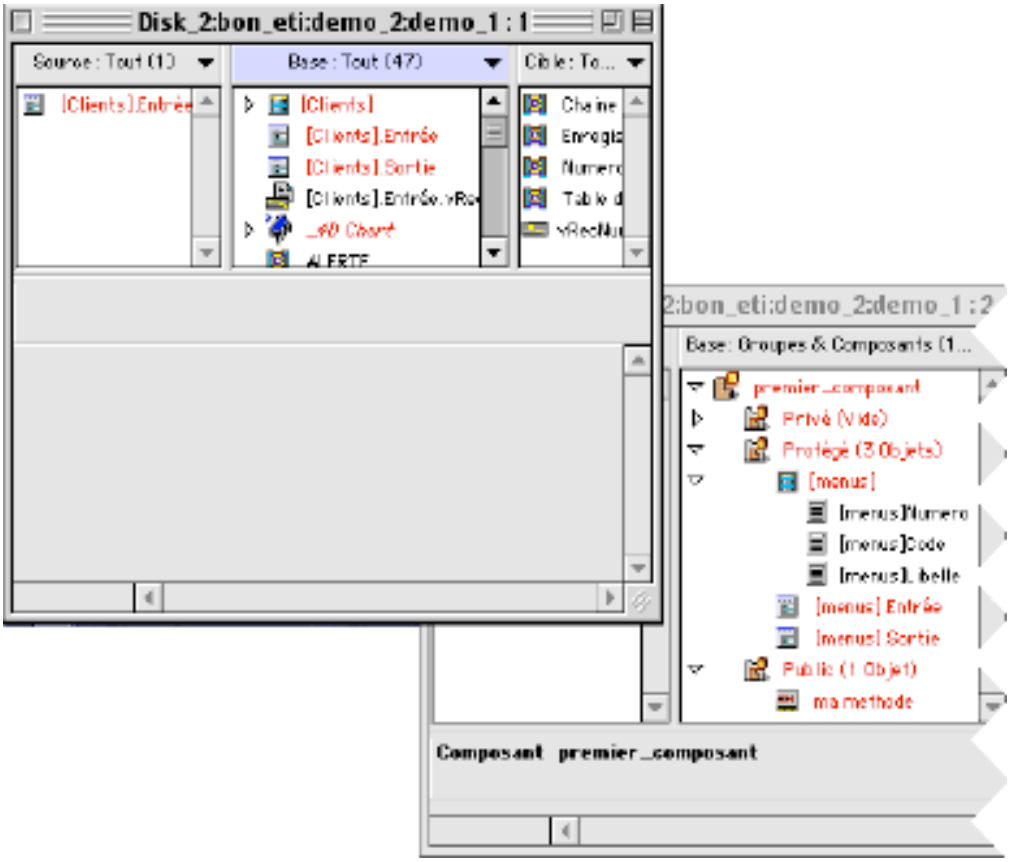
Association d'une table avec un formulaire qui utilise des champs de la table.

Il n'y a pas de difficultés pour mettre une table dans un composant, il faut simplement se souvenir qu'une table ne peut faire partie du dossier privé.

Ci-dessous la nouvelle table " menus " créée avec deux formulaires associés :



Ci-dessous la représentation du composant modifié qui contient la table "menus" et les deux formulaires. Tous ces objets se situent dans le dossier "protégé" (voir base démo_2 avec le composant version 2).



Attention aux tables : si par mégarde vous avez des formulaires ou des champs dans votre composant qui font référence à des tables, vous aurez un message de 4D Insider vous indiquant qu'il faut intégrer ces tables. Dans ce cas posez-vous la question : ai-je des tables à intégrer ?
Si vous n'intégrez pas ces tables, il vous sera impossible de générer le composant.

Lors de l'intégration du composant, si 4D Insider passe l'étape des conflits sur les objets (nous détaillerons plus loin cette étape), il n'y aura pas de message concernant l'intégration de la table.

Association d'une table destinée à recevoir des formulaires qui serviront pour l'interface du composant.

Il est possible de mettre des formulaires, dans un composant, qui seront utilisés dans la base de destination. Cependant ces formulaires ne doivent posséder que des variables.
Ces formulaires se trouvent, à l'origine, dans une table de votre base. Ils pourront être intégrés, via le composant, dans une autre table de la base de destination. L'association de ces deux tables s'effectue par l'intermédiaire d'un dialogue proposé par 4D INSIDER lors de l'intégration du composant. Une fois l'intégration effectuée, les formulaires font partie intégrante de la table de la base de destination, mais appartiennent toujours au composant.

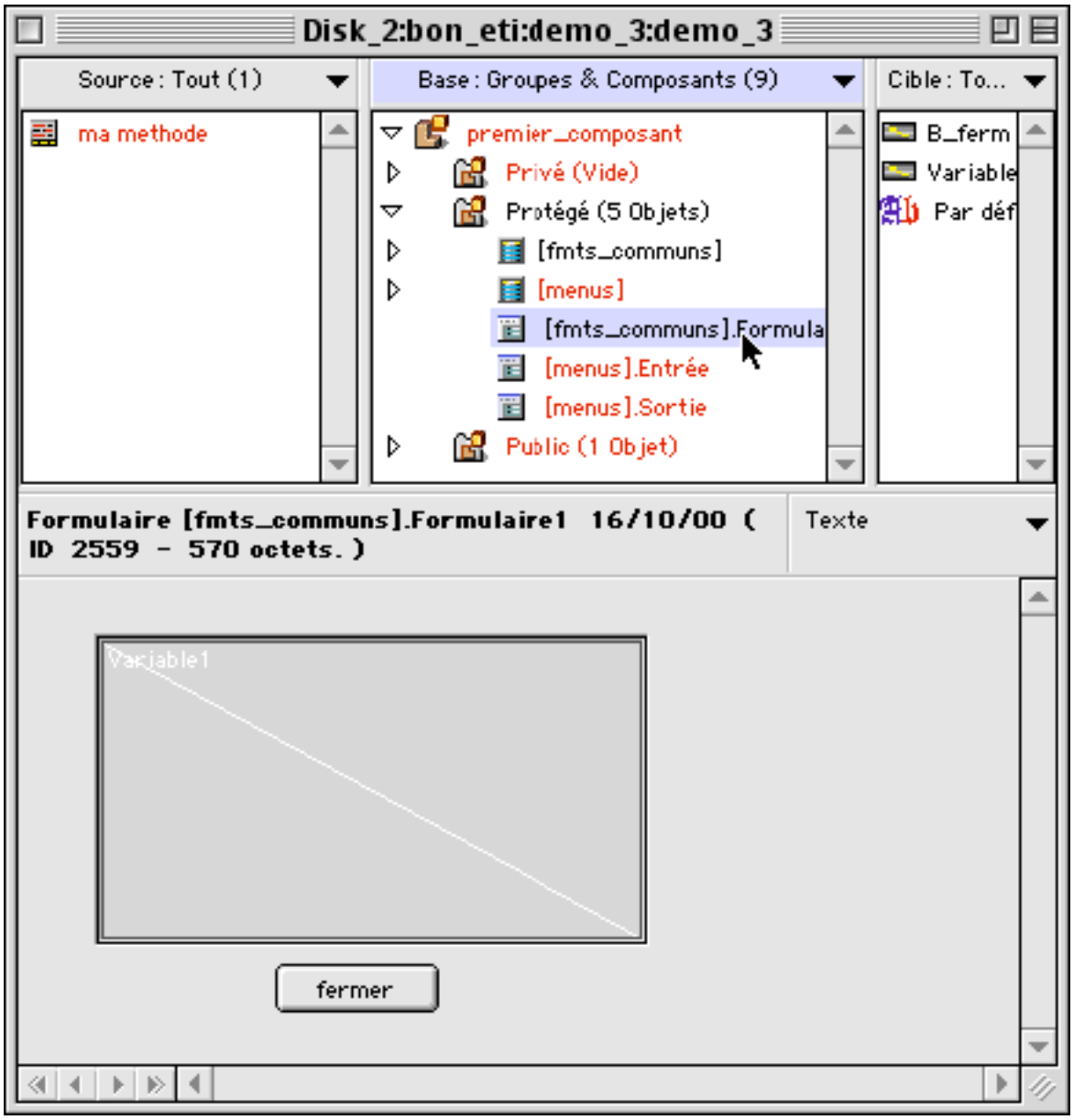
Nous allons démontrer ce principe ci-dessous (voir base démo_3 avec le composant version 3).

Pour illustrer cette démonstration nous allons transformer la méthode " ma méthode " en y incluant le code suivant :

Ma méthode.

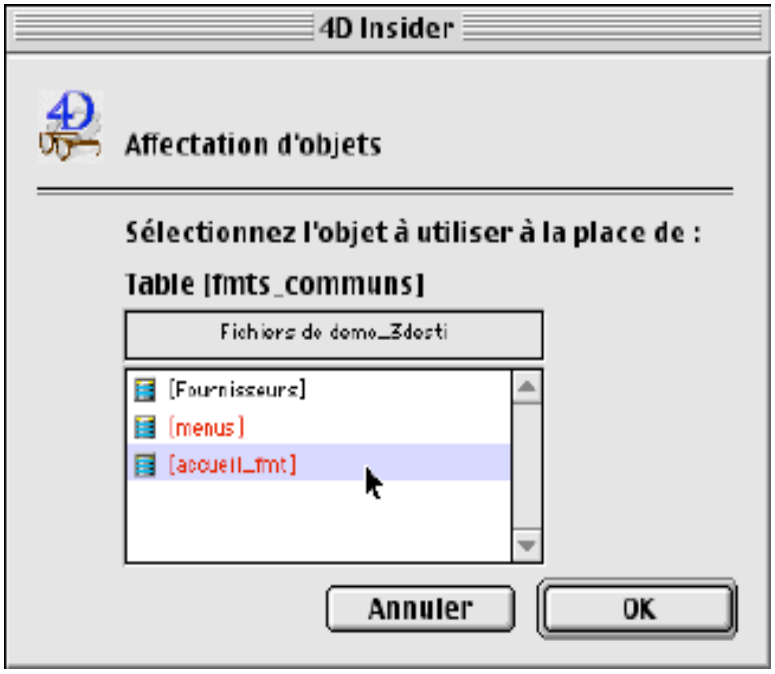
```
`ma première méthode classée dans le dossier public


`ALERTE("mon premier composant")

variable1:="mon premier composant"
Creer  fenetre(100;100;400;300;3)

DIALOGUE([fmts_communs];"Formulaire1")
```

L'image ci-dessous montre le formulaire intégré dans le composant :



Les deux copies d'écran suivantes représentent les phases d'intégration du formulaire lors de l'étape d'intégration du composant dans la base de destination. Ce formulaire sera associé a une table. Vous avez cependant la solution d'intégrer la table et le formulaire.

Le formulaire est associé à la table "accueil_fmt ".



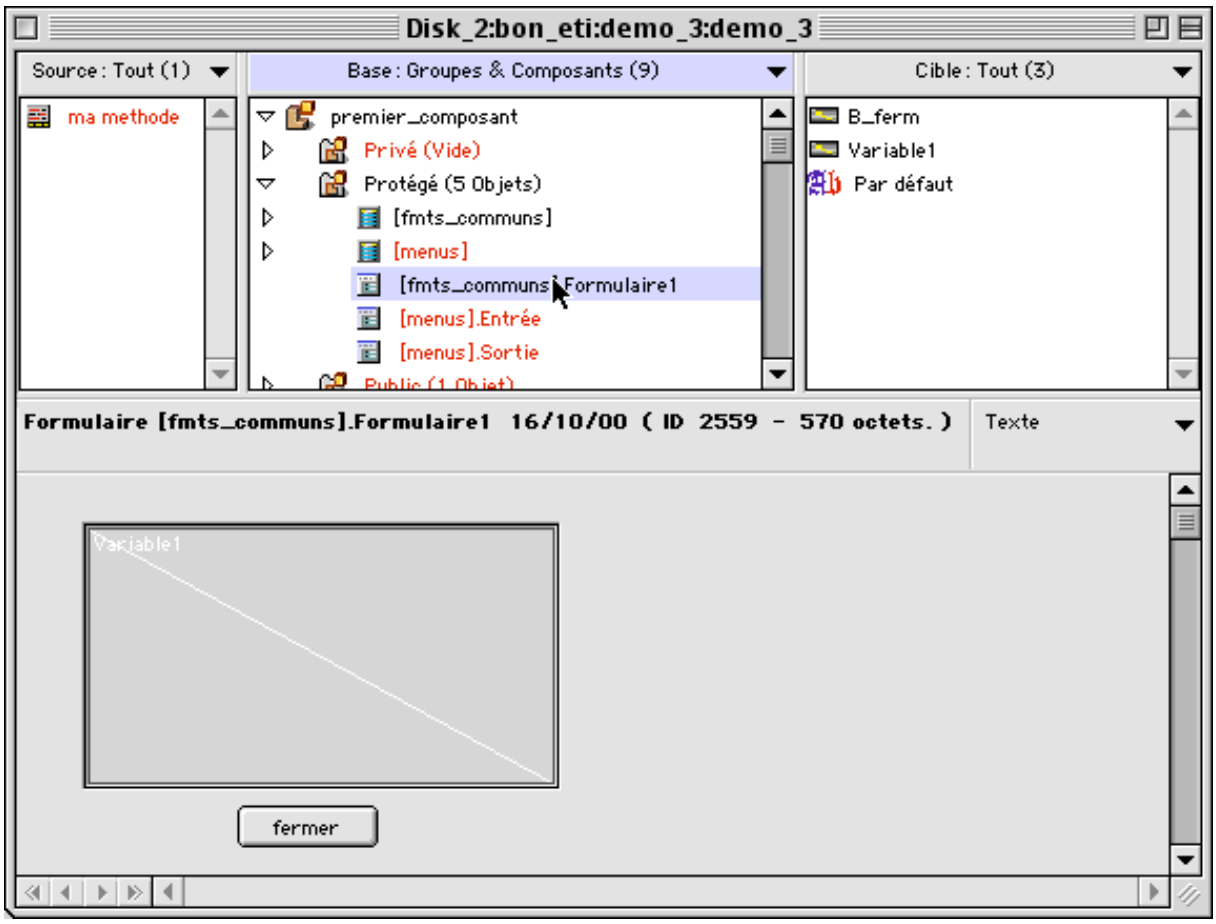
Cette étape n'est accessible qu'une fois que vous avez franchi l'étape des conflits que nous détaillerons ultérieurement.

VII - Quelques images représentant la lisibilité des objets d'un composant

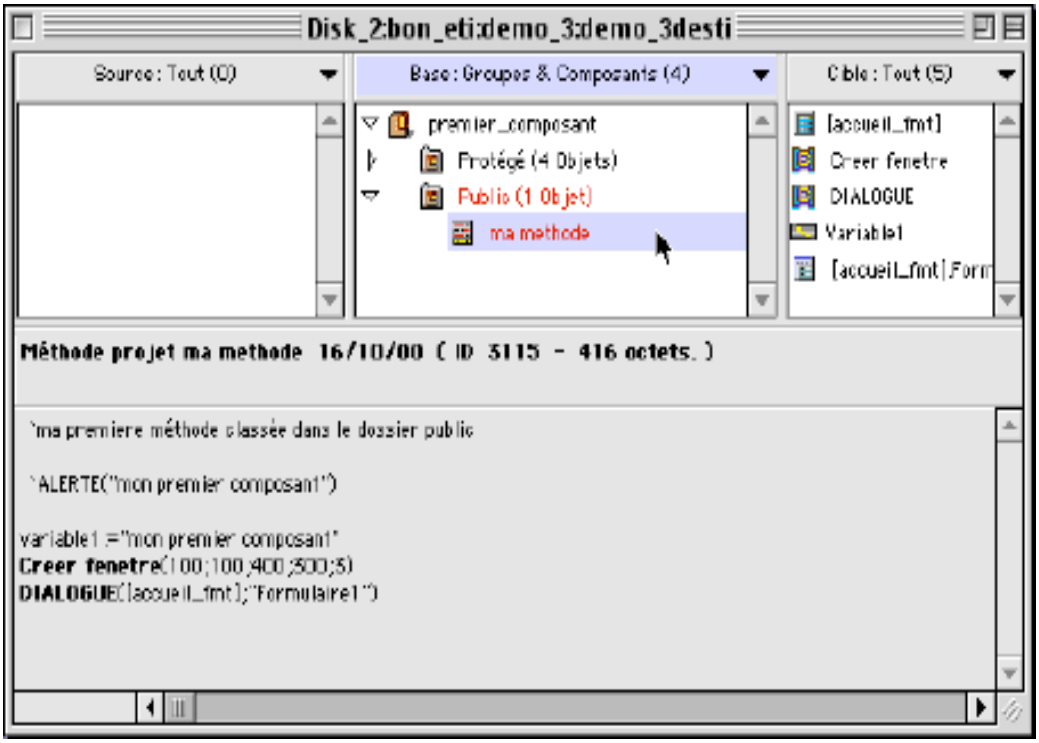
1) Au travers de 4D Insider.

Pour le créateur du composant :

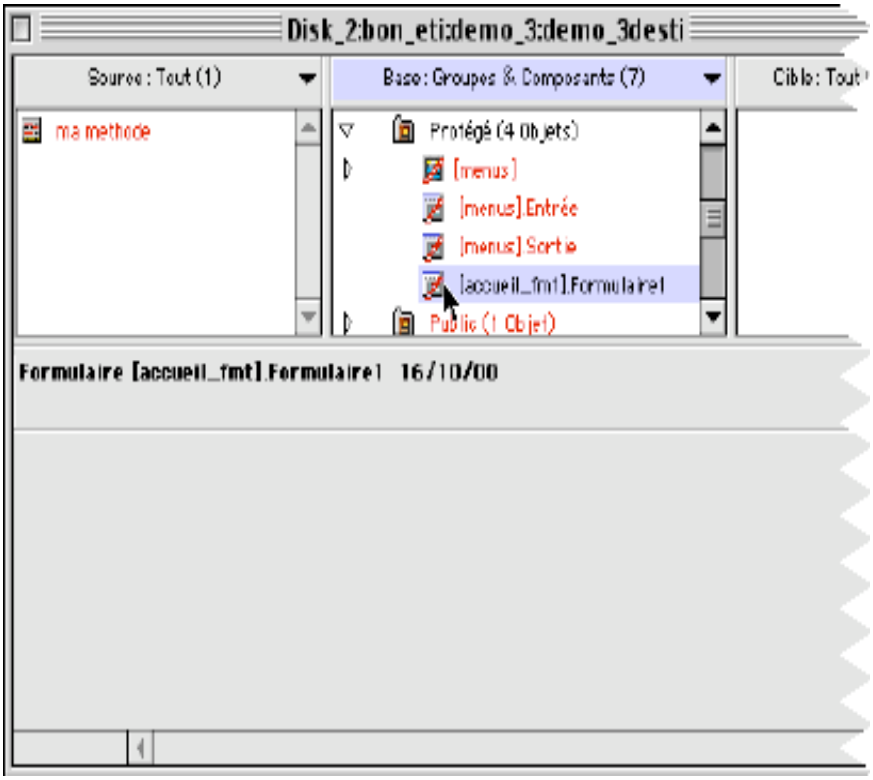
Quel que soit le dossier où l'objet est situé, ce dernier est visible. Comme illustré par les images ci-dessous :



Pour le client utilisateur du composant :



Dans la figure ci-dessus on voit que l'objet se trouve dans le dossier " public " nous avons donc accès à son contenu.



On peut remarquer dans l'illustration ci-dessus, que l'objet n'est visible que par son nom car il fait partie du dossier "protégé".

Remarque : nous pouvons constater que le dossier " privé " n'est pas présent car nous sommes dans la base qui accueille le composant. Cette constatation est une indication très précieuse pour une personne qui ne maîtrise pas particulièrement les composants car c'est un moyen efficace pour déterminer si le composant est un composant en cours de développement ou non (composant installé).

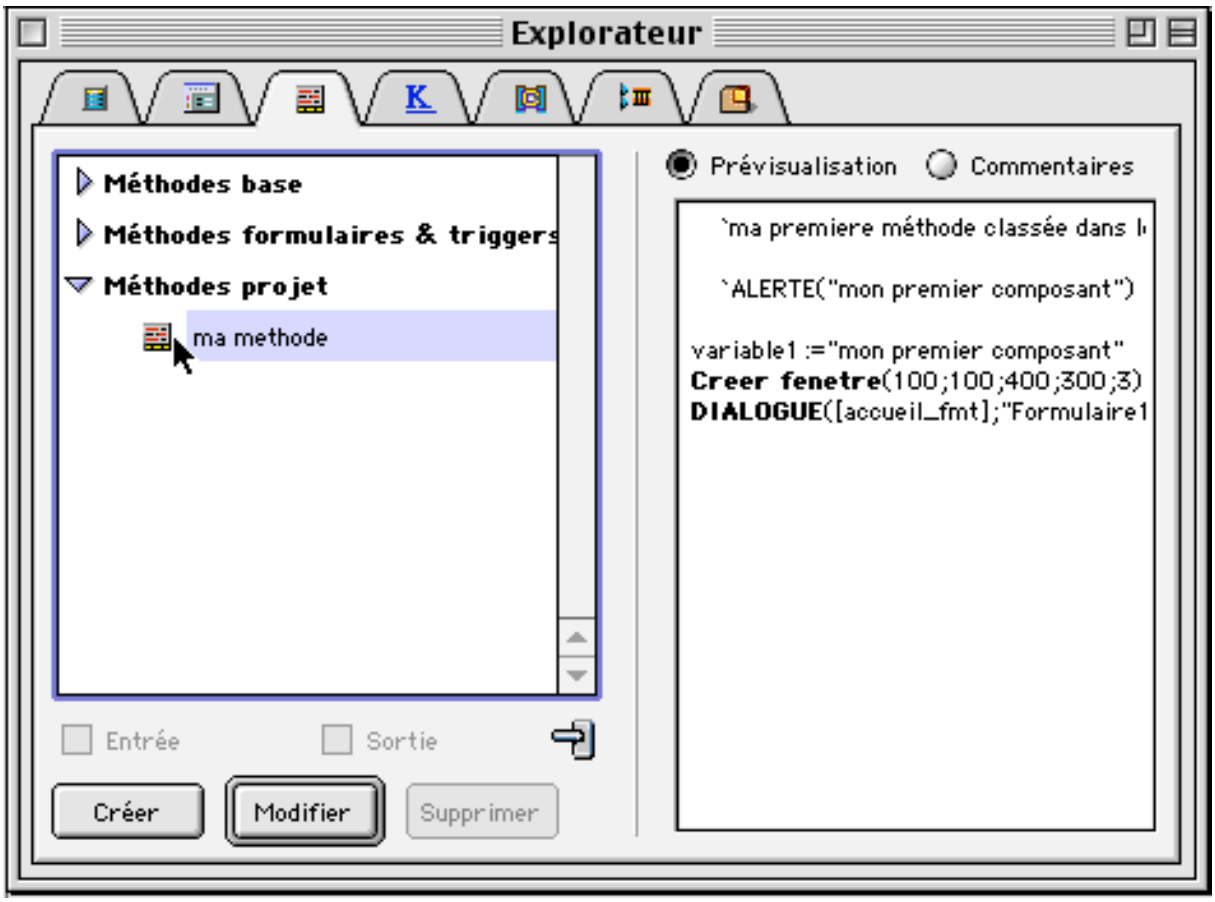
2) Au travers de 4D

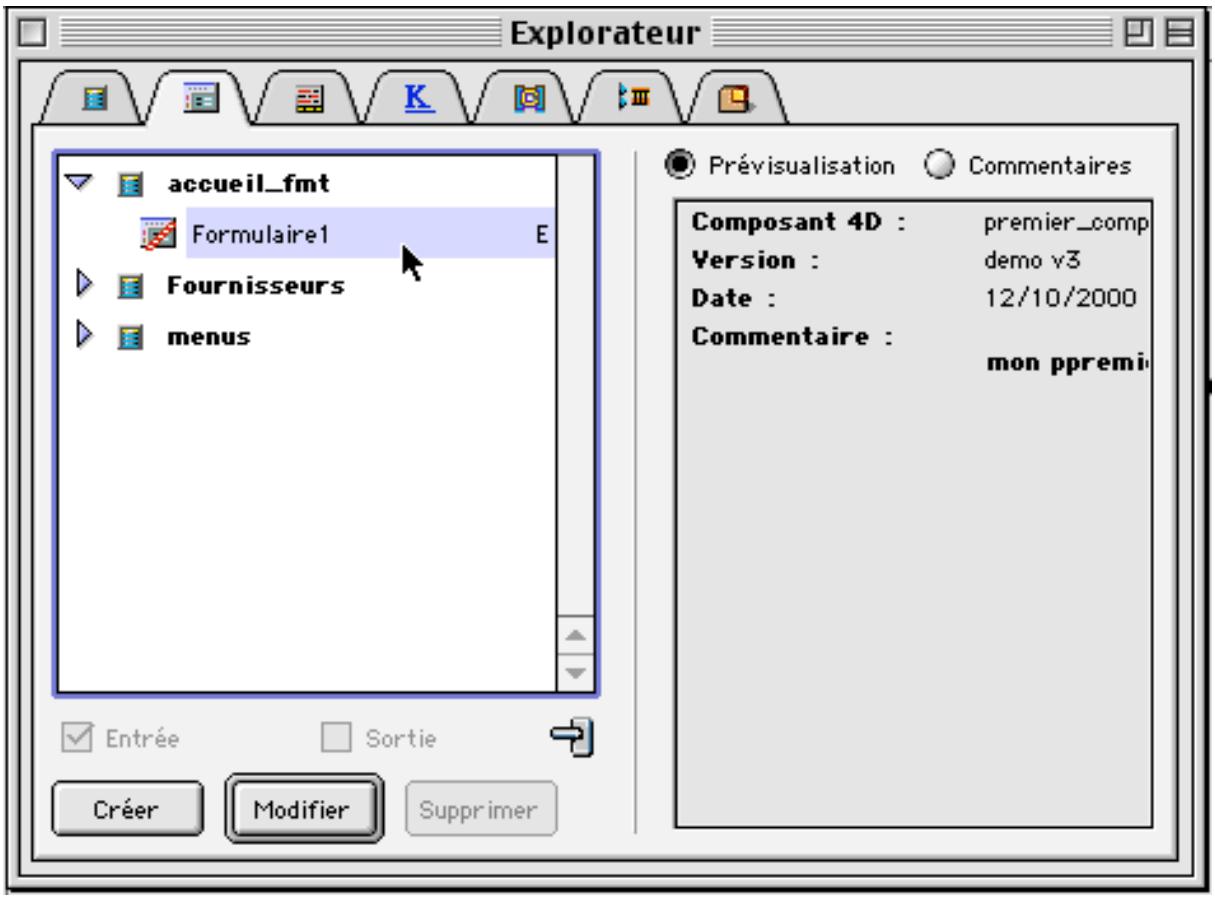
Pour le créateur du composant :

Tout est visible, il faut simplement souligner qu'il n'y a pas de différence de représentation des objets qui appartiennent à un composant.

Pour le client utilisateur du composant :

Le principe de visibilité des composants est identique dans la structure de destination comme nous le montre ces quelques images.


Explorateur
Méthodes base
Méthodes formulaires & triggers
Méthodes projet
ma methode
Prévisualisation
Commentaires
`ma premiere méthode classée dans l
`ALERTE("mon premier composant")
variable1 :="mon premier composant"
Creer fenetre(100;100;400;300;3)
DIALOGUE([accueil_fmt];"Formulaire1
Entrée
Sortie
Créer
Modifier
Supprimer

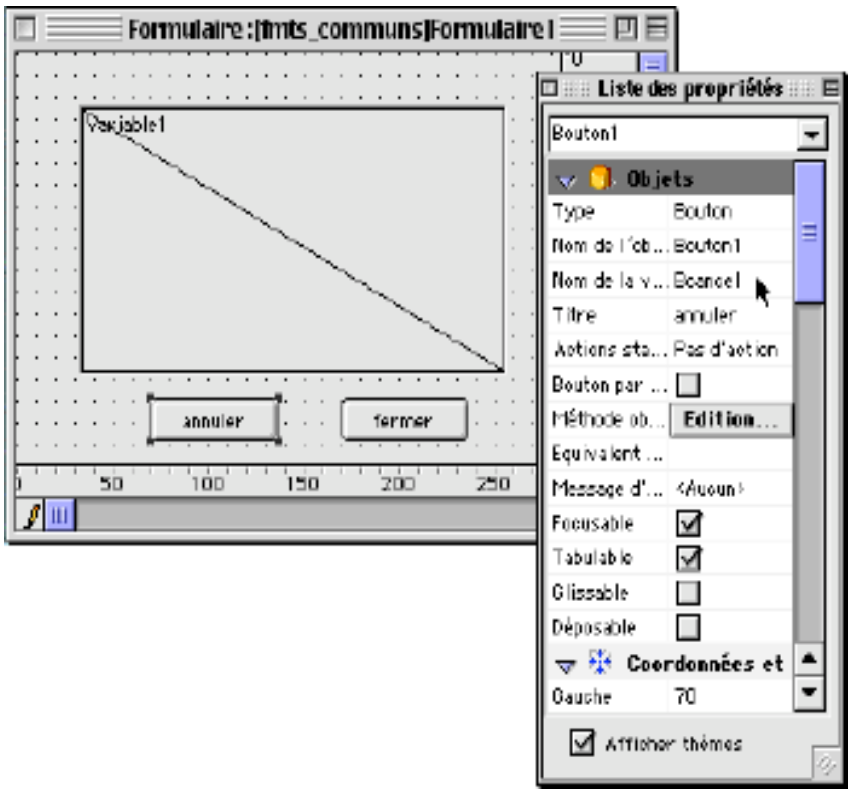

Explorateur
accueil_fmt
Formulaire1
E
Fournisseurs
menus
Prévisualisation
Commentaires
Composant 4D :
premier_comp
Version :
demo v3
Date :
12/10/2000
Commentaire :
mon ppremi
Entrée
Sortie
Créer
Modifier
Supprimer

VIII - Les problèmes rencontrés lors de l'intégration d'un composant

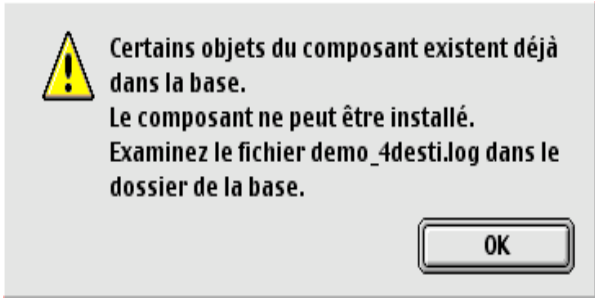
1) Intégration et mise à jour d'un composant : précautions

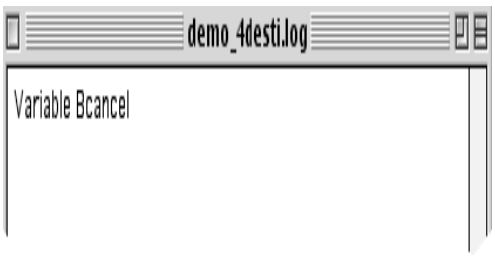
Si vous n'avez pas respecté toutes les règles concernant la création d'un composant, 4D Insider, lors de l'intégration va générer un fichier de log qui vous mentionnera les problèmes.

Dans l'exemple de la démo version 4 nous avons créé un bouton nommé " Bcancel " dont les propriétés sont illustrées ci-dessous. Celui-ci existe aussi dans un formulaire de la base de destination.



Le fait que 4D Insider génère un fichier de log implique l'arrêt immédiat de l'installation.
Les deux figures ci-dessous montrent le message généré par 4D Insider et le fichier log qui peut être ouvert par un traitement de texte.





2) Règles pour éviter la génération d'un fichier d'erreurs

Tous d'abord 4D Insider vérifie l'unicité du nom de tous les objets (nom du composant compris). On entend par objet "les objets Insider". Il ne faut pas confondre avec le nom de l'objet que vous voyez dans la liste des propriétés de l'objet dans votre base.

Cela veut dire qu'il est strictement interdit d'avoir un même nom d'objet présent dans le composant et dans la base qui va recevoir le composant. On entend par objet, tout objet 4D. Par exemple, si dans la base de destination, une méthode a pour nom " mon_objet " et que dans le composant une variable porte aussi ce nom, 4D Insider générera une erreur.

Cependant les objets internes à 4D ne sont pas concernés. Exemple le "user set".

Il existe une exception à cette règle. Il s'agit des objets prédéfinis par 4D comme, par exemple, les boutons qui sont créés automatiquement dans un formulaire. Un autre objet est concerné, il fait partie des formulaires " ENTREE " et indique le nombre d'enregistrements, il s'agit de la variable "Vrecnum".

Par contre 4D et 4D Insider gèrent tous les objets systèmes comme les variables " OK " ou " DOCUMENT ".

Il faut aussi souligner qu'il y a un certain nombre de noms d 'objets qui ne sont pas vérifiés par 4D Insider. En effet, pour des raisons historiques, 4D Insider ne tient pas compte des objets qui sont placés entre guillemets. Par exemple : "mon ensemble".

X - La mise à jour d'un composant

Le principe est le même que lors d'une première intégration à la différence du dialogue ci-dessous.

Il faut souligner cependant que vous n'êtes pas obligé d'intégrer tous les objets dans la nouvelle version du composant. Les objets qui ne sont pas présents dans la nouvelle version restent en l'état. Les objets qui ont fait l'objet d'une modification sont remplacés.

Attention aux objets publics modifiés par l'utilisateur dans la base de destination.

X - Lecture du contenu d'une table n'appartenent pas au composant

Principe

Il est possible de lire une table dans la base de destination qui n'appartient pas au composant grâce à l'utilisation des fonctions "table" et "nom de table" dans une méthode.

Le principe est de créer des tableaux avec le nom des tables et des champs. Puis de les utiliser avec les fonctions vues ci-dessus pour permettre de créer des pointeurs sur ces tables et champs (associés à ces tables).

Pour l'échange de données, nous avons vu, paragraphe VIII-2, qu'il est possible d'avoir des noms d'ensembles identiques entre la base de destination et le composant à intégrer donc la communication par ensemble est possible.

Exemple

Exemple de code qui permet d'accéder à une table qui ne se trouve pas à l'intérieur d'un composant.

```
TABLEAU  ALPHA (45 ; Tmonma;0)


C_ENTIER($nbt;$i;$ind1;$ind2)
C_POINTEUR($ptr1;$ptr2)

$nbt:=Nombre de tables
TABLEAU  ALPHA(15;$tnom;$nbt)

Boucle ($i;1;$nbt)
  $Tnom{$i}:=Nom de la table($i)
Fin de boucle

$ind1:=Chercher dans tableau($tnom;"liste_mnu")
S i ($ind1>0)
  $ptr1:=Table($ind1)
Sinon
 ALERTE("erreur sur le nom d'une Table")
Fin de si
```

```
$ind2:=Chercher dans tableau($tnom;"lgn_mnu")
S i ($ind2>0)
  $ptr2:=Table($ind2)
Sinon
 ALERTE("erreur sur le nom d'une Table")
Fin de si

CHERCHER($ptr1->;Champ($ind1;1)->="denomina")
LIEN RETOUR($ptr1->)
SELECTION VERS TABLEAU(Champ($ind2;3)->;Tmonma)
REDUIRE SELECTION($ptr2->;0)

REDUIRE SELECTION($ptr1->;0)
```

Vous pouvez bien entendu optimiser ce code.

XI - Conseils sur la préparation d'un composant

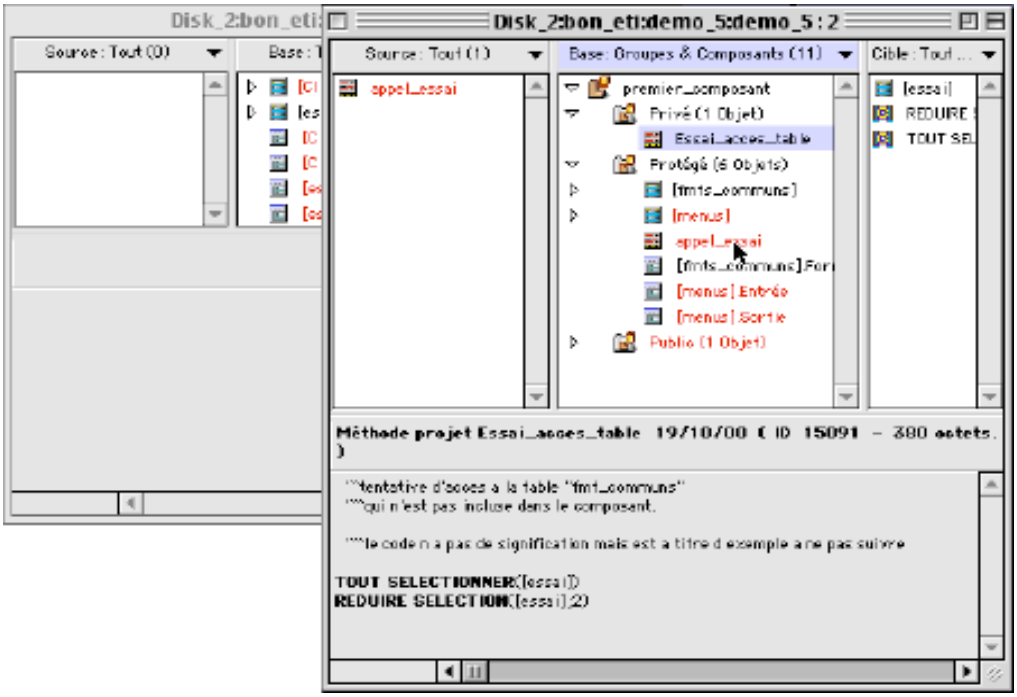
1) Précautions élémentaires

Donner des noms de variables avec un préfixe de composant. Cela est inutile pour les variables locales car leurs portés est la méthode.

Note : une base de données regroupant les préfixes déjà utilisés sera probablement proposée sur le web

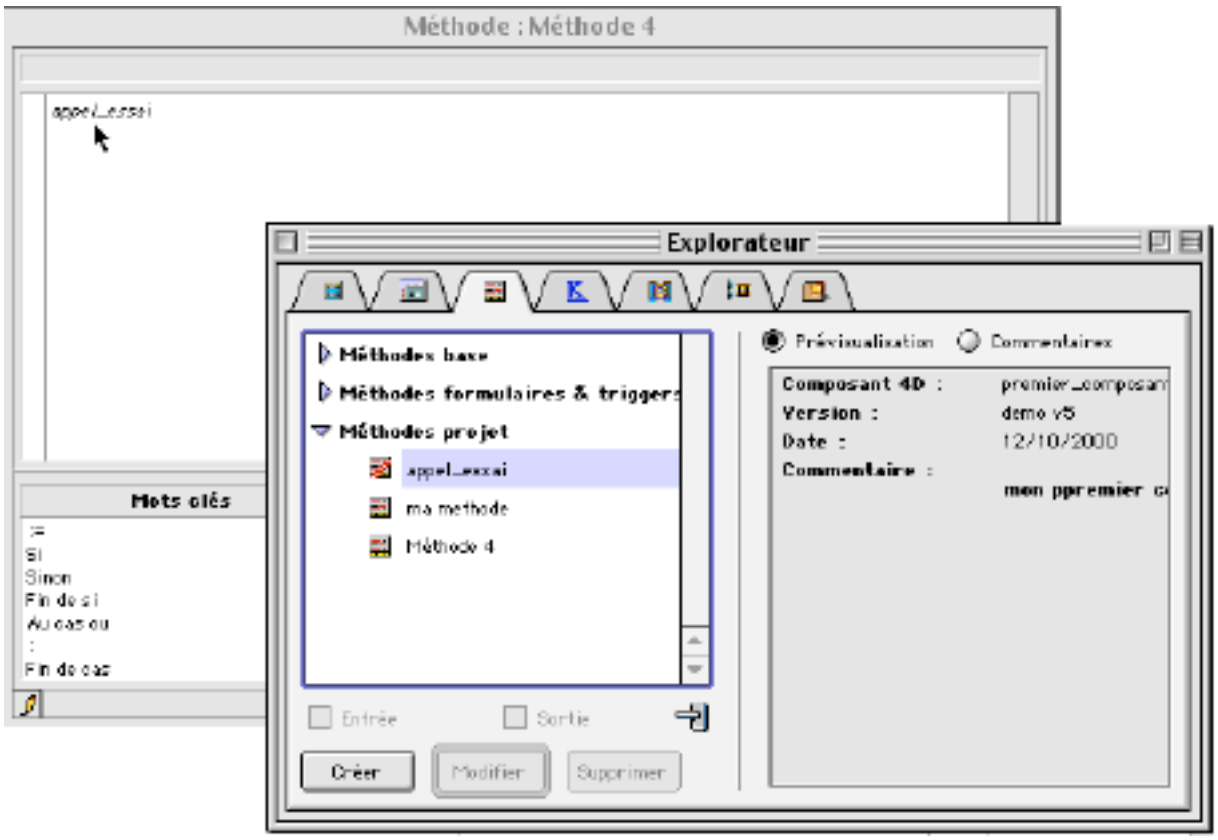
Attention : si vous avez des références de tables et que vous n'avez pas intégré ces tables, vous obtiendrez des erreurs à l'utilisation du composant. Plus grave encore, les commandes incriminées peuvent, dans leur description syntaxique, avoir des paramètres optionnels faisant références aux tables. Dans ce cas 4D peut utiliser le nom de la table par défaut dans ces commandes. Ceux-ci est rendu possible car 4D Insider a intégré vos commandes sans les noms de tables. Ce n'est peut-être pas la table que vous voulez utiliser.

Ci-dessous quelques images qui illustrent le paragraphe précédent (voir base démo 5)



Ci-dessus la méthode "essai_acces_table" fait une tentative d'accès à la table "[essai]" qui n'appartient pas au composant. Vous pourrez constater, lors de la génération du composant, que 4D Insider ne génère aucune erreur. De même, il n'y aura pas d'erreur à l'intégration du composant.

Les problèmes vont apparaître lors de l'utilisation de la méthode "essai_acces_table" par l'intermédiaire de la méthode "appel_essai" que nous avons intégré dans la base de destination comme le montre la figure ci-dessous :



Ci-dessous l'erreur générée

Comment vous pouvez le constater, lors de la génération du composant, 4D Insider n'a pas intégré le nom de la table.

Ceci est un comportement normal de 4D Insider. Car la vérification des objets se fait sur l'existence de l'objet et non sur le contenu de l'objet (donc du code) qui ferait appel a des objets.

2) Composant et compilation

- Un composant seul ne peut pas être compilé. C'est la base d'accueil qui doit l'être.
- NE JAMAIS livrer un composant sans avoir fait l'essai d'une compilation.
- Utiliser les méthodes " compiler_xx " pour typer vos variables.
- Typez toutes vos variables.

Le compilateur analyse, en premier, les composants.

Attention : le compilateur ne vérifie pas l'appartenance aux dossiers "privé", "protégé", "public" des objets de type variable à l'intérieur du composant. Cela veut dire qu'il est tout a fait possible de modifier le contenu d'un objet, par exemple privé, une fois que le composant a été intégré. Cette modification pourrait entraîner un mauvais fonctionnement de votre composant.

Note : il est impératif que vos objets soient préfixés et qu'ils ne contiennent pas de noms usuels afin d'éviter que l'utilisateur du composant puisse découvrir par hasard des noms d'objets et les utiliser.

Attention : si le composant engendre des erreurs à la compilation, si celles-ci se trouvent dans les dossiers "protégé" ou "privé", le client qui veut intégrer le composant ne peut corriger ces erreurs.

3) Qu'est ce qu'un point d'entrée ?

C'est généralement une méthode projet qui est plutôt dans le dossier dit "protégé. Cette méthode permet

d'utiliser le composant à travers elle. Généralement le développeur du composant vous indiquera dans quel cadre il faudra faire appel à cette méthode et vous précisera quels sont les paramètres à passer le cas échéant.

Cependant un composant peut avoir plusieurs points d'entrée.

4) Suppression d'un composant

Il faut savoir que si le composant possède des tables, les tables ne sont pas supprimées.
Attention tous les objets publics sont supprimés. Cela peut être gênant si l'utilisateur du composant utilise le code dans sa base pour d'autres évènements qui ne sont pas en rapport avec le composant.

Attention à la désinstallation de composant intermédiaire. Il faut comprendre par cela qui si vous avez installé plusieurs versions de votre composant, c'est tout le composant qui sera désinstallé et non la dernière version. IL EST DONC IMPERATIF DE FAIRE UNE SAUVEGARDE AVANT L'INTEGRATION D'UN COMPOSANT.

XII - Composants contenant des ressources ou utilisant la bibliothèque d'images

1) Les ressources

Les ressources PICT ne sont pas dans un formulaire :
4D Insider ne gère pas les ressources PICT qui pourraient être utilisées dans une base. Il ne les copiera donc pas. C'est aux développeurs de faire le travail nécessaire pour qu'elles soient présentes dans la base de destination.

Les ressources PICT sont dans un formulaire :
4D Insider copie la ressource PICT dans la base de destination. Quel que soit le cas, 4D Insider ne touche pas aux PICT de la base de destination.

Cas d'un formulaire :

Si les numéros d'ID des PICT ne sont pas en conflit avec ceux de la base de destination, 4D Insider copie les PICT du composant sans toucher à leurs numéros d'ID.

Si les numéros d'ID des PICT sont en conflit avec ceux de la base de destination, 4D Insider re-numérote l'ID des PICT du composant installé.

Si on a besoin d'utiliser la commande "LIRE RESSOURCE IMAGE", on peut passer comme paramètre la nouvelle fonction "Lire ID ressource composant" dans laquelle on passera le numéro d'origine de l'ID (et non celui affecté par 4D Insider lors de l'installation du composant).

Ainsi, on n'a pas besoin de vérifier s'il existe des conflits d'ID et quel est le numéro affecté par 4D Insider.

2) 4D Insider et les ressources PICT

Dans le cas d'un formulaire :

Si les numéros d'ID des PICT ne sont pas en conflit avec ceux de la base de destination, 4D Insider copie les PICT sans toucher leurs numéros d'ID.
Si les numéros d'ID des PICT sont en conflit avec ceux de la base de destination, 4D Insider re- numérote les ID des PICT de la base source. Si on a besoin d'utiliser la commande "LIRE RESSOURCE IMAGE", on doit d'abord aller vérifier quels sont les nouveaux numéros affectés aux PICT copiées dans la base de destination.

3) La bibliothèque d'images

Comportement de 4D Insider vis-à-vis de la bibliothèque d'images

Les images sont toujours copiées dans la base de destination qu'elles soient utilisées ou non dans un composant.
4D Insider ne touche pas aux ID des images de la base de destination.

Si on fait appel aux images de la bibliothèque dans un composant

Si les numéros des images ne sont pas en conflit avec ceux de la base de destination, 4D Insider copie les images sans modifier les numéros.

Si les numéros des images sont en conflit avec ceux de la base de destination, 4D Insider re-numérote l'ID des images du composant. Si, par exemple, on utilise la commande "LIRE IMAGE DANS BIBLIOTHEQUE" dans une méthode appartenant au composant et que celle-ci est placée dans le dossier "Protégé" ou "Privé", il faut obligatoirement avoir passé le nom de l'image (et non son numéro d'ID) comme paramètre à la commande.

Certes, il est possible de connaître le nouveau numéro que 4D Insider a attribué aux images (qu'elles soient protégées ou privées), par contre, il n'est plus possible de modifier le contenu des méthodes protégées et privées qui font appel à ces images.

Note : Les images protégées et privées sont affichées en rouge dans la bibliothèque d'images. Leur nom et leur numéro d'ID sont visibles mais pas leur contenu.

Si on fait appel aux images de la bibliothèque en dehors d'un composant

Si les numéros des images n'entrent pas en conflit avec ceux de la base de destination, 4D Insider copie les images sans modifier les numéros.

Si les numéros des images entrent en conflit avec ceux de la base de destination, 4D Insider re- numérote l'ID des images du composant.