VisualAge Pacbase

Applications eBusiness & Pacbench C/S
Clients Graphiques

Version 3.5

Note

Avant d'utiliser le présent document et le produit associé, prenez connaissance des informations générales figurant à la section "Notices" après la Table des Matières.

Vous pouvez consulter ou télécharger la documentation de VisualAge Pacbase, régulièrement mise à jour, à partir du site Web du Support Technique :

http://www.ibm.com/software/awdtools/vapacbase/productinfo_f.htm

La section Catalogue dans la page d'accueil de la Documentation vous permet d'identifier la dernière édition disponible du présent document.

1ère édition (juin 2004)

La présente édition s'applique à :

- VisualAge Pacbase Version 3.5

Vous pouvez nous adresser tout commentaire sur ce document (en indiquant sa référence) via le site Web de notre Support Technique à l'adresse suivante : http://www.ibm.com/software/awdtools/vapacbase/support.htm ou en nous adressant un courrier à :

IBM Paris Laboratory
1, place Jean–Baptiste Clément
93881 Noisy-le-Grand, France.

IBM pourra disposer comme elle l'entendra des informations contenues dans vos commentaires, sans aucune obligation de sa part.

Sommaire

La *Table des Matières* détaillée est proposée dans les pages suivantes.

Table des Matières

Remarques

Ce document peut contenir des informations ou des références concernant certains produits, logiciels ou services IBM. Cela ne signifie pas qu'IBM ait l'intention de les annoncer dans tous les pays où la compagnie est présente.

Toute référence à un produit, logiciel ou service IBM n'implique pas que seul ce produit, logiciel ou service puisse être utilisé. Tout autre élément fonctionnellement équivalent peut être utilisé, s'il n'enfreint aucun droit d'IBM.

Il est de la responsabilité de l'utilisateur d'évaluer et de vérifier lui-même les installations et applications réalisées avec des produits, logiciels ou services non expressément référencés par IBM.

IBM peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet. Si vous désirez recevoir des informations concernant l'acquisition de licences, veuillez en faire la demande par écrit à l'adresse suivante :

Intellectual Property and Licensing

International Business Machines Corporation

North Castle Drive, Armonk, New-York 10504-1785

USA

Les détenteurs de licences du présent produit souhaitant obtenir des informations sur celui-ci à des fins : (i) d'échange d'informations entre des programmes développés indépendamment et d'autres programmes (y compris celui-ci) et (ii) d'utilisation mutuelle des informations ainsi échangées doivent s'adresser à :

IBM Paris Laboratory

Département SMC

1 place J.B. Clément

93881 Noisy-le-Grand Cedex

FRANCE

De telles informations peuvent être mises à la disposition du Client et seront soumises aux termes et conditions appropriés, y compris dans certains cas au paiement d'une redevance.

IBM peut modifier ce document, le produit qu'il décrit ou les deux.

Marques

IBM est une marque d'International Business Machines Corporation, Inc.

AIX, AS/400, CICS, CICS/MVS, CICS/VSE, COBOL/2, DB2, IMS, MQSeries, OS/2, PACBASE, RACF, RS/6000, SQL/DS, TeamConnection et VisualAge sont des marques d'International Business Machines Corporation, Inc. dans certains pays.

Java et toutes les marques et logos incluant Java sont des marques de Sun Microsystems, Inc. dans certains pays.

Microsoft, Windows, Windows NT et le logo Windows sont des marques de Microsoft Corporation dans certains pays.

UNIX est une marque enregistrée aux Etats-Unis et/ou dans d'autres pays et utilisée avec l'autorisation exclusive de la société X/Open Company Limited.

D'autres sociétés peuvent être propriétaires des autres marques, noms de produits ou logos qui pourraient apparaître dans ce document.

Préambule

Avant de lire ce volume, vous devez connaître le contenu du manuel *Applications eBusiness & Pacbench C/S : Concepts & Architecture*.

Ce manuel ne pouvant contenir toutes les informations relatives au développement d'applications VisualAge Pacbase, vous trouverez ces informations dans les documentations suivantes :

- Manuel *Applications Pacbench C/S: Services Applicatifs*,
- Manuel *Applications eBusiness & Pacbench C/S : Interface de programmation des Proxies*,
- Aide en ligne de Developer workbench,
- Aide en ligne des outils eBusiness,
- *Guide d'utilisation du Middleware*,
- Documentation associée à l'utilisation de WSAD ou de la Station VisualAge Java, notamment la documentation en ligne HTML ou celle associée à tout autre outil Java utilisé.
- Documentation associée à votre environnement de développement COM.

Conventions typographiques

La police `Courier New` est utilisée pour toute chaîne de caractères à saisir, affichée ou correspondant à du code généré.

Les titres des Manuels ainsi que les titres des chapitres dans les renvois sont en *italiques*.

Les symboles suivants sont utilisés :

☞ note, remarque.

renvoi à un autre emplacement dans la documentation.

truc ou astuce, précision utile.

manipulation à effectuer dans un Outil ou Editeur.

précaution à prendre (manipulation risquée ou irréversible...).

Conventions terminologiques

- Une PVD désigne une Proxy Vue de Dossiers.
- Une PVL désigne une Proxy Vue Logique.
- Les termes Proxy Vues de Dossier, Proxy Elémentaires et Proxy Vues Logiques désignent de manière générique les objets Proxy et Composants Proxy.
- Par souci de commodité, dans les parties communes aux environnements Java et COM, le terme *propriété* est utilisé pour désigner à la fois une propriété Java et un attribut COM et le terme *méthode* désigne à la fois une méthode Java et une action COM.

Chapitre 1 : Introduction

L'objet de ce manuel est de guider le développeur dans la réalisation d'applications graphiques dans un environnement Java ou COM, en utilisant les composants Serveur développés avec le module VisualAge Pacbase eBusiness ou Pacbench C/S.

Ces applications peuvent être exécutées sur micro-ordinateur mais elles sont aussi accessibles depuis le Web (Intranet ou Internet).

Contenu du manuel

Ce manuel se compose des chapitres suivants :

- L'introduction décrit brièvement les étapes de développement côté client graphique, de la génération des éléments extraits du Référentiel à la construction graphique des applications.
- Le chapitre suivant présente l'étape de génération et décrit en détail les résultats de cette génération pour chaque cible (Java ou COM).
- Les deux chapitres suivants présentent l'interface publique des composants générés ainsi que la gestion des erreurs et la gestion de la communication.
- Deux autres chapitres présentent des exemples commentés pas à pas de construction graphique d'applications standard et Web. Ils fournissent également des informations sur le test et le packaging d'une application.
- Enfin, l'annexe présente la liste des actions/methodes, attributs/propriétés, événements et classes que vous retrouverez dans ce manuel.

Etapes du développement

La réalisation d'une application graphique comprend les étapes suivantes :

- Le développement des composants serveur avec le module VisualAge Pacbase eBusiness ou Pacbench C/S.

  ☞ Pour plus de détails sur le développement des composants serveur, consultez l'aide en ligne de Developer workbench si vous utilisez Developer workbench, ou bien le manuel *Applications Pacbench C/S : Services Applicatifs* si vous utilisez la Station de Travail VisualAge Pacbase.

- La génération des Proxies à partir de ces composants serveur, via les outils eBusiness, puis le test des Proxies.

  La phase de génération produit des classes qui exécuteront des services associés aux composants serveur.

  Le générateur prend en entrée le fichier d'extraction initialement transferé sur le poste de développement client.

  ☞ Pour plus de détails sur la génération, consultez le Chapitre 2: Génération des Proxies.

- La réalisation d'une application cliente.

  Le développement d'une application cliente inclut l'intégration des objets Proxy ainsi que l'appel de leurs services. L'application graphique peut être développée avec WSAD, VisualAge Java (ou tout autre outil de développement supportant JDK à partir de la version 1.1) ou encore tout outil de développement utilisant la technologie COM.

  ☞ Vous trouverez un exemple commenté de développement d'une application standard et Web dans le Chapitre 4 : Développement d'un Client VisualAge Java et Chapitre 5 : Développement d'un Client au standard COM.

Compatibilité entre les Composants Elémentaires / Objets Proxy

Les Composants Elémentaires et les objets Proxy doivent être générés sous le même numéro de version, car cela peut engendrer un décalage entre les versions et provoquer des incohérences dans l'application cliente.

Pour éviter toute possibilité de décalage entre les versions, vous devez mettre en place un contrôle des versions en positionant une option dans le Composant Elémentaire. Cette option envoie un message d'erreur si une différence de numéro de version est détectée lors de l'appel du Composant Elémentaire par le Client.

Dans Developer workbench, le champ **`Version,`** situé dans l'onglet Définition du Composant Elémentaire, doit être renseigné.

☞ Si vous utilisez VA Pac via la Station de Travail VisualAge Pacbase, vous devez indiquer l'option **`NUVERS`**. Pour plus de détails, consultez le manuel *Applications Pacbench C/S : Services Applicatifs*.

Chapitre 2: Génération des Proxies

Classes génériques livrées (Java)

Pour développer un Client eBusiness dans VisualAge, vous utilisez des classes générées mais également un grand nombre de classes génériques qui sont livrées avec le produit pour éviter la multiplication des éléments à chaque nouvelle génération. Contrairement aux classes générées documentées plus loin, les classes génériques ne dépendent pas des caractéristiques de la Vue Logique traitée.

Vérifiez que les classes génériques sont bien installées sur votre poste avant de commencer tout développement.

Les classes génériques sont livrées dans les packages suivants :

- **com.ibm.vap.generic**

Ce package contient :

- les classes ancêtres des classes **ProxyLv** générées, c'est-à-dire les classes suivantes :
  - **ProxyLv**
  - **HierarchicalProxyLv**
  - **DependentProxyLv**
  - **Folder**
  - **ReferenceProxyLv**
- les classes **Data** génériques (ancêtres des classes **selectionCriteria** et **DataDescription** générées)
- les classes du cache local, c'est-à-dire les classes suivantes :
  - **Node**
  - **HierarchicalNode**
  - **DependentNode**
  - **RootNode**
  - **ReferenceNode**
- les exceptions et erreurs Fonctions eBusiness, c'est-à-dire les classes suivantes :
  - **VapException**
  - **LocalException**
  - **ServerException**
  - **CommunicationError**
  - **SystemError**
- les classes de présentation sous forme de liste des classes **DataDescription** générées (manipulées par la propriété **rows** des objets Proxy).
- les classes décrivant les propriétés des objets Proxy telles qu'elles sont définies dans VisualAge Pacbase :
  - **VapProxyProperties**
  - **VapHierarchicalProxyProperties**
  - **VapDependentProxyProperties**
  - **VapFolderProperties**
  - **VapReferenceProxyProperties**

- Les classes de manipulation de flots XML :
  - **XMLMapping**
  - **XMLWrapper**

- **com.ibm.vap.exchange**

Ce package contient les classes de gestion du Gestionnaire d'échanges.

Documentation en ligne des classes génériques

Une documentation au format HTML est livrée avec le runtime fonction eBusiness. Cette documentation porte sur :

- les classes génériques, ancêtres des classes d'exécution **ProxyLv** et **data** générées,
- les erreurs et exceptions levées par ces classes d'exécution,
- les beans associés aux propriétés de type Rubriques VA Pac, utilisés dans le Composition Editor pour le maquettage rapide de ces données.

Dans la palette **awt**, ces beans sont les suivants :

- **Pacbase Text Field**
- **Pacbase Integer Field**
- **Pacbase Decimal Field**
- **Pacbase Date Field**
- **Pacbase Time Field**
- **Pacbase Long Field**
- **Pacbase Text Choice**
- **Pacbase Integer Choice**
- **Pacbase Decimal Choice**
- **Pacbase Date Choice**
- **Pacbase Time Choice**
- **Pacbase Long Choice**

Dans la palette **swing**, ces beans sont les suivants :

- **Pacbase Swing Text Field**
- **Pacbase Swing Integer Field**
- **Pacbase Swing Decimal Field**
- **Pacbase Swing Date Field**
- **Pacbase Swing Time Field**
- **Pacbase Swing Long Field**
- **Pacbase Swing Text ComboBox**
- **Pacbase Swing Integer ComboBox**
- **Pacbase Swing Decimal ComboBox**
- **Pacbase Swing Date ComboBox**
- **Pacbase Swing Time ComboBox**
- **Pacbase Swing Long ComboBox**
- **Pacbase Swing Date RadioButtonGroup**
- **Pacbase Swing Decimal RadioButtonGroup**
- **Pacbase Swing Integer RadioButtonGroup**

Lancement du générateur

A partir du module eBusiness de Developer workbench

Vous lancez le générateur de Proxies à partir de l'onglet "Applications" ou "Dossiers" de Developer workbench.

Pour plus d'informations sur l'interface du générateur de Proxies, consultez l'aide en ligne des outils eBusiness.

A partir de WSAD (ou Eclipse)

Pour lancer le générateur de Proxies à partir de WSAD ou d'Eclipse,

- sélectionnez "Fichier" → "Nouveau" → "Autre".
- ou faites un clic droit sur un projet Java et choisissez « Nouveau » → « Autre ».

Enfin sélectionnez « VisualAge Pacbase eBusiness »→ « eBusiness Proxy Generator ».

A partir de VisualAge for Java

Pour lancer le générateur de Proxies directement à partir de VisualAge for Java, sélectionnez, dans le menu "Workspace", "Outils" → "VisualAge Pacbase eBusiness" → "Générateur de Proxies". Dans ce cas :

- le type de génération est forcé à Java.
- vous devez indiquer le nom du projet VisualAge for Java dans lequel les classes seront importées.

Pour plus d'informations sur l'interface du générateur de Proxies, consultez l'aide en ligne des outils eBusiness

Si vous souhaitez paramétrer le lancement du générateur de Proxies, sélectionnez , dans le menu "Workspace", "Outils" → "VisualAge Pacbase eBusiness" → " Propriétés du générateur de Proxies".

A partir du fichier .exe

Lancez le fichier **vapgen.exe**.

Vous pouvez paramétrer le lancement du générateur de Proxies en utilisant les paramètres suivants afin de préparer et d'accélérer la génération :

-lang<LANGUAGE> : Ce paramètre permet de choisir la langue de l'interface graphique du générateur : **fr** pour Français, **en** pour English. Par défaut, la langue du système est utilisée.

-input<INPUT_FILE> : cette option permet de spécifier le fichier d'extraction contenant les Dossiers à générer.

-output<OUTPUT_DIR> : Ce paramètre permet de désigner le répertoire dans lequel le résultat de la génération sera stocké.

-java : en choisissant ce paramètre, vous indiquez que la génération est effectuée pour une cible Java.

-com : en choisissant ce paramètre, vous indiquez que la génération est effectuée pour une cible COM.

-includereference : ce paramètre vous permet d'intégrer les noeuds référence à la génération.

-i18n : ce paramètre vous permet de rendre active l'option d'internationalisation pour la génération.

Par ailleurs, les paramètres suivants sont spécifiques à une génération Java :

-proxy<PROXY_PACKAGE> (seulement pour les Proxies provenant du Métamodèle 2.5) : cette option vous permet d'indiquer dans quel package Java seront générées les classes correspondant aux composants proxy.

-data<DATA_PACKAGE> (seulement pour les Proxies provenant du Métamodèle 2.5) : cette option vous permet d'indiquer dans quel package Java seront générées les classes de données.

-config<CONFIGURATION_FILE> : cette option vous permet d'indiquer le nom du fichier de configuration Java qui sera utilisé.

-classpath<PATH> : cette option vous permet d'indiquer le chemin à emprunter pour la compilation des proxies Java générées.

A partir d'une machine virtuelle Java

Le lancement du générateur de proxies à partir d'une machine virtuelle Java s'exécute via le fichier **java_vapGen.bat**.

Ce fichier .bat est un exemple que vous devez modifier selon la localisation de votre JDK (Java Developer ToolKit) ou JRE (Java Runtime Environment).

Vous pouvez indiquer les options de génération ci-dessus pour préparer et accélérer le traitement de la génération.

Résultats de la génération

Les éléments générés dépendent toujours du type de service effectué par le Composant Elémentaire. Ils ne seront pas les mêmes selon que le Composant Elémentaire effectue des services de mise à jour ou simplement de lecture.

Le fichier généré contient uniquement les classes qui dépendent des caractéristiques de la Vue Logique traitée.

Java

Introduction

A l'issue de la génération, les fichiers suivants sont créés :

- Le fichier des localisations, **VAPLOCAT.INI** est créé dans le répertoire de sortie de génération.

  Vous complétez ce fichier en utilisant l'Editeur de Localisation. Cet outil permet de paramétrer le fichier de localisation, évitant ainsi les erreurs de syntaxe qui pourraient inhiber la communication entre le composant Client (Proxy) et les serveurs.

  ✍ Pour plus d'informations, consultez l'aide en ligne de l'Editeur de Localisation.

  Ce fichier doit être fourni en entrée de la gateway.

- Les fichiers sources des classes générées et ressources nécessaires à leur édition.

  ☞ Si vous avez coché l'option **Générer des beans**, une classe **BeanInfo** est générée pour chaque classe d'exécution (**ProxyLv** et **data**). La classe **BeanInfo** associée à une classe donnée (bean) porte le même nom que ladite classe, suivi du suffixe **BeanInfo**.

  Par exemple, lorsqu'une classe **CustomerProxyLv** *(Proxy Racine dans notre exemple) est générée, une classe* **CustomerProxyLvBeanInfo** *est systématiquement générée.*

  Une classe **BeanInfo** contient des informations d'édition (libellé, icône, etc.) pour la classe d'exécution associée. Elle contient les méthodes publiques qui fournissent des informations sur la classe du bean associé, comme le nom de classe, les propriétés, méthodes et événements exposés par le bean.

  ☞ Le choix de l'option **Générer un schéma et des données XML**, provoque la génération des classes **XMLMapping** et **XMLWrapper** pour la classe Proxy Racine (**ProxyLv**).

  Par exemple : **CustomerProxyLvXMLMapping** *et* **CustomerProxyLvXMLWrapper**.

Classes générées

Dans VisualAge, les éléments générés par la Fonction eBusiness ou le module Pacbench C/S correspondent à des classes dont la codification est composée d'un préfixe et d'une partie codée en dur dans le générateur. Le préfixe correspond au nom en clair de la Vue Logique qui comporte 36 caractères au maximum.

☞ Si le préfixe originaire du serveur ne vous convient pas, vous avez la possibilité de le modifier lors de la génération.

- Classe **[Préfixe]Data**

  Cette classe représente la description d'une instance de Vue Logique. Elle possède un ensemble de propriétés correspondant aux Rubriques de la Vue Logique.

- Classe **[Préfixe]SelectionCriteria**

  Cette classe représente la description des critères de sélection. Elle possède un ensemble de propriétés correspondant aux Rubriques de type identifiant et paramètres d'extraction associés à la Vue Logique.

- Classe **[Préfixe]Buffer**

  Cette classe représente la description des informations contextuelles. Elle possède un ensemble de propriétés correspondant aux Rubriques du buffer utilisateur.

- Classe **[Préfixe]DataUpdate**

  Cette classe hérite de la classe **[Préfixe]Data**. Par rapport à la classe parente, elle possède 2 propriétés supplémentaires dont les valeurs varient en fonction des modifications en cours. Ces deux propriétés sont les suivantes :

  - **action** : action de mise à jour qui peut valoir **Read**, **Modified**, **Created** ou **Deleted**. Cette propriété n'est visible que dans la liste de la propriété **UpdatedFolders**.
  - **updatedInstancesCount** : nombre de mises à jour serveur utiles associé à l'instance de Dossier concernée qui peut valoir de 1 à n.

- Classe **[Préfixe]UserData**

  Cette classe hérite de la classe **[Préfixe]Data**. Elle contient une propriété supplémentaire qui correspond à la Rubrique clé de l'instance parente.

- Classe **[Préfixe]TableModel**

  Cette classe n'est générée que si vous avez coché l'option **Utiliser Swing** lors de la génération. Elle hérite de la classe **Pacbase TableModel** et implémente le composant swing **TableModel**, servant à alimenter un tableau swing. Elle permet d'afficher la liste des instances de la classe **[Préfixe]Data** générées dans le tableau swing.

- Classe **[Préfixe]UpdateTableModel**

  Cette classe n'est générée que si vous avez coché l'option **Utiliser Swing** lors de la génération. Elle hérite de la classe **Pacbase UpdateTableModel** et implémente le composant swing **TableModel**, servant à alimenter un tableau swing. Elle permet d'afficher la liste des instances de la classe **[Préfixe]DataUpdate** générées dans le tableau swing.

Si vous avez coché l'option **Générer des Classes Proxies** EJB, les classes suivantes sont générées :

- Classe **[Préfixe]Session**

  Cette classe représente l'interface distante de la session EJB générée.

- Classe **[Préfixe]SessionBean**

  Cette classe correspond à la session EJB générée.

- Classe **[Préfixe]SessionHome**

  Cette classe est la classe Home Interface pour la session EJB générée.

Si vous avez coché l'option **Génération XML**, deux classes et un schéma XML sont générés :

- Classe **[Préfixe]ProxyLvXMLMapping**

  Cette classe représente la description des éléments du mapping spécifique à un Dossier ou une Vue de Dossier. Elle hérite de la classe **XMLMapping**.

- Classe **[Préfixe]ProxyLvXMLWrapper**

  Cette classe offre des méthodes permettant pour chaque noeud d'identifier la demande. Elle hérite de la classe **XMLWrapper**

  Par exemple, la méthode **getCustomerProxyLvHierarchicalDetailXML()** *renvoie*

  ```
  super.getXML(getCustomerProxyLv(), false, true);
  ```

- Schéma XML **[Folder_Name].xsd**

  Ce schéma correspond à la structure complète d'un Dossier ou d'une Vue de Dossier. Il est constitué d'une structure (**ComplexType**) correspondant au noeud racine, celui-ci appelant les structures des nœuds dépendants (**ComplexType**), etc. tels que décrit dans le Dossier et la Vue de Dossier.

De plus, selon le type de **Description de déploiement** choisi, un des fichiers suivants peut être généré :

- **[Préfixe].xml**

  Ce fichier est généré si le type de **Description de déploiement** est positionné à **XML**.

- **[Préfixe].ser**

Ce fichier est généré si le type de **Description de déploiement** est **Sérialisé**. Ce type est utilisé pour une génération EJB conforme à la spécification EJB 1.0.

Import

Si le générateur de Proxies a été lancé à partir de Visualage for Java, cette étape est inutile.

Si, au contraire le générateur de Proxies n'a pas été lancé à partir de VisualAge for Java, vous devez indiquer le nom du projet dans lequel les classes générées seront importées.

Pour ce faire, suivez les étapes suivantes :

- A partir du workbench VisualAge, sélectionnez le choix **Importer...** dans le menu **Fichier**.
- Dans le **SmartGuide –** fenêtre **Type d'import**, entrez le nom du projet dans lequel les classes sont importées ou sélectionnées, via le bouton **Parcourir** (un projet déjà créé).

  ☞ Nous vous déconseillons d'importer des classes générées dans un projet standard VisualAge Pacbase ou Java.

- Comme pour le type d'import, sélectionnez l'option **Répertoire complet (avec ressources)**.

☞ L'option **fichiers Java** peut être aussi sélectionnée. Dans ce cas, les icones spécifiques à la Fonction eBusiness ne seront pas disponibles.

- Cliquez sur **Suivant** pour obtenir le détail suivant. Sélectionnez, via le bouton **Parcourir**, le répertoire dans lequel les classes ont été générées.
  ☞ Utilisez le répertoire qui a été sélectionné pour la génération dans la zone **répertoire de sortie**, dans le détail option de la génération. Dans notre exemple : **C:\vap\gen\java**.

Documentation en ligne des classes générées

La documentation de l'interface publique des classes générées est directement intégrée au code sous forme de commentaires.

Vous pouvez donc consulter cette documentation en accédant directement au source de l'élément de l'interface publique souhaité dans le workbench ou un browser VisualAge.

Vous pouvez également choisir de générer cette documentation à l'aide de l'utilitaire Javadoc livré avec le JDK.

- **Génération de la documentation**

Il est possible de générer la documentation associée à un projet, un package, une classe ou encore une méthode.

Pour lancer Javadoc, vous disposez de deux solutions.

- Dans VisualAge, depuis le workbench ou un browser :
  - sélectionnez le projet, le package, la classe ou la méthode souhaité(e).
  - accédez au menu contextuel associé et sélectionnez le choix **Generate javadoc**. Par défaut, la documentation est générée dans le répertoire de VisualAge, dans le sous-répertoire …**\Ide\javadoc**.
- A partir d'une session DOS ou OS/2, en utilisant les paramètres par défaut du générateur des composants Proxy, lancez la commande :

```
javadoc -classpath c:\vap\generated\java -d c:\doc -public
com.ibm.vap.generated.data com.ibm.vap.generated.proxies
```

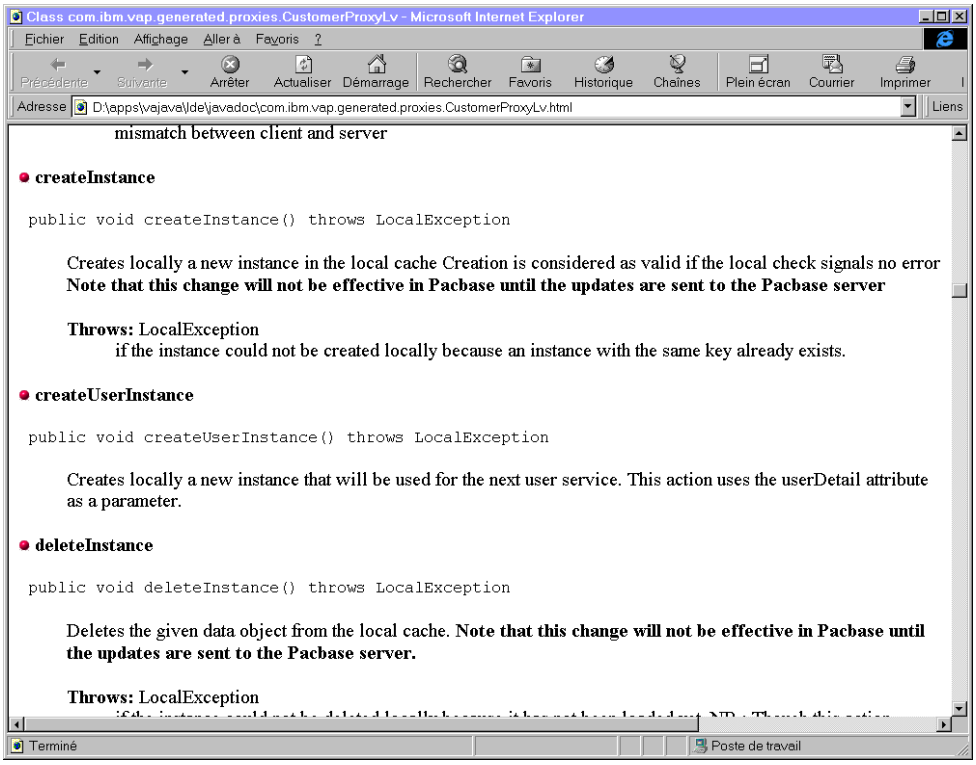
- **Résultat obtenu**

La documentation générée ne contient que des informations pertinentes pour le développeur car elle correspond exactement à chaque classe générée.

En effet, seule la documentation des éléments de l'interface publique - propriétés ou méthodes – **effectivement générés** – est extraite. Cette documentation est au format HTML et inclut les liens hypertextes.

Pour chaque méthode, outre les commentaires, sa signature – paramètre(s), code retour et exceptions - est également extraite.

A titre indicatif en voici un exemple :



☞ Vous pouvez également consulter le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies* où les propriétés, méthodes et événements sont documentés par ordre thématique.

Personnalisation des classes

Toute classe générée hérite d'une classe générique.

Les classes génériques sont chargées une seule fois à l'installation du produit.

Si vous souhaitez implémenter de nouvelles fonctionnalités dans les composants Proxy générés, vous devez vous appuyer sur ce mécanisme d'héritage pour créer de nouvelles classes qui seront réservées à ces traitements. Les classes mères et les classes générées ne doivent pas être modifiées.

- **En suivant le système d'héritage**

Vous pouvez suivre le système d'héritage pour créer de nouvelles classes et implémenter de nouvelles fonctionnalités.

- Personnalisation des classes ProxyLv

  Il est possible d'ajouter un comportement commun à toutes les Proxy Racines, Références ou Dépendantes.

  Il suffit pour cela de créer une classe héritant de **`DependentProxyLv`**, **`FolderProxyLv`** ou **`ReferenceProxyLv`**, d'y implémenter la nouvelle fonctionnalité souhaitée et de modifier ensuite la classe mère des objets Proxy générés.

  ☞ Cette dernière modification devra être refaite à chaque nouvelle génération.

- Personnalisation des classes Data

  Il est possible de changer les classes **Data**, **UserData** et **SelectionCriteria** utilisées par une Proxy donnée.

  Pour cela, il faut définir :

  - une classe héritant de la classe **DataDescription**, y effectuer les implémentations nécessaires, puis modifier les méthodes **newData()** et **newData(String[] values)** de la Proxy.
  - une classe héritant de la classe **UserDataDescription**, y effectuer les implémentations nécessaires, puis modifier les méthodes **newUserData()** et **newUserData(String[] values)** de la Proxy.
  - une classe héritant de la classe **SelectionCriteria**, y effectuer les implémentations nécessaires, puis modifier les méthodes **newSelectionCriteria()** et **newSelectionCriteria(String[] values)** de la Proxy.

    ☞ La classe **UserData** est générée seulement si un service utilisateur a été demandé pour la Proxy.

- **En modifiant la Hiérarchie des classes générées**

Au lancement du générateur, vous pouvez indiquer un fichier de configuration. Ce fichier de configuration permet de modifier la génération standard des Proxies Java. Cela permet, par exemple, de modifier la hiérarchie des classes générées en indiquant la classe mère de chaque type de classe générée. Le format de ce fichier est le format standard du fichier Java **.properties**.

Exemple de fichier de configuration :

```
#
# Default Configuration File for the VisualAge Pacbase for
Java proxy generator.

#
# Folder proxy superclass
Folder.superclass = Folder

# DependentProxy superclass
DependentProxyLv.superclass = DependentProxyLv

# ReferenceProxyLv superclass
ReferenceProxyLv.superclass = ReferenceProxyLv

# UserBuffer superclass
UserBuffer.superclass = DataGroup

# DataDescription superclass
DataDescription.superclass = DataDescription

# DataDescriptionUpdate superclass
DataDescriptionUpdate.superclass = DataDescriptionUpdate

# SelectionCriteria superclass
SelectionCriteria.superclass = DataGroup

# PacbaseTableModel superclass
PacbaseTableModel.superclass =
com.ibm.vap.beans.swing.PacbaseTableModel
```

```
# PacbaseUpdateTableModel superclass
PacbaseUpdateTableModel.superclass =
com.ibm.vap.beans.swing.PacbaseUpdateTableModel

# Locale language
#Locale.language = fr
# Locale country
#Locale.country = FR
# Locale variant
#Locale.variant = EURO

# Default VapConfigurator class
Configurator.class = com.ibm.vap.generator.VapConfigurator

# Default PacbaseUpdateTableModel Status Column header
PacbaseUpdateTableModel.status.header = Status

# Default PacbaseUpdateTableModel Update Count Column
header
PacbaseUpdateTableModel.update_count.header = Update Count
# Default PacbaseUpdateTableModel LockTimestamp Column
header
PacbaseUpdateTableModel.lock_timestamp.header = Lock
Timestamp

# Default DataDescriptionUpdate Read Status Label
PacbaseUpdateTableModel.read.label = Read
# Default DataDescriptionUpdate Created Status Label
PacbaseUpdateTableModel.created.label = Created
# Default DataDescriptionUpdate Modified Status Label
PacbaseUpdateTableModel.modified.label = Modified
# Default DataDescriptionUpdate Deleted Status Label
PacbaseUpdateTableModel.deleted.label = Deleted
```

COM

Introduction

A l'issue de la génération, les fichiers suivants sont créés :

- Le fichier des localisations, **VAPLOCAT.INI** est créé dans le répertoire de sortie de génération.

  Vous devez compléter ce fichier en utilisant l'Editeur de Localisation. Cet outil permet de paramétrer le fichier de localisation, évitant ainsi les erreurs de syntaxe qui pourraient inhiber la communication entre le composant Client (Proxy) et les serveurs.

  Pour plus d'informations, consultez l'aide en ligne de l'Editeur de Localisation.

  Ce fichier doit être fourni en entrée de la gateway.

- Les fichiers sources des classes générées et ressources nécessaires à leur édition dans le répertoire **<Nomdossier><version>** (dans notre exemple : **VDCLNT2.0**).

Classes générées

Les élements générés par la Fonction eBusiness ou le module Pacbench C/S correspondent aux classes dont le code est composé d'un préfixe et d'une partie en dur attribués par le générateur. Le préfixe représente le nom de la Vue Logique, sa longueur est de 36 caractères maximum.

- Classe **[Préfixe]Data**

  Cette classe représente la description d'une instance de Vue Logique. Elle possède un ensemble d'attributs correspondant aux Rubriques de la Vue Logique.

- Classe **[Préfixe]SelectionCriteria**

  Cette classe représente la description des critères de sélection. Elle possède un ensemble d'attributs correspondant aux Rubriques de type identifiant et paramètres d'extraction associés à la Vue Logique.

- Classe **[Préfixe]Buffer**

  Cette classe représente la description des informations contextuelles. Elle possède un ensemble d'attributs correspondant aux Rubriques du buffer utilisateur.

- Classe **[Préfixe]DataUpdate**

  Cette classe hérite de la classe **[Préfixe]Data**. Par rapport à la classe parente, elle possède 2 attributs supplémentaires dont les valeurs varient en fonction des modifications en cours. Ces deux attributs sont les suivants :

  - **action** : action de mise à jour qui peut valoir **Read**, **Modified**, **Created** ou **Deleted**. Cet attribut n'est visible que dans la liste de l'attribut **UpdatedFolders**.
  - **updatedInstancesCount** : nombre de mises à jour serveur utiles associé à l'instance de Dossier concernée qui peut valoir de 1 à n.

- Classe **[Préfixe]UserData**

  Cette classe hérite de la classe **[Préfixe]Data**. Elle contient un attribut supplémentaire qui correspond à la Rubrique clé de l'instances parente.

Si vous avez sélectionné l'option **Génération XML**, deux classes et un schéma XML sont générés :

- classe **[Prefix]ProxyLvXMLMapping**

  Cette classe représente la description des éléments du mapping spécifique à un Dossier ou à une Vue de Dossier. Elle hérite de la classe **XMLMapping**.

- classe **[Préfixe]ProxyLvXMLWrapper**

  Cette classe offre des méthodes permettant pour chaque nœud d'identifier la demande. Elle hérite de la classe **XMLWrapper**.

- Schéma XML **[Nom_Dossier].xsd**

  Ce schéma correspond à la structure complète d'un Dossier ou d'une Vue de Dossier. Il est constitué d'une structure (**ComplexType**) correspondant au noeud racine, celui-ci appelant les structures des nœuds dépendants (**ComplexType**), etc. tels décrit dans le Dossier et Vue de Dossier.

Pour plus d'informations, consultez *la Documentation eBusiness & Pacbench C/S : Interface de Programmation des Proxies.*

Une description sommaire de l'interface publique est disponible en ligne. Vous pouvez la visualiser si votre outil client le permet.

Résultats de la compilation

Si la compilation s'est déroulée correctement, les éléments compilés se trouvent dans le répertoire **<Nomdossier><version>\Release** (dans notre exemple **VDCLNT2.0\Release**).

Un compte-rendu est édité dans le fichier **<Nomdossier><version>\Resume.txt**.

Une fois la génération et la compilation effectuées, la proxy est directement exploitable sur la machine où est installé le générateur.

Compilation avec la version 5.0 et 6.0 de Visual C++

Suite à la génération, vous obtenez un ensemble de sources que vous allez utiliser pour compiler votre Proxy.

Deux fichiers sont générés :

- **C<Foldername>.mak** pour la Bibliothèque statique C++,
- **<Foldername>.mak** pour le composant COM (DLL ou OCX)

Vous pouvez compiler chaque composant en lançant, à partir d'une fenêtre DOS, les fichiers **compil_C<Foldername>.bat** et **compil_<Foldername>.bat**. Ces fichiers sont traités par le compilateur Visual Studio C++.

Chapitre 3: Principes généraux de développement

Après présentation des principes généraux, ce chapitre détaille pas à pas cette étape fondamentale : insertion des objets Proxy avec les liens de programmation impliquant les méthodes, propriétés et événements, gestion des erreurs et gestion de la communication.

☞ Si votre Proxy ne contient qu'une seule Proxy Elémentaire (nécessairement une Proxy Racine), les propriétés, méthodes et événements associés aux lectures massives ou aux nœuds références ou dépendants ne sont pas disponibles.

☞ Par commodité, nous conviendrons que le terme *propriété* désigne à la fois une propriété Java et un attribut COM, et le terme *méthode* désignes à la fois une méthode Java et une action COM.

Représentation visuelle des objets Proxy dans l'environnement cible

Environnement Java

Une fois importée dans la Station VisualAge, la Proxy Vue de Dossier est utilisable par un bean graphique.

Sont présentées ci-dessous les icônes correspondant aux différents objets Proxy, fournies à l'installation.

Icônes	Types de Proxy Elémentaire possibles
	Proxy Racine
	Proxy Dépendante 0,N
	Proxy Dépendante 0,1
	Proxy Dépendante 1,N
	Proxy Dépendante 1,1
	Proxy Référence 0,1
	Proxy Référence 1,1

Environnement COM

Après avoir été générée et compilée, la Proxy ActiveX peut être intégrée à tout langage client supportant un standard COM. La Proxy est représentée graphiquement par l'icone suivante :

Utilisation des propriétés

Une propriété correspond à une information gérée par un objet Proxy. Cette information définit une donnée élémentaire, une liste de données élémentaires ou une liste d'instances de données composées. Une propriété peut correspondre à une constante, à un paramètre ou à un résultat de méthode. En fonction du contexte, il est initialisé par l'application graphique ou la Proxy.

On distingue deux types de propriétés :

- celles qui représentent des variables technologiques. Ils permettent d'affiner le comportement des composants proxy dans VisualAge.
- celles qui correspondent aux données de la Vue Logique.

La disponibilité d'une propriété est fonction du type de Proxy. Toutes les propriétés de l'interface publique sont documentées dans le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies*.

Contrôles locaux

La Proxy Elémentaire exécute automatiquement les contrôles locaux lors de la création et de la modification d'une instance, via les méthodes **`createInstance`** et **`modifyInstance`**. Chaque Rubrique appartenant à la Vue Logique est contrôlée.

Les contrôles exécutés sont les suivants :

- Contrôles des listes de valeurs définies dans la description des Rubriques.
- Contrôles des intervalles définis dans la description des Rubriques.
- Contrôles de présence obligatoire définis au niveau de l'appel des Rubriques dans une Vue Logique. La présence des Rubriques de type identifiant ou de type foreign key pour une relation référence de cardinalité minimum de 1 est contrôlée automatiquement.

Si les contrôles locaux détectent une erreur, un message d'erreur est positionné via le Gestionnaire d'erreurs.

Ces contrôles peuvent être déclenchés de manière sélective pour chaque noeud de type racine ou dépendant du Dossier concerné. L'attribut permettant d'activer ou de désactiver le contrôle des Rubriques sur le serveur est **`serverCheckOption`**.

Que l'on soit en émission ou réception de message, la détection d'une Rubrique vide est automatique.

En revanche, la Proxy n'assure pas les contrôles de numéricité et de date, qui sont pris en charge par les contrôles graphiques.

Contrôle de longueur des champs de la propriété detail

Pour chaque champ de la propriété **`detail`**, lors des contrôles effectués pour une création ou une modification locale d'instance, la longueur de la valeur contenue dans la propriété ne doit pas dépasser la longueur maximale de la valeur de cette propriété.

Le contrôle s'effectue systématiquement (sauf si la propriété n'appartient pas au sous-schéma courant) même si la propriété a été définie comme "non à contrôler dans le client".

Une erreur locale est envoyée en cas de longueur excessive.

☞ Il n'y a pas de contrôle de longueur sur les champs inclus dans les buffers utilisateur. Si la longueur est excessive, elle est tronquée à la longueur maximale.

Sélection du tri local ou serveur sur une liste d'instances

La propriété **localSort** permet de spécifier si la Proxy trie les instances de la propriété **rows** à partir du tri défini en local (**true**) ou laisse les instances triées dans l'ordre d'envoi du serveur (**false**).

Vous pouvez à tout moment modifier le type de tri.

☞ Cette propriété n'est pas effectif dans les services utilisateur.

Tri local

Le tri local des instances de la propriété **rows** correspond au fonctionnement standard de la Proxy si le paramétrage n'a pas été modifié après sa génération. Dans ce contexte, deux types de tri sont possibles :

- Si aucun critère de tri n'est défini en local (voir paragraphe Spécification du critère de tri local (Java exclusivement)), la Proxy trie implicitement les instances dans l'ordre croissant des identifiants définis sur la Vue Logique.
- Si un critère de tri est défini en local, la Proxy trie les instances dans l'ordre défini par ce dernier.

Dans tous les cas, la création locale d'une instance insère celle-ci en fonction du critère de tri courant appliqué dans la propriété **rows**.

Le changement dynamique ou la suppression du critère de tri local induit immédiatement un tri sur les instances contenues dans la propriété **rows**.

Tri serveur

Le tri serveur des instances de la propriété **rows** est déclenché si la propriété **localSort** est à **false**. Dans ce contexte, les instances contenues dans la propriété **rows** sont présentées dans l'ordre dans lequel elles ont été reçues du serveur, quel que soit le contenu du critère de tri défini en local.

Dans le contexte de gestion manuelle des collections ou de pagination en mode extend, les instances reçues sont ajoutées en fin de la collection existante dans la propriété **rows**.

Toute instance créée localement est systématiquement ajoutée à la fin de la collection existante dans la propriété **rows**. Dans ce contexte, une instance qui n'est pas positionnée en fin d'une collection qui est supprimée et recréée localement est transférée à la fin de la collection contenue dans la propriété **rows**.

Spécification du critère de tri local (Java exclusivement)

Il est possible de modifier dynamiquement le critère de tri utilisé pour présenter les instances dans la propriété **rows**, en utilisant la propriété **dataComparator** de chaque Proxy : **void setDataComparator(Comparator c)**.

Le paramètre requis est une instance d'une classe implémentant l'interface **com.ibm.vap.generic.Comparator**.

Cette interface se compose d'une méthode représentant la relation d'ordre suivante : **int compare(Object a, Object b)**.

Cette méthode doit renvoyer :

- un nombre négatif si a < b
- 0 si a = b
- et un nombre positif si a > b.

Par exemple :

```
import com.ibm.vap.generic.Comparator ;
public final class CustomerComparator implements Comparator {
public static final int NAME = 0 ;
public static final int COMPANY = 1 ;
public int criteria ;
public int compare(Object a, Object b) {
try {
switch(criteria) {
case NAME:
return ((CustomerData)a).getName().compareTo(
(CustomerData)a).getName());
break;
case COMPANY:
return ((CustomerData)a).getComp().compareTo(
(CustomerData)a).getComp());
break;
                }
} catch (IllegalCastException ice) {
return 0;
         }
  }
}
```

TableModel (Java exclusivement)

Cette propriété est disponible en lecture/écriture sur tous les types de nœud si vous avez choisi l'option de génération **Utiliser Swing**.

Cette propriété permet d'intégrer dans une application une JTable, composant swing constitué de plusieurs lignes et de plusieurs colonnes.

Par défaut, cette propriété est initialisée avec une nouvelle instance de TableModel générée.

Gestion des sous-schémas

Les sous-schémas spécifiés dans la description de la Vue Logique peuvent être pris en compte par les méthodes de sélection/lecture si les Composants Applicatifs gèrent la présence des Rubriques Elements (activation des paramètres "Génération des vecteurs de présence " ou "Contrôle de données " dans l'onglet Définition du Composant élémentaire de Developer workbench, ou options **VECTPRES=YES** ou **CHECKSER=YES** dans la Station de Travail VA Pac).

Chaque nœud possède deux propriétés :

- **subSchema**, qui permet d'attribuer le sous-schéma désiré lors d'une sélection, lecture ou mise à jour du Composant Elémentaire du nœud. Cette propriété peut être alimentée à partir de la propriété **subSchemaList**.
- **subSchemaList**, qui permet de lister tous les sous-schémas disponibles pour le nœud. Comme il n'est pas possible dans le Référentiel VisualAge Pacbase d'affecter un nom au sous-schéma, chaque sous-schéma est désigné par **SubSchema<n>** (avec **n** de 01 à 10).

☞ Dans l'environnement COM, l' attribut **subSchema** ne peut pas être accédé directement mais seulement via les actions **getSubschemaCount** et **getSubSchemaElementAt (index Int)**.

Utilisation des méthodes

Une méthode correspond à un traitement qu'un objet Proxy peut exécuter. Elle est déclenchée à l'aide d'une connexion entre un événement de l'application graphique et le code d'une méthode d'une Proxy.

✍ La disponibilité d'une méthode est fonction du type de Proxy. Toutes les méthodes de l'interface publique sont documentées dans le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies.*

Les différents types de méthodes serveur

Les méthodes serveur exécutent des traitements implémentés dans un ou plusieurs Composants Applicatifs associés au Dossier. Ces méthodes émettent une requête vers les Composants Applicatifs qui renvoient un résultat sur la station de travail. Les requêtes et les réponses contiennent généralement des paramètres techniques, des instances de Vue Logique associées à un ou plusieurs noeuds et des informations contextuelles définies dans un buffer utilisateur.

Il faut distinguer deux types de méthodes serveur.

- celles qui accèdent systématiquement au serveur :
  - **selectInstances**,
  - **readInstance** : Cette méthode permet d'extraire une instance seulement.
  - **readInstances** : Cette méthode permet d'extraire une ou plusieurs instances, en fonction des clés indiquées.
  - **readInstanceAndLock**,

- **readInstanceWithFirstChildren** (Java) ou **readWithFirstChildren** (COM)
- **readInstanceWithAllChildren** (Java) ou **readWithAllChildren** (COM)
- **readInstanceWithFirstChildrenAndLock** (Java) ou **readWithFirstChildrenAndLock** (COM),
- **readInstanceWithAllChildrenAndLock** (Java) ou **readWithAllChildrenAndLock** (COM),
- **readAllChildrenFromDetail.**
- **readAllChildrenFrom** (Java) ou **readWithAllChildrenFrom** (COM)**:**

- celles qui ne font pas un accès systématique au serveur :
  - **readNextPage** Il y a accès au serveur sauf si lors de la précédente sélection, l'événement (**noPageAfter** pour Java ou **NO_PAGE_AFTER** pour COM) indiquant qu'il n'y a pas de page après la page courante a été renvoyé.
  - **readPreviousPage** : il y a accès au serveur sauf si lors de la précédente sélection, l'événement **noPageBefore** pour Java ou **NO_PAGE_AFTER** pour COM) indiquant qu'il n'y a pas de page après la page courante a été renvoyé.
  - **readFirstChildrenFromDetail** : il y a accès au serveur, sauf si la propriété, qui permet de contrôler le nombre maximum de demande d'instances, (**maximumNumberOfRequestedInstances** pour Java ou **maxNumberOfRequestedInstances** pour COM) des Proxy Dépendantes est positionnée à 0 et si la propriété **globalSelection** est positionnée à false.
  - **checkExistenceOfDependentInstances** pour Java ou **checkExistenceOfDependencies** pour COM: il y a accès au serveur sauf s'il existe en local la possibilité de vérifier l'existence d'instances dépendantes.
  - **updateFolder** : il n'y a d'accès au serveur que s'il existe au moins une instance du noeud concerné modifiée dans sa propriété **updatedFolders.**

Gestion des lectures d'un Dossier

La lecture massive de la racine d'un Dossier permet de lire sur un composant client toutes les occurrences du noeud racine du Dossier présentes dans la base de données. Les méthodes concernées sont **selectInstances** et **readNextPage**.

Lecture massive par anticipation des noeuds dépendants

Dans la cinématique des architectures client/serveur, une application graphique manipulant un Dossier cherche à acquérir des informations par anticipation pour minimiser les échanges avec les serveurs.

Dans un réseau hiérarchique, plusieurs méthodes d'anticipation des lectures peuvent être envisagées :

- La première méthode de type '**allChildren**' lit *toutes* les instances dépendantes de l'instance sélectionnée dans la propriété **detail** de la Proxy Elémentaire parente.

- La deuxième méthode de type '**firstChildren**' ne lit que les instances *immédiatement* dépendantes de l'instance sélectionnée dans la propriété **detail** de la Proxy Elémentaire parente.

La première méthode de lecture massive par anticipation n'est disponible que sur la Proxy Racine.

La deuxième méthode de lecture massive par anticipation est disponible sur la Proxy Racine ou sur les Proxy Dépendantes qui possèdent également des Proxy Dépendantes.

Transfert d'instance entre les propriétés rows et detail

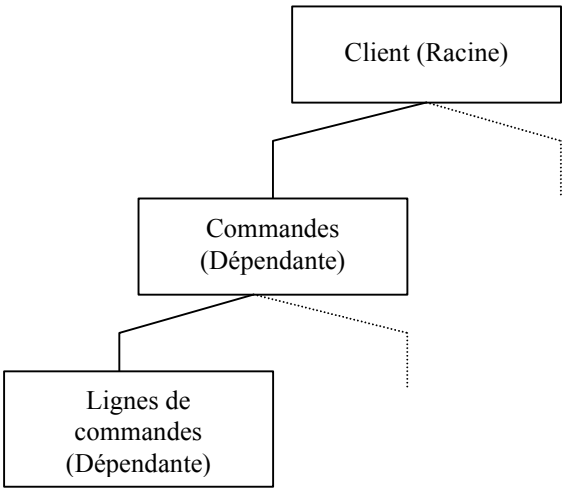
Le transfert d'instance entre la propriété **rows** et **detail** permet d'alimenter la propriété **detail** avec une instance initialement lue par une méthode de lecture massive.

Ce transfert n'est disponible que sur les Proxy Racines et Dépendantes. Il correspond à une méthode de lecture locale qui alimente également toutes les instances locales des Proxy Dépendantes connues par la Proxy Vue de Dossier. Le transfert est réalisé par l'intermédiaire de la méthode **getDetailFromDataDescription** pour Java ou **getDetailFromData** pour COM.

Lecture massive et transfert d'instance entre les propriétés rows et detail : cinématique de fonctionnement

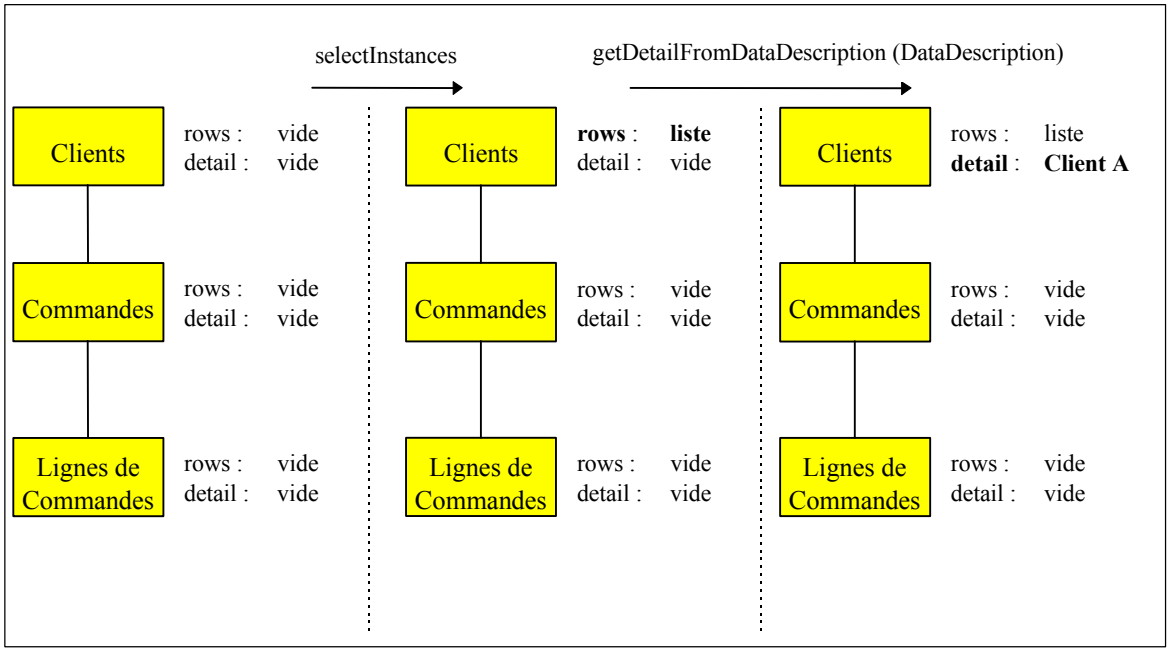
Cet exemple illustre l'alimentation de **detail** avec une instance préalablement transférée dans **rows** par une méthode de lecture massive d'un noeud racine ou dépendant et le principe de la lecture massive par anticipation des noeuds dépendants.

Il est basé sur une Proxy Vue de Dossier composée de trois Proxy Elémentaires :



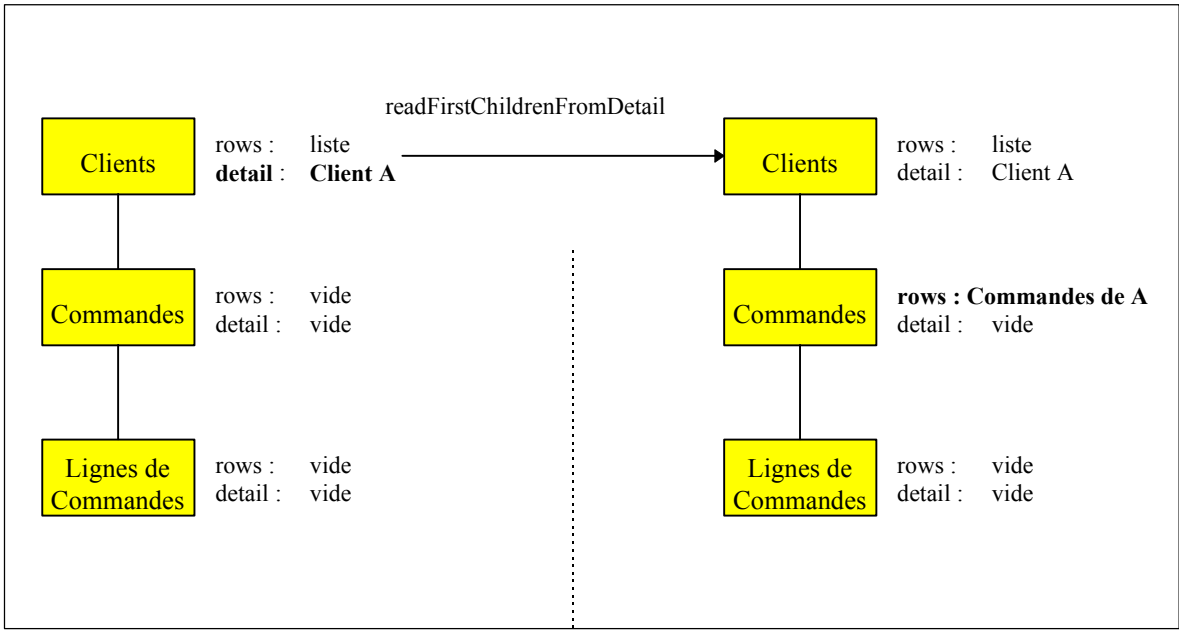
Les schémas ci-dessous présentent la cinématique de fonctionnement basé sur ce principe.

☞ Les codes des méthodes utiliséesdans cet exemple sont des codes Java codes mais le même principe s'applique à l'environnement COM.



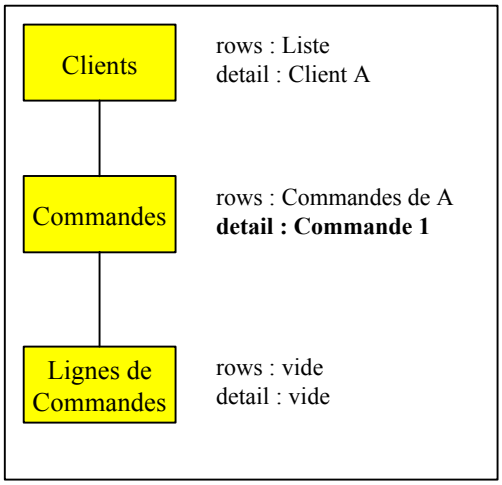
La propriété **`detail`** du noeud Clients contient à présent le Client A. Ensuite, pour lire les instances dépendantes du Client A, c'est-à-dire ses commandes, vous disposez de trois solutions.

SOLUTION 1



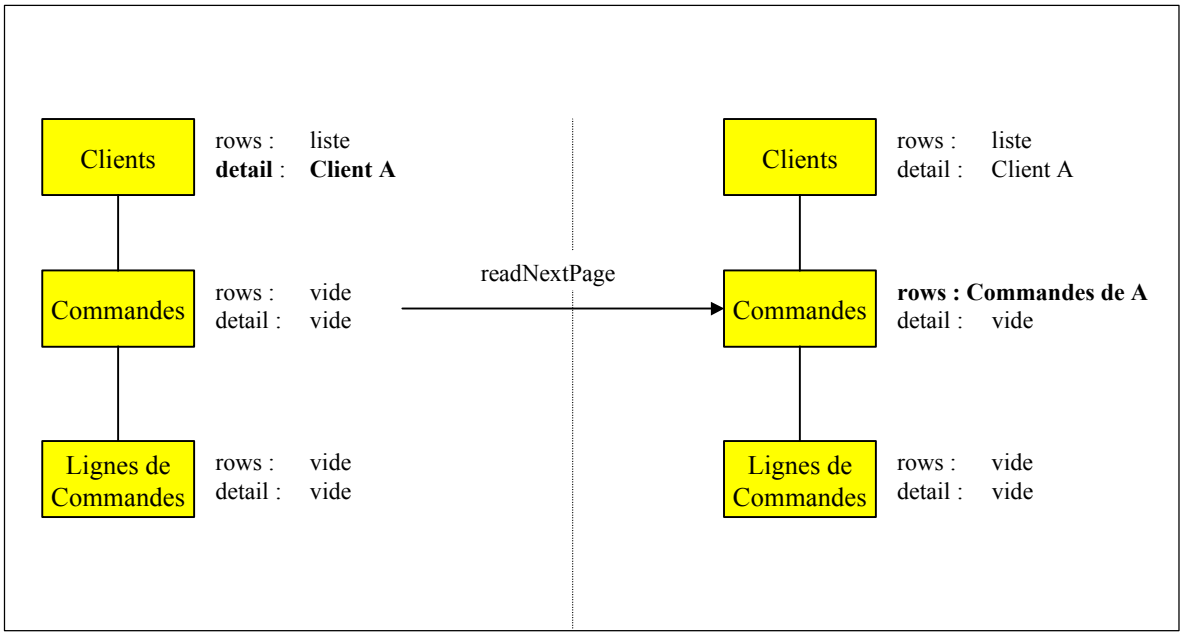
La méthode **`readFirstChildrenFromDetail`** sur la Proxy Racine Clients alimente non seulement la propriété **`rows`** de la Proxy Dépendante Commandes pour le Client A mais aussi la propriété **`rows`** des éventuelles autres Proxy Elémentaires directement dépendantes de la Proxy Racine.

Dans ce contexte, la méthode **`getDetailFromDataDescription (DataDescription)`** sur la Proxy Dépendante Commandes provoque :



Pour lire ensuite les lignes de commandes de la Commande 1 du Client A, utilisez la méthode **`readFirstChildrenFromDetail`** sur Commandes ou la méthode **`readNextPage`** sur Lignes de commandes.

SOLUTION 2

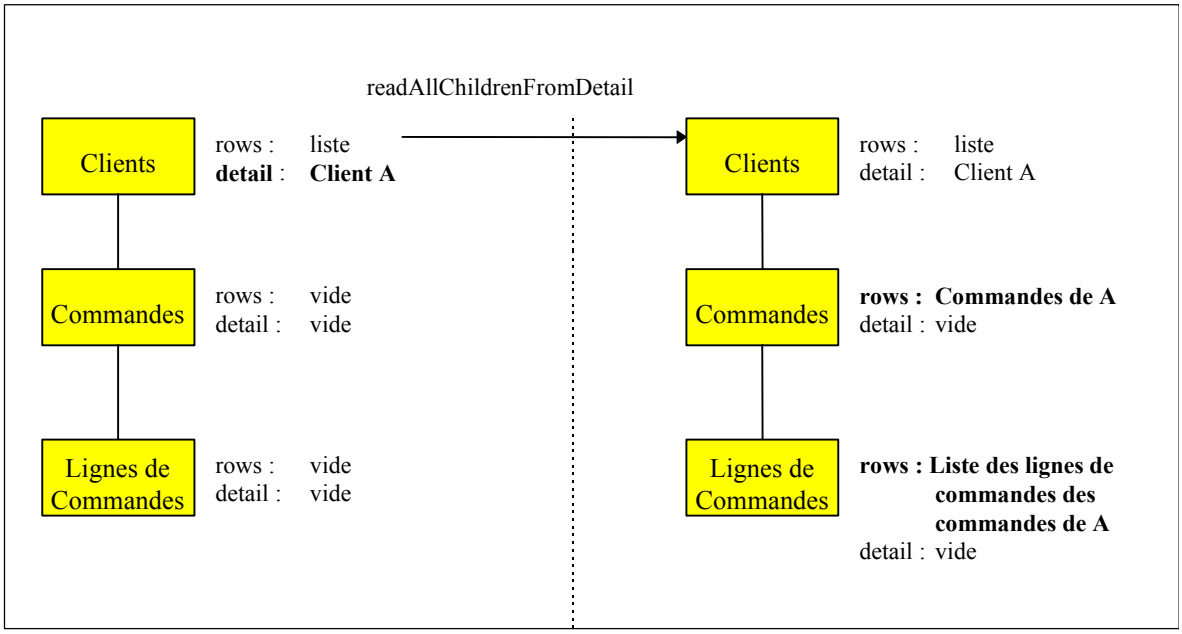


La méthode **`readNextPage`** sur la Proxy Dépendante Commandes alimente sa propriété **`rows`**. Les instances des éventuelles autres Proxy Elémentaires directement dépendantes de la Proxy Racine ne sont pas lues.

Dans ce contexte, la méthode **`getDetailFromDataDescription (DataDescription)`** sur la Proxy Dépendante Commandes provoque le même résultat que dans la solution 1.

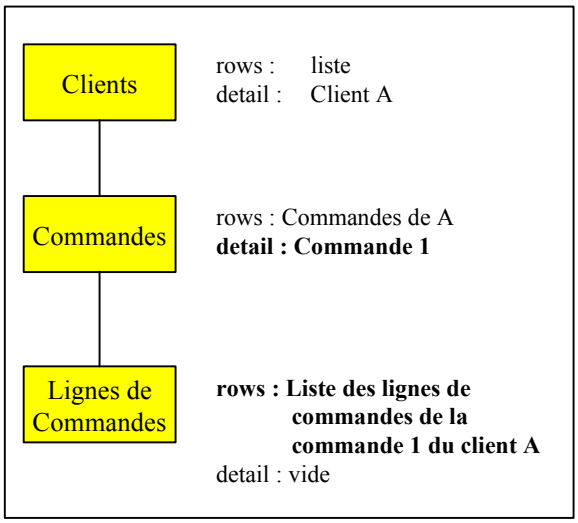
Pour lire ensuite les lignes de commandes de la Commande 1 du Client A, il faut procéder de la même manière que dans la solution précédente.

SOLUTION 3



La méthode **`readAllChildrenFromDetail`** sur la Proxy Racine Clients lit non seulement toutes les commandes de A mais aussi toutes les éventuelles autres instances dépendantes de A quelque soit leur niveau hiérarchique : ainsi dans notre exemple, toutes les lignes de commandes de toutes les commandes de A sont lues.

Dans ce contexte, la méthode **`getDetailFromDataDescription (DataDescription)`** sur la Proxy Dépendante Commandes provoque :



La méthode **`getDetailFromDataDescription (DataDescription)`** alimente automatiquement la propriété **`rows`** de la Proxy Dépendante Lignes de commandes avec les lignes de commandes de la Commande 1 du Client A, ces lignes ayant été transférées en local au préalable par la méthode **`readAllChildrenFromDetail`**.

Lecture massive des noeuds références

La lecture d'un noeud référence est considérée comme une aide sur critères. Elle permet de montrer à l'utilisateur final une liste d'informations potentiellement référençables sur un noeud dépendant.

Les informations présentées à l'utilisateur correspondent aux informations nécessaires et suffisantes pour assister l'utilisateur final dans son choix.

Pour optimiser le volume des caractères transmis pour ce type de service, les Vues Logiques disposent d'un sous-schéma de type « aide sur critères » qui permet de sélectionner les Rubriques concernées au niveau du serveur.

Les lectures massives des noeuds références sont exécutées sur demande et ne peuvent pas être intégrées dans le traitement des lectures par anticipation. Les méthodes concernées sont **`selectInstances`** et **`readNextPage`**.

Principe de pagination dans les noeuds d'un Dossier

Deux types de pagination sont proposés sur les noeuds d'un Dossier :

- Le premier type de pagination, appelé pagination en mode *non-extend*, permet de paginer en avant et en arrière sur une collection prédéfinie, par l'intermédiaire de méthodes spécifiques. Chaque méthode exécute une demande de lecture au serveur et son résultat remplace le résultat de la lecture précédente. Ce type de pagination n'est applicable que sur les noeuds racines ou références.
- Le deuxième type de pagination, appelé pagination en mode *extend*, permet de lire graduellement les instances d'une collection définie, au fur et à mesure des demandes de lectures des pages suivantes. Dans ce contexte, la lecture des pages précédentes disparaît et cette fonction est reprise en local par les ascenseurs du contrôle graphique qui présente la liste des instances. Ce type de pagination est applicable à tous les noeuds d'un Dossier.

Critères de sélection associés aux lectures massives

Les critères de sélection associés aux lectures massives sont des propriétés élémentaires ou composées associées à chaque noeud d'un Dossier. Ils se répartissent en deux types :

- Les critères de sélection fonctionnels correspondent à l'identifiant et aux éléments nécessaires à la définition des méthodes d'extraction de la Vue Logique associée au noeud. Ces critères correspondent aux propriétés suivantes :
  - **`selectionCriteria`** définit les Rubriques de type identifiant et paramètres par valeur.
  - **`extractMethodCode`** définit le code de la méthode d'extraction souhaitée.
  - **`extractMethodCodes`** contient la liste des méthodes d'extraction disponibles.
    - ☞ Dans la cible COM, les actions **`getExtractMethodCodesCount`** et **`getExtractMethodCodesElementAt(Int i)`** vous permettent d'accéder à la liste des méthodes d'extraction disponibles.

- Les critères de sélection organiques correspondent aux informations permettant de maîtriser le volume des instances sélectionné sur chaque nœud.
  - **globalSelection** est une propriété booléenne qui, lorsqu'elle est positionnée à true, permet de lire la totalité des instances du noeud sur une requête de sélection.
  - **maximumNumberOfRequestedInstances** (Java) ou **maxNumberOfRequestedInstances** (COM) est une propriété numérique qui indique, lorsque la propriété **globalSelection** est à false, le nombre d'instances d'un noeud à lire sur une requête de sélection. Cette propriété peut contenir la valeur 0. Dans ce cas, la branche n'est pas lue lors d'une lecture massive par anticipation.

Limitation du champ des lectures massives

Lors d'une lecture massive avec utilisation de critères de sélection, seules les instances qui correspondent à ces critères sont lues.

Cependant, dans le cas d'une lecture massive commandée par une lecture par anticipation de type **allChildren**, on considère que la propriété **globalSelection** est positionnée à true sur chaque noeud.

Lecture d'une instance d'un noeud racine ou dépendant

La lecture d'une instance d'un noeud racine ou dépendant permet d'alimenter directement la propriété **detail** d'un noeud sans passer au préalable par une sélection massive d'une collection des instances de ce noeud.

Ce type de lecture est considéré comme une sélection de collection et annule donc la sélection précédente même si celle-ci correspondait à une lecture massive. L'alimentation de la propriété **detail** entraîne donc l'initialisation de la propriété **rows**.

Les méthodes qui implémentent cette fonction de sélection permettent :

- Une lecture de l'instance sans ses dépendances (**readInstance**).
- Une lecture de l'instance avec ses dépendances de premier niveau (**readInstanceWithFirstChildren**) pour Java ou **readWithFirstChildren** pour COM.
- Une lecture de l'instance avec toutes ses dépendances (**readInstanceWithAllChildren**).

Lecture d'une instance d'un noeud référence

La lecture d'une instance d'un noeud référence ne peut pas activer de processus de lecture massive par anticipation. Elle permet de lire la totalité de la description de l'instance du noeud appelé dans sa propriété **detail**.

Seule la méthode **readInstance** est donc disponible sur un noeud référence.

Les noeuds références sont dépourvus de méthode de mise à jour. Leurs propriétés **rows** et **detail** sont indépendantes. Le résultat de la méthode **readInstance** n'initialise donc pas la propriété **rows**.

La propriété **rows** permet de visualiser les informations suffisantes pour permettre d'affecter une des instances du noeud référence à l'instance du noeud référençant.

☞ Dans l'environnement COM, l'accès à l'attribut **rows** ne peut se faire que par l'intermédiaire des actions **getRowsCount()** et **getRowsElementAt(Int i)**.

La propriété **detail** permet de consulter la totalité de la description d'une instance référencée.

La structure de ces deux propriétés peut donc être différente.

Critères de sélection associés aux lectures d'une instance

L'identifiant de l'instance du noeud à lire est spécifié dans les critères de sélection fonctionnels associés au noeud.

Pour les noeuds dépendants, l'identifiant du noeud correspond à l'identifiant de la Vue Logique associée au noeud débarrassé des Rubriques définissant les identifiants des Vues Logiques hiérarchiquement supérieures. Ces identifiants hiérarchiques sont initialisés automatiquement par la Proxy Vue de Dossier en fonction de la navigation dans le Dossier.

Gestion des mises à jour d'un Dossier

Mises à jour locales

Les services de mise à jour locales sont disponibles sur chaque noeud de type racine ou dépendant du Dossier.

- Création d'une instance de noeud,
- Modification d'une instance de noeud,
- Annulation d'une instance de noeud.

Cependant, il existe d'autres règles inhérentes à la gestion des Dossiers :

- La création d'une instance d'un noeud dépendant n'est autorisée que si la hiérarchie des instances contenues dans les propriétés **detail** des noeuds supérieurs existe.
- L'annulation d'une instance d'un noeud entraîne l'annulation récursive des instances locales des noeuds dépendants.

Pour permettre au développeur de gérer des messages permettant d'avertir l'utilisateur de l'impact d'une annulation en cascade, une méthode de vérification d'existence d'instances dépendantes est disponible sur les noeuds de type racine ou dépendant.

Cette méthode (**checkExistenceOfDependentInstances**) pour Java ou **checkExistenceOfDependencies** pour COM) émet un résultat booléen true ou false. Si aucune dépendance n'est trouvée dans le cache local du Dossier et que l'instance concernée n'a pas été créée localement, elle émet une requête de vérification sur le serveur.

Pour qu'un utilisateur puisse revenir en arrière sur les manipulations locales d'une instance de Dossier, une méthode **undoAllLocalFolderUpdates** permet d'éliminer toutes les mises à jour locales sur tous les noeuds du Dossier appliquées depuis la dernière exécution d'une méthode de mise à jour serveur. Cette méthode n'est disponible que sur le noeud racine du Dossier. Une autre méthode **undoLocalFolderUpdates** permet d'éliminer toutes les mises à jour associées à l'instance du nœud de Racine que vous aurez paramétré.

Mises à jour serveur

La Proxy Racine est seule à disposer des méthodes de mise à jour serveur.

Les mises à jour serveur correspondent aux méthodes permettant à un composant client d'envoyer au serveur l'ensemble des mises à jour locales effectuées depuis la dernière mise à jour de ce type.

Ces mises à jour concernent toutes les instances dépendantes modifiées. Elles peuvent concerner plusieurs instances de Dossier.

Lorsqu'une méthode de mise à jour serveur renvoie des erreurs, elle laisse le Dossier en statut « modifié localement » et seule la correction des erreurs et le renvoi des mises à jour, ou de la méthode **undoAllLocalFolderUpdates** ou **undoLocalFolderUpdates** permettent d'effectuer de nouvelles sélections de collection.

Les méthodes de mise à jour serveur exécutent, avant l'émission de la requête sur le serveur, un contrôle de l'intégrité du Dossier pour les instances créées localement. Cette méthode contrôle, pour chaque instance de noeud créé en local, les cardinalités minimum de chaque lien et émet une erreur si le nombre des instances dépendantes ne respecte pas les propriétés des liens associés.

Une méthode de mise à jour serveur peut être accompagnée d'une demande de rafraîchissement des instances mises à jour dans le cas où certaines Rubriques de ces instances, comme les identifiants, sont calculées par le serveur. Cette demande de rafraîchissement passe par l'utilisation de la propriété **refreshOption**.

Gestion des mouvements utiles

La gestion des mouvements utiles, assurée par le cache local, est entièrement automatique et transparente pour le développeur.

Elle consiste à calculer la mise à jour résultante de plusieurs mises à jour locales effectuées sur la même instance de noeud du Dossier. Elle contrôle la création d'instances en double. Si plusieurs mises à jour locales ont été effectuées sur une même instance de noeud, seule la dernière sera envoyée au serveur.

Réinitialisation des instances du cache local

La méthode **resetCollection** permet d'éliminer explicitement toutes les instances contenues dans le cache d'une Proxy Vue de Dossier avant de reconstituer une nouvelle collection. Cette méthode est disponible sur tous les types de nœuds d'une Proxy Vue de Dossier disposant de la propriété **Rows**.

L'accès à l'attribut **rows** ne peut se faire que par l'intermédiaire des méthodes **getRowsCount()** et **getRowsElementAt(Int i)**.

Gestion des collections d'instances

La gestion des collections d'instances peut se faire soit automatiquement, soit manuellement par le positionnement de la propriété booléenne **`manualCollectionReset`** disponible sur tous les types de nœuds d'une Proxy Vue de Dossier. Le mode de gestion manuel permet de constituer des collections hétérogènes par une succession de méthodes de sélection de collections et de pagination.

Peuplement du cache local sans accès serveur

La méthode **`initializeInstance`** permet de stocker dans le cache local une instance de Vue Logique non lue sur le serveur et qui n'a pas été créée localement. Elle permet donc d'envoyer la mise à jour de l'instance de Vue Logique sans qu'elle ait été lue prélablement sur le serveur. Cette méthode est disponible sur tous les types de nœuds et est valide dès lors que l'instance de Vue Logique n'existe pas en local.

Méthodes asynchrones

Principes

La programmation asynchrone permet de dissocier la méthode d'émission de la requête de celle qui permet de récupérer sa réponse. Vous pouvez utiliser ce type de programmation que vous utilisiez un protocole de communication asynchrone ou non.

Il est possible d'utiliser les composants Proxy en mode asynchrone, et ceci de façon totalement indépendante du middleware utilisé. Ainsi, on peut travailler en mode asynchrone au niveau Proxy, en utilisant une *location* dont le middleware est synchrone, et réciproquement.

Dans ce contexte, l'utilisateur final améliore sa productivité en soumettant à l'avance des requêtes dont il récupèrera la réponse au moment où il en aura besoin. Cette technique de travail par anticipation supprime la perception négative de l'inactivité associée aux temps de réponse.

De plus, les protocoles de communication permettent de sécuriser les requêtes ou les réponses dans des files de messages locales en garantissant l'acheminement du message quel que soit l'état du réseau.

Le mode de conversation est défini par la propriété booléenne **`Asynchronous`** au niveau Dossier.

La méthode **`getLastReplyContext`** permet d'extraire le contexte d'émission de la requête (**`serverActionContext`**).

☞ Dans Java, vous pouvez également utiliser l'exception au contexte d'émission de la requête **`com.ibm.vap.generic.AsynchronousRequestException`**.

Dans le cas où aucune erreur n'est survenue et que la réponse a pu être traitée, **`getReply(context)`** renvoie true. Consultez les exemples.

Méthodes globales ou associées à une instance

Certaines méthodes serveur, désignées comme méthodes globales, sont indépendantes de toute sélection, alors que d'autres dépendent d'une instance donnée contenue dans le cache local. En mode asynchrone, les méthodes globales stockent les identifiants de réponse dans une collection et chaque requête exécutée ajoute son identifiant dans cette collection ; les méthodes associées à une instance de Vue Logique stockent les identifiants de réponse dans une collection associée à l'instance concernée. Les collections d'identifiants de réponse associées aux méthodes globales sont perdues lorsque l'application qui utilise la Proxy est fermée. Une collection d'identifiants de réponse associée à une instance contenue dans le cache local est perdue lorsque cette instance est supprimée localement ou suite à un changement de collection.

- **méthodes globales**

Les réponses associées aux méthodes suivantes peuvent toujours être traitées, indépendamment de la collection courante :

- `executeUserService ()`
- `readInstance ()` (RAC)
- `readInstances()` (ROOT)
- `readInstanceWithFirstChildren ()` (RAC)
- `readInstanceWithAllChildren ()` (RAC)
- `readInstanceAndLock ()` (RAC)
- `readInstanceWithFirstChildrenAndLock ()` (RAC)
- `readInstanceWithAllChildrenAndLock ()` (RAC)
- `readNextPage ()` (RAC)
- `selectInstances ()`
- `lock ()`
- `unlock ()`
- `readPreviousPage ()`

- **Méthodes associées à une instance**

Les méthodes suivantes dépendent soit de l'instance `detail`, soit de l'instance passée en paramètre, soit de l'instance parente `detail` pour les nœuds dépendants :

- `checkExistenceOfDependentInstances ()`
- `readAllChildren(data)`
- `readFirstChildren(data)`
- `readAllChildrenFromCurrentInstance ()`
- `readAllChildrenFrom{}Data` (DEP)
- `readFirstChildrenFromCurrentInstance ()`
- `readFirstChildrenFrom{}Data` (DEP)
- `readInstance ()` (DEP)
- `readInstances()` (DEP si la cardinalité maximale est 'n')
- `readInstanceWithFirstChildren ()` (DEP)
- `readInstanceWithAllChildren ()` (DEP)
- `readInstanceAndLock ()` (DEP)

- **readInstanceWithFirstChildrenAndLock ()** (DEP)
- **readInstanceWithAllChildrenAndLock ()** (DEP)

Exemples d'utilisation

Vous pouvez utiliser les méthodes asynchrones de deux manières, illustrées ci-après.

- **Polling**

Ce système consiste à « guetter » l'arrivée d'une réponse dans un *thread* pour ne pas bloquer l'application en attendant l'affichage des informations. Le code de la réponse se trouve dans un *thread* distinct de celui de l'application principale, mais possédant un pointeur sur la Proxy Racine. On suppose également qu'il connaît un contexte associé à une méthode asynchrone.

```
while (!myFolder().getReply(context)) {
wait(1000);
}
```

- **Accès en tâche de fond**

L'exemple suivant montre comment procéder pour acquérir des informations sur les instances dépendantes avant que l'utilisateur final ne les demande explicitement, et cela sans que l'application soit bloquée.

- Lorsque l'utilisateur sélectionne une collection d'instances radicales

```
myFolder.setAsynchronous(false);
myFolder.selectInstances();
```

  ☞ Son résultat étant nécessaire pour la suite, la méthode **selectInstances** est utilisée en mode synchrone.

- Ensuite, on va parcourir **rows** afin de lire toutes les instances dépendantes de chaque instance lue par la méthode **selectInstances**.

  Au préalable, il est nécesaire de passer en mode asynchrone :

```
myFolder.setAsynchronous(true);
Enumeration rows = myFolder.rows.elements();
Vector contexts = new Vector();
while (rows.hasMoreElements()) {
Data currentData = (Data)rows.nextElement();
try {
myFolder.readAllChildren(currentData);
} catch (VapException ve) {
} catch (AsynchronousRequestException are) {
contexts.addElement(are.getContext();
    }
}
myFolder.setAsynchronous(false);
```

- Ensuite, lorsque l'utilisateur décide de travailler sur une donnée data :

```
try {
int index = myFolder.rows().indexOf(data);
    myFolder.getReply(contexts.elementAt(index));
} catch (VapException) {} //Tout s'est bien passé //
myFolder.getDetailFromDataDescription(data);
```

Ainsi, les instances dépendantes apparaissent dans l'arbre de sélection « spontanément ». Il est à noter que les réponses des méthodes ne sont traitées qu'au dernier moment, ce qui n'est pas une obligation.

Sauvegarde du contexte d'une Proxy

Vous pouvez sauvegarder le contexte de lecture d'une Proxy.

L'utilisation de la méthode **`getProxyContext`** sur le noeud racine, permet la sauvegarde des clés courantes sur chaque noeud, des clés suivantes et précédentes, des critères de sélection, du détail courant et des mises à jour locales.

En utilisant la méthode **`initFromProxyContext`** et en donnant un contexte préalablement mémorisé, toutes les clés de lecture de votre Proxy seront restaurées. You pourrez donc savoir où vous vous étiez arrêté avant de sauvegarder votre contexte.

Externalisation de la gestion des requêtes

Il est possible de gérer les demandes de services dans un objet spécifique qui est l'instance de la classe **`MainRequest`**. Vous pouvez ensuite signaler des demandes de services, elles seront envoyées par différentes Proxies avant qu'une seule demande ne soit envoyée au serveur.

Les proxies partagent alors le même contexte d'execution.

La méthode **`createRequest`** permet l'initialisation d'une requête sur toute Proxy racine et la méthode **`setRequest`** permet d'indiquer quelles Proxies racine participent à la requête. La demande est envoyée au serveur via la méthode **`sendRequest`**.

Les Proxies participant à la requête doivent appartenir à la même application eBusiness que la Proxy qui crée la requête.

Services utilisateur

Java

Chaque noeud de type racine ou dépendant dispose pour mettre en oeuvre un service utilisateur des éléments suivants :

- Une propriété permettant d'avoir la liste des services utilisateurs disponibles sur ce noeud plus **`nil`**. Cette propriété est **`userServiceCodes`**.
- Une propriété permettant d'initialiser le code du service utilisateur à exécuter. Cette propriété est **`userServiceCode`**.
- Une propriété permettant de stocker localement des instances de Vue Logique à traiter pour le prochain service utilisateur. Cette propriété est **`userServiceInputRows`**.
- Une propriété permettant de présenter plusieurs instances de Vue Logique renvoyées par un service utilisateur. Cette propriété est **`userServiceOutputRows`**.
- Une propriété présentant les instances de Vue Logique candidates à l'exécution du prochain service utilisateur. Cette propriété est **`userDetail`**.

- Des méthodes locales permettant de mémoriser chaque instance de Vue Logique à envoyer au serveur pour l'exécution d'un service utilisateur. Ces méthodes sont **createUserInstance**, **modifyUserInstance** et **deleteUserInstance**.

Le noeud racine du Dossier dispose en plus des éléments suivants :

- Une méthode permettant d'exécuter l'ensemble des services utilisateur paramétrés sur chaque noeud du Dossier. Cette méthode est **executeUserServices**.
- Une méthode permettant d'effacer toutes les instances locales mémorisées pour tous les noeuds du Dossier. Cette méthode est **resetUserServiceInputInstances**.
- Une méthode permettant d'effacer l'instance courante mémorisée en local. Cette méthode est **resetUserServiceCodes**.

Ce principe permet d'exécuter dans la même requête un ensemble de 1 à n services utilisateur répartis sur des noeuds différents. L'ordre d'exécution de ces services correspond à l'ordre hiérarchique des noeuds, en parcourant l'arbre de haut en bas et de gauche à droite.

COM

Chaque noeud de type racine ou dépendant contient les éléments suivants pour la mise en œuvre d'un service utilisateur :

- Deux actions permettant d'avoir la liste des services utilisateurs disponibles sur le noeud. Ces actions sont **getUserServiceCodesCount()** et **getUserServiceCodesElementAt(Int i)**.
- Deux actions permettant de stocker localement des instances de Vue Logique à traiter pour le prochain service utilisateur. Ces actions sont **getUserInputRowsCount()** et **getUserInputRowsElementAt(Int i)**.
- Deux actions permettant de présenter plusieurs instances de Vue Logique renvoyées par un service utilisateur. Ces actions sont **getUserOutputRowsCount()** et **getUserOutputRowsElementAt(Int i)**.
- Un attribut présentant les instances de Vue Logique candidates à l'exécution du prochain service utilisateur. Cet attribut est **userDetail**.
- Des actions locales permettant de mémoriser chaque instance de Vue Logique à envoyer au serveur pour l'exécution d'un service utilisateur. Ces actions sont **createUserInstance**, **modifyUserInstance** et **deleteUserInstance**.

Le noeud racine du Dossier dispose en plus des éléments suivants :

- Une action permettant d'exécuter l'ensemble des services utilisateur sur chaque noeud du Dossier. Cette action est **executeUserServices**.
- Une action permettant d'effacer toutes les instances locales mémorisées pour tous les noeuds du Dossier. Cette action est **resetUserRows**.

- Une action permettant d'effacer l'instance courante mémorisée en local. Cette action est **resetUserServiceCodes**.

Ce principe permet d'exécuter dans la même requête un ensemble de 1 à n services utilisateur répartis sur des noeuds différents. L'ordre d'exécution de ces services correspond à l'ordre hiérarchique des noeuds, en parcourant l'arbre de haut en bas et de gauche à droite.

Verrouillage logique de la base

Les mécanismes d'upload-download associés à un Dossier augmentent le temps écoulé entre la lecture d'un Dossier et l'affichage de la mise à jour.

Dans ce contexte, sans mécanisme de verrouillage, deux utilisateurs peuvent modifier la même instance de Dossier. Les mises à jour cumulées deviennent difficiles à gérer.

Pour permettre à l'utilisateur d'utiliser un Dossier dans un mode d'appropriation exclusif, deux types de verrouillage d'une instance de noeud sont à sa disposition :

- Le verrouillage optimiste qui fonctionne sur le principe de la vérification de l'évolution d'un **TimeStamp** avant d'exécuter le traitement de mise à jour.
- Le verrouillage pessimiste qui s'approprie en mise à jour exclusive une entité au moyen de l'enregistrement d'une ressource spécifique. Dans ce cas, le traitement de mise à jour du Dossier est effectué avant de libérer la ressource exclusive.

Le traitement de verrouillage serveur est déclenché en fonction de l'exécution explicite d'une méthode spécifique disponible sur la Proxy Racine. Cette méthode est **lock**.

Le traitement de déverrouillage serveur est déclenché automatiquement avec l'exécution d'une méthode de mise à jour serveur ou explicitement en fonction de l'exécution d'une méthode spécifique disponible sur la Proxy Racine. Cette méthode spécifique est **unLock**.

L'écriture du traitement de verrouillage dans le Composant Elémentaire racine est à la charge du développeur.

Ce traitement reçoit l'identifiant de l'instance de Vue Logique à verrouiller ainsi que le type de demande (verrouillage ou déverrouillage) à exécuter.

Il doit positionner en retour un statut permettant d'accorder ou de refuser le verrouillage ainsi que le **TimeStamp** ou le nom de la ressource utilisée.

En cas de refus de verrouillage, la Proxy Racine émet un événement **lockFailed** pour Java et **LOCK_FAILED** pour COM.et toutes les méthodes de mise à jour locales ou serveur exécutées ultérieurement sont invalidées pour l'instance de Dossier spécifiée. Dans ce cas, le Dossier passe en statut « lecture uniquement ».

Le verrouillage logique est défini dans l'entité Dossier ou dans le Composant Elémentaire en cas de développement mono-vue.

Lorsque l'option de verrouillage logique est active sur un Dossier, toute demande de lecture de la propriété **detail** de la Proxy Racine peut être accompagnée d'une demande de blocage logique au niveau du serveur.

Personnalisation des colonnes d'une Jtable (Java uniquement)

Une JTable est un composant swing disponible à partir de la version 2 de VisualAge Java.

Si vous intégrez ce composant tel quel, il affichera, à l'exécution, toutes les colonnes correspondant à toutes les Rubriques de la Vue Logique, avec les noms en clair définis dans la Vue Logique.

Pour sélectionner les colonnes à afficher, modifier leur en-tête ou encore créer une nouvelle colonne affichant des informations calculées en local, vous devez personnaliser la JTable.

Pour cela, vous devez d'abord créer une nouvelle classe publique héritant soit du TableModel généré (utile pour récupérer une partie de son implémentation) soit directement de **PacbaseTableModel** (package **com.imb.vap.beans.swing**).

Il suffit ensuite de personnaliser les méthodes suivantes :

- **public Int getColumnCount()** : retourne le nombre de colonnes à afficher.
- **public String getColumnName(int col)** : retourne le titre de la colonne **col** (commence à 0).
- **public Object getValueAt(int row, int column)** : retourne l'objet (String en général) à afficher en ligne **row**, colonne **column**.

L'exemple suivant hérite d'un TableModel généré. Il réduit le nombre de colonnes à afficher de 7 à 3. Les deux premières colonnes représentent deux Rubriques standard (numéro et nom du client). La troisième représente l'adresse du client, c'est-à-dire le résultat de la concaténation de la rue, du code postal et de la ville.

```
package test.swing;


import com.ibm.vap.generated.reuse.CustomerData;
public class NewCustomerTableModel extends
com.ibm.vap.generated.reuse.CustomerTableModel {


public int getColumnCount (){
  return 3;
}


public String getColumnName (int i){
  if (i == 0) return "Id";
  if (i == 1) return "Name";
  if (i == 2) return "Address";
  return "";
}
```

```
public Object getValueAt(int row, int column){
  try {
          CustomerData data = (CustomerData) getRows().elementAt(row);
          if (column == 0) return data.getCusId();
          if (column == 1) return data.getCusNam();
          if (column == 2) {
                  String result = "" +
                          data.getStreet() +"." +
                          data getZipcod() + "-"+
                          data.getTown();
                  return result;
          }
  } catch (Throwable t) {}
  return null;
}

}
```

Gestion de la présence des Rubriques

Java

Les deux méthodes suivantes vous permettent de gérer la présence des Rubriques de la Vue Logique au niveau de la Proxy.

La méthode **is<corub>Present** permet de tester la présence ou l'absence de la Rubrique **corub**. Elle est générée pour toutes les classes **DataDescription** et **UserDataDescription**.

La méthode **set<corub>Present(boolean aBoolean)** permet d'indiquer la présence ou l'absence de la Rubrique avant toute méthode de mise à jour locale. Elle est générée pour toutes les classes **DataDescription** et **UserDataDescription**.

Par défaut, les Rubriques sont considérées comme étant absentes, sauf si une valeur par défaut a été indiquée dans la description VisualAge Pacbase.

COM

Les deux méthodes suivantes vous permettent de gérer la présence des Rubriques de la Vue Logique au niveau de la Proxy.

La méthode **is<corub>Present** permet de tester la présence ou l'absence de la Rubrique **corub**. Elle est générée pour toutes les classes **DataDescription** et **UserDataDescription**.

La méthode **set<corub>Present(aBoolean)** permet d'indiquer la présence ou l'absence de la Rubrique avant toute méthode de mise à jour locale. Elle est générée pour toutes les classes **DataDescription** et **UserDataDescription**.

Par défaut, les Rubriques sont considérées comme étant absentes, sauf si une valeur par défaut a été indiquée dans la description VisualAge Pacbase.

Gestion du contrôle des Rubriques

Java

Les trois actions suivantes vous permettent de gérer le contrôle des Rubriques de la Vue Logique au niveau de la Proxy.

La méthode **get<corub>Index** indique l'indice du champ **corub** dans la classe **DataDescription**. Cet indice est utilisé dans l'activation du contrôle serveur sur la Rubrique **corub**.

La méthode **setcheck(int index, boolean aBoolean)** permet d'activer ou d'inhiber les contrôles serveur sur une Rubrique (pointée par l'indice) avant toute mise à jour locale. Elle est générée pour toutes les classes **DataDescription** des nœuds racines ou dépendants dont le Composant Elémentaire possède le paramètre "génération des vecteurs de présence" (Developer workbench) ou les options **NULLMNGT=YES** et **CHECKSER=YES** dans la Station de Travail VA Pac.

Par défaut, toutes les Rubriques sont à contrôler (si l'attribut **serverCheckOption** est positionné à **true**).

COM

Les deux actions suivantes vous permettent de gérer le contrôle des Rubriques de la Vue Logique au niveau de la Proxy.

L'action **setCheck<fieldIndex,aBoolean>** permet d'activer ou d'inhiber les contrôles serveur sur une Rubrique avant toute mise à jour locale.

L'action **getCheck<fieldIndex>** permet de tester si les contrôles serveur sont activés sur une Rubrique.

Ces eux actions sont générées pour toutes les classes **DataDescription** classes des nœuds racine ou dépendants dont les Composants applicatifs inclus le paramètre " génération des vecteurs de présence " (Developer workbench) ou les options **NULLMNGT=YES** and **CHECKSER=YES** dans la Station de Travail VA Pac.

Par défaut, toutes les Rubriques doivent être contrôlées (si l'attribut **serverCheckOption** est positionné à **true**).

Gestion des sous-schémas

Toute méthode serveur de sélection, lecture ou mise à jour prend en compte le sous-schéma présent dans la propriété **subSchema** et retourne donc les valeurs des Rubriques du sous-schéma. Si une sélection est suivie d'une pagination, le sous-schéma pris en compte est celui associé à la méthode de sélection.

Toute création locale est effectuée hors sous-schéma.

Toute modification/suppression locale est effectuée sur le sous-schéma associé à l'instance, c'est à dire :

- si la modification/suppression est effectuée sur une instance créée localement, le sous-schéma est vide.

- si la modification/suppression est effectuée sur une instance lue, le sous-schéma est celui associé à la sélection de cette instance.

De plus, les trois méthodes suivantes sont spécifiques à la gestion des sous-schémas.

La méthode **`resetSubSchema`** permet de réinitialiser la propriété **`subSchema`** à vide, c'est à dire de ne sélectionner aucun sous-schéma.

La méthode **`completeInstance`** permet de récupérer, par appel du Composant Elémentaire associé à la Vue Logique, les valeurs des Rubriques n'appartenant pas au sous-schéma.

La méthode **`belongsToSubschema`** permet de savoir si la Rubrique passée en paramètre appartient au sous-schéma associé à l'instance contenue dans l'attribut **`detail`**.

Utilisation des événements

Java

Les événements émis par une Proxy permettent de déclencher des méthodes applicatives appartenant à l'application graphique. Ces traitements sont exécutés en connectant un événement de la Proxy à une ou plusieurs méthodes de l'application graphique. L'exécution conditionnelle des méthodes est facilitée par le fait qu'un événement est toujours accompagné de son événement contraire, les deux événements ne pouvant jamais être émis simultanément.

La disponibilité d'un événement est fonction du type de Proxy. Tous les événements de l'interface publique sont documentés dans le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies*.

Gestion événementielle des lectures massives

La gestion événementielle des lectures massives permet au développeur d'avoir des informations sur l'état de la collection d'instances contenu dans un noeud. Chaque méthode de pagination disponible propose son propre système de pagination événementielle.

La méthode de pagination en mode non-extend peut émettre les quatre événements suivants :

- **`noPageBefore`** : Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue est la première de la collection courante.

- **`pageBefore`** : Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue n'est pas la première de la collection courante.

- **noPageAfter** : Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue est la dernière de la collection courante.

- **pageAfter** : Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue n'est pas la dernière de la collection courante.

La méthode de pagination en mode extend peut émettre les deux événements suivants :

- **pageAfter** : Cet événement est émis par tout type de noeud à la fin de l'exécution d'une méthode de sélection de collection ou de pagination avant lorsque celle-ci ne renvoie pas d'erreur et que le nombre d'instances contenu dans le noeud ne correspond pas au nombre d'instances total contenu dans la base.

- **noPageAfter** : Cet événement est émis par tout type de noeud à la fin de l'exécution d'une méthode de sélection de collection ou de pagination avant lorsque celle-ci ne renvoie pas d'erreur et que le nombre d'instances contenu dans le noeud correspond au nombre d'instances total contenu dans la base au moment de la requête.

Gestion événementielle des lectures d'une instance

Une Vue Logique peut être mappée sur une ou plusieurs entités de stockage physique. Dans ce contexte, la gestion événementielle de la lecture d'une instance de Vue Logique peut émettre les événements suivants :

- **notFound** lorsque l'instance recherchée n'existe pas dans la base de données. Cet événement peut être émis lorsque la Vue Logique est mappée sur une ou plusieurs tables.

COM

Les événements émis par une Proxy permettent de déclencher des méthodes applicatives appartenant à l'application graphique. Ces traitements sont exécutés en connectant un événement de la Proxy à une ou plusieurs méthodes de l'application graphique. L'exécution conditionnelle des méthodes est facilitée par le fait qu'un événement est toujours accompagné de son événement contraire, les deux événements ne pouvant jamais être émis simultanément. Une fois envoyé, l'événement est stocké dans un stack. Cependant, l'application graphique doit accéder régulièrement au stack en utilisant les méthodes **getServerEventsCount** et **popServerEvent**.

La disponibilité d'un événement est fonction du type de Proxy. Tous les événements de l'interface publique sont documentés dans le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies.*

Gestion événementielle des lectures massives

La gestion événementielle des lectures massives permet au développeur d'avoir des informations sur l'état de la collection d'instances contenu dans un noeud. Chaque méthode de pagination disponible propose son propre système de pagination événementielle.

La méthode de pagination en mode non-extend peut émettre les quatre événements suivants :

- **NO_PAGE_BEFORE**: Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue est la première de la collection courante.

- **PAGE_BEFORE**: Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue n'est pas la première de la collection courante.

- **NO_PAGE_AFTER**: Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue n'est pas la dernière de la collection courante.

- **PAGE_AFTER**: Cet événement est émis par un noeud racine ou référence à la fin de l'exécution d'une méthode de sélection de collection ou de pagination lorsque celle-ci ne renvoie pas d'erreur et que la page lue n'est pas la dernière de la collection courante.

La méthode de pagination en mode extend peut émettre les deux événements suivants :

- **PAGE_AFTER**: Cet événement est émis par tout type de noeud à la fin de l'exécution d'une méthode de sélection de collection ou de pagination avant lorsque celle-ci ne renvoie pas d'erreur et que le nombre d'instances contenu dans le noeud ne correspond pas au nombre d'instances total contenu dans la base.

- **NO_PAGE_AFTER**: Cet événement est émis par tout type de noeud à la fin de l'exécution d'une méthode de sélection de collection ou de pagination avant lorsque celle-ci ne renvoie pas d'erreur et que le nombre d'instances contenu dans le noeud correspond au nombre d'instances total contenu dans la base au moment de la requête.

Gestion événementielle des lectures d'une instance

Une Vue Logique peut être mappée sur une ou plusieurs entités de stockage physique. Dans ce contexte, la gestion événementielle de la lecture d'une instance de Vue Logique peut émettre l' événement suivant :

- **NOT_FOUND** lorsque l'instance recherchée n'existe pas dans la base de données. Cet événement peut être émis lorsque la Vue Logique est mappée sur une ou plusieurs tables

Gestion des erreurs

Il n'y a pas de message d'erreur lors des phases d'extraction et de génération.

Il est donc indispensable d'avoir compilé correctement le Composant Elémentaire que l'on veut extraire car

- la commande GVC de la procédure GPRT extrait toujours sans afficher d'erreur, même si le Composant Elémentaire ne contient aucune Vue Logique,
- le générateur ne fait aucun contrôle sur le fichier qu'il prend en entrée.

En revanche, des messages d'erreur peuvent apparaître lors des phases de développement, de test et d'exécution de l'application. Ces messages proviennent d'erreurs locales, serveur ou de communication.

Introduction

Il existe quatre types d'erreurs :

- Erreurs locales envoyées par le Composant Client qui correspondent aux erreurs de manipulation ou de saisie dans le Composant Client.
- Erreurs serveurs envoyées par le Composant Elémentaire. Ce sont des erreurs d'accès aux données et des erreurs utilisateur positionnées dans le serveur.
- Erreurs système envoyées par le Composant Elémentaire et qui correspond à un décallage entre la Proxy et les Composants applicatifs,
- Erreurs de communication.

Vous trouverez ci-dessous la liste des erreurs locales et des erreurs de communication que vous pourriez rencontrer ainsi que la structure de la clé d'erreur pour les erreurs serveur et système..

Vous trouverez les informations sur la gestion des erreurs spécifique à l'environnement Java dans le Chapitre 4 : Développement d'un Client VisualAge Java, sous-chapitre Gestion des erreurs et les informations sur la gestion des erreurs spécifique à l'environnement COM dans le Chapitre 5 : Développement d'un Client au standard COM, sous-chapitre Gestion des erreurs.

Erreurs locales

Liste des erreurs locales

- ASYNCHRONOUS_VIOLATION

  Cette erreur se produit en cas de demande de verrouillage, de déverrouillage ou de vérification d'existence d'instances dépendantes en mode asynchrone.
- CARDINALITY_VIOLATION

  Cette erreur se produit en cas de non respect des cardinalités lors de l'activation d'une méthode de mise à jour.
- CREATION_CONTEXT_INVALID

Cette erreur se produit lorsque l'on essaie de sauvegarder un contexte Proxy alors qu'une demande a déjà été faite pour cette Proxy.

- CURRENT_INSTANCE_MISSING

  Cette erreur se produit quand une méthode est appliquée à la propriété **`detail`** alors que cette propriété ne contient pas d'instance.

- CURRENT_USER_INSTANCE_MISSING

  Cette erreur se produit quand une méthode utilisateur est appliquée à la propriété **`userDetail`** alors que cette propriété ne contient pas d'instance.

- FOLDER_USER_CONTEXT_LENGTH_ERROR

  Cette erreur se produit quand le contenu d'une rubrique du buffer utilisateur a une taille supérieure à la taille autorisée pour la rubrique.

- INSTANCE_ALREADY_LOCKED

  Cette erreur se produit en cas de demande de verrouillage d'une instance déjà verrouillée sur le serveur.

- INSTANCE_NOT_LOCKED

  Cette erreur se produit en cas de demande de déverrouillage sur une instance non verrouillée sur le serveur.

- INVALID_CHANGE

  Cette erreur se produit lorsque l'instance à modifier n'existe pas dans le cache local.

- INVALID_CREATION

  Cette erreur se produit en cas de création d'une instance qui existe déjà dans le cache local.

- INVALID_DELETION

  Cette erreur se produit lorsque l'instance à supprimer n'existe pas dans le cache local.

- INVALID_INITIALIZATION

  Cette erreur se produit dans le cas d'une tentative d'initialisation d'une instance déjà connue du cache local quel que soit son statut (READ, CREATED, MODIFIED ; DELETED).

- INVALID_INSTANCE

  Cette erreur se produit lorsqu'une clé primaire de l'instance courante n'est pas valide.

- LENGTH_ERROR

  Cette erreur se produit quand le contenu d'une rubrique de l'instance courante a une taille supérieure à la taille autorisée pour la rubrique.

- LOCK_SERVICE_ALREADY_REQUESTED

  Cette erreur se produit lors d'une tentative de vérouillage d'un enregistrement alors que celui-ci est déjà vérouillée dans une requête.

- NO_SERVER_RESPONSE_REQUESTED

  Cette erreur se produit lorsque l'action utilisateur est envoyée au serveur car aucune réponse n'est attendue.

- PARENT_INSTANCE_MISSING

Cette erreur se produit en cas de sélection d'une instance de noeud dépendant alors que l'instance supérieure est inexistante.

- READ_SERVICE_ALREADY_REQUESTED

  Cette erreur se produit lors d'une tentative d'action de lecture alors que celle-ci a déjà été demandée.

- REFERENCE_USER_CONTEXT_LENGTH_ERROR

  Cette erreur se produit quand la taille du contenu d'une Rubrique du buffer utilisateur associé à un nœud référence est supérieure à la taille autorisée pour la Rubrique.

- REFERING_INSTANCE_MISSING

  Cette erreur se produit quand la méthode **`transferReference`** ne trouve pas d'instance dans le **`detail`** du noeud référençant.

- REQUEST_ALREADY_EXISTS

  Cette erreur se produit lors d'une tentative de création d'une requête externe alors qu'elle existe déjà.

- REQUEST_BAD_APPLICATION

  Cette erreur se produit lorsqu'un noeud essaie de se connecter à une une demande n'appartenant pas à la même application C/S.

- REQUEST_BAD_USER_BUFFER

  Cette erreur se produit lors d'une tentative de connexion à une demande dont le buffer est différent.

- REQUEST_NOT_ACTIVE

  Cette erreur se produit lorsque l'on essaie d'agir sur une requête non activée.

- REQUEST_TOO_LARGE

  Cette erreur se produit lorsque l'on essaie d'ajouter un service à une requête qui contient déjà le nombre maximum de requêtes (9999 services).

- SERVER_UPDATE_REQUIRED

  Cette erreur se produit lorsqu'une méthode est appliquée à une instance alors que l'instance supérieure créée localement n'existe pas encore dans la base. Une mise à jour serveur préalable de l'instance supérieure est requise.

- SUBSCHEMA_ERROR

  Cette erreur se produit lorsqu'une rubrique de l'instance courante est modifiée alors qu'elle n'a pas été renseignée pour cette instance à l'issue d'un accès serveur paramétré avec un sous-schéma.

- UNKNOWN CONTEXT

  Cette erreur se produit lorsque le contexte est inconnu.

- UNKNOWN_INSTANCE

  Cette erreur se produit en cas de sélection d'une instance inconnue du cache local.

- VALUE_ERROR

  Cette erreur se produit quand le contenu d'une rubrique de l'instance courante n'est pas valide.

- VALUE_REQUIRED

  Cette erreur se produit quand le contenu d'une rubrique de l'instance courante, pourtant obligtoire, est considéré comme étant inexistant.

Erreurs serveur

Vous devez connaître la clé d'erreur si vous souhaitez afficher des libellés d'erreur personnalisés dans le Client.

Pour des informations sur la manière dont sont déterminées les clés d'accès aux libellés locaux des erreurs serveur, reportez-vous au manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies*.

Col 1-3	Col 4-9	Col 10-13	Col 14-19	Col 20	Col 21	Col 22-25	
lib	ser	seg			E	DUPL	Création à tort
lib	ser	seg			E	NFND	Modif/annul à tort
lib	ser				E	Code erreur	Erreur utilisateur
lib	ser	vue	rub	2	E		Rubrique obligatoire
lib	ser	vue	rub	5	E		Erreur de valeur
lib	ser		LOCKED		E		Déjà verrouillée
lib	ser		NTLOCK		E		Déverrouillage à tort

*Légend*e :

ebib = code bibliothèque vue = code Vue Logique ser = code serveur

rub = code Rubrique seg = code segment d'accès physique

E = Exception (Erreur serveur)

Erreurs système

Vous devez connaître la clé d'erreur si vous souhaitez afficher des libellés d'erreur personnalisés dans le Client.

Pour des informations sur la manière dont sont déterminées les clés d'accès aux libellés locales des erreurs système, reportez-vous au manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies*.

Erreurs système reçues du Composant Elémentaire

Col 1-3	Col 4-9	Col 10-13	Col 14-19	Col 20	Col 21	Col 22-25	
bib	ser		LKABSC		S		Timestamp non défini pour verrouillage
bib	ser	vue	ACCESS		S		Erreur d'accès données
bib	ser	STRU			S		Erreur structure vue
bib	ser	VERS			S		Erreur version
bib	ser	VIEW			S		Vue inconnue
bib	ser	SERV			S		Service inconnu
bib	ser	LTH			S		Longueur vue erronée
			MISPCV		S		Déphasage composants

Légende :

bib = code bibliothèque vue = code Vue Logique ser = code serveur

rub = code Rubrique seg = code segment d'accès physique

S = Erreur système

L'erreur de type **`Service inconnu`** apparaît lorsque le service demandé par la Proxy n'est pas reconnu par le Composant Elémentaire.

L'erreur de type **`Longueur de vue erronée`** apparaît lorsqu'il y a un changement de format dans une Vue Logique associée à la Proxy et que celle-ci n'a pas été regénérée.

Pour résoudre ce problème, vous devez regénérer la Proxy.

L'erreur de type **`Déphasage composants`** survient lorsqu'il y a un déphasage entre les composants Client et Serveur.

Pour résoudre ce problème, vous devez regénérer la Proxy.

Erreurs système reçues du Moniteur de Communication

Col 1-3	Col 4-9	Col 10-13	Col 21	
bib	mon	LSRV	S	Erreur de longueur du message reçu
bib	mon	NSPA	S	Erreur de structure sur les paramètres du service suivant
bib	mon	PCOD	S	Erreur de structure du paramètre "code de service"
bib	mon	PCVF	S	Erreur de structure du message en provenance du client
bib	mon	PCVS	S	Erreur de structure d'une demande de service en provenance du client
bib	mon	PNOD	S	Code du Dossier inconnu du Moniteur de Communication
bib	mon	PNUM	S	Erreur de structure du paramètre "numéro de service"
bib	mon	TAND	S	Erreur Tandem/Pathway
bib	mon	WF00	S	Erreur d'accès au fichier de travail ou à la base de données (open/close)

Légende bib = code bibliothèque mon = code Moniteur de Communication S = erreur système

Erreurs système reçues du Gestionnaire de Services

Ces erreurs sont, pour la plupart, des erreurs internes que vous résoudrez en contactant le support VisualAge Pacbase.

Col 1-3	Col 4-9	Col 10-13	Col 21	
lib	serv	BUF1	S	Erreur de structure du buffer utilisateur
lib	serv	CHK1	S	Paramètre "Check Option" absent
lib	serv	CHK2	S	Erreur de longueur sur le paramètre "Check Option"
lib	serv	CP01	S	Erreur de structure d'un champ de "Selection Criteria" en provenance du Composant Elémentaire
lib	serv	CP02	S	Erreur de structure d'un champ d'une instance de Vue Logique en provenance du Composant Elémentaire
lib	serv	DANA	S	Erreur de structure du code de champ erroné dans le message d'erreur utilisateur
lib	serv	DOS1	S	Paramètre "nom du Dossier" absent
lib	serv	DOS2	S	Erreur de longueur du paramètre "nom du Dossier"
lib	serv	ERK1	S	Erreur de structure de la clé de message d'erreur utilisateur
lib	serv	ERKY	S	Erreur de structure de la clé du "Selt Message" en provenance du Composant Elémentaire
lib	serv	ERL1	S	Erreur de structure du label de message d'erreur utilisateur
lib	serv	ERLA	S	Erreur de structure du label du "Selt Message" en provenance du Composant Elémentaire
lib	serv	EXT1	S	Erreur de structure du paramètre "méthode d'extraction"
lib	serv	EXT2	S	Méthode d'extraction inconnue dans le Dossier
lib	serv	FRRE	S	Erreur d'accès en lecture au fichier de travail avant mise à jour
lib	serv	FRRD	S	Erreur d'accès en lecture au fichier de travail
lib	serv	FRW2	S	Erreur d'écriture du dernier enregistrement du fichier de travail
lib	serv	FRWR	S	Erreur d'accès en écriture au fichier de travail
lib	serv	FRRW	S	Erreur d'accès en mise à jour au fichier de travail
lib	serv	LCK1	S	Paramètre "Lock Timestamp" absent
lib	serv	LCK2	S	Erreur de longueur sur le paramètre "Lock Timestamp"
lib	serv	LNG1	S	Erreur de conversion de la longueur du service sur une requête multi-messages
lib	serv	LNG2	S	Erreur de conversion de la longueur du service sur une requête mono-message
lib	serv	LTH	S	Erreur du paramètre "longueur" dans le Composant Elémentaire élémentaire
lib	serv	NOCP	S	Erreur de structure du paramètre "nombre d'occurrences"
lib	serv	NOC1	S	Erreur de longueur sur le paramètre "nombre d'occurrences"
lib	serv	NOC2	S	Erreur de conversion du paramètre "nombre d'occurrences"
lib	serv	NOD1	S	Paramètre "nom du noeud" absent
lib	serv	NOD2	S	Erreur de longueur du paramètre "nom du noeud"
lib	serv	NOD3	S	Nom du noeud inconnu dans le Dossier
lib	serv	NOS1	S	Erreur de structure du paramètre "numéro de service"
lib	serv	NOS2	S	Erreur de conversion du paramètre "numéro de service"
lib	serv	NUVE	S	Erreur du paramètre "numéro de version" dans le Composant Elémentaire élémentaire
lib	serv	OCNB	S	Erreur de structure, dans le message d'erreur utilisateur, du code champ sur lequel porte l'erreur
lib	serv	PC01	S	Erreur de conversion du paramètre "critère de sélection"
lib	serv	PC02	S	Erreur de conversion d'un champ du buffer utilisateur

Col 1-3	Col 4-9	Col 10-13	Col 21	
lib	serv	PC03	S	Erreur de conversion d'un champ d'une instance de Vue Logique en provenance du client
lib	serv	PC05	S	Erreur de conversion d'un champ de "Selection Criteria" en provenance du client
lib	serv	PC06	S	Erreur de conversion d'un champ d'une instance de Vue Logique à mettre à jour
lib	serv	PCV1	S	Erreur de structure du paramètre "Selection Criteria"
lib	serv	PCV3	S	Erreur de structure d'un champ d'une instance de Vue Logique en provenance du client
lib	serv	PCV4	S	Erreur de structure du paramètre "Selection Criteria" en provenance du client
lib	serv	PCV5	S	Erreur de structure du code action d'une instance de Vue Logique à mettre à jour
lib	serv	PCV6	S	Erreur de structure d'un champ d'une instance de Vue Logique à mettre à jour en provenance du client
lib	serv	PCV7	S	Erreur de structure du vecteur de présence d'un champ d'une instance de Vue Logique à mettre à jour
lib	serv	PCVF	S	Erreur de structure du message en provenance du client
lib	serv	PCVS	S	Erreur de structure d'une demande de service en provenance du client
lib	serv	PILO	S	Erreur d'accès à l'enregistrement pilote du fichier de travail
lib	serv	RFH1	S	Paramètre "Refresh Option" absent
lib	serv	RFH2	S	Erreur de longueur sur le paramètre"Refresh Option"
lib	serv	SCH1	S	Erreur de structure du paramètre "code sous-schéma"
lib	serv	SCH2	S	Code sous-schéma inconnu dans le Dossier
lib	serv	SERV	S	Erreur du paramètre "code opération" dans le Composant Elémentaire
lib	serv	SRV1	S	Service non trouvé dans le fichier de travail
lib	serv	SRV2	S	Code service inconnu dans le Dossier
lib	serv	SRV3	S	Erreur de structure du paramètre "code service"
lib	serv	STRU	S	Erreur du paramètre “structure” dans le Composant Elémentaire
lib	ges	TAND	S	Erreur Tandem/Pathway
lib	serv	TYNO	S	Service non autorisé sur le noeud
lib	serv	USR1	S	Paramètre "service utilisateur" absent
lib	serv	USR2	S	Erreur de longueur sur le paramètre"service utilisateur"
lib	serv	USR3	S	Service Utilisateur inconnu dans le Dossier
lib	serv	VER2	S	Erreur de longueur du pramètre "numéro de version"
lib	serv	VIEW	S	Erreur du paramètre "code de la Vue Logique" dans le Composant Elémentaire élémentaire
lib	serv	WF00	S	Erreur d’accès au fichier de travail

Légende bib = code bibliothèque ges = code Gestionnaire de services S = erreur système

Erreurs de communication

Si un message d'erreur de communication apparaît, vous devez d'abord en informer le responsable de la communication car la cause peut être un encombrement de ligne, une ligne défectueuse, un serveur indisponible…

Trois messages d'erreur de communication sont susceptibles de s'afficher :

- `Open server error`
- `Call server error`
- `Close server error`

Chapitre 4 : Développement d'un Client VisualAge Java

Après avoir généré les objets Proxy puis les avoir importés dans la station VisualAge, il ne vous reste plus qu'à les intégrer dans l'application graphique.

Ce chapitre détaille pas à pas le déploiement d'un Client (Applet ou standalone) dans VisualAge for Java. Cette description comprend : l'insertion des objets Proxy avec les liens de programmation impliquant les actions, attributs et événements, ainsi que la gestion des erreurs, la gestion de la communication et enfin le test et le déploiement de l'application.

Exemple d'une applet

Introduction

Cet exemple présente une applet VisualAge Java, c'est-à-dire un programme destiné à être exécuté dans un browser.

Cette applet est développée successivement avec la version 1 de VisualAge Java puis avec la version 2 de VisualAge Java, ce qui permet d'illustrer les spécificités de développement de chaque version.

Afin de faciliter le développement et la réutilisabilité des clients mis en oeuvre avec VisualAge, il vous est conseillé de créer un projet pour chaque application fonctionnelle. Le projet doit comporter un ou plusieurs packages de classes graphiques et un package de classes générées.

Les composants Proxy insérés dans l'applet ont été générés avec les options **`Générer des Beans`** et :

- **`Utiliser les classes IBM Enterprise Access Builder`** pour la version 1.
- **`Utiliser Swing`** pour la version 2.

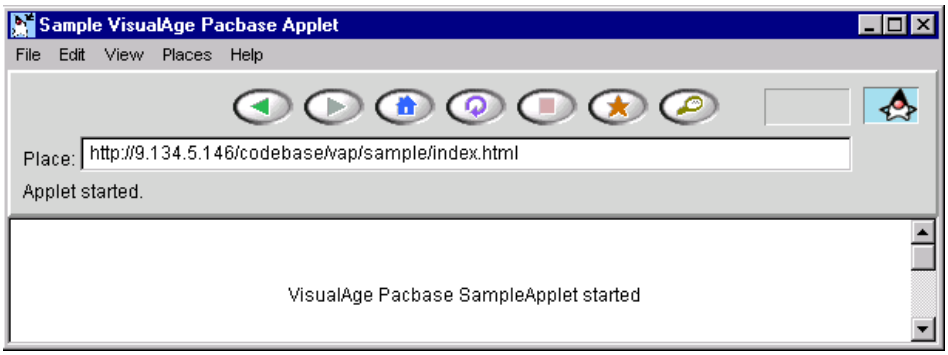
L'exemple manipule trois Proxy Elémentaires de la PVD :

- la Proxy Racine correspondant au noeud **`Clients`** qui gère les clients dans le système d'information décrit par le Dossier.
- la Proxy Dépendante correspondant au noeud **`Commandes`** qui gère les commandes dans le système d'information décrit par le Dossier.
- la Proxy Dépendante correspondant au noeud **`Lignes de Commandes`** qui gère les lignes de commandes dans le système d'information décrit par le Dossier.

Présentation de l'interface utilisateur

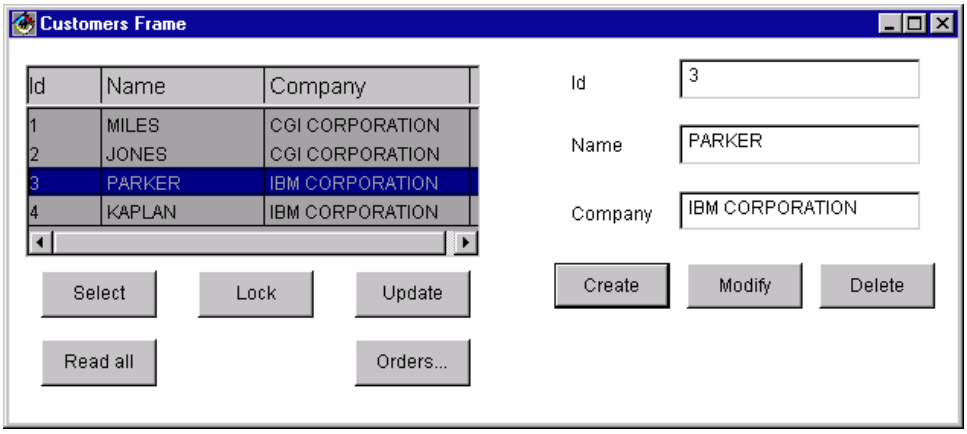
L'interface graphique utilisateur développée comprend l'applet elle-même et deux fenêtres.

- **L' applet**



L'applet n'a pas de fonctionnalité dans l'application utilisateur. Elle constitue un point de départ et permet à l'ensemble de l'application d'être accessible par le Web.

- **La fenêtre Customers**

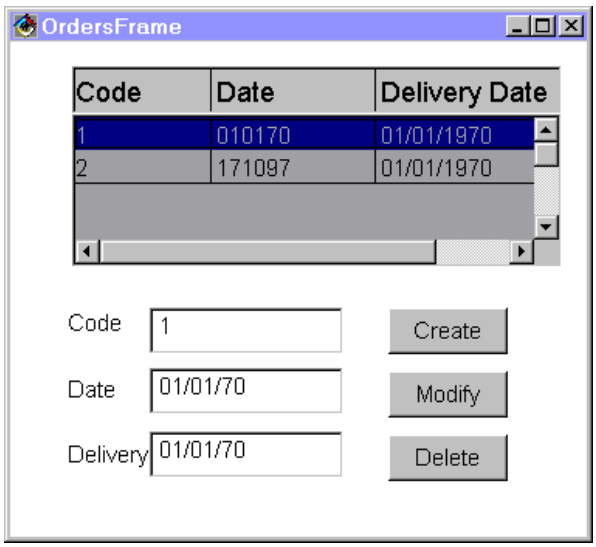


Cette fenêtre s'ouvre automatiquement lors du lancement de l'applet. Ses fonctionnalités sont les suivantes :

- Le bouton **`Select`** permet l'affichage de la liste des clients.
- Le bouton **`Read all`** permet, à partir de cette fenêtre, de demander la lecture des commandes du client sélectionné.
- Parce qu'un service de verrouillage a été spécifié dans le Dossier, l'utilisateur doit, avant toute mise à jour, cliquer sur le bouton **`Lock`** pour verrouiller l'instance sélectionnée dans la liste avant toute mise à jour.
- Le bouton **`Update`** permet de mettre à jour la base de données
- Le bouton **`Orders...`** permet la navigation vers la fenêtre de gestion des commandes.
- Le bouton **`Modify`** permet la modification d'un client à partir de la fiche, après verrouillage de cette instance.

  Après sélection d'un client dans la liste, l'utilisateur doit cliquer sur **`Lock`** pour s'approprier momentanément ce client. Il doit ensuite cliquer sur **`Modify`** après saisie des modifications souhaitées.

- **La fenêtre Orders**



La fenêtre **Orders** s'affiche lorsque l'utilisateur clique sur le bouton **Orders...** de la fenêtre **Customers**.

Pour le client sélectionné dans la fiche de la fenêtre **Customers**, cette fenêtre permet :

- de consulter la liste de ses commandes.
- de sélectionner une commande dans la liste et de l'afficher dans la fiche.
- de créer, modifier, supprimer des commandes.

Développement de l'interface utilisateur avec VisualAge Java V1

Le développement de l'interface graphique utilisateur comprend la programmation de l'applet et la construction des fenêtres **Customers** et **Orders**.

Dans la description de ces différentes étapes, l'utilisation des différents outils et fonctionnalités de VisualAge n'est pas détaillée. S'ils ne vous sont pas familiers, reportez-vous à la documentation appropriée.

Tout en gardant autant que possible une démarche séquentielle dans le développement, nous divisons, le cas échéant, la programmation des fenêtres en plusieurs parties selon des groupes cohérents de fonctionnalités.

A l'issue du traitement de chaque partie, nous présentons le Composition Editor tel qu'il doit apparaître en cachant éventuellement les liens précédemment développés pour mieux isoler ceux nouvellement créés.

Mise en place de l'exemple et création de l'applet

- Dans un projet, créez un package **vap.sample** destiné à contenir tous les composants de l'application.
- Dans ce package, créez une applet **SampleApplet**. Dans la fenêtre **SmartGuide - Create Applet**, indiquez l'option **Design the applet visually**, de manière à ce que le browser de classes s'ouvre directement sur l'onglet **Visual Composition**.

- **Affichage du texte**

Dans le Visual Composition Editor, effectuez les opérations suivantes :

- Redimensionnez l'applet.
- Dans l'applet, posez un bean **Label** (catégorie **Data Entry**). Dans la fenêtre **Properties** du bean, dans le champ **text**, entrez **VisualAge Pacbase SampleApplet started**.

- **Intégration de la Proxy Racine**

Pour poser la Proxy Racine sur la Free Form Surface, effectuez les opérations suivantes :

- Dans le menu **Options**, sélectionnez le choix **Add Bean**.
- Dans la fenêtre **Add Bean** qui s'ouvre alors, vous devez entrer **com.ibm.vap.generated.proxies.CustomerProxyLv** en le saisissant dans la zone correspondante (ou en utilisant le bouton **Browse**....).

  Le nom de classe, dans notre exemple **CustomerProxyLv**, doit être précédé du nom du package choisi lors de la génération, dans notre exemple, **com.ibm.vap.generated.proxies**.

- **Définition de la communication avec la Gateway**

A ce sujet, voir également le sous-chapitre. Gestion de la communication

Dans cet exemple, on suppose que la Proxy Vue de Dossier communique avec son host, c'est-à-dire avec le serveur HTTP où l'applet est stockée. Pour que cette communication puisse s'effectuer, procédez de la manière suivante :

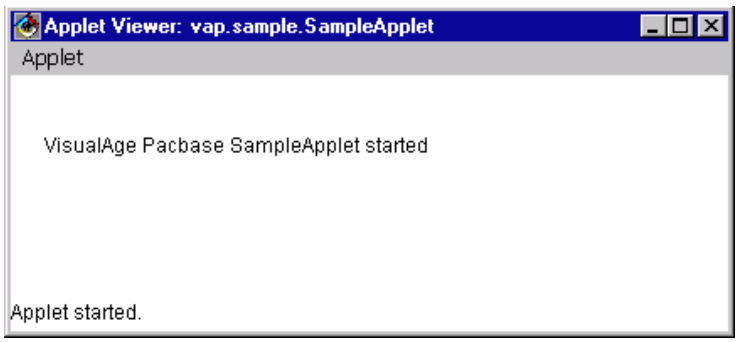
- Depuis un endroit quelconque de la Free Form Surface, accédez au menu contextuel.
- Sélectionnez **Tear-Off Property**, puis **codeBase (URL)**.
- Cliquez sur la Free Form Surface. Un bean variable **codeBase1** s'affiche alors.
- Accédez au menu contextuel du bean **codeBase1**, sélectionnez **Connect**, puis **All Features**.....
- Dans la fenêtre **Start connection from**, sélectionnez la propriété **host**. Un lien en pointillé s'affiche.
- Cliquez sur la Proxy Racine. Le menu contextuel s'affiche.
- Sélectionnez **All Features**.... La fenêtre **Connect property named:** s'ouvre.
- Affichez toutes les propriétés disponibles pour la Proxy Racine en cochant **Show expert features**.
- Sélectionnez la propriété **Host**, puis cliquez sur **OK**.

La propriété **host** de **codeBase1** est maintenant connectée à la propriété **Host** de la Proxy Racine.

Ces connexions sont équivalentes à la ligne de code suivante dans l'applet :

```
getCustomerProxyLv1().setHost(getCodeBase().getHost());
```

Vous pouvez maintenant tester votre applet. La fenêtre **`Applet Viewer`** s'affiche :



- **Programmation de l'ouverture de la fenêtre CustomersFrame depuis l'Applet**

Cette étape se compose de deux parties, toutes deux à réaliser une fois les opérations décrites dans les paragraphe *Création de la fenêtre et intégration de la PVD dans le Composition Editor*, *Promotion de la Proxy Racine* effectuées :

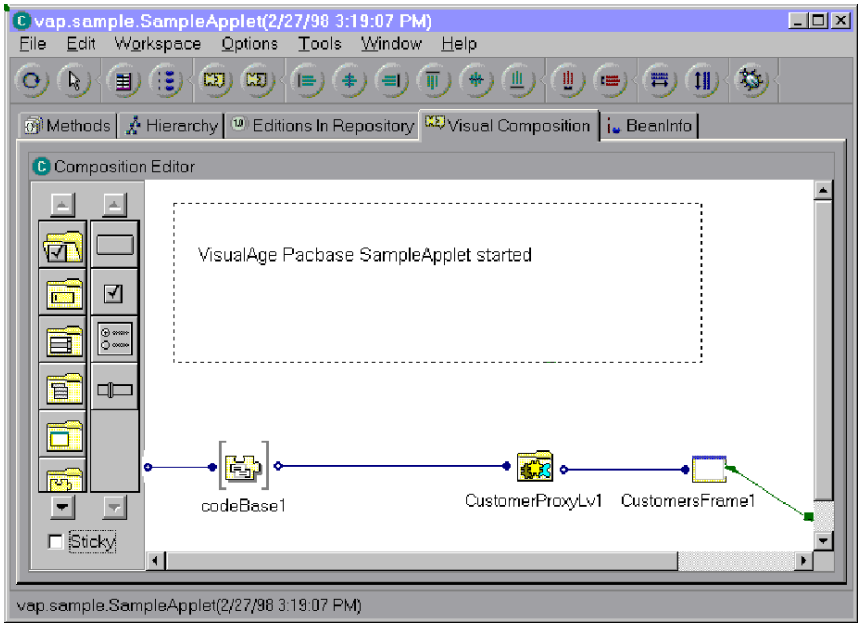
- Appel de la Proxy Racine dans l'applet
  - Dans le Composition Editor de l'applet, posez un bean constant de type **`CustomersFrame`** sur la Free Form Surface.
  - Connectez la propriété **`this`** de la Proxy Racine à la propriété **`customerProxyLv1This`** du bean **`CustomersFrame`**.
- Programmation du lancement de la fenêtre depuis l'applet

  Il s'agit maintenant de provoquer l'affichage automatique de la fenêtre **`CustomersFrame`** immédiatement après le lancement de l'applet.

  Pour cela, connectez l'événement **`componentShown`** de l'applet à la méthode **`show`** du bean **`CustomersFrame`**.

- **Résultat dans le Composition Editor**

L'applet est maintenant terminée. Voici le Composition Editor tel qu'il doit se présenter à ce stade :



Développement de la fenêtre Customers

Maquettage de rows et detail

Cette étape, après l'insertion de la Proxy Racine et sa promotion, détaille le maquettage de ses propriétés **rows** et **detail**.

- **Création de la fenêtre et intégration de la PVD dans le Composition Editor**

Créer une fenêtre de gestion des clients consiste à créer la classe correspondante dans le workbench.

- Dans le workbench, créez une classe **CustomersFrame** dans le package **vap.sample**.
- Dans la fenêtre **SmartGuide –Create Class or Interface** qui s'ouvre alors :
  - entrez **java.awt.Frame** dans le champ **Superclass** : vous spécifiez ainsi que la classe **CustomersFrame** hérite de la classe **java.awt.Frame**.
  - activez l'option **Design the class visually** de manière à ce que le browser de classes s'ouvre directement sur l'onglet **Visual Composition**.
- Pour intégrer la Proxy Vue de Dossier dans le Visual Composition Editor, effectuez les étapes décrites au paragraphe *Création de la fenêtre et intégration de la PVD dans le Composition Editor, Promotion de la Proxy Racine*, à une différence près : dans la fenêtre **Add Bean**, sélectionnez l'option **Variable** dans la zone **Bean Type**.

  ☞ La Proxy Racine déposée ici est en effet de type variable car elle doit représenter la même instance de Proxy que celle appelée dans l'applet.

- **Promotion de la Proxy Racine**

L'objectif est maintenant d'assurer l'égalité permanente de cette Proxy variable avec la Proxy constante instanciée dans l'applet. Pour cela, il est nécessaire que la Proxy variable soit visible depuis l'extérieur de la classe **CustomersFrame**, pour qu'une connexion propriété - propriété entre la Proxy constante et la Proxy variable puisse être réalisée dans l'applet.

Effectuez les opérations suivantes :

- Depuis le menu contextuel de la Proxy, choisissez **Promote Bean feature**.
- Dans la colonne **Property** choisissez **this** puis cliquez sur **Promote**.

  La classe **CustomersFrame** a maintenant une propriété publique, en lecture/écriture, nommée **customerProxyLv1This** de type Proxy Racine.
- Sauvegardez la fenêtre (menu **File**, choix **Save Bean** ou **CTRL-F2**).

- **Maquettage de la propriété Rows**

☞ Le maquettage proposé utilise un container EAB ; il n'est donc possible que si vous avez coché l'option **Utiliser les classes IBM Enterprise Access Builder** lors de la génération.

Cette étape inclut les opérations suivantes :

- A l'intérieur du bean **CustomersFrame**, posez un bean **Multi Column List Box**, container de la catégorie **Access Enterprise**. Redimensionnez ce bean.
- Vous devez maintenant maquetter dans le container les colonnes correspondant aux Rubriques associées au nœud racine.

  Ouvrez la fenêtre **Properties** du container. A partir du champ **columns** de cette fenêtre, ajoutez une colonne par Rubrique de la Vue Logique dans l'ordre dans lequel les Rubriques sont appelées dans le Référentiel. Vous pouvez maquetter autant de colonnes que la **DataDescription** de la Proxy a de propriétés ou ne maquetter que les premières.

  ☞ Les colonnes à maquetter correspondent aux valeurs retournées par la méthode **getAttributeStrings()** de la **DataDescription**.
- Une fois les colonnes créées, connectez la propriété **IRows** de la Proxy à la propriété **elements** du container.

  ☞ **IRows** est identique à **Rows** mais il renvoie une instance de la classe **IVector** qui, contrairement à la classe **DataDescriptionVector** retournée par **Rows**, est compatible avec la propriété **elements** du bean **Multi Column List Box**.

  Cette opération équivaut réellement au maquettage de la propriété **Rows** de la Proxy.

- **Maquettage de la propriété Detail**

Cette étape nécessite les opérations suivantes :

- Depuis la Proxy Racine, effectuez un Tear-Off de la propriété **detail**.
- Insérez un bean **label** par propriété à maquetter.
- Vous devez maintenant maquetter un champ de saisie par propriété à afficher.

  Pour cela, utilisez les beans **Pacbase Text Field**, **Pacbase Integer Field**, **Pacbase Decimal Field**, **Pacbase Date Field** et **Pacbase Time Field** fournis par Pacbench C/S lors de l'importation du runtime.

  Dans la palette, ces beans se trouvent dans la catégorie **VisualAge Pacbase**.

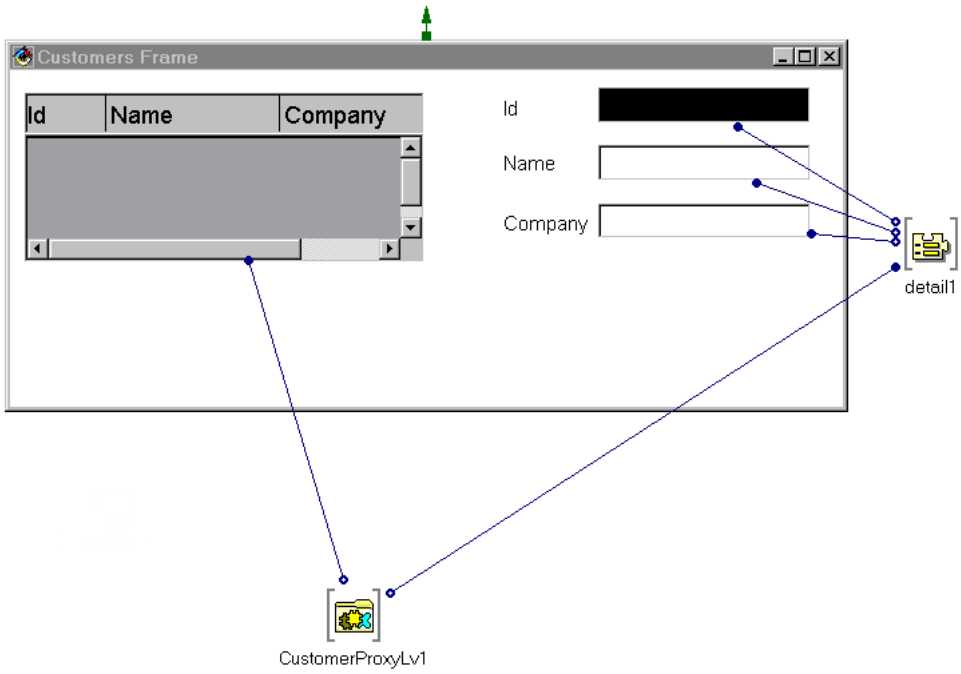
  Vous pouvez également insérer ces beans en sélectionnant dans le menu **Options** le choix **Add bean** ; entrez le nom du bean précédé du nom complet du package. Par exemple : **com.ibm.vap.beans.PacbaseDateField**.

  ☞ Pour des détails complémentaires sur ces beans, reportez-vous à la documentation en ligne des classes génériques.
- Insérez un bean de type approprié pour chaque propriété à maquetter.
- Ensuite, pour chaque champ maquetté, connectez la propriété du bean **detail** à sa propriété correspondante de type **String**, **Int**, **Decimal**, **Date** ou **Time**.

Dans notre exemple, pour le champ de saisie correspondant au numéro du client, il faut connecter la propriété **Customer Number** du bean **detail** à la propriété **Int** du champ.

- **Résultat dans le Composition Editor**

A l'issue de ces étapes, le Composition Editor se présente comme ci-dessous :



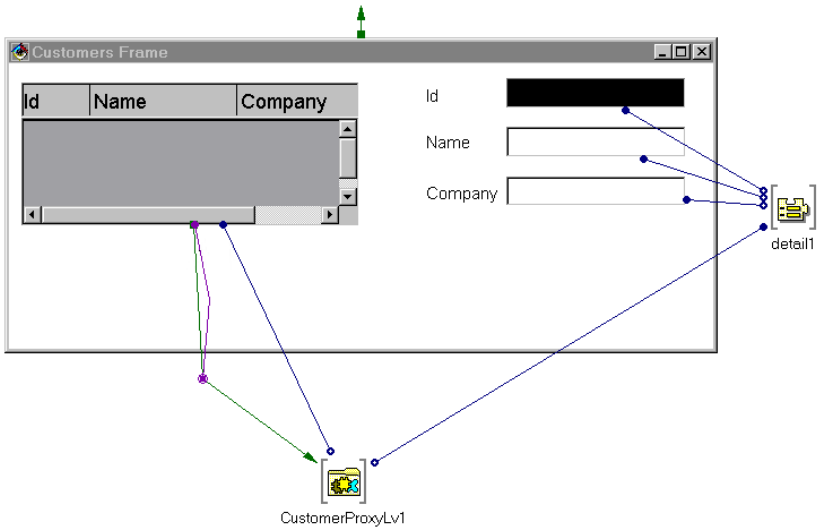
Sélection d'une instance dans rows et transfert dans detail

L'objectif est maintenant de programmer le transfert de l'instance sélectionnée par l'utilisateur depuis le container dans la propriété **detail** de la Proxy.

Pour cela, effectuez les étapes suivantes :

- Connectez l'événement **itemStateChanged** de la MultiColumnListBox à la méthode **Get Detail From DataDescription** de la Proxy Racine.
- Le lien apparaît en pointillés puisque la méthode **Get Detail From DataDescription** nécessite un paramètre.
- Pour spécifier ce paramètre, connectez la propriété **selectedObject** de la MultiColumnListBox au paramètre de type data du lien, ici **customerData**. Ce paramètre est accessible lorsque le pointeur se trouve au milieu de la connexion.

Voici le Composition Editor tel qu'il doit se présenter à l'issue de cette étape :



Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders

- **Déclenchement des méthodes de la Proxy**

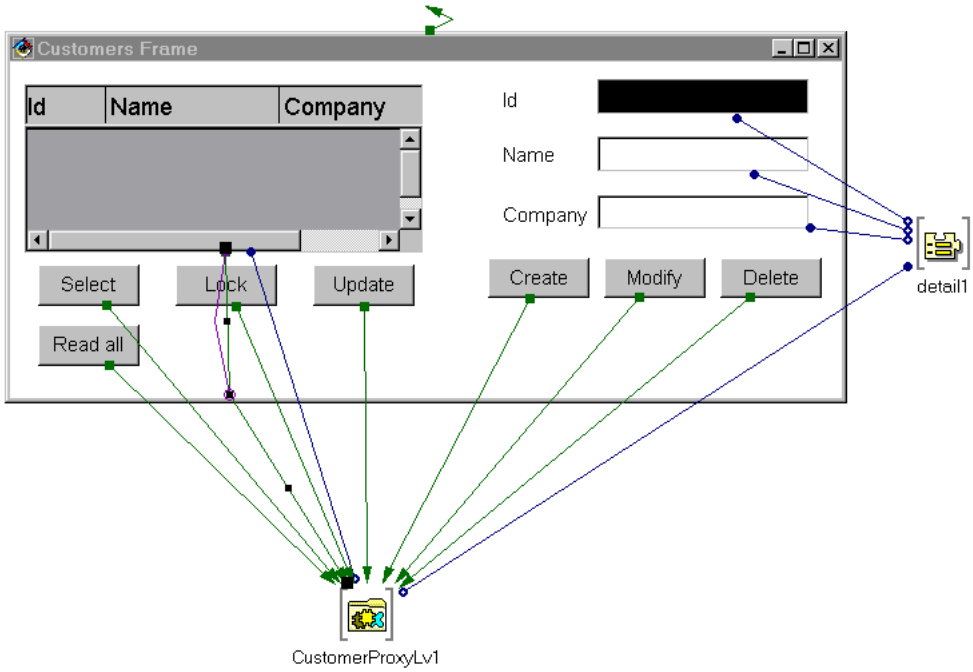
Pour chaque méthode que vous souhaitez programmer, posez un bouton-poussoir dans la fenêtre. Le libellé est modifiable dans la fenêtre **`Properties`** du bouton.

Pour déclencher une méthode, connectez l'événement **`actionPerformed`** d'un bouton donné à la méthode adéquate de la Proxy. Au besoin, si la méthode souhaitée n'est pas disponible, cochez l'option **`Show expert features`**.

Par exemple :

- pour programmer le déclenchement du bouton **`Update`**, il faut connecter le bouton, via l'événement **`actionPerformed`** à la Proxy, via la méthode **`Update Folder`**.
- pour programmer le déclenchement du bouton **`Read all`**, il faut connecter le bouton, via l'événement **`actionPerformed`** à la Proxy, via la méthode **`Read All Children From Detail`**.

A l'issue de ces opérations, le Composition Editor doit se présenter comme ci-dessous :

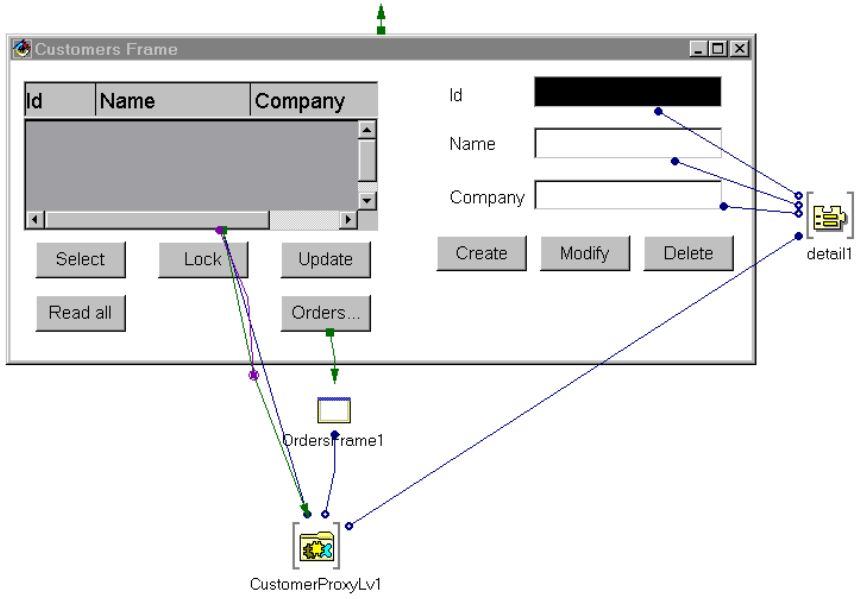


- **Gestion de la navigation**

Cette étape est à réaliser après la promotion de la Proxy Dépendante Orders.

- Sur la Free Form Surface, posez un bean constant **`OrdersFrame`**.
- Connectez la propriété **`Order Proxy`** de la Proxy Racine à la propriété **`orderProxyLv1This`** du bean **`OrdersFrame`**. Cette connexion assure le lien des deux objets Proxy entre les deux fenêtres.
- Posez un bouton **`Orders...`** dans le bean **`CustomersFrame`**.
- Connectez l'événement **`actionPerformed`** du bouton à la méthode **`show`** du bean **`OrdersFrame`**.

Voici le Composition Editor tel qu'il doit se présenter à l'issue de cette étape :



Pour une meilleure lisibilité, les liens entre les autres boutons et la Proxy Racine sont cachés.

Développement de la fenêtre Orders

Cette fenêtre présente les informations relatives aux commandes. Elle s'affiche lorsque l'utilisateur clique sur le bouton **Orders...** depuis la fenêtre **Customers**.

Le container affiche automatiquement les commandes du client sélectionné dans la fenêtre **Customers** si l'utilisateur clique sur **Read all** avant de cliquer sur **Orders...**.

Le développement de cette fenêtre qui manipule la Proxy Dépendante **OrderProxyLv** comporte les étapes décrites ci-dessous.

Création de la fenêtre Orders

Dans le workbench, de la même manière que pour la création de la fenêtre **Customers**, créez une une classe **OrdersFrame** héritant de **java.awt.Frame** dans le package **vap.sample**.

Intégration et promotion de la Proxy Dépendante

Dans l'onglet Visual Composition, posez un bean variable **OrderProxyLv** sur la Free Form Surface.

Pour plus de détails sur l'intégration de la Proxy, reportez-vous au paragraphe Développement de la fenêtre Customers.

Il faut maintenant promouvoir cette Proxy variable de manière à la rendre visible de l'extérieur.

Pour cela, effectuez les opérations suivantes :

- Depuis le menu contextuel de la Proxy, choisissez **Promote Bean feature**.
- Dans la colonne **Property** choisissez **this** puis cliquez sur **Promote**.

  La classe **OrdersFrame** a maintenant une propriété publique, en lecture/écriture, nommée **OrderProxyLv1This** de type Proxy Dépendante.
- Sauvegardez la fenêtre (menu **File**, choix **Save Bean** ou **CTRL-F2**).

Maquettage de rows et detail de la Proxy Dépendante

Les opérations à réaliser ici sont identiques à celles requises pour le maquettage des propriétés **rows** et **detail** de la Proxy Racine.

Le même container EAB est utilisé pour le maquettage de **rows**.

Pour plus de détails, reportez-vous au paragraphe *Développement de la fenêtre Customers*.

Sélection d'une instance dans rows et transfert dans detail

Les opérations à effectuer ici sont identiques à celles nécessaires pour les propriétés **rows** et **detail** de la Proxy Racine.

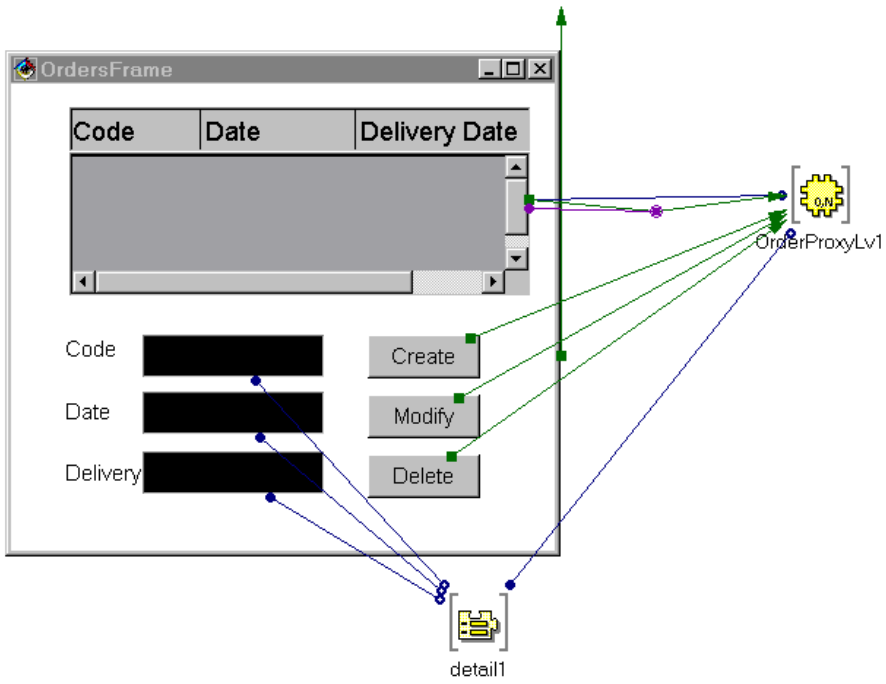
Pour plus de détails, reportez-vous au paragraphe *Développement de la fenêtre Customers*.

Déclenchement des méthodes de la Proxy Dépendante

Utilisez les mêmes principes que ceux décrits dans le paragraphe *Développement de la fenêtre Customers*.

Résultat dans le Composition Editor

Une fois toutes ces opérations effectuées, le Composition Editor se présente comme ci-dessous :



Développement de l'interface utilisateur avec VisualAge Java V2

Cette section détaille le développement, avec VisualAge Java V2, de l'application dont l'interface utilisateur est présentée à la section *Présentation de l'interface utilisateur*.

Les objets Proxy intégrés dans l'application ont été générés avec les options **`Generate Beans`** et **`Utiliser Swing`**. Tous les composants insérés sont des beans Swing.

Nous partons du principe que nous créons l'application depuis le début. Nous ne reprenons donc pas l'application développée avec VisualAge Java V1.

☞ Si vous avez déjà développé une application avec VisualAge Java V1, vous pouvez la sauvegarder et continuer à la développer avec VisualAge Java V2. Cependant, si vous voulez insérer des composants Swing, vous devez regénérer les objets Proxy avec l'option **`Utiliser Swing`** afin que ces objets Proxy communiquent correctement avec les beans swing.

⚠ VisualAge Java V2 ne reconnaît pas les containers EAB, qui sont des composants spécifiques de VisualAge Java V1. Si l'application développée avec la V1 inclut de tels composants, vous devez les remplacer par d'autres composants (composant swing JTable par exemple).

Mise en place de l'exemple et création de l'applet

- Dans un projet, créez un package **example.swing** destiné à contenir tous les composants de l'application.
- Dans ce package, créez une applet **SwingApplet**. Dans la fenêtre **SmartGuide – Create Applet**, sélectionnez, via le bouton **Browse**, **JApplet** dans la zone **SuperClass** et cochez l'option **Compose the applet visually**, de manière à ce que le browser de classes s'ouvre directement sur l'onglet **Visual Composition**.

- **Affichage du texte**

Dans le Visual Composition Editor, effectuez les opérations suivantes :

- Redimensionnez l'applet.
- Dans l'applet, posez un bean **JLabel**. Dans la fenêtre **Properties** du bean, dans le champ **text**, entrez **VisualAge Pacbase SwingSampleApplet started**.

- **Intégration de la Proxy Racine**

Pour poser la Proxy Racine sur la Free Form Surface, effectuez les opérations suivantes :

- Cliquez sur l'icône **Choose Bean** située au-dessus de la palette.
- Sélectionnez le type de bean **Class**.
- Avec le bouton Browse, sélectionnez le nom de la classe **CustomerProxyLv**.

- **Spécification de la communication avec la gateway**

A ce sujet, voir également le sous-chapitre, *Gestion de la communication*.

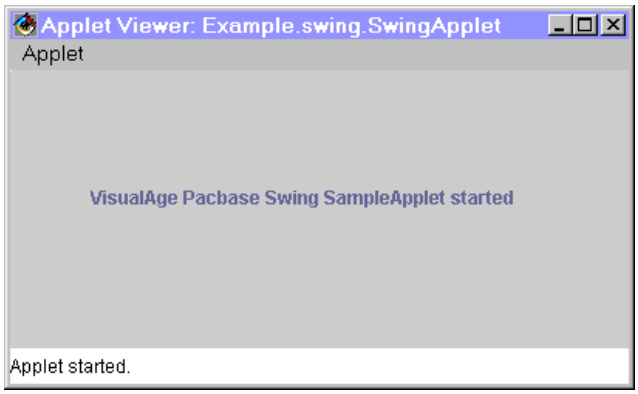
Dans cet exemple, on suppose que la Proxy Vue de Dossier communique avec son host, c'est-à-dire avec le serveur HTTP où l'applet est stockée. Pour que cette communication puisse s'effectuer, procédez de la manière suivante :

- Depuis un endroit quelconque de la Free Form Surface, accédez au menu contextuel.
- Sélectionnez **Tear-Off Property**, puis **codeBase (URL)**.
- Cliquez sur la Free Form Surface. Un bean variable **codeBase1** s'affiche alors.
- Accédez au menu contextuel du bean **codeBase1**, sélectionnez **Connect**, puis **Connectable Features**....
- Dans la fenêtre **Start connection from**, sélectionnez la propriété **host**. Un lien en pointillé s'affiche.
- Cliquez sur la Proxy Racine. Le menu contextuel s'affiche.
- Sélectionnez **Connectable Features**.... La fenêtre **End Connection to** s'ouvre.
- Affichez toutes les propriétés disponibles pour la Proxy Racine en cochant **Show expert features**.
- Sélectionnez la propriété **Host**, puis cliquez sur **OK**.

La propriété **host** de **codeBase1** est maintenant connectée à la propriété **Host** de la Proxy Racine.

☞ Ces connexions sont équivalentes à la ligne de code suivante dans l'applet : `getCustomerProxyLv1().setHost(getCodeBase().getHost());`

Vous pouvez maintenant tester votre applet. La fenêtre **Applet Viewer** s'affiche :



- **Programmation de l'ouverture de la fenêtre CustomersFrame depuis l'applet**

Cette étape se compose de deux parties, toutes deux à réaliser une fois les opérations décrites dans les paragraphe **Error! Reference source not found.** et **Promotion de la Proxy Racine** effectuées :

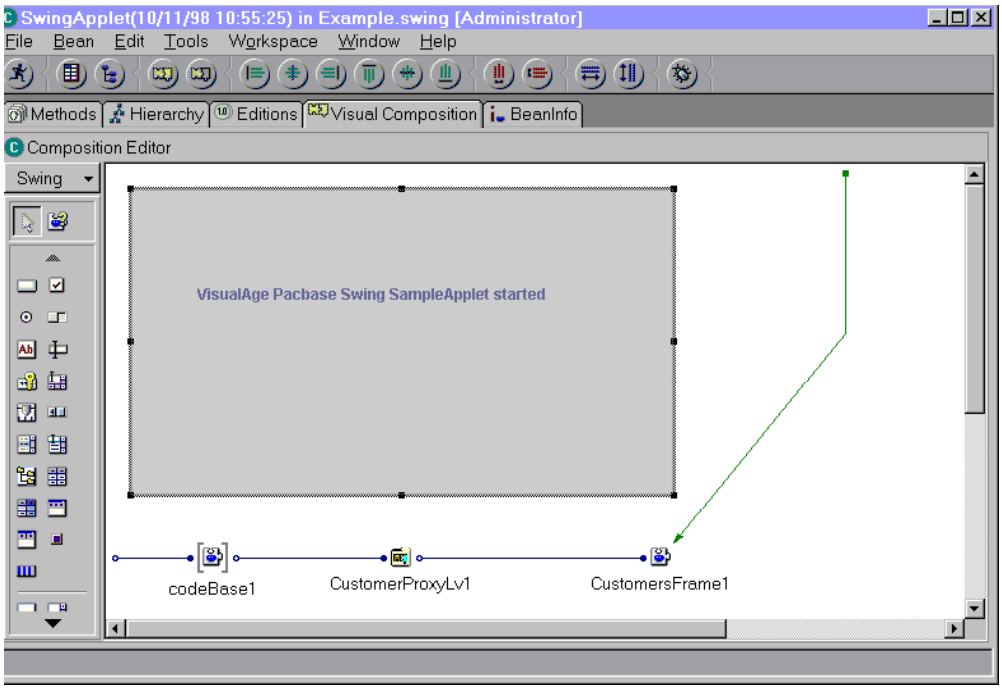
- Appel de la Proxy Racine dans l'applet
  - Dans le Composition Editor de l'applet, posez un bean constant de type **CustomersFrame** sur la Free Form Surface.
    Pour cela : sélectionnez l'icône **Choose Bean** au-dessus de la palette, choisissez le type de bean **Class** et sélectionnez, avec le bouton Browse, la classe **CustomersFrame**.
  - Connectez la propriété **this** de la Proxy Racine à la propriété **customerProxyLv1This** du bean **CustomersFrame**.
- Programmation du lancement de la fenêtre depuis l'applet

  Il s'agit maintenant de provoquer l'affichage automatique de la fenêtre **CustomersFrame** immédiatement après le lancement de l'applet.

  Pour cela, connectez l'événement **componentShown** de l'applet à la méthode **show** du bean **CustomersFrame**.

- **Résultat dans le Composition Editor**

L'applet est maintenant terminée. Voici le Composition Editor tel qu'il doit se présenter à ce stade :



Développement de la fenêtre Customers

Maquettage de rows et detail

Cette étape, après l'insertion de la Proxy Racine et sa promotion, détaille le maquettage de ses propriétés **`rows`** et **`detail`**.

- **Création de la fenêtre et intégration de la PVD dans le Composition Editor**

Créer une fenêtre de gestion des clients consiste à créer la classe correspondante dans le workbench.

- Dans le workbench, créez une classe **`CustomersFrame`** dans le package **`example.swing`**.
- Dans la fenêtre **`SmartGuide -Create Class or Interface`** qui s'ouvre alors :
  - sélectionnez **`JFrame`** avec le bouton **`Browse`** dans le champ **`Superclass`** : vous spécifiez ainsi que la classe **`CustomersFrame`** hérite de la classe **`com.sun.java.swing.JFrame`**.
  - activez l'option **`Compose the class visually`** de manière à ce que le browser de classes s'ouvre directement sur l'onglet **`Visual Composition`**.
- Pour intégrer la Proxy Vue de Dossier dans le Visual Composition Editor :
- Cliquez sur l'icône **`Choose Bean`** situé au-dessus de la palette.
- Sélectionnez le type de bean **`Variable`**.
- Avec le bouton Browse, sélectionnez le nom de la classe **`CustomerProxyLv`**.

☞ La Proxy Racine déposée ici est en effet de type variable car elle doit représenter la même instance de Proxy que celle appelée dans l'applet.

- **Promotion de la Proxy Racine**

L'objectif est maintenant d'assurer l'égalité permanente de cette Proxy variable avec la Proxy constante instanciée dans l'applet. Pour cela, il est nécessaire que la Proxy variable soit visible depuis l'extérieur de la classe **CustomersFrame**, pour qu'une connexion propriété - propriété entre la Proxy constante et la Proxy variable puisse être réalisée dans l'applet.

Effectuez les opérations suivantes :

- Depuis le menu contextuel de la Proxy, choisissez **Promote Bean feature**.
- Dans la colonne **Property** choisissez **this** puis cliquez sur **>>**.

  La classe **CustomersFrame** a maintenant une propriété publique, en lecture/écriture, nommée **customerProxyLv1This** de type Proxy Racine.
- Sauvegardez la fenêtre (menu **Bean**, choix **Save Bean**).

- **Maquettage de la propriété rows**

A l'intérieur du bean **CustomersFrame**, posez un bean **JTable** et redimensionnez-le.

Pour voir apparaître les colonnes de la JTable, vous devez exécuter la JFrame. Les colonnes de la JTable sont initialisées avec les Rubriques de la Vue Logique et les lignes représentent les instances présentes dans **Rows**. Le contenu de la JTable est rafraîchi après chaque mise à jour de **Rows** (sélections, créations…).

Deux maquettages sont possibles :

- Si vous voulez afficher toutes les colonnes correspondant à toutes les Rubriques de la Vue Logique avec les noms en clair définis dans la Proxy, il vous suffit de connecter la propriété **Table model** de la Proxy à la propriété **model** de la JTable.
- En revanche, si vous voulez sélectionner les colonnes à afficher, modifier leur en-tête, ou créer une nouvelle colonne affichant des informations calculées en local, vous devez personnaliser la JTable.

  Pour cela, vous devez créer une nouvelle classe **TableModel** et personnaliser ses méthodes.

  Cette nouvelle classe doit être publique et doit hériter:
  - soit de **CustomerTableModel**, situé dans le package **com.ibm.vap.generated.reuse**. Ainsi, cette classe héritera automatiquement de toutes les méthodes existant dans **CustomerTableModel** et vous ne modifierez que les méthodes qui ne conviennent pas.
  - soit directement de **PacbaseTableModel**, situé dans le package **com.imb.vap.beans.swing**. Dans ce cas, vous devrez réécrire toutes les méthodes que vous voulez utiliser puisqu'elles ne sont pas récupérées automatiquement.

Pour créer la nouvelle classe, sélectionnez le choix **Add Class** dans le menu contextuel du package (**com.ibm.vap.generated.reuse** ou **com.ibm.vap.beans.swing**). Donnez-lui un nom (par exemple, **NewCustomerTableModel**) et dans le champ **Superclass**, sélectionnez la classe (**CustomerTableModel** ou **PacbaseTableModel**) dont cette nouvelle classe va hériter.

Vous devez ensuite personnaliser les méthodes de cette classe en insérant du code directement dans la partie **source**. Dans notre exemple, nous avons choisi de faire hériter la nouvelle classe **NewCustomerTableModel** de la classe **CustomerTableModel**.

Nous devons personnaliser la méthode **public int getColumnCount()** pour limiter le nombre de colonnes de la JTable à 3, alors que la Vue Logique contient 7 Rubriques.

```
public class NewCustomerTableModel extends
com.ibm.vap.generated.reuse.CustomerTableModel {
public int getColumnCount (){
  return 3;
}
}
```

Les autres méthodes permettant de personnaliser les colonnes de la JTable sont documentées dans le *Chapitre* 3: Principes généraux de développement, *Utilisation des propriétés, Personnalisation des colonnes d'une Jtable (Java uniquement)*.

Dans le Composition Editor, il vous suffit alors de :

- Poser une instance du nouveau **TableModel**,
- Connecter cette instance, via sa propriété **this**, à la propriété **TableModel** de la Proxy,
- Connecter cette instance, via sa propriété **this**, à la propriété **model** de la JTable.

- **Maquettage de la propriété detail**

Cette étape nécessite les opérations suivantes :

- Depuis la Proxy Racine, effectuez un Tear-Off de la propriété **detail**.
- Insérez un bean **JLabel** par propriété à maquetter.
- Vous devez maintenant maquetter un champ de saisie par propriété à afficher.

  Pour cela, utilisez les beans **Pacbase Swing Text Field** et **Pacbase Swing Integer Field** fournis par Pacbench C/S lors de l'importation du runtime.

  Vous pouvez également insérer ces beans en cliquant sur l'icône **Choose Beans** située en haut de la palette. Sélectionnez le type de bean **Class** ou **Variable**. Puis sélectionnez, à l'aide du bouton Browse, le nom du bean : par exemple : **PacbaseJTextField**.

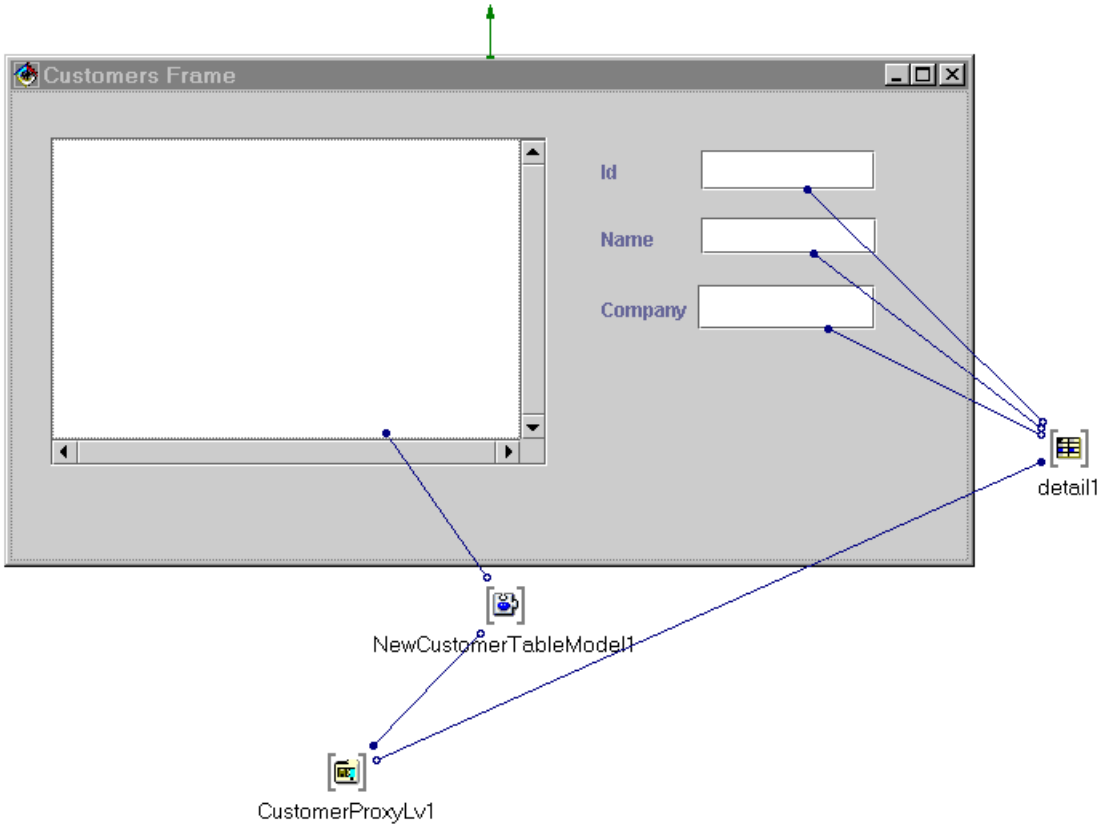
  Pour des détails complémentaires sur ces beans, reportez-vous à la documentation en ligne des classes génériques.
- Insérez un bean de type approprié pour chaque propriété à maquetter.

- Ensuite, pour chaque champ maquetté, connectez la propriété du bean **detail** à la propriété correspondante de type **String** ou **Int**.

  Dans notre exemple, pour le champ de saisie correspondant au numéro du client, il faut connecter la propriété **Customer Number** du bean **detail** à la propriété **Int** du champ.

- **Résultat dans le Composition Editor**

A l'issue de ces étapes, le Composition Editor se présente comme ci-dessous :



Nous avons choisi ici de personnaliser la JTable. C'est pourquoi nous avons inséré un bean NewCustomerTableModel1.

Sélection d'une instance dans rows et transfert dans detail

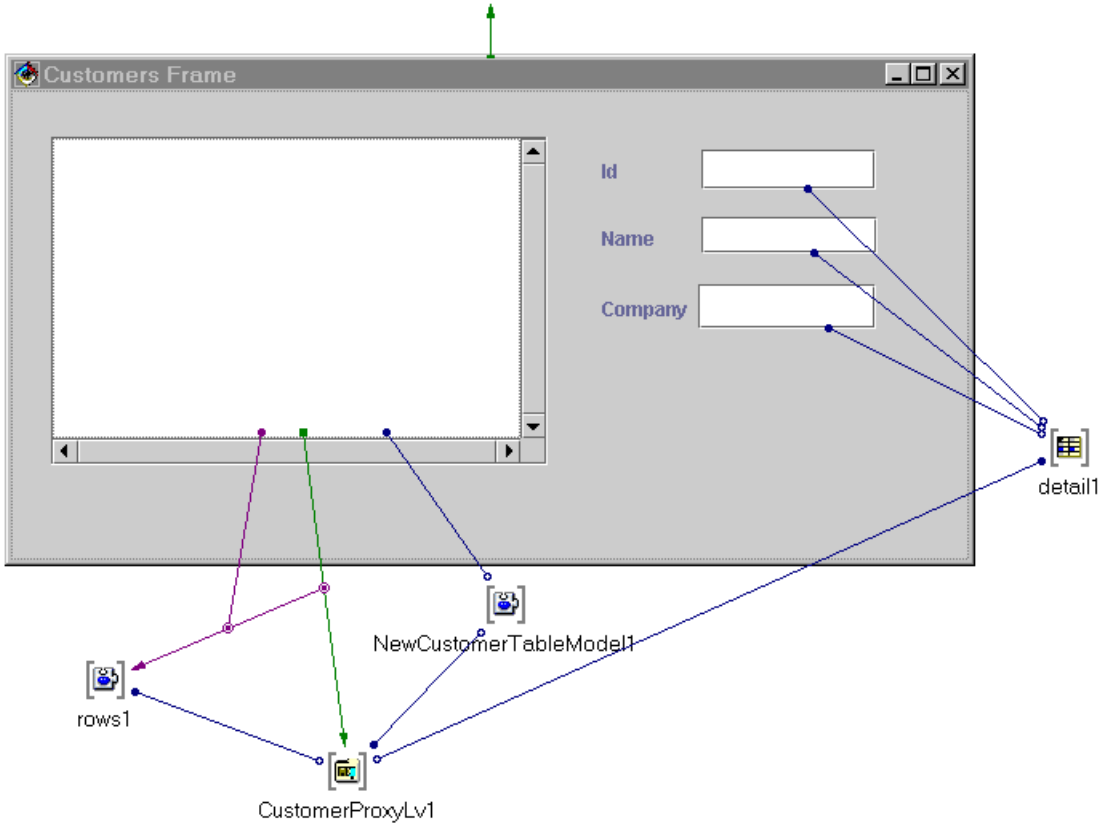
L'objectif est maintenant de programmer le transfert de l'instance sélectionnée par l'utilisateur depuis la table dans la propriété **detail** de la Proxy.

Pour cela, effectuez les étapes suivantes :

- Connectez les événements **keyEvents** et **mouseEvents** de la JTable à la méthode **Get Detail From DataDescription** de la Proxy Racine.
- Les liens apparaissent en pointillés puisque la méthode **Get Detail From DataDescription** nécessite un paramètre.
- Faites un **Tear-Off** de la propriété **Rows** de la Proxy Racine.Vous obtenez alors un bean variable **rows1**.
- Pour spécifier le paramètre requis par la méthode **Get Detail From DataDescription**, cliquez successivement sur les liens en pointillés. Connectez le paramètre de type data du lien, ici **customerData**, à la méthode **elementAt(int)** de **rows1**.

- Connectez ensuite la propriété **`SelectedRow`** de la JTable au paramètre **`Arg1`** du lien créé lors de l'étape précédente. Ce paramètre est accessible lorsque le pointeur se trouve au milieu de la connexion.

Voici le Composition Editor tel qu'il doit se présenter à l'issue de cette étape :



Pour plus de clarté, nous ne présentons la connexion que d'un des deux événements de la JTable.

Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders

- **Déclenchement des méthodes de la Proxy**

Le déclenchement des méthodes est identique à celui détaillé pour la Proxy Racine de l'exemple VisualAge Java V1.

Pour plus de détails, reportez-vous au au paragraphe *Développement de l'interface utilisateur avec VisualAge Java V1, Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders.*

- **Gestion de la navigation**

La gestion de la navigation est identique à celle détaillée pour la Proxy Racine de l'exemple VisualAge Java V1.

Pour plus de détails, reportez-vous au au paragraphe *Développement de l'interface utilisateur avec VisualAge Java V1, Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders*.

Développement de la fenêtre Orders

La fenêtre Orders est la même que celle développée dans l'exemple VisualAge Java V1.

☞ Pour plus de détails, reportez-vous au paragraphe *Développement de l'interface utilisateur avec VisualAge Java V1, Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders*.

Création de la fenêtre Orders

Dans le workbench, de la même manière que pour la création de la fenêtre **`Customers`**, créez une classe **`OrdersFrame`** héritant de **`JFrame`** dans le package **`example.swing`**.

☞ Pour plus de détails, reportez-vous au paragraphe Développement de la fenêtre Customers

Intégration et promotion de la Proxy Dépendante

Dans l'onglet Visual Composition, posez un bean variable **`OrderProxyLv`** sur la Free Form Surface.

☞ Pour plus de détails sur l'intégration de la Proxy, reportez-vous dans cette section, au paragraphe ***Création de la fenêtre et* intégration de la PVD dans le Composition Editor**.

Il faut maintenant promouvoir cette Proxy variable de manière à la rendre visible de l'extérieur.

Pour cela, effectuez les opérations suivantes :

- Depuis le menu contextuel de la Proxy, choisissez **`Promote Bean feature`**.
- Dans la colonne **`Property`** choisissez **`this`** puis cliquez sur **`>>`**.

  La classe **`OrdersFrame`** a maintenant une propriété publique, en lecture/écriture, nommée **`OrderProxyLv1This`** de type Proxy Dépendante.
- Sauvegardez la fenêtre (menu **`Bean`**, choix **`Save Bean`**).

Maquettage de rows et detail de la Proxy Dépendante

Le maquettage est identique à celui détaillé pour la Proxy Racine.

☞ Pour plus de détails, reportez-vous dans cette section, au paragraphe *Développement de la fenêtre Customers*.

Sélection d'une instance dans rows et transfert dans detail

Les opérations à effectuer ici sont identiques à celles nécessaires pour les propriétés **`rows`** et **`detail`** de la Proxy Racine.

☞ Pour plus de détails, reportez-vous dans cette section, au paragraphe *Développement de la fenêtre Customers*.

Déclenchement des méthodes de la Proxy Dépendante

Utilisez les mêmes principes que ceux décrits dans *Développement de l'interface utilisateur avec VisualAge Java V1, Développement de la fenêtre Customers, Déclenchement des méthodes de la Proxy et navigation vers la fenêtre Orders*.

Particularités du développement d'une application standalone

Introduction

Ce sous-chapitre a pour objectif de souligner les différences entre le développement d'une applet et celui d'une application *standalone*.

☞ Une application *standalone*, contrairement à une applet, n'est pas destinée à être exécutée dans un browser Web.

Les différences fondamentales sont les suivantes :

- Dans le développement d'une application *standalone*, la classe de base utilisée est de type **`Frame`** (pour awt) ou **`JFrame`** (pour swing). Cette classe hérite de la classe **`java.awt.Frame`** (pour awt) ou **`com.sun.java.swing.JFrame`** (pour swing) alors qu'une applet hérite de la classe **`java.applet.Applet`** (pour awt) ou **`java.applet.JApplet`** (pour swing).
- Dans le développement d'une application *standalone*, l'instanciation de la Proxy Racine est conditionnée par l'utilisation de la méthode **`setLocationsFile`** et non par celle des propriétés **`Host`** et **`Port`**. Cette méthode permet de positionner le fichier des *locations* **`VAPLOCAT.INI`** pour que l'application accède directement au middleware et non via la gateway.

Exemple

Cet exemple montre comment modifier l'applet précédemment développée en application *standalone*. Il réutilise les fenêtres **`CustomersFrame`**, désormais le point d'entrée de votre application, et **`OrdersFrame`**.

☞ Pour les fenêtres développées avec l'option Swing de VisualAge Java V2, il suffit de remplacer les classes awt présentes dans le code par les classes correspondantes swing.

Aucune modification n'est nécessaire dans le Composition Editor. Il vous suffit d'insérer du code spécifique dans la méthode **`main`** de la classe **`CustomersFrame`**. La méthode **`main`** constitue le point d'entrée d'une application *standalone*.

Ce code spécifique vous est présenté ci-dessous entre les lignes de commentaires ***`//begin`*** et ***`//end`***.

```
/**
 * main entrypoint - starts the part when it is run as an application
 * @param args java.lang.String[]
 */
public static void main(java.lang.String[] args) {
   try {
      vap.sample.CustomersFrame aCustomersFrame = new vap.sample.CustomersFrame();
      try {
         Class aCloserClass = Class.forName(« uvm.abt.edit.WindowCloser »);
         Class parmTypes[] = { java.awt.Window.class };
         Object parms[] = { aCustomersFrame };
         java.lang.reflect.Constructor aCtor = aCloserClass.getConstructor(parmTypes);
         aCtor.newInstance(parms);
   } catch (java.lang.Throwable exc) {};
   //begin
   aCustomersFrame.setCustomerProxyLv1(new
   com.ibm.vap.generated.proxies.CustomerProxyLv());
   aCustomersFrame.getCustomerProxyLv1().setLocationsFile(« d:\\user\\vapb\\vaplocat.ini
    »);
```

```
//end
aCustomersFrame.setVisible(true);
catch (Throwable exception) {
  System.err.println(« Exception occurred in main() of java.awt.Frame »);
}
}
```

La première instruction spécifie la création d'une nouvelle Proxy Racine. La précédente est déjà instanciée dans l'applet et ne peut être réutilisée ici comme bean constant.

La seconde instruction positionne le fichier des *locations*.

Gestion des erreurs

Principes

Introduction

La gestion des erreurs associées à la manipulation des objets Proxy VisualAge Pacbase se base sur le mécanisme des levées d'exceptions propres au langage Java.

Les objets Proxy peuvent lever quatre types d'erreurs Fonction eBusiness ainsi que l'ensemble des erreurs Java.

Pour la gestion des erreurs Fonction eBusiness, l'utilisateur dispose de quatre classes génériques dont les méthodes permettent de véhiculer les erreurs levées par les objets Proxy. Chacune de ces classes correspond à un type d'erreurs :

- Erreurs locales
- Erreurs serveur
- Erreurs système
- et erreurs de communication

Ces classes héritent toutes de la classe **`java.lang.Throwable`** et se trouvent dans le package **`com.ibm.vap.generic`**.

☞ Pour plus de détails, consultez le Chapitre 3: Principes généraux de développement, sous-chapitre Gestion des erreurs. Vous trouverez la liste des erreurs locales et de communication possibles, ainsi que la structure de la clé d'erreur pour les erreurs serveur et système.

Programmation

Les exceptions doivent obligatoirement être interceptées par la programmation dans le composant Client.

La programmation des erreurs nécessite d'une part l'écriture de code Java, et d'autre part la construction graphique de la fenêtre qui permet d'afficher les erreurs.

☞ Un exemple de gestion des erreurs vous est proposé dans la section *Exemple de gestion des erreurs*.

☞ Pour plus de détails sur les exceptions susceptibles d'être levées par les objets Proxy, consultez le manuel *Applications eBusiness & Pacbench C/S : Interface de Programmation des Proxies* ou, dans votre station VisualAge, reportez-vous à la signature des méthodes correspondantes.

Erreurs locales

Les erreurs locales provoquent l'exception **com.ibm.vap.generic.LocalException**. Cette exception porte une propriété de type **int** permettant d'identifier le type de l'erreur.

Vous trouverez la liste des erreurs à l'origine de cette exception dans la section Erreurs locales (Chapitre 3: Principes généraux de développement). Elles sont également décrites dans la documentation HTML associée aux classes génériques : Package com.ibm.vap.generic.

Ces intitulés d'erreurs correspondent à des constantes de la classe **com.ibm.vap.generic.LocalException** et représentent des types d'erreurs. Préfixé par **LOCAL** _, chaque constante détermine une clé d'erreur. Cette clé d'erreur permet d'identifier le libellé d'erreur associé dans le fichier local des libellés d'erreurs **vaperror.properties**.

Erreurs serveur

Les erreurs serveur provoquent l'exception **com.ibm.vap.generic.ServerException**.

Cette exception est levée à la suite de la réception d'un message d'erreur logique détectée par le composant Serveur. Elle porte la clé et le libellé associés.

Vous trouverez les informations sur la clé d'erreur des erreurs serveur dans la section *Erreurs serveur (*Chapitre 3: Principes généraux de développement*).*

Erreurs système

Les erreurs système (physiques) provoquent l'erreur **com.ibm.vap.generic.SystemError**. Ce type d'erreur représente une erreur interne et irrécupérable.

Vous trouverez les informations sur la clé d'erreur des erreurs système dans la section Erreurs système (Chapitre 3: Principes généraux de développement).

Erreurs de communication

Les erreurs dans la chaîne de communication avec le Serveur provoquent l'erreur **com.ibm.vap.generic.CommunicationError**. Comme toute erreur Java, une erreur de communication VisualAge Pacbase ne renvoie pas de clé. Pour identifier la cause de l'erreur, il faut récupérer le message associé à l'erreur (méthode **getMessage()** de la classe).

Vous trouverez les informations sur les erreurs de communication dans la section Erreurs de communication (Chapitre 3: Principes généraux de développement).

Exemple de gestion des erreurs

Introduction

Cet exemple illustre une méthode qui permet de gérer tous les types d'erreurs susceptibles d'être levées par les objets Proxy. Pour cela, trois classes sont définies :

- **`StandardErrorMessageFactory`** : classe permettant de créer un vecteur de **`StandardErrorMessage`** à partir de l'exception levée
- **`StandardErrorMessage`** : classe unifiant les caractéristiques des différents types d'erreurs
- **`ErrorManagerExample`** : classe graphique utilisée pour afficher les erreurs

Présentation des classes non visuelles utilisées par l'exemple

- classe **`StandardErrorMessageFactory`**

Cette classe offre la méthode **`public static java.util.Vector getStandardErrorMessages (Throwable th)`** qui permet de créer un vecteur de **`StandardErrorMessage`** à partir d'une exception donnée quel que soit son type.

- classe **`StandardErrorMessage`**

Chaque objet de type **`StandardErrorMessage`** a des propriétés qui sont initialisées selon le type d'erreur (locale, serveur, système, communication, etc.) :

- la **Proxy hiérarchique associée à l'erreur** si l'erreur concerne une instance de Vue Logique particulière.
- la **clé de l'erreur** (pour les erreurs locale, serveur ou système) :

  C'est une chaîne de caractères retournée par le serveur, dans le cas d'une erreur serveur ou système.

  Dans le cas d'une erreur locale, c'est une chaîne de caractères correspondant au nom de la constante associée au type de l'erreur et définie dans la classe **`LocalException`**, préfixé par **`LOCAL_`**.

  ☞ Pour connaître tous les types d'erreurs locales, reportez-vous au Chapitre 3: Principes généraux de développement**,** Erreurs locales

  ☞ Pour connaître les clés d'erreur correspondant aux erreurs serveur et système, reportez-vous au Chapitre 3: Principes généraux de développement, respectivement *Erreurs serveur* et *Erreurs système*.
- le **libellé local de l'erreur** :
  - dans le cas d'une erreur locale, serveur ou système, le libellé résulte de la traduction par le Client de la clé de l'erreur. Dans le fichier **`vaperror.properties`**, une table de correspondance permet d'identifier ce libellé à partir d'une clé donnée.

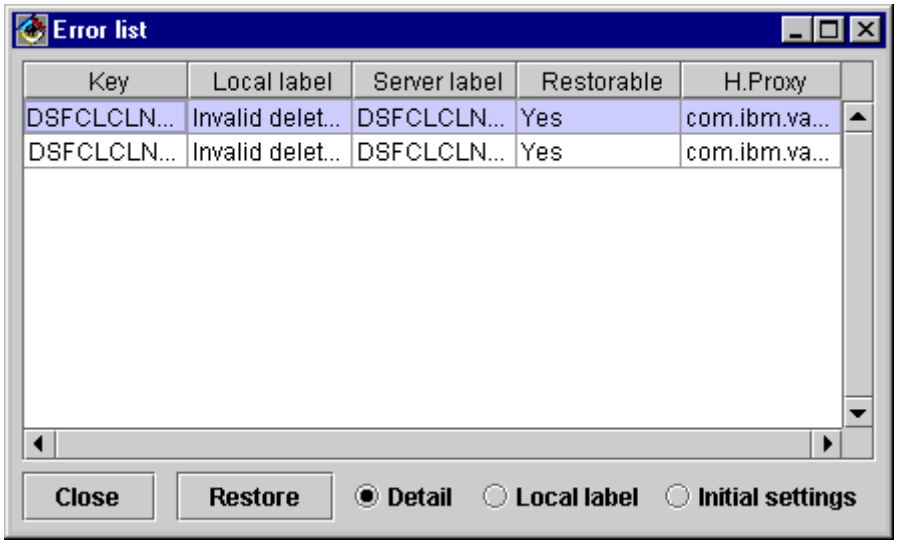
    ☞ Pour des informations sur le fichier **`vaperror.properties`**, reportez-vous à la *Documentation du Développeur : Interface de Programmation des Proxies*.
  - Pour les autres types d'erreurs, le libellé correspond directement au message contenu dans l'exception Java.

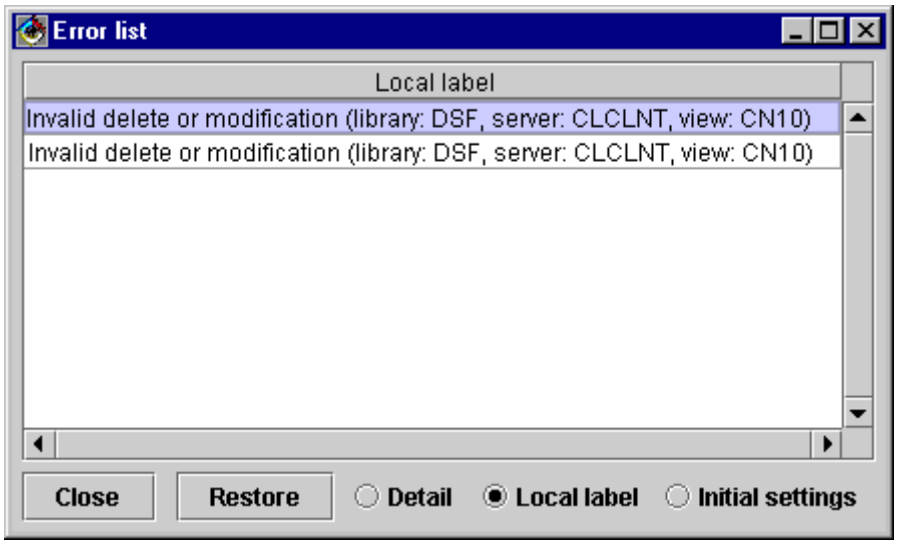
- le **libellé serveur de l'erreur**, pour les erreurs serveur et système, si un serveur de libellés d'erreurs a été codé dans les Services Applicatifs.
- la propriété booléenne `Restorable`. Cette propriété est vraie :
  - si une Proxy est associée à l'erreur.
  - s'il est possible de réafficher l'instance à l'origine de l'erreur dans le détail de la fenêtre qui appelle la Proxy.

Pour des informations complémentaires sur la gestion des erreurs, reportez-vous à la *Documentation du Développeur : Interface de Programmation ds Proxies*.

Présentation de la classe visuelle ErrorManagerExample

Interface graphique





Fonctionnalités de la classe

Cette classe est utilisée pour présenter une liste de messages d'erreur correspondant à des instances de la classe `StandardErrorMessage`.

Tous les contrôles graphiques utilisés dans cette fenêtre sont proportionnels à sa taille.

Les fonctionnalités de cette fenêtre sont les suivantes :

- Les instances de **StandardErrorMessage** calculées par la classe **StandardErrorMessageFactory** sont passées à cette fenêtre dans un Vector par l'intermédiaire de la méthode **addStandardErrorMessages(Vector)**. Tant que la fenêtre n'est pas fermée, les messages d'erreur passés sont cumulés aux messages d'erreur déjà affichés. Les messages d'erreur affichés peuvent être supprimés en utilisant la méthode **resetCurrentStandardErrorMessages()**.

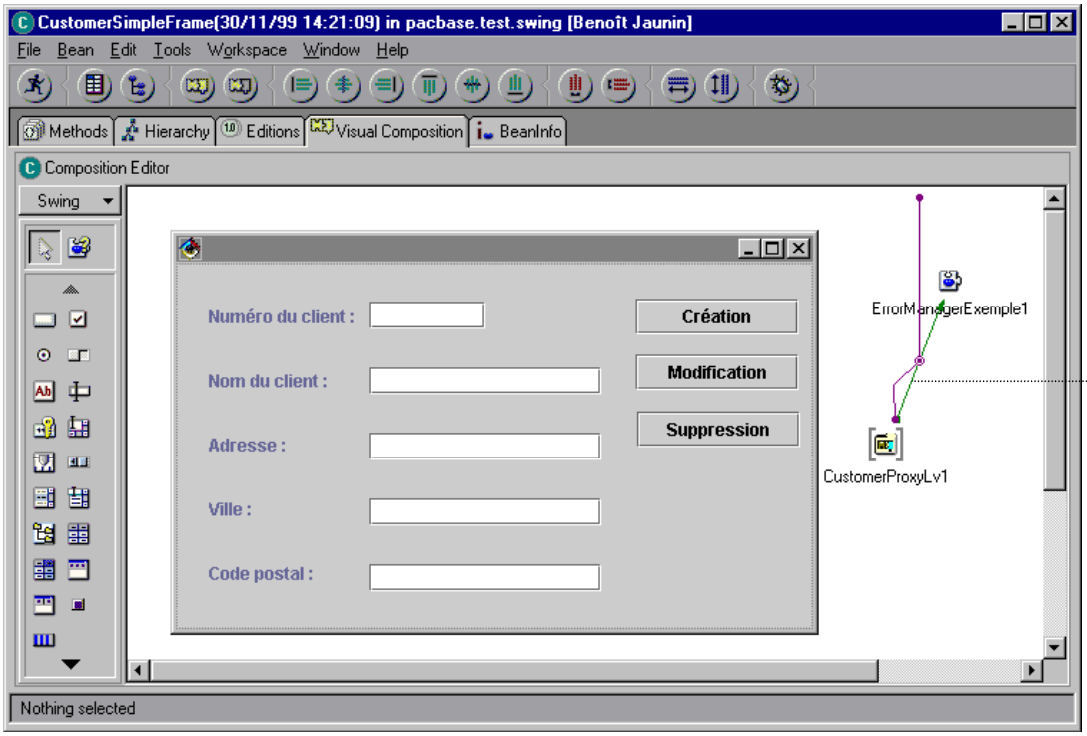
- Les propriétés des instances de **StandardErrorMessage** que l'on souhaite visualiser sont définies dans l'attribut **visibleColumns** des propriétés de la fenêtre **ErrorManagerExample**. Cet attribut correspond à un tableau de **int** qui peuvent prendre les valeurs suivantes :
  - **1** pour visualiser la clé du message d'erreur
  - **2** pour visualiser le libellé local du message d'erreur
  - **3** pour visualiser le libellé serveur du message d'erreur
  - **4** pour visualiser le statut de restauration de l'erreur
  - **5** pour visualiser le nom de la classe associé à la Proxy hiérarchique

- Trois boutons radios ont été insérés pour permettre de sélectionner dynamiquement des présentations différentes de la liste des instances de **StandardErrorMessage** :
  - **Detail** pour visualiser toutes les propriétés de chaque instance de **StandardErrorMessage**.
  - **Local label** pour ne visualiser que le libellé local de chaque instance de **StandardErrorMessage**.
  - **Initial settings** pour visualiser les propriétés définies par l'attribut **visibleColumns**.

- Le bouton **Restore** est activé lorsqu'une instance de **StandardErrorMessage** est sélectionnée et qu'il est possible de restaurer le contexte de l'erreur. Lorsque l'utilisateur clique sur ce bouton, le contexte de l'erreur est restauré et la fenêtre qui gère l'instance de la Vue Logique qui a provoqué l'erreur est affichée.

  Pour mettre en œuvre cette fonctionnalité, il est nécessaire de faire connaître à la fenêtre **ErrorManagerExample,** chaque fenêtre gérant l'attribut **detail** du nœud d'une Proxy en utilisant la méthode **addProxyManagingbyWindow**. Cette méthode prend en paramètres le nœud de la Proxy (**HierarchicalProxyLv**) et la fenêtre de gestion des erreurs (**Jframe**).

Exemple



Cet exemple illustre le principe qui permet d'enregistrer l'association d'une Proxy hiérarchique et la fenêtre qui l'utilise et de faire connaître cette association à la fenêtre d'erreur.

Le lien (1) connecte l'événement **`this`** de la Proxy à la méthode **`addProxyManagingbyWindow`** de la fenêtre d'erreurs (**`ErrorManagerExample`**). Dans ce contexte, cette méthode sera exécutée dès l'instanciation de la Proxy.

Les deux autres liens permettent de passer l'instance de la Proxy hiérarchique (**`this`** de la Proxy sur paramètre **`hp`** de la connexion) et l'instance de la fenêtre (**`this`** de **`CustomerSimpleFrame`** sur paramètre **`wi`** de la connexion).

Code pour l'affichage de la fenêtre d'erreur

La méthode proposée est adaptée aux applications conçues dans VisualAge Java à l'aide de l'outil de composition visuelle.

Cette méthode suppose que la fenêtre de gestion des erreurs (classe **`ErrorManagerExample`**) a été insérée en tant que bean de type *class* ou *variable* dans l'éditeur de composition visuelle qui contient la fenêtre qui l'appelle.

Les exceptions associées à cette fenêtre sont traitées en insérant dans la méthode **handleException(Throwable)** de celle-ci le code suivant :

```
private void handleException(Throwable exception) {
java.util.Vector standardErrorMessages =
pacbase.test.swing.ErrorManager.StandardErrorMessageFactory.getStandardErrorMessages(exception);
   if (standardErrorMessages != null) {
                if (standardErrorMessages.size() >= 0) {
                            getErrorManagerExemple1().addStandardErrorMessages(standardErrorMessages);
                            getErrorManagerExemple1().showStandardErrorMessages();
                }
   }
}

}
```

Gestion de la communication

Ce sous-chapitre contient toutes les informations nécessaires au développement du middleware utilisé par les applications générées avec le module Business ou Pacbench C/S.

Les informations relatives au middleware utilisé par les applications lors de leur déploiement sont données dans le *Guide utilsateur du Middleware*.

Traitement d'une requête

Dans VisualAge, les services du middleware sont exécutés à partir d'un ensemble de classes de communication spécifiques livrées à l'installation du produit.

La communication avec le Serveur s'effectue via l'interface **ServerAdapter**. Deux implémentations de cette interface existent :

- **MiddlewareAdapter** qui permet un accès direct aux DLLs natives (en C++) du Middleware installées localement. Elle permet également de définir le paramètre du contexte de communication (**location**, **userId**, **password**, **clientEncoding**…) et son mode de fonctionnement (**traceLevel**, **nbMaxConnection**, **connectionCleaningInterval**, …).
- **GatewayAdapter** qui passe par l'intermédiaire du module VAP Gateway ou VAP Relay. Cette classe est particulièrement adaptée aux applications de type Applet. Les paramètres (**host**, **port**, **userid**, **password**, **clientEncoding**…) qui définissent la communication par l'intermédiaire de **VapGateway** ou **vaprelay** sont également indispensables. En revanche, les paramètres permettant de définir la communication avec le serveur d'application doivent être définis au niveau du module **VapGateway**.

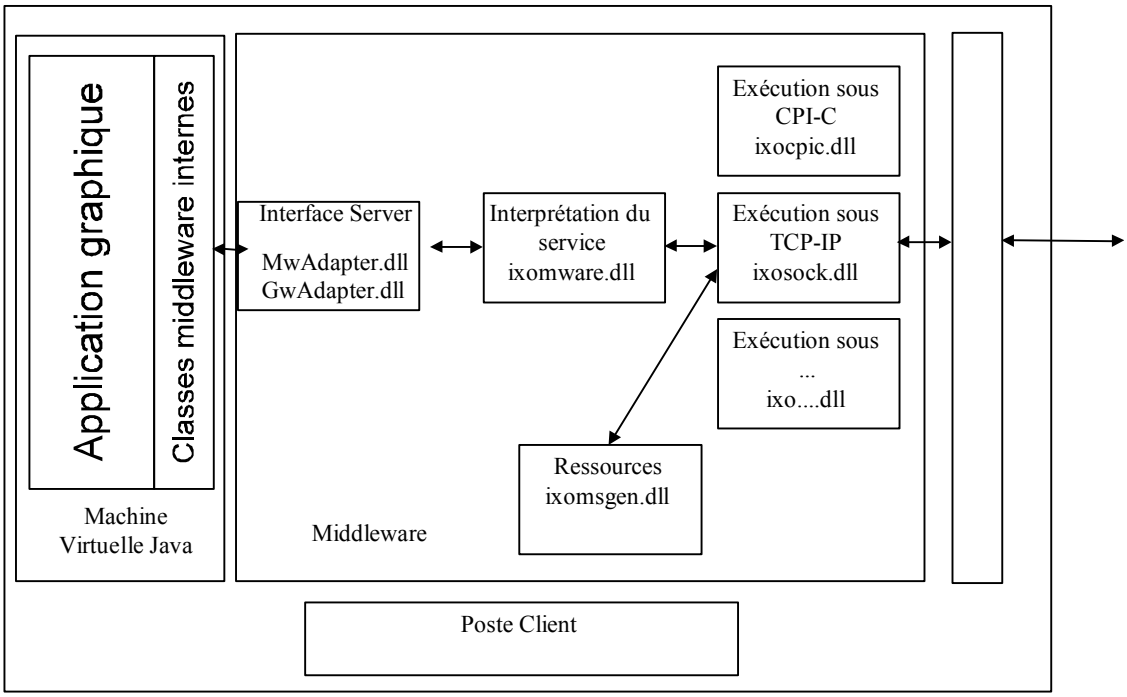
Pour plus d'informations, consultez le *Guide utilisateur du Middleware*.

Lorsqu'un objet **ServerAdapter** est instancié, celui-ci créé systématiquement un autre objet de classe **Requester** qui met en oeuvre les conditions techniques pour communiquer avec le Serveur, en implémentant notamment les méthodes d'envoi et de réception des messages. Deux implémentations de la classe **Requester**, associées respectivement aux classes **MiddlewareAdapter** et **GatewayAdapter** sont fournies :

- **NativeRequester** qui fait appel directement (via **JNI**) aux fonctions C des DLLs du Middleware.
- **GatewayRequester** qui implémente les messages via **vaprelay** ou **VapGateway**.

Accès direct au Middleware

Schéma de traitement d'une requête



Afin d'instancier un Dossier en mode direct, vous pouvez utiliser le constructeur par défaut : **CustomerProxyLv()**

Par exemple :

```
myProxy = new CustomerProxyLv() ;
myproxy.setLocationsFile("c:\vap\gen\java\VAPLOCAT.INI") ;
```

La seconde instruction spécifie le fichier de locations à utiliser. Par défaut, on recherche un fichier **VAPLOCAT.INI** se trouvant dans le répertoire courant.

Accès via une Gateway

Deux constructeurs peuvent dans ce cas être utilisés :

- **CustomerProxyLv(String host)**

  Par exemple : `new CustomerProxyLv("9.134.5.146")`
- **CustomerProxyLv(String host, int port)**

  Par exemple : `new CustomerProxyLv("9.134.5.146", 6001)`

☞ Si **host** vaut null, on considère que le middleware est local.

Accès via un adaptateur particulier

Pour accéder au Middleware via un adaptateur particulier, vous utiliserez la propriété **`setServerAdapter (ServerAdapter)`** ou **`setServerAdapterName(String)`**.

Exemple : proxy.setServerAdapterName("GwAdapter")

Changement dynamique des paramètres d'accès au middleware

Plusieurs méthodes de la Proxy Racine permettent de changer ces paramètres dynamiquement :

- **`void setHost(String host)`**

  ☞ Si **`host`** vaut null, on considère que le middleware est local.

- **`void setPort(int port)`**
- **`void setLocationsFile(String filename)`** (mode direct)
- **`void setTraceFile(String filename)`**
- **`void setTraceLevel(int traceLevel)`** (mode direct)

Définition du contexte d'utilisation

La gestion de la communication nécessite la définition du contexte d'utilisation du middleware.

Ce contexte est défini via l'outil Editeur de Localisation dans un fichier spécifique appelé **`vaplocat.ini`**.

☞ Pour plus de détails sur l'interface de l'Editeur de Localisation, consultez l'aide en ligne des outils eBusiness. Vous pouvez également vous reporter au *Guide utilisateur du Middleware*, chapitre *Protocoles Description & Configuration*, vous y trouverez la liste et la fonction des parameters que vous devez définir pour chaque localisation, selon le protocole employé.

Il existe plusieurs façon de l'Editeur de Localisation :

A partir du module eBusiness de Developer workbench

L'Editeur de Localisation se lance à partir de l'onglet "Applications" ou "Dossiers" du workbench.

A partir de VisualAge for Java

Pour lancer l'outil de Test des Services directement à partir de VisualAge for Java, sélectionnez dans le menu "Workspace", "Outils" → "VisualAge Pacbase eBusiness" → "Editeur de Localisation".

A partir du fichier .exe

Lancez le fichier the **`vapLocationEditor.exe`**.

Vous pouvez paramétrer le lancement de l'Editeur de Localisation via les options suivantes :

`inputfile<INPUT_FILE>` : ce paramètre vous permet d'indiquer le chemin d'accès au fichier de localisation que vous souhaitez utiliser. Ce fichier est soit un fichier de localisation existant, soit un fichier `.gvc` (contenant l'extraction des proxies eBusiness). Toutes les caractéristiques des Moniteurs de Communication sont présentes dans le fichier en entrée.

Au lieu d'initialiser l'éditeur par l'intermédiaire d'un fichier en entrée, vous pouvez lancer l'éditeur en mode expert de façon à créer et modifier les Moniteurs de Communication. Dans ce cas, vous devrez utiliser l'option **`-expertmode`**.

A partir d'une machine virtuelle Java

Pour effectuer le lancement de l'Editeur de Localisation à partir d'une machine virtuelle Java, lancez le fichier **`java_vapLocationEditorTool.bat`**.

Ce .bat est un exemple que vous devez modifier en fonction de la localisation de votre JDK (Java Developer ToolKit) ou JRE (Java Runtime Environment).

Vous pouvez indiquer l'option spécifiée au dessus pour paramétrer le lancement de l'Editeur de Localisation.

Test de l'application générée – Packaging

Test de l'application générée

Test des composants Serveur avec l'outil de Test des Services

Vous pouvez tester les composants client de votre application sans développer obligatoirement l'interface graphique de l'application. Le test est fait via l'outil de Test des Services.

Cet outil permet de voir les attributs de la Proxy, tester les méthodes disponibles et la communication.

Vous pouvez identifier ensuite les problèmes liés :

- à la conception et à la mise en oeuvre des Dossiers et des Composants Elémentaires,
- à la génération des Proxies,
- à la Communication et au middleware.

Consultez l'aide en ligne de l'outil de Text des Services.

Il existe plusieurs possibilités pour lancer l'outil de Test des Services :

A partir du module eBusiness de Developer workbench

L'Editeur de Localisation se lance à partir de l'onglet "Applications" ou "Dossiers" du workbench.

A partir de WSAD (ou Eclipse)

Pour lancer l'outil de Test des Services directement à partir WSAD (ou Eclipse), faites un clic-droit sur :

- un projet Java contenant des proxies VisualAge Pacbase,
- un ou plusieurs packages contenant des proxies VisualAge Pacbase,

- un ou des classes Java correspondant à des proxies VisualAge Pacbase.

A partir de VisualAge for Java

Pour lancer l'outil de Test des Services directement à partir de VisualAge for Java, sélectionnez dans le menu "Workspace", "Outils" → "VisualAge Pacbase eBusiness" → "Editeur de Localisation".

A partir du fichier .exe

Lancez le fichier **`vapServicesTestFacility.exe`**.

Vous pouvez paramétrer le lancement de l'Editeur de Localisation via les options suivantes :

`-classpath<PATH>`: ce paramètre permet d'indiquer le chemin d'accès aux classes de proxies à tester.

`-folders<PROXY_CLASS_NAME>`: ce paramètre permet d'indiquer quelles proxies doivent être testées.

A partir d'une machine virtuelle Java

Pour effectuer le lancement de l'Editeur de Localisation à partir d'une machine virtuelle Java, lancez le fichier **`java_vapServicesTestFacility.bat`** file.

Ce .bat est un exemple que vous devez modifier en fonction de la localisation de votre JDK (Java Developer ToolKit) ou JRE (Java Runtime Environment).

Vous pouvez indiquer l'option spécifiée au dessus pour paramétrer le lancement de l'Outil de Test des Services .

Contrôle des versions

Si l'option contrôle des versions détecte des divergences entre les versions, cela signifie que le Composant Elémentaire et l'objet Proxy n'ont pas été générés sous le même numéro de version. Vous devez alors :

- regénérer l'objet Proxy si vous avez regénéré seulement une nouvelle version du Composant Elémentaire,
- ou implémenter, dans VisualAge, l'application graphique générée comportant le nouveau composant proxy si cela n'a pas déjà été fait,
- ou implémenter le Composant Elémentaire si cela n'a pas déjà été fait.

Packaging

Packager une applet ou une application permet de passer des phases de développement et de test à la phase d'exploitation, c'est-à-dire faire en sorte de pouvoir utiliser l'applet ou l'application en dehors de l'environnement de développement. Cela consiste à exporter cette applet ou cette application.

Rappel : Prérequis

En phase d'exploitation, le fichier **VAPLOCAT.INI** doit se trouver dans le même répertoire que les applications finales. Lors de l'installation de ces applications, le développeur a la charge de s'en assurer.

- Applet

Pour la mise en exploitation d'une applet Java, les éléments suivants doivent être installés sur le poste de l'utilisateur :

- Un serveur HTTP
- Un browser Web Java 1.1 *enabled*
- La gateway et le relay sont installés sur la machine du serveur HTTP

- Application *standalone*

Pour la mise en exploitation d'une application *standalone* Java, le Java Runtime Environment (JRE) doit être installé sur le poste de l'utilisateur.

Pour plus de détails sur l'environnement d'exécution, reportez-vous à la *documentation Applications eBusiness & Pacbench C/S : Concepts & Architecture*.

Export

Que faut-il exporter ?

Il est impératif d'exporter toutes les classes d'exécution utilisées par l'application fonctionnelle qui ne font pas partie des classes de base. Les classes d'édition telles que les classes **BeanInfo** ou les beans utilisés pour le maquettage rapide des propriétés de type Rubriques Pacbase sont facultatives.

Il faut donc exporter :

- tous les packages du projet contenant le runtime, sauf les packages **com.ibm.vap.beans** et/ou **com.ibm.vap.beans.swing** (selon le package utilisé dans votre application). Pour ces deux packages, n'exportez que les classes réellement utilisées.
- le projet contenant les composants Proxy générés,
- l'applet ou l'application elle-même, c'est-à-dire tout le projet dans lequel celle-ci se trouve ou alors le ou les package(s) qui la compose(nt). Dans notre exemple, il s'agit du package **vap.sample** (pour l'exemple V1) ou **example.swing** (pour l'exemple V2).
- éventuellement les beans externes : dans l'exemple V1, nous utilisons le bean **IMulticolumnListbox** ; il convient donc d'exporter tout le package qui le contient, soit le package **COM.ibm.ivj.javabeans**.

Mise en place

- **Pour une applet**

Avant de réaliser l'export d'une applet, vous devez créer un répertoire sur la racine du serveur HTTP, destiné à recevoir le résultat de l'export. Par exemple **c:\www\html\codebase**. Ce répertoire constitue la racine des classes Java dans le serveur HTTP.

☞ Vous pouvez exporter les fichiers **class** directement dans l'arborescence du serveur HTTP, soit dans **codebase** dans notre exemple, ou dans un autre répertoire. Dans ce dernier cas, il faudra copier ces fichiers dans ce répertoire avant la mise en exploitation, en veillant à respecter l'arborescence des packages.

- **Pour une application *standalone***

Dans ce cas, l'emplacement du répertoire destiné à recevoir le résultat de l'export importe peu. Il suffit de déclarer ce répertoire dans la variable **CLASSPATH**.

Comment exporter ?

- **A partir de WSAD**

Dans le menu **File**, sélectionnez le choix **Export**. L'utilitaire d'exportation s'ouvre alors.

✍ Pour plus de détails, reportez-vous à l'aide en ligne de WSAD.

- **A partir de VisualAge for Java**

Pour l'export proprement dit, réalisez les opération suivantes :

- dans le workbench de VisualAge, sélectionnez tous les éléments à exporter,
- dans le menu **File**, sélectionnez le choix **Export**,
- dans la fenêtre **SmartGuide - Type of Export**, sélectionnez l'option **Class Files**, puis cliquez sur **Next**,
- dans la fenêtre **SmartGuide – Export to files**, entrez le nom du répertoire de sortie ou sélectionnez-le à l'aide du bouton **Browse**, en utilisant l'option **Create package subdirectories**,

✍ Pour des informations complémentaires sur l'export, reportez-vous à la documentation en ligne VisualAge Java. (menu **Help**, Choix **Tasks**., puis **Exporting to the file system**) ou à partir de Windows Explorer (ouvrir le répertoire **Tasks**, sélectionner un fichier HTML dans le répertoire **Export**, le fichier **overview.htm** est proposé comme fichier en d'entrée).

Optimiser le temps de téléchargement des fichiers .class

Pour cette option, le JDK doit être installé dans l'environnement d'exécution.

Une fois l'export sous forme de fichiers **.class** effectué, vous pouvez optimiser le temps de téléchargement des classes lors de l'exécution en transformant tous ces fichiers en un seul fichier archive **.jar**.

Dans une fenêtre DOS ou OS/2, positionnez-vous dans le répertoire de sortie de l'export, puis entrez la commande :

```
jar cvf sample.jar com vap
```

où :

- **sample.jar** est le nom du fichier archive,
- **com** et **vap** représentent deux répertoires contenant tous les fichiers **.class** nécéssaires au fonctionnement de l'application finale.

Pour une applet, le fichier **.jar** obtenu doit être copié dans dans l'arborescence du serveur HTTP.

Ecriture d'un fichier HTML (applet seulement)

Enfin, l'écriture d'un fichier HTML contenant l'applet est nécessaire à son exécution dans un browser Web. Ce fichier permet de positionner certains paramètres, tels que la largeur et la hauteur de l'applet.

Pour cela, toujours dans le cadre de notre exemple, créez dans le répertoire **c:\www\html\codebase\vap\sample**, un fichier **index.html** contenant le texte suivant :

```
<HTML>
<TITLE>
Sample Applet
</TITLE>
<BODY>
<CENTER>
<APPLET      code="vap.sample.SampleApplet.class"      WIDTH=1000"
HEIGHT=1000
codebase="/codebase"></APPLET>
</CENTER>
</BODY>
</HTML>
```

ou le texte suivant, si vous avez constitué un fichier archive **.jar**:

```
<HTML>
<TITLE>
Sample Applet
</TITLE>
<BODY>
<CENTER>
<APPLET      code="vap.sample.SampleApplet.class"      WIDTH=1000
HEIGHT=1000
archive="/codebase/sample.jar"
codebase="/codebase"></APPLET>
</CENTER>
</BODY>
</HTML>
```

Déploiement de l'application

Pour déployer l'application, suivez les explications indiquées dans la documentation de WSAD ou de VisualAge Java.

Mais vous devez aussi installer certains fichiers liés à l'utilisation de la Fonction eBusiness ou du module Pacbench C/S.

- **Pour une applet**

Le poste de l'utilisateur final doit être simplement équipé d'un navigateur.

- **Pour une application *standalone* :**
  - Si aucune gateway n'est utilisée, vous devez installer, sur le poste de l'utilisateur final :
    - le package middleware,
    - **`VAPLOCAT.INI`**,
    - **`VAPRUN.JAR`**,
    - **`VAPSWING.JAR`** si l'application utilise **`Swing`** ou **`VAPAWT.JAR`** si l'application utilise **`AWT`**.
  - Si une gateway est utilisée, vous devez installer, sur le poste où est installée la gateway :
    - **`GATEWAY.EXE`**,
    - le package middleware,
    - **`VAPLOCAT.INI`**,

    Dans ce cas, vous ne devez installer aucun de ces fichiers sur le poste de l'utilisateur final. En revanche, vous devez y installer le fichier :
    - **`VAPSWING.JAR`** si l'application utilise **`Swing`** ou
    - **`VAPAWT.JAR`** si l'application utilise **`AWT`**.

Chapitre 5 : Développement d'un Client au standard COM

Une fois les objets Proxy générés et compilés, ils doivent être utilisés dans un langage client capable d'assurer la gestion des objets ActiveX. Dans l'exemple donné ci-après, le langage client est Visual Basic.

Vous trouverez dans ce chapitre une description détaillée du développement d'un client avec les étapes suivantes : insertion des objets Proxy avec liens de programmation comprenant des actions, des attributs et des événements, la gestion des erreurs, la gestion de la communication et les tests de l'application.

Exemple d'utilisation de Proxy COM en Visual Basic

Cet exemple représente le développement d'un programme exécutable Visual Basic utilisant une Proxy COM. Cette Proxy est générée puis compilée sur une station de travail où est installé Visual Studio 6.0.

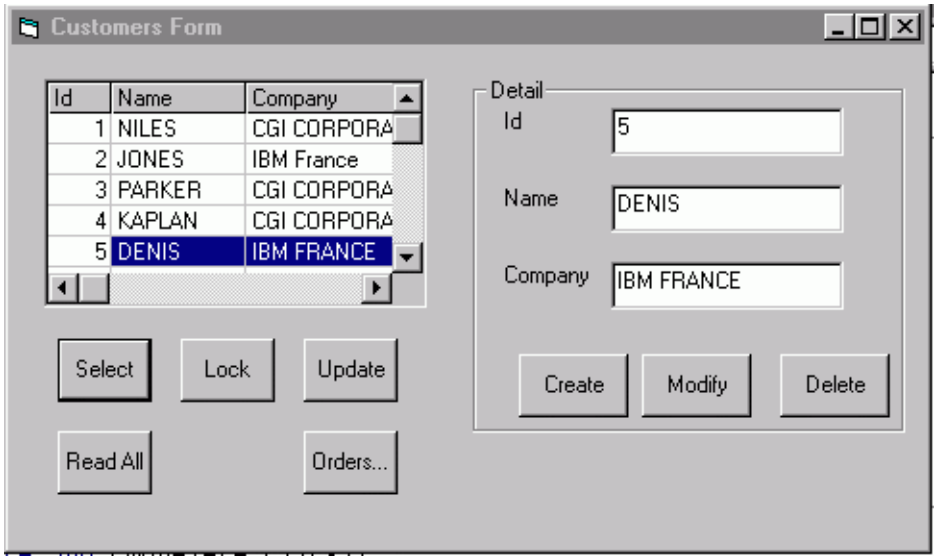
L'exemple manipule trois Proxy Elémentaires de la PVD :

- la Proxy Racine correspondant au noeud **`Clients`** qui gère les clients dans le système d'information décrit par le Dossier.
- la Proxy Dépendante correspondant au noeud **`Commandes`** qui gère les commandes dans le système d'information décrit par le Dossier.
- la Proxy Dépendante correspondant au noeud **`Lignes de Commandes`** qui gère les lignes de commandes dans le système d'information décrit par le Dossier.

Présentation de l'Interface Utilisateur

L'interface graphique utilisateur développée comprend deux fenêtres.

- **La fenêtre Customers**



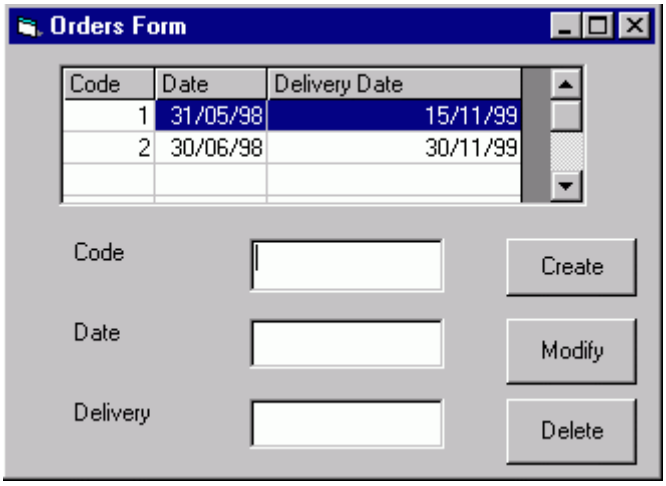
Cette fenêtre s'ouvre automatiquement au lancement du programme exécutable Visual Basic.

Elle contient les fonctionnalités suivantes :

- Le bouton **Select** permet l'affichage de la liste des clients.
- Le bouton **Read all** permet, à partir de cette fenêtre, de demander la lecture des commandes du client sélectionné.
- Parce qu'un service de verrouillage a été spécifié dans le Dossier, l'utilisateur doit, avant toute mise à jour, cliquer sur le bouton **Lock** pour verrouiller l'instance sélectionnée dans la liste avant toute mise à jour.
- Le bouton **Update** permet de mettre à jour la base de données
- Le bouton **Orders...**permet la navigation vers la fenêtre de gestion des commandes.
- Le bouton **Modify** permet la modification d'un client à partir de la fiche, après verrouillage de cette instance.

  Après sélection d'un client dans la liste, l'utilisateur doit cliquer sur **Lock** pour s'approprier momentanément ce client. Il doit ensuite cliquer sur **Modify** après saisie des modifications souhaitées.

- **La fenêtre Orders**



La fenêtre **Orders** s'affiche lorsque l'utilisateur clique sur le bouton **Orders...**de la fenêtre **Customers**.

Pour le client sélectionné dans la fiche de la fenêtre **Customers**, cette fenêtre permet :

- de consulter la liste de ses commandes.
- de sélectionner une commande dans la liste et de l'afficher dans la fiche.
- de créer, modifier, supprimer des commandes.

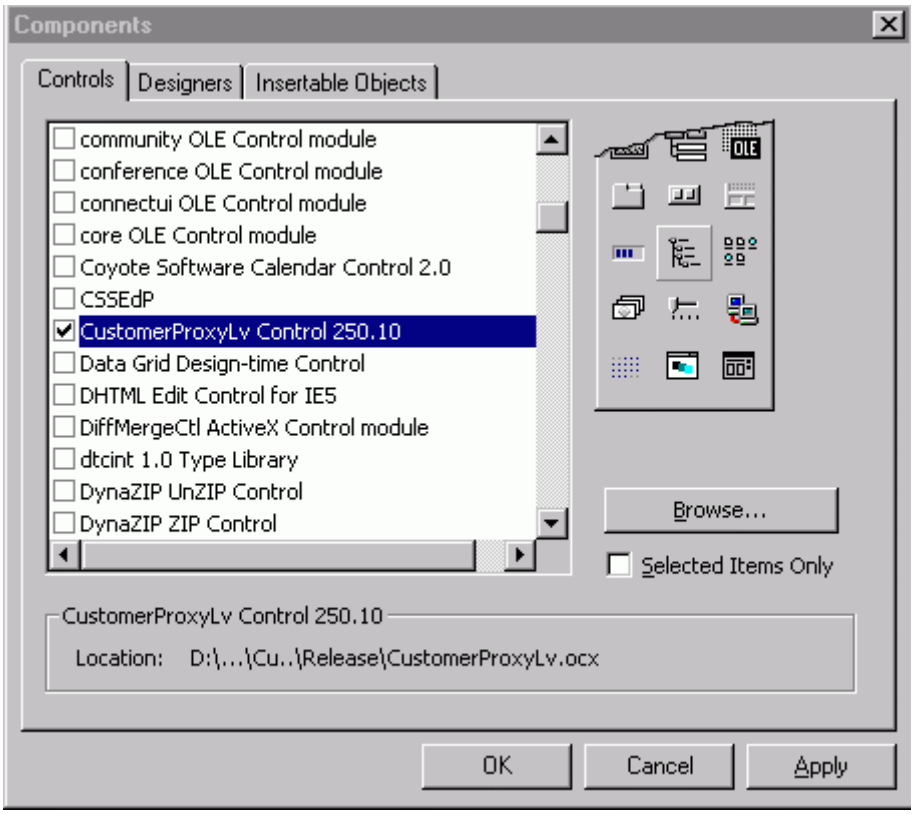
Exemple de Développement en Visual Basic

Le développement de l'interface graphique utilisateur comprend la construction des fenêtres **Customers** et **Orders** ainsi l'interface avec la Proxy COM.

Dans la description de ces différentes étapes, l'utilisation de Visual Basic et de ses fonctionnalités n'est pas détaillée. S'ils ne vous sont pas familiers, reportez-vous à la documentation appropriée.

Intégration de la Proxy COM dans le Projet Visual Basic

Pour intégrer graphiquement une Proxy à la barre d'outils Visual Basic, sélectionnez le menu **Project**, puis **Components**. Lorsque la boîte de dialogue s'ouvre, choisissez la Proxy à intégrer.



☞ Si votre Proxy ne figure pas dans la liste des objets, vous pouvez utiliser directement le bouton **Browse** pour la rechercher en précisant l'objet *nameProxyLV.ocx* qui se trouve dans le répertoire de génération **Release**.

La Proxy sélectionnée s'affiche ensuite dans la barre d'outils. Il ne vous reste plus qu'à l'intégrer à votre Form.

Positionnement d'une Proxy en Mode Conception d'Application

Certains attributs de Proxy tels que le type de localisation, le niveau de trace ou le fichier trace, etc., peuvent être paramétrés lors de la conception de l'application.

Vous pouvez visualiser et modifier ces attributs dans la fenêtre **Properties** accessible via le menu **View**, choix **Properties window**. Sélectionnez la Proxy de votre choix dans la liste des objets.

Sélection et Remplissage de la Grille Représentant l'Attribut Rows

Vous trouverez ci-après un exemple de code utilisé afin de renseigner la grille de type MSFlexGrid obtenue suite à une action de sélection, lorsque l'utilisateur a cliqué sur le bouton **Select**.

```
Private Sub CmdSelect_Click()
    CustomerProxy.selectInstances
    'check if no error has occurred
    processErrors

    ' Put the customer rows in the grid
    For i = 0 To CustomerProxy.getRowsCount - 1
        Set customerDetail = CustomerProxy.getRowsElementAt(i)
        GrdCustomer.AddItem customerDetail.Nuclie & vbTab &
                    customerDetail.Nomcli & vbTab &
customerDetail.Raisoc, i + 1
        Set customerDetail = Nothing
    Next i
End Sub
```

Traitement des Erreurs

Vous trouverez ci-dessous un exemple qui vous permettra de gérer les erreurs potentielles envoyées par la Proxy. Cette fonction peut être appelée après chaque appel d'action capable d'envoyer une erreur.

```
Sub processErrors()
Dim i As Integer
Dim vap_errors As VapCollection
```

```
Dim vap_error As VapError

    Set vap_errors = CustomerProxy.getErrors
    i = vap_errors.Count

    While i > 0
        Set vap_error = vap_errors.Item(i - 1)
        MsgBox vap_error.getLabel, vbInformation, "Error has
occurred"
        Set vap_error = Nothing
        i = i - 1
    Wend

    Set vap_errors = Nothing
End Sub
```

☞ Les objets **VapCollection** et **VapError** sont fournis par l'utilitaire **VapTools.dll**. Ils doivent être référencés dans l'application Visual Basic. Pour cela, sélectionnez le menu **Project**, puis **References**..., et choisissez **VapTools**.

Remplissage de l'Attribute détail

L' exemple ci-après vous montre comment l'attribut **détail** de la Proxy est renseigné lorsqu'une ligne est sélectionnée à partir de la grille de l'attribut **rows**.

```
Private Sub GrdCustomer_Click()
    ' Retrieve the current select row in the detail
    Set customerDetail =
CustomerProxy.getRowsElementAt(GrdCustomer.Row - 1)
    TxtId = customerDetail.Nuclie
    TxtName = customerDetail.Nomcli
    TxtCompany = customerDetail.Raisoc
    CustomerProxy.getDetailFromData customerDetail
    Set customerDetail = Nothing
```

Gestion des erreurs

Il existe quatre types d'erreurs :

- Les erreurs locales,
- Les erreurs serveur,
- Les erreurs système,
- Les erreurs de communication.

☞ Vous trouverez les informations sur toutes les erreurs dans le Chapitre 3: Principes généraux de développement.

Dans un environnement, la gestion des erreurs se fait vial'interface **VAPERROR**. Elle contient les attributs, les actions et les événements qui vous permettent de gérer tous les types d'erreur.

L'interface **VAPERROR** est disponible à partir de la bibliothèque **VapTools** livrée avec le générateur.

☞ Les attributs, actions et événements proposées par l'interface **VAPERROR** sont documentées dans le manuel *Applications eBusiness & Pacbench C/S : Interface de programmation des Proxies*, chapitre *Gestion des Erreurs*.

Gestion de la communication

Ce sous-chapitre contient toutes les informations nécessaires au développement du middleware utilisé par les applications générées avec le module Business ou Pacbench C/S.

☞ Les informations relatives au middleware utilisé par les applications lors de leur déploiement sont données dans le *Guide utilsateur du Middleware*.

Traitement d'une requête

Dans VisualAge, les services du middleware sont exécutés à partir d'un ensemble de classes de communication spécifiques livrées à l'installation du produit.

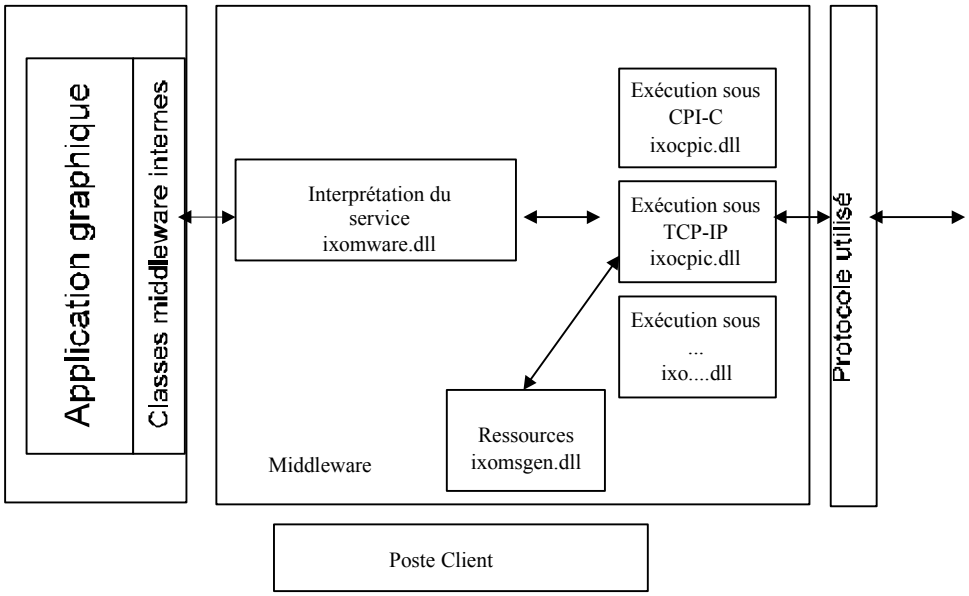
La communication avec le Serveur s'effectue via l'interface **`ServerAdapter`**. Deux implémentations de cette interface existent :

- **`MiddlewareAdapter`** qui permet un accès direct aux DLLs natives (en C++) du Middleware installées localement. Elle permet également de définir le paramètre du contexte de communication (**`location`**, **`userId`**, **`password`**, **`clientEncoding`**…) et son mode de fonctionnement (**`traceLevel`**, **`nbMaxConnection`**, **`connectionCleaningInterval`**, …).
- **`GatewayAdapter`** qui passe par l'intermédiaire du module VAP Gateway ou VAP Relay. Cette classe est particulièrement adaptée aux applications de type Applet. Les paramètres (**`host`**, **`port`**, **`userid`**, **`password`**, **`clientEncoding`**…) qui définissent la communication par l'intermédiaire de **`VapGateway`** ou **`vaprelay`** sont également indispensables. En revanche, les paramètres permettant de définir la communication avec le serveur d'application doivent être définis au niveau du module **`VapGateway`**.

☞ Pour plus d'informations, consultez le *Guide utilisateur du Middleware*.

Lorsqu'un objet **`ServerAdapter`** est instancié, celui-ci créé systématiquement un autre objet de classe **`Requester`** qui met en oeuvre les conditions techniques pour communiquer avec le Serveur, en implémentant notamment les méthodes d'envoi et de réception des messages. Deux implémentations de la classe **`Requester`**, associées respectivement aux classes **`MiddlewareAdapter`** et **`GatewayAdapter`** sont fournies :

- **`NativeRequester`** qui fait appel directement (via **`JNI`**) aux fonctions C des DLLs du Middleware.
- **`GatewayRequester`** qui implémente les messages via **`vaprelay`** ou **`VapGateway`**.



Définition du contexte d'utilisation via l'éditeur de localisation

La gestion de la communication nécessite la définition du contexte d'utilisation du middleware.

Ce contexte est défini via l'outil Editeur de Localisation dans un fichier spécifique appelé **vaplocat.ini.**

Pour plus de détails sur l'interface de l'Editeur de Localisation, consultez l'aide en ligne des outils eBusiness. Vous pouvez également vous reporter au *Guide utilisateur du Middleware*, chapitre *Protocoles Description & Configuration*, vous y trouverez la liste et la fonction des parameters que vous devez définir pour chaque localisation, selon le protocole employé.

Il existe plusieurs façon de l'Editeur de Localisation :

A partir du module eBusiness de Developer workbench

L'Editeur de Localisation se lance à partir de l'onglet "Applications" ou "Dossiers" du workbench.

Lancement à partir du fichier .exe

Lancez le fichier the **vapLocationEditor.exe**.

Vous pouvez paramétrer le lancement de l'Editeur de Localisation via les options suivantes :

`inputfile<INPUT_FILE>` : ce paramètre vous permet d'indiquer le chemin d'accès au fichier de localisation que vous souhaitez utiliser. Ce fichier est soit un fichier de localisation existant, soit un fichier `.gvc` (contenant l'extraction des proxies eBusiness). Toutes les caractéristiques des Moniteurs de Communication sont présentes dans le fichier en entrée.

Au lieu d'initialiser l'éditeur par l'intermédiaire d'un fichier en entrée, vous pouvez lancer l'éditeur en mode expert de façon à créer et modifier les Moniteurs de Communication. Dans ce cas, vous devrez utiliser l'option **-expertmode**.

Déploiement de l'application

Le déploiement d'une application cliente via ActiveX Proxy consiste à installer les runtimes et fichiers nécessaires pour l'exploitation de l'application cliente sur toutes les postes, i.e. pour s'assurer que l'application peut être utilisée en dehors de son environnement, et sur d'autres postes. Vous devez donc installer :

- **MFC42.DLL** (Microsoft),
- **MSVCRT.DLL** (installé en standard avec Windows NT - Microsoft),
- **OLEAUT32.DLL** (langages client installés en standard et compatibles au standard COM – Microsoft),
- Le package middleware,
- **VAPLOCAT.INI**,
- **VAPRUNTIME.DLL** (runtime des Proxies COM générées),
- **VAPTOOLS.DLL** (utilitaire des Proxies COM générées).

☞ Pour activer la fonction multilingue des Proxies, le fichier des messages d'erreur (**VapErrorMsg.pro**) ainsi que le fichier des libellés et valeurs permises spécifique à chaque Proxy (**_<*folder*>_Labels.pro**) doivent être stockés sur le poste de l'utilisateur final.

Chapitre 6 : Index

Cet index ne constitue pas la liste exhaustive des éléments de l'interface publique, que ce soit pour un Client Java ou COM.

☞ Pour obtenir cette liste, consultez le manuel *Applications eBusiness & Pacbench C/S : Interface de programmation des Proxies*.

E

M

P

S