Glisser Déposer

Par
Thomas D'Urso, Technicien Support Clients 4DUS
Note technique 4D-200008-23-FR
Version 1

Date 1 Août 2000

Résumé

Cette Note Technique fournit une vue d'ensemble d'implémentation de " Glisser Déposer ".
Même si ce mécanisme fait appel à des pointeurs, il reste très facile à implémenter. La base de données livrée avec cette note technique contient tous les exemples décrits (ils peuvent êtres copiés et facilement intégrés dans votre base de données).

4D Notes techniques

Introduction

Cette note technique reprend les principes fondamentaux de l'intégration du " Glisser Déposer " dans une base de données 4D, et introduit les différents concepts liés à son implémentation comme les évènements formulaires, les pointeurs et la programmation générique.

Le " Glisser Déposer" permet :

- De faciliter l'insertion et le déplacement des données pour l'utilisateur.

- De fournir une interface intuitive facilement compréhensible.

- Une interactivité entre l'utilisateur et l'application, sans l'utilisation du clavier.

Organisation de la note technique :

- Le " Glisser Déposer " entre deux variables.

- Le " Glisser Déposer " entre deux tableaux.

- Le " Glisser " à partir d'une liste hiérarchique.

- Le " Glisser Déposer " entre deux process.

Des exemples de codes génériques sont aussi fournis pour faciliter l'implémentation et l'entretien du " Glisser Déposer " dans votre base 4D.

Implémentation du " Glisser Déposer " entre deux variables

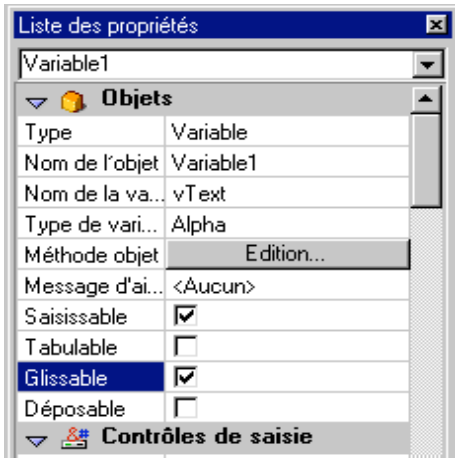
Le déplacement de données entre deux variables, constitue l'exemple le plus simple d'implémentation du " Glisser Déposer ". Dans l'exemple 1, ci-dessous, vText est la variable source qui contient le texte à glisser, et vText2 est la variable destination.

Exemple 1 de la base de données comme il se présente en mode structure

L'implémentation du " Glisser Déposer " présente trois étapes :

1/ Propriété Glissable :

Activer les propriétés glissables et déposables dans la liste ou la palette des propriétés de l'objet.



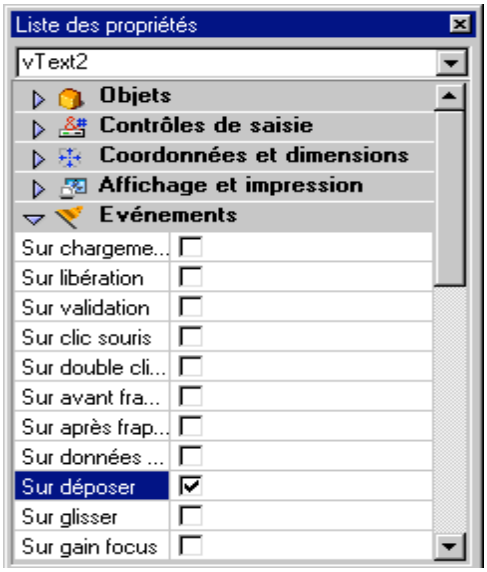
Les propriétés " Glisser Déposer " de la variable source vText

La propriété glissable doit être sélectionnée pour un objet pouvant être déplacé. Pour un objet, pouvant recevoir un autre objet glissable, la propriété déposable doit être sélectionnée. Un objet peut avoir les deux propriétés à la fois, ou l'une des deux, ou encore aucune.

Dans notre exemple, vText (la variable source) est glissable et vText2 (la variable destination) est déposable. vText2 ne peut que recevoir des objets déplaçables. En revanche, nous ne pouvons glisser que vText.

2/ Propriété Déposable :

Pour tout objet déposable, l'événement formulaire sur " déposer " doit être coché, soit sur la liste des propriétés, soit dans la palette des propriétés de l'objet. Si cet événement est coché, la méthode " objet ", de l'objet déposable, s'exécute quand ce dernier reçoit l'objet glissable. Lorsque l'utilisateur dépose le texte au-dessus de vText2, le code de la méthode associée à vText2 s'exécute.



La liste des propriétés de la variable destination vText2

3/L'action :

Lorsque l'utilisateur effectue le " Glisser Déposer ", le contenu de vText est copié par programmation dans vText2.

Le code minimal pour implémenter ceci :

```
PROPRIETES  GLISSER  DEPOSER ($source_objet;$source_element;$source_process)

VText2:=$source_objet->
```

La commande " **PROPRIETES GLISSER DEPOSER** " permet de récupérer des informations sur l'objet source lorsque l'événement " Sur glisser " ou " Sur déposer " est déclenché pour un objet.
Le premier paramètre est un pointeur vers l'objet source, c'est-à-dire l'objet qui a été glissé et déposé (vText).

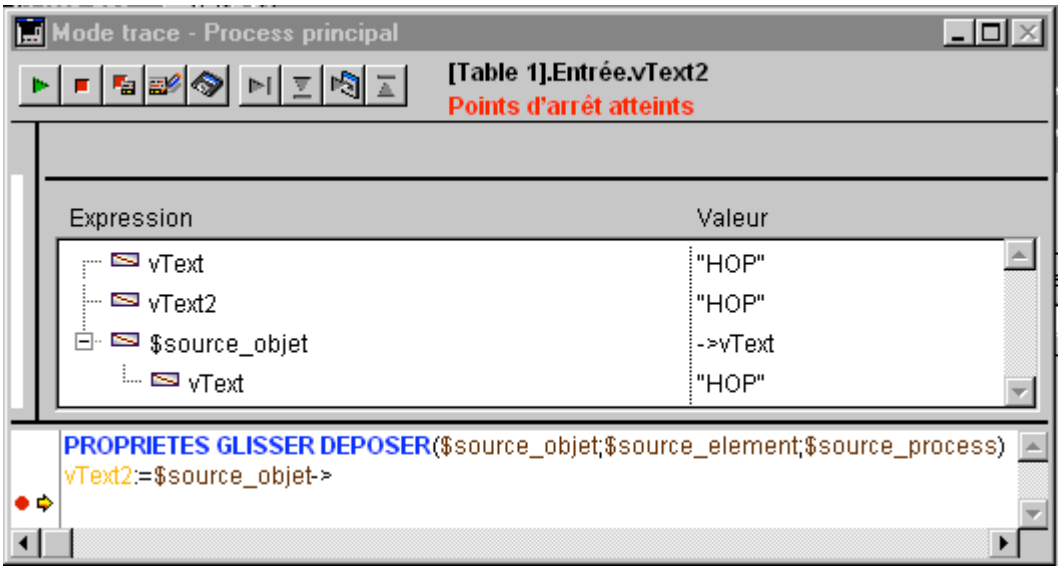
Un pointeur est une variable qui " pointe " vers un autre objet. Ceci est très pratique pour plusieurs raisons. En effet, il permet de dire que l'objet, pointé par le premier paramètre " $source_object", est l'objet que l'utilisateur a glissé.

Quand " **PROPRIETES GLISSER DEPOSER** " est exécutée et que " $source_object " contient un pointeur vers l'objet source, on peut récupérer la valeur de l'objet source en dépointant " $source_object ".

D'où la ligne de code suivante :

```
VText2:=$source_objet->
```

qui peut être lue de la façon suivante : affecter à VText2 la valeur de vText



Visualisation dans le Débogueur de l'affectation de la valeur de la variable source vText à la variable destination vText2.

Implémentation du " Glisser Déposer " entre deux tableaux

Un tableau est différent d'une variable dans le sens où il contient plusieurs valeurs au lieu d'une seule. Chaque valeur du tableau est appelée " élément ". Le " Glisser Déposer " entre deux tableaux est implémenté avec les mêmes principes que précédemment, mais dans ce cas, on aura besoin de déterminer quel élément est glissé, et, où l'utilisateur le dépose dans le tableau de destination.

Le code est le suivant :

```
PROPRIETES GLISSER DEPOSER($source_objet;$source_element;$source_process)

$élément_déposable:=Position deposer `Recupérer l'élément déposable
S i ($élément_déposable=-1) `Si le tableau de destination ne contient aucun élément
  $élément_déposable:=Taille tableau(Tab2)+1
  INSERER ELEMENT(Tab2;$élément_déposable) `Insérer un élément dans le tableau
Fin de si

Tab2{$élément_déposable}:=$source_objet->{$source_element} `Affecter la valeur source à l'élément
```

Ce code tient compte de la possibilité de glisser un élément sur un tableau où aucun élément n'est affiché.

"Position Déposer" retourne le numéro de l'élément du tableau sur lequel un objet a été (glissé et) déposé. Cette fonction retourne " -1 " si l'élément source a été déposé après le dernier élément du tableau.

Si l'utilisateur ne dépose pas l'élément source sur élément existant, ce code ajoutera un élément à la fin du tableau.

La dernière ligne de code affecte la valeur de l'élément glissé à l'élément du tableau sur lequel l'utilisateur a cliqué.

L'élément source est défini dans le deuxième paramètre de la commande "**PROPRIETES GLISSER DEPOSER**". La ligne peut être lue ainsi : affecter l'élément du tableau source que l'utilisateur a glissé, à l'élément du tableau de destination.

" Glisser Déposer " à partir d'une liste hiérarchique

Comme pour les tableaux, glisser des éléments à partir d'une liste hiérarchique exige que vous suiviez l'élément glissé de la liste source et que vous déterminiez où l'utilisateur va le déposer.
Pour manipuler les éléments d'une liste hiérarchique, il faut faire appel à quelques commandes du thème " Liste Hiérarchique ".

Quand vous utilisez le " Glisser Déposer " à partir d'une liste hiérarchique, la commande "**PROPRIETES GLISSER DEPOSER**" vous fourni non seulement la référence de la liste hiérarchique source, mais aussi la position de l'élément glissé sur cette liste. Cette position est retournée dans le deuxième paramètre.
Dans notre exemple, c'est la variable " $source_position ".

L'utilisation de la commande "**INFORMATION ELEMENT**" permet d'affecter la valeur, de l'élément glissé, de la liste hiérarchique source à la variable où il sera déposé.

Ce code affecte la valeur, d'un élément d'une liste hiérarchique à une variable :

```
PROPRIETES GLISSER DEPOSER($source_objet;$source_position;$source_process)

INFORMATION ELEMENT($source_objet->;$source_position;$element_Numero_réference;vText4)
```

" Glisser Déposer " entre deux process

Il est également possible de " glisser déposer " des objets entre deux process différents. Dans ce cas, vous aurez besoin des IDs (Identifiants) des deux process.

Le troisième paramètre de la commande "**PROPRIETES GLISSER DEPOSER**" fournit l'ID du process à partir duquel l'utilisateur fait glisser l'objet.

Cet ID sera le premier paramètre de la commande "**LIRE VARIABLE PROCESS**", qui permet de copier la variable à partir du process source au process de destination.

Code utiliser pour " glisser déposer " entre process :

```
C_TEXTE(variable_process_local)
PROPRIETES GLISSER DEPOSER($source_objet;$source_element;$source_process_id)
LIRE VARIABLE PROCESS($source_process_id;$source_objet->;variable_process_local)
```

Nous devons noter que la variable "local_process_variable" n'était pas déclarée au préalable. Lorsque la commande "LIRE VARIABLE PROCESS" est exécutée, la valeur de la variable est transférée entre les deux process. Par conséquent, les variables du process de destination doivent être déclarées. De plus, la valeur peut ne pas être affectée si la variable destination n'est pas typée.

Code générique

Le code générique facilite et accélère la création et la maintenance d'une base de données. Pour illustrer le code générique, les exemples suivant vous sont proposés.
Le premier exemple permet à un utilisateur de glisser et de déposer sur une variable spécifique vText2 :

```
PROPRIETES GLISSER DEPOSER($source_objet;$source_position;$source_process)

vText2:=$source_objet->
```

Le deuxième exemple générique, peut être utilisé à la place du premier :

```
PROPRIETES GLISSER DEPOSER($source_objet;$source_position;$source_process)

Self->:=$source_objet->
```

Ce dernier exemple utilise la commande " SELF " qui retourne un pointeur sur l'objet.
La dernière ligne est lue : affecter à cet objet la valeur de l'objet de la source.
L'intérêt exposé dans cet exemple est l'utilisation de ce code, pour un autre objet, sans aucune modification. Le fait que le code soit générique signifie qu'il n'est pas écrit seulement pour des objets spécifiques.

Nous avons la possibilité d'insérer ce code dans une autre méthode qui peut être appelée partout dans votre base de données. Prenons, par exemple, le code de la méthode " Deposer " suivante :

```
`Méthode : Déposer
$Objet_déposé:=$1
Si (Evenement   formulaire=Sur déposer )
```

```
  PROPRIETES  GLISSER  DEPOSER($source_objet;$source_element;$source_process)
  $Objet_déposé->:=$source_objet->

Fin  de  si
```

Cette méthode reçoit la fonction " SELF " comme paramètre. Par exemple :

```
`méthode objet de la variable vText

Deposer (Self)
```

Quand cette méthode est exécutée, un pointeur passe à l'objet appelé par la fonction " SELF ".
Ce pointeur est affecté à une autre variable " $Drop_Object ". Utiliser une variable dont le nom décrit ce que l'objet contient est plus utile que le nom d'une variable générique $1, puisque cette méthode peut être appelée dans différents endroits de la base de données.

Cette méthode teste si l'événement détecté est bien l'événement " sur déposer ". La terminaison de code est semblable à celle ci-dessus, exception faite de l'utilisation de la fonction " SELF " qui permet de retourner le pointeur vers l'objet.

Avantages à rendre une méthode générique :

1) Après avoir été écrite, une méthode générique peut être utilisée partout dans une base de données, simplement en l'appelant et en lui donnant les paramètres requis. Nous n'avons, donc, plus de soucis à nous faire quant au contenu d'une méthode générique écrite et documentée correctement.

2) Pour apporter une modification, nous n'avons qu'à modifier le code générique plutôt que d'explorer toutes les copies de la méthode se trouvant dans la base de données.

3) Le code générique n'est présent qu'une seule fois dans la base de données, réduisant ainsi la taille de celle-ci.

Pointeurs et Variables Locales

Il est important, à ce niveau, de rappeler la limite d'utilisation des variables locales et des pointeurs.
Il ne faut jamais utiliser des pointeurs vers des variables locales. La raison de cette interdiction est basée sur la façon dont les variables locales sont manipulées dans la mémoire.

Une variable locale est stockée dans la mémoire, dans une structure appelée " pile ", qui est une zone mémoire réservée au stockage des valeurs temporaires. 4D effectue des changements fréquents dans la pile. Le pointeur peut donc être déplacé, il ne pointera plus vers la même adresse.
Par conséquent, un pointeur, sur une variable locale, ne retourne pas forcément une valeur correcte. Cela dépend des changements, ou non, effectués par 4D, dans la pile. Si vous tentez de dépointer un pointeur vers une adresse erronée dans la pile, vous ferez cracher l'application, ou vous récupèrerez une valeur incorrecte. Il n'y a donc aucun moyen de savoir quand, ou comment 4D a modifié la pile.

Par contre, les variables process et Interprocess sont stockées dans une zone fixe de la mémoire (Heap). Si la mémoire est modifiée, les adresses des variables process et Interprocess sont mises à jour correctement.

Exemples de pointeurs vers des variables :

```
$ptrVar:=->Var  `ce code est bon

ptrVar:=->$Var  `celui-ci n'est pas bon
```

Note : Pour plus d'information sur les variables, consultez la section " Variable " du Manuel de Langue de 4D.