VDOC

Version 2010

Référence SDK

Sommaire

Support technique

Le service VDoc Software

Pour toute question à propos du Kit de développement VDoc ou pour tout complément d'information, n'hésitez pas à vous rendre sur le site support VDoc à l'adresse suivante :

http://www.myvdocservices.net

Vous trouverez sur ce site les derniers « comment faire (HOWTO) » ainsi que les dernières mises à jour des exemples de code.

En vous connectant sur ce site vous profiterez également du **forum** qui traite généralement d'un nombre de cas important et qui fournit des solutions adaptées.

Adresse VDoc Software

L'équipe commerciale de VDoc Software est à votre disposition à l'adresse suivante :

VDoc Software

26 Rue Benoît Bennier

69260 Charbonnières les Bains

Tel : (04) 78 87 29 26 (support technique)

Fax : (04) 78 87 29 00

http://www.vdocsoftware.com

Chapitre 1 : Introduction

Avant-propos

Le Kit de développement (SDK) VDoc constitue un ensemble d'outils graphiques, de configuration et de bibliothèques de développement qui vous permettent à la fois d'enrichir et d'étendre les fonctionnalités livrées en standard par le produit.

VDoc Software ne fournit cette documentation qu'à titre indicatif. Aucun support technique ne peut être envisagé, car les compétences nécessaires à la mise en place des éléments de ce Kit de développement sont à la fois mixtes et importantes.

Toutefois, VDoc Software propose des formations adaptées afin de maîtriser les différents points de ce Kit de développement.

Pour plus de renseignements, veuillez contacter la société VDoc Software.

Pré-requis techniques

Ce document est d'ordre technique et suppose que le lecteur soit familiarisé avec le ou les produits concernés et les technologies environnantes.

Il s'adresse tout particulièrement aux consultants, chefs de projet, développeurs et distributeurs participant à la mise en production de la suite VDoc dans un environnement Client.

Les pré-requis techniques sont les suivants :

- Familiarisation avec les langages de script (Java Script) ;
- Quelques notions HTML, XML ;
- Expérience en programmation Java dans l'environnement serveur (Servlet) ;
- Connaissance de JDBC.

Références documentaires

Les manuels VDoc Software

Guide Designer.pdf
Guides Administrateur VDoc.pdf
Guides Utilisateur VDoc.pdf

Les technologies utilisées

Référence EJB : http://java.sun.com/products/ejb/
Référence JSP: http://java.sun.com/products/jsp/
Référence JDBC : http://java.sun.com/products/jdbc/

Les principaux produits Open source utilisés

Référence sur JBOSS : http://www.jboss.org
Référence sur Tomcat : http://tomcat.apache.org/
Référence sur Rhino : http://www.mozilla.org/rhino/
Référence sur Lucene : http://lucene.apache.org/

Objectifs fonctionnels du produit

VDoc permet aux entreprises de concevoir, déployer et gérer des processus d'entreprise fondés sur des formulaires. Son principal objectif est de leur faire gagner en flexibilité et en productivité.

VDoc permet de créer rapidement une application de gestion de processus sans programmation. Son modélisateur graphique simplifie considérablement la conception et la réalisation des processus tout en préservant des possibilités d'extension.

VDoc offre aujourd'hui les éléments suivants :

- un outil client web de conception de processus très convivial et garantissant l'intégrité des relations entre les éléments graphiques ;
- des Portlets intégrées aux modules Easysite, Portal pour la gestion des activités, l'accès aux documents et applications, et à la recherche ;
- une administration adaptée pour permettre l'intégration d'applications, par connexion JDBC 2.0 par exemple ;
- des agentspour garantir les échéances et l'envoi de mails adaptés à la nature de l'évènement.

Les cas d'utilisation suivants mettent en évidence, de manière simplifiée, les interactions possibles entre les éléments du produit VDoc et leurs acteurs associés :

- un administrateur installe et configure le produit sur une machine.
- un concepteur conçoit un processus, dans le web Designer, et le génère dans une application.
- un responsable fonctionnel paramètre les processus d'une application en affectant les personnes aux rôles.
- un utilisateur accède, via le portail, aux applications puis aux processus associés, et initialise des documents (demandes).

Pour plus d'informations sur l'utilisation de tous ces éléments, reportez vous au Guide administrateur livré sur le CD d'installation.

Présentation du Kit de développement

Le Kit de développement VDoc est composé de plusieurs niveaux :

- des commandes HTTP (Url) personnalisables permettant de réaliser certains traitements comme la création de documents ;
- un environnement de développement d'application web (le même que celui utilisé par VDoc pour la réalisation de son interface graphique) ;
- des extensions et abonnements catégorisés permettant de manipuler dynamiquement les documents et les documents liés sans pour autant forcer le stockage de ces derniers ;
- des ouvertures de code back-office permettant de réaliser des traitements synchrones tels que des imports ou exports depuis ou vers une base de données ;
- des interfaces de programmation (API) situées sur deux niveaux permettant à la fois d'utiliser et respecter les règles de gestion de VDoc, et de modifier ou surcharger le comportement de certaines de ces règles.

Ce Kit de développement doit permettre aux développeurs la reprise des processus opérationnels d'une entreprise, et leur mise en œuvre tels qu'ils existaient au sein de celle-ci.

L'entreprise ne doit pas changer ses processus pour s'adapter à l'outil. Les processus réalisés avec VDoc doivent refléter les processus et règles de l'entreprise sans changer les habitudes de travail.

Avec ce Kit de développement, vous pourrez effectuer les développements suivants :

- intervenir sur les éléments de l'interface web (ajout de nouveaux types de champ sur le document, ajout d'une page dans un assistant d'annulation, etc.) ;
- construire des applications manuelles nécessitant une interaction de l'utilisateur pour simplifier, étendre ou surcharger des fonctionnalités standard (ex. réalisation d'assistants).
- intervenir sur le routage des documents (manipulation des rôles, mise à jour des systèmes externes lors de l'activation d'étape manuelle) ;
- personnaliser la visualisation graphique du document (skin, positionnement des éléments, masquage de champs lors du chargement) ;
- utiliser les abonnements inter-champs ou inter-documents pour éviter la saisie inutile de données (internes ou externes) ;
- piloter les processus (changements d'étape automatiques sur une condition).
- enfin, lancer des processus depuis des applications tierces (ex : utilisation des web Services).

L'utilisation du SDK nécessite des compétences de base sur les technologies web (HTML, XML), et une expérience importante sur l'environnement Java (+Servlet, JSP).

Les points d'entrée suivants sont abordés dans ce manuel :

- L'utilisateur visualise et participe à la vie des documents dans les portlets et des applications.
- Le concepteur met à disposition de nouveaux processus en lançant le générateur avec le Web Designer.
- Le développeur réalise des développements spécifiques qui peuvent répondre aux événements déclenchés par VDoc ou utiliser les APIs de ce dernier pour effectuer des traitements automatiques.
- L'administrateur peut mettre en place des références externes et configurer des relations entre les éléments de VDoc. Il peut également déployer des développements spécifiques sans pour autant modifier la structure du produit.

Chapitre 2 : Architecture

Architecture matérielle

VDoc suit une architecture n-tier. Il est, en effet, composé de plusieurs couches qui ont chacune un rôle particulier.

1er tier

La couche de **présentation** est assurée par le navigateur web. Il est chargé du traitement de l'interaction avec l'utilisateur.

Le web Designer se situe exclusivement sur ce premier tier.

2ème tier

L'authentification, la récupération des données de l'utilisateur connecté (environnement, langue, skin, etc.) et la **production de l'interface graphique** sont assurées par ce deuxième tier.

Le serveur web utilisé est Tomcat. En production, ce dernier est généralement placé derrière un serveur Apache ou IIS pour accélérer la desserte des données statiques tels que les images, et les fichiers HTML.

3ème tier

La couche d'**application** n'est disponible qu'à partir du troisième tier. C'est sur cette couche que sont réalisés tous les traitements applicatifs prenant en compte les règles de gestion définies par l'application VDoc.

Cette couche joue le rôle de tampon entre la présentation et les données.

Le serveur d'application utilisé est JBoss qui fait office de container d'EJB (Entreprise Java Bean).

4ème tier

La manipulation des données est effectuée sur ce tier qui nécessite également la présence d'un serveur d'EJB. JBoss doit être présent sur ce tier.

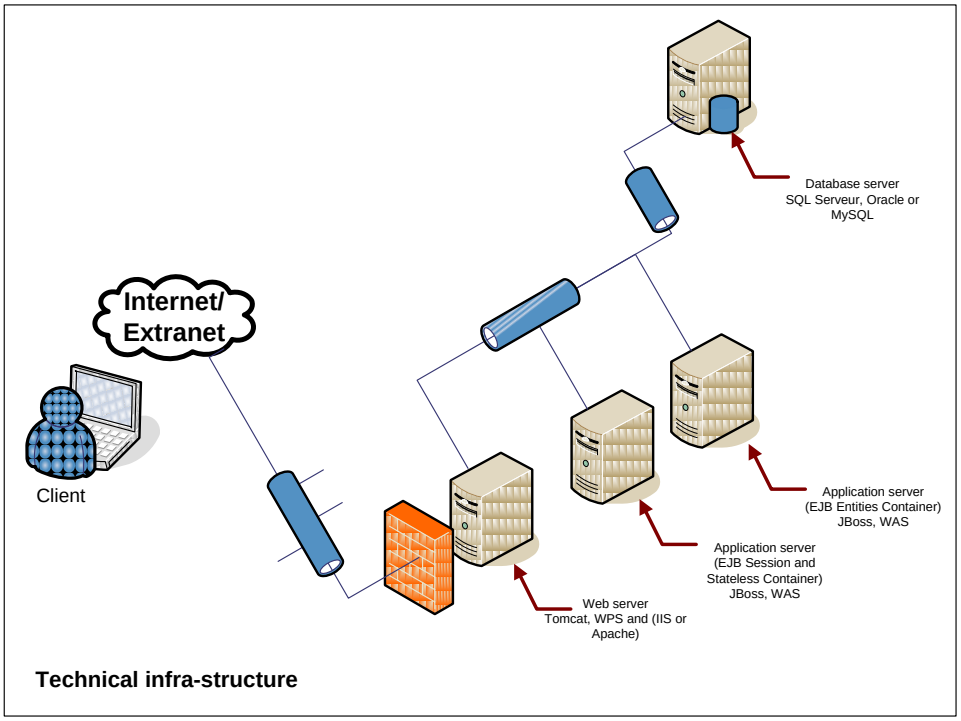
5ème tier

Enfin, sur ce dernier tier sont présentes les données pérennes de l'entreprise et de l'application VDoc.

Ce dernier supporte aujourd'hui quatre moteurs de base de données :

- Microsoft SQL Server
- MySQL
- Oracle
- DB2.

Le diagramme suivant représente l'architecture n-tier supportée par VDoc.



Technical infra-structure

A terme, cette séparation en plusieurs couches simplifie les procédures d'installation de logiciels, le partage d'informations entre applications et implique la réutilisation de composants.

Entre chacune de ces couches, il est possible d'interposer un pare-feu pour renforcer la sécurité et protéger au maximum les données.

Architecture logicielle

L'application VDoc regroupe un ensemble de services métier répartis dans une architecture n-tier. Elle utilise et étend certains services offerts par la couche J2EE essentiellement basés sur les containers de servlets, de portlets et d'EJB (Enterprise Java Bean).

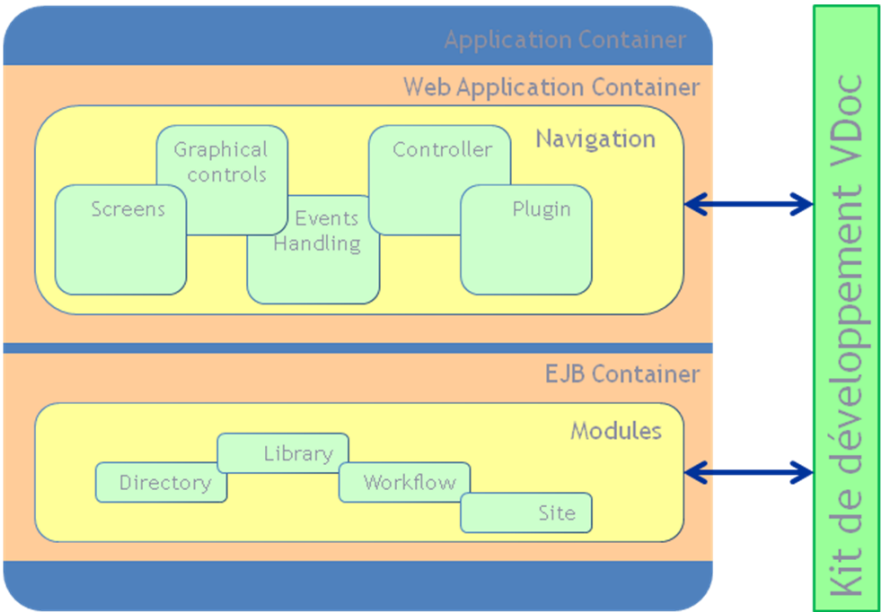
L'application VDoc utilise de manière restreinte des technologies « Open Source » sélectionnées pour répondre à des besoins précis :

- **Rhino** : moteur d'exécution de Java script côté serveur. Cette technologie a été mise en place pour permettre d'étendre les traitements back-office du moteur de workflow directement à l'aide de scripts Java Script.
- **Lucene** : moteur d'indexation.

L'application VDoc intègre nativement l'ensemble des services liés à l'annuaire et au portail : authentification, import et export LDAP et Excel, Portlets, skins.

> *Le module Processus possède un mini annuaire pour sa gestion interne. Nous aurons l'occasion de manipuler les objets de ces deux annuaires dans les prochains chapitres.*

Le diagramme suivant illustre les différents points d'entrée de l'application VDoc et permet de visualiser les diverses couches existantes.



Points d'entrée de l'application

La suite VDoc propose aujourd'hui plusieurs formes d'API qui permettent aux développeurs d'ajouter du fonctionnel, d'intégrer le produit avec d'autres systèmes, ou encore de rendre les documents plus dynamiques.

API Graphique

Ensemble de classes permettant de créer de nouvelles interfaces graphiques basées sur les modèles standard d'écrans. Toute la partie graphique est contenue dans le projet VDocUINavigation.jar

API SDK

Ensemble de classes simplifiant la manipulation des APIs actuelles et étendant le système à d'autres applications. Cette API est une uniformisation des APIs multi-produits.

API SDK Client

Ensemble de classes clientes simplifiant la manipulation des éléments de la suite VDoc.

Couches de l'application

Couche d'interface graphique

Cette couche représente le framework d'interface graphique utilisé dans de nombreux modules de l'application VDoc. Elle est entièrement décrite par le biai de fichiers XML. En effet, ces fichiers recensent tous les éléments suivants :

- la navigation dans l'administration ;
- la navigation dans les applications : processus, site ;
- la navigation dans les portlets : portal, site ;
- l'ensemble des assistants (générateur, import / export d'application, changement d'étape, etc.).

Cette couche est également prévue pour permettre de réaliser de nouveaux écrans avec beaucoup de simplicité.

Couche des services

L'API SDK donne accès à la partie métier de l'application VDoc.

Couche d'accès aux données

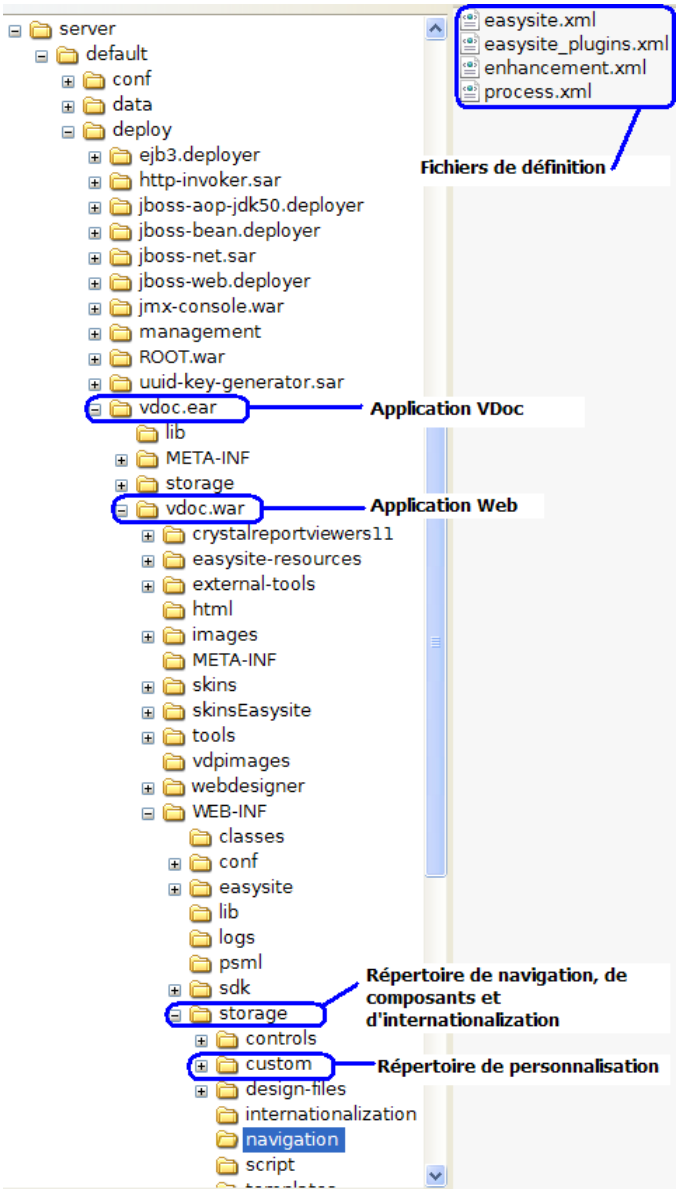
L'application VDoc utilise un mode de stockage relationnel (Microsoft SQL Serveur 2000, MySQL, Oracle).

Les données sont accessibles via les API.

Structure des dossiers

Arborescence

La structure des dossiers de la suite VDoc intègre JBoss et Tomcat (jbossweb-tomcat.sar).
Vous noterez sur ce schéma les pointeurs importants pour le développement.



Le répertoire de déploiement des Jars spécifiques est **server/default/lib**. Il vous permet de déposer vos propres bibliothèques de classes (Jar). Une fois présentes dans ce répertoire, elles seront prises en compte au prochain démarrage du serveur VDoc.

Pour plus d'informations sur la structure des dossiers reportez-vous aux Guides Administrateur VDoc joints avec le CD.

Chapitre 3 : Framework de navigation

L'interface graphique de VDoc est basée sur un modèle dynamique de production d'interface.

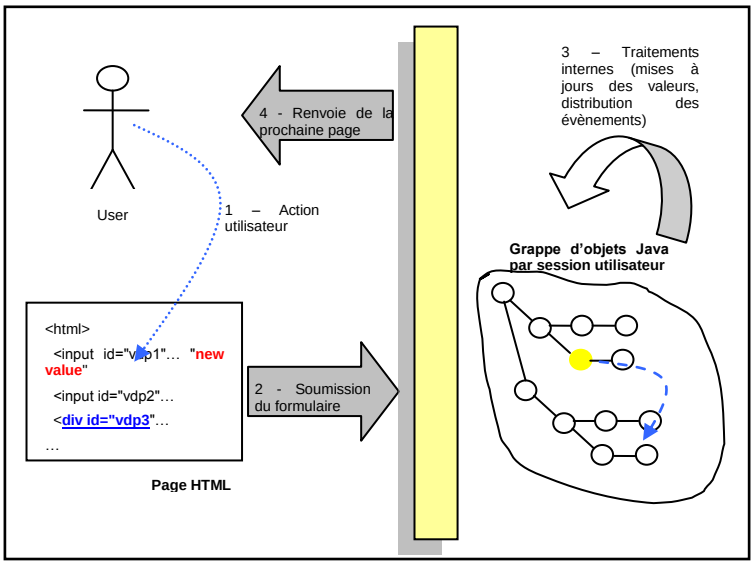
Elle s'appuie sur une combinaison de composants modèles HTML, de composants serveur (Java), le tout s'exécutant côté serveur, pour produire du HTML qui fonctionne sur les navigateurs du marché tels que Firefox et Internet Explorer. L'interface graphique se construit à partir d'une page HTML unique qui se complète au fur et à mesure de la navigation.

Le principe adopté est celui du mécanisme du « **post-back** » qui implique, côté serveur, la présence des objets Java reflétant le contexte de l'interface graphique perçue par l'utilisateur. Ainsi, chaque objet Java sur le serveur est repérable grâce à son identifiant généré de manière unique (dans la page). Ces identifiants, associés aux objets Java du serveur, sont donc inscrits dans les balises du document HTML.

Lorsque l'utilisateur clique sur un élément de la page (ex : un bouton), le formulaire HTML est soumis sur le serveur. Le système de navigation parcourt l'ensemble des « ID » envoyés et active les objets Java présents dans la session HTTP de l'utilisateur. D'autres paramètres sont également passés permettant de retrouver l'élément cliqué, l'évènement déclenché (ex. onClick, onOk, etc.) et le contexte de la page.

Le schéma ci-dessous présente en quatre étapes ce qui se produit lorsque l'utilisateur réalise une action :

1. L'utilisateur saisit certains champs du document, puis clique sur un bouton d'action ;
2. Le formulaire est transmis au serveur ;
3. Le système retrouve l'ensemble des objets Java correspondant aux identifiants de la page HTML postés. Il exécute les mises à jour et déclenche les évènements nécessaires ;
4. Le système renvoie la page nouvellement construite.

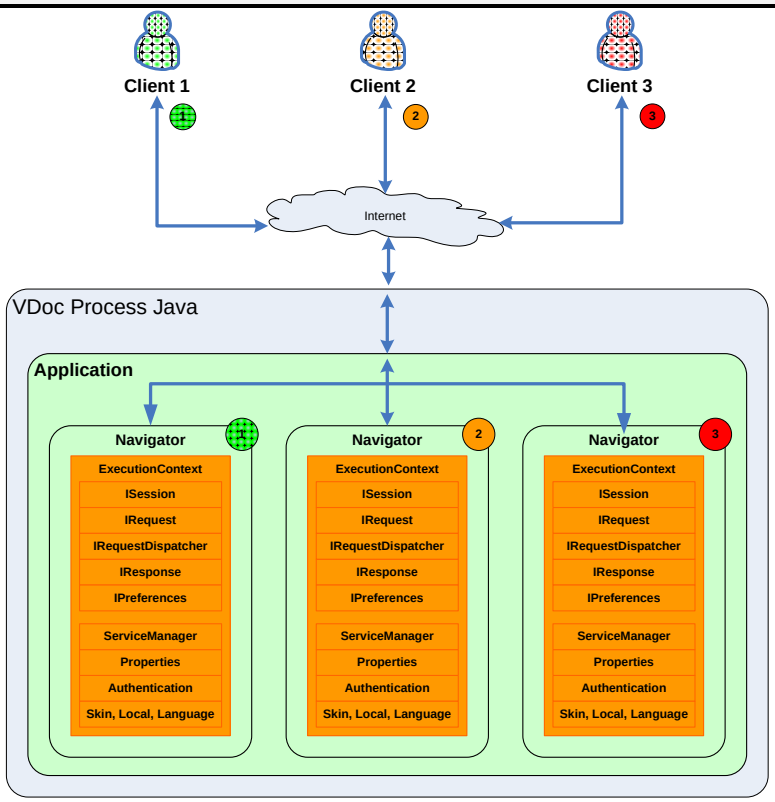


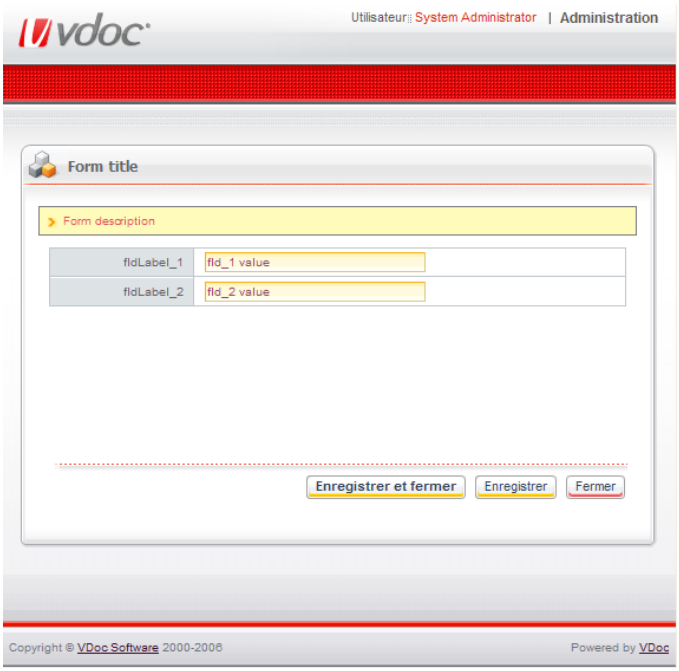
Les étapes de la navigation

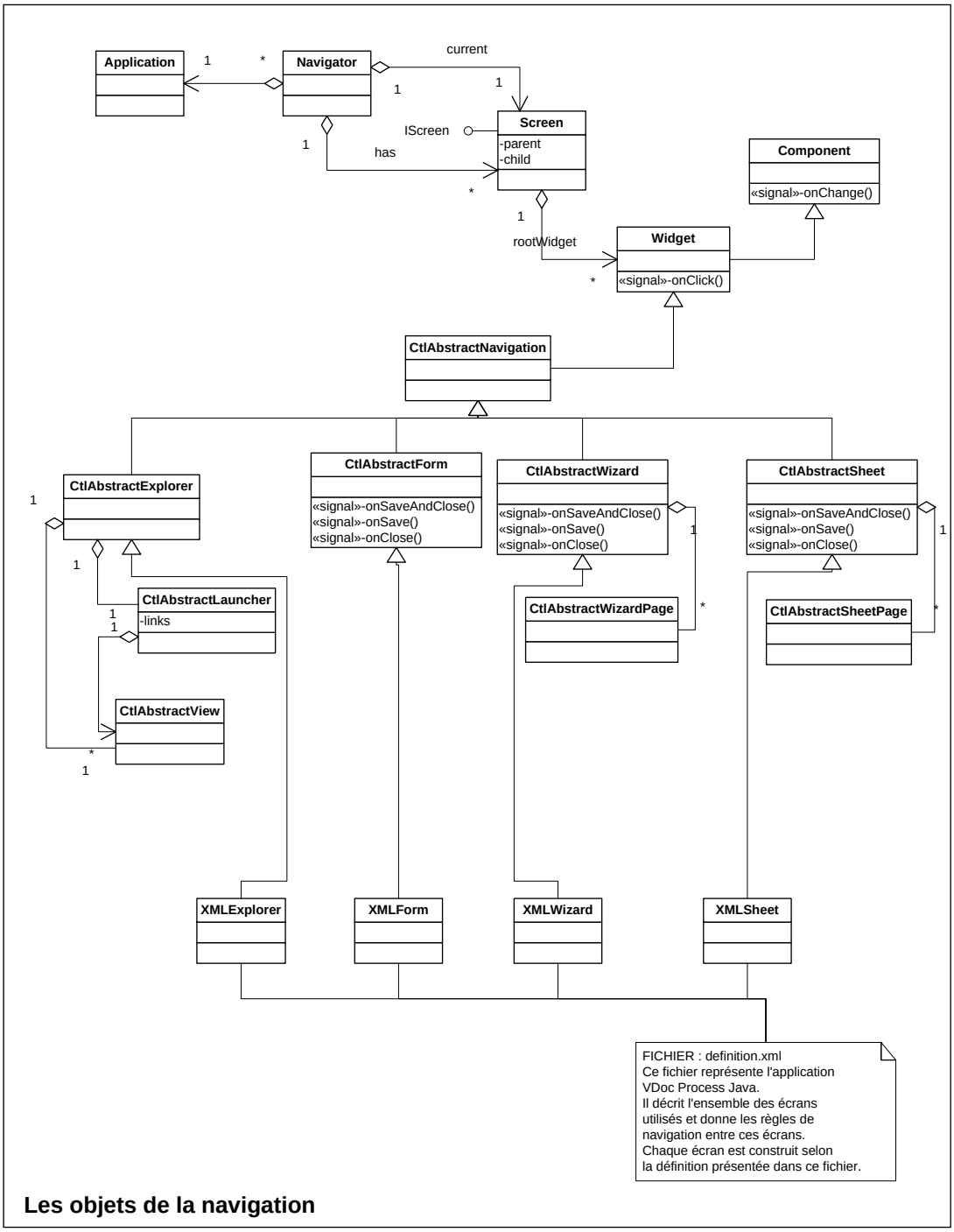
Le tableau suivant décrit les étapes de navigation dans le framework UI de VDoc.

Tableau

Etape	Description
Le serveur VDoc démarre	A l'initialisation de la servlet principale de VDoc, un premier objet est créé. Il se nomme **com.axemble.vdp.ui.framework.foundation.Application**. Cet objet constitue un point central de partage d'éléments entre toutes les sessions HTTP.
Un utilisateur demande une ressource de l'application VDoc.	A la première réception d'une requête issue du navigateur web, le système va créer un objet nommé **com.axemble.vdp.ui.framework.foundation.Navigator** associé au navigateur web. C'est le représentant du navigateur web sur le serveur. Cet object **Navigator** sera disponible pendant toute la durée de vie de la session HTTP et sera accessible pour tous les échanges client web / serveur.
Un objet de contexte d'exécution est associé à la requête.	A la réception d'une requête, un objet de contexte est créé. Il correspond au « contexte d'exécution ». Cet objet, disponible pour toute la durée de la requête HTTP, implémente l'interface nommée : **com.axemble.vdp.ui.framework.runtime.IExecutionContext**. *L'utilisation d'une interface est justifiée par le fait que le système VDoc ait besoin de fonctionner, de la même façon, dans les deux environnements JBoss et WebSphere Portal Server (container de Servlets pour JBoss et de Portlets pour WebSphere Portal Server). De ce fait, l'interface **IExecutionContext** encapsule un grand nombre des méthodes communes aux objets HTTP de ces deux types de container.* L'objet **ExecutionContext** permet d'accéder à deux types d'éléments issus du contexte dynamique. Il tient en mémoire des objets HTTP suivants : • ISession : représentant de la session HTTP ; • IRequest : représentant de la requête HTTP ; • IRequestDispatcher : représentant de l'objet RequestDispatcher HTTP ; • IResponse : représentant de la réponse HTTP ; • IPreferences (exclusivement pour les Portlets JSR#168). Il tient également des informations liées à l'exécution dans VDoc. Depuis cet objet **ExecutionContext** les objets suivants sont accessibles : • ServiceManager : interface d'accès aux services et gestionnaire Process ; • Properties : toutes les propriétés spécifiées à l'initialisation ; • Authentication : objet d'authentification ; • Et l'objet Navigator : représentant du navigateur web.

Etape	Description
	
Le système construit le composant racine à partir des éléments de la requête	Le système VDoc examine les paramètres passés dans l'URL et demande à l'objet **Navigator** de construire un écran implémentant l'interface nommée **com.axemble.vdp.ui.framework.foundation.screens.IScreen**. Cet objet écran (**Screen**) tient en mémoire un objet de navigation unique implémentant l'interface nommée **com.axemble.vdp.ui.core.providers.INavigationListenerImpl**. Cet objet de navigation, qui représente l'écran affiché, se base sur un modèle de représentation. Les modèles de représentation d'écran sont nécessaires pour la construction de l'interface graphique. Plusieurs modèles sont proposés en standard : • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractExplorer** : utilisé dans l'administration et l'application. Il permet d'afficher des liens avec des vues associées. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractForm** : c'est la représentation la plus simple d'un formulaire avec une section composée de d'entrées de champs et des boutons d'action. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractSheet** : c'est un formulaire un peu plus évolué muni d'onglets. Pour chacun des onglets des entrées de champs et des boutons d'action peuvent être définis. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractWizard** : c'est un assistant. Il est composé de pages contenant des entrées de champs et des boutons d'action. Les boutons **Précédent** et **Suivant** permettent de naviguer entre les pages. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractDocument** : c'est le modèle de représentation du document processus. Il s'appuie sur la description XML des formulaires définis dans l'administration. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractSelector** : c'est l'élément abstrait des sélecteurs. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractSingleSelector** : sélecteur d'élément simple utilisé pour les propriétés, les acteurs et les rôles. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractMultipleSelector** : sélecteur multiple d'éléments utilisé pour les propriétés, les acteurs et les rôles. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractJsp** : c'est la représentation qui permet d'embarquer une page JSP. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractPortlet** : c'est le modèle de représentation des Portlets utilisé dans le contexte de Websphere Portal Server. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractView** : c'est le modèle de représentation des vues. • **com.axemble.vdp.ui.framework.composites.base.CtlAbstractGroup** : c'est le modèle de représentation des groupes d'écrans (exemples : [form + view] ou [sheet + view]).

Etape	Description
	Ces modèles permettent la création typée d'écrans web et la manipulation des éléments disponibles tels que les boutons, les onglets, les entrées de champs. **Exemple** Le code présenté ci-dessous construit un formulaire simple composé d'un en-tête, un pied de page, et une zone de contenu dans laquelle sont présents deux champs TextBox. `// création d'un nouvel écran` `IScreen screen = new Screen();` `// création d'un formulaire basé sur un modèle simple` `CtlForm fooForm = new CtlForm("fooForm");` `fooForm.setLabel( new CtlText("Form title") );` `fooForm.setInformation(new CtlText("Form description"));` `// ajout des entrées de champs. Chaque entrée est constituée` `// d'un titre et d'un composant graphique (CtlTextBox, CtlDate, CtlComboBox, etc.)` `fooForm.addEntry( "fld 1",` `new CtlText("fldLabel_1"),new CtlTextBox("fld_1 value") );` `fooForm.addEntry( "fld_2",` `new CtlText("fldLabel_2"), new CtlTextBox("fld_2 value") );` `// positionnement du formulaire en tant que composant principal de l'écran` `screen.setRootWidget( fooForm );` `// positionnement l'écran courant du navigateur` `Navigator.getNavigator().setCurrentScreen( screen );` L'exécution de ce code produit l'écran suivant :  Les écrans de VDoc sont constitués d'un certain nombre d'objets graphiques tels que l'en-tête, le pied de page et une zone de contenu « dynamique ». Cette zone « dynamique » s'alimente à partir de l'objet désigné comme composant racine. Dans la plupart des écrans de VDoc le composant racine correspond à l'une des classes suivantes : • XMLExplorer : permet de construire des écrans de type explorateur. Ces écrans sont constitués d'un « lanceur » (partie gauche de l'écran) et de « vues » (partie droite). Chaque vue est associée à un lien du lanceur. Ce type d'écran est utilisé principalement dans l'administration et dans l'application ; • XMLForm : permet de construire un formulaire très simple. Une liste de champs est décrite ; • XMLSheet : permet de construire un formulaire à onglet ; • XMLWizard : permet de construire un assistant composé de pages ; • XMLDocument : permet de contruire un document process ; • XMLView : permet de contruire une vue ; • XMLGroup : permet de contruire un écran composite composé de plusieurs modèles d'écran ;

Etape	Description
	• XMLJsp : permet d'intégrer une page JSP ; • XMLPortlet : permet de construire un portlet en respectant la spécification JSR 168. En général, toutes les classes préfixées par « XML » et implémentant l'interface **INavigationListenerImpl** ont la faculté de se construire à partir d'un document XML de définition. En réalité, plusieurs documents de définition peuvent être créés (déclarés). Ils sont fusionnés au chargement du serveur ou lors de l'exécution d'un rafraîchissement manuel depuis l'administration Gestion des processus (Configuration).
Construction de l'interface graphique à partir du document XML de définition et des paramètres de requête.	Le schéma suivant présente les principaux objets qui permettent de mettre en œuvre la navigation dans VDoc.  **Les objets de la navigation** Ce document XML de définition est présent sous le dossier « vdoc.ear/vdoc.war/WEB-INF/storage » de l'application VDoc. Il se nomme « definition.xml ». Il contient l'ensemble de la description XML des écrans présents dans l'application VDoc. Cette définition est valable pour tout le serveur VDoc. Dans ce fichier XML, chaque écran est identifié par les attributs « name » et « action ». Lorsqu'une requête est traitée sur le serveur les paramètres « class » et « method » sont évalués. Ces paramètres sont comparés aux attributs « name » et « method » de l'écran et l'écran trouvé est construit.

Etape	Description
	Exemple Une requête de ce type http://localhost:8080/vdoc/process?class=DEMO&method=action1 aboutit à la lecture de l'élément XML présenté ci-dessous (partie du document XML de définition personnalisé) et à la construction de l'écran ainsi identifié, basé sur le modèle de représentation associé. Dans le cas de l'exemple, le modèle de représentation associé à la balise « form » est un formulaire simple. L'écran final sera construit avec deux champs TextBox. `<form name="DEMO" action="action1" label="LG_VIEW_PROP"` `  information="LG PUT VIEW INFO"` `  provider="com.axemble.education.providers.demo.DemoFormProvider">` `  <fields>` `  <field name="fldLabel_1" label="LG_LABEL1"` `  ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField"` `  mandatory="false" />` `  <field name="fldLabel_2" label="LG_LABEL2"` `  ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField"` `  mandatory= »false » />` `  </fields>` `</form>`
Interaction avec l'écran créé	Une fois l'écran construit, le système de navigation de VDoc prévoit l'interaction à l'aide d'un objet appelé « provider » implémentant l'interface **com.axemble.vdp.ui.core.providers.IProvider**. C'est au travers de cet objet Java que le développeur pourra renseigner les données de l'écran et interagir avec l'interface graphique. L'association entre l'objet « provider » et l'écran est réalisée par le biais de l'attribut « provider » présent dans le fichier de définition XML des écrans. Chaque provider est spécialisé selon le type d'écran manipulé. **Liens hiérarchiques entre les interfaces de « provider »** `[IProvider]` `  [IDocumentProvider] - gestion des documents génériques (GenericDocument)` `  [IFormProvider] - gestion des formulaires simples` `  [ISheetProvider] - gestion des formulaires à onglets` `  [IProcessDocumentProvider] - gestion des documents process (CoreDocument)` `  [IWizardProvider] - gestion des assistants` `  [IExplorerProvider] - gestion des écrans type explorateur` `  [ILauncherProvider] - gestion de la partie droite de l'écran explorateur` `  [IViewProvider] - gestion de toutes les vues` `  [IFilterableViewProvider] - gestion des vues filtrables` `  [IPortletProvider] - gestion des portlets` `  [IJspProvider] - gestion des écrans basés sur des JSP` `  [ISelectorProvider] - gestion des sélecteurs`

Les providers de document

Un provider est une classe Java qui peut être associée à un type d'écrans graphiques tels qu'un formulaire simple, un formulaire multi-onglets, un assistant, un explorateur, etc. Cette association est mise en place dans le fichier de définition XML.

Depuis une classe « provider », il est possible d'altérer le contenu et l'aspect graphique d'un écran car tous les objets graphiques tels que les zones de boutons, les entrées de champs, le titre, la zone information sont accessibles.

Provider de document : classes mises en œuvre

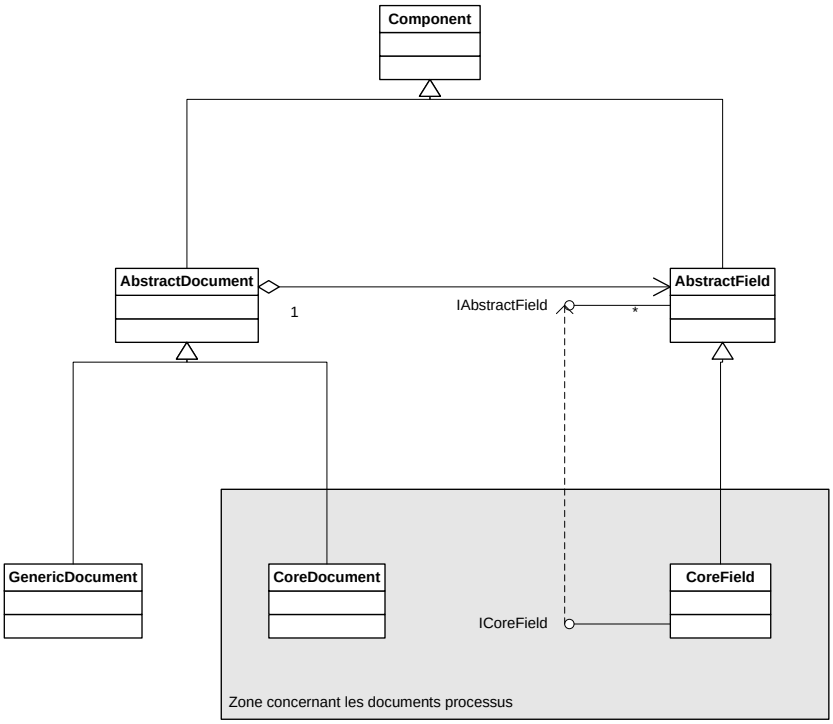
Dans le cas particulier d'un provider dit « de document », le provider reçoit en paramètre un document interne lui permettant de gérer le stockage des valeurs de champs associés au formulaire. Ainsi, le provider peut affecter les valeurs de ces champs à l'initialisation du formulaire en plus de compléter dynamiquement l'aspect graphique.

La classe modèle

Une classe « modèle » permettant de créer un écran de type form, sheet ou wizard (ex. CtlAbstractForm).

La classe de document

Une classe représentant le document : **AbstractDocument**. Cette classe maintient en mémoire une liste de champs correspondant aux « field » décrits dans le document XML de définition des écrans.



De cette classe de document, deux classes en dérivent :

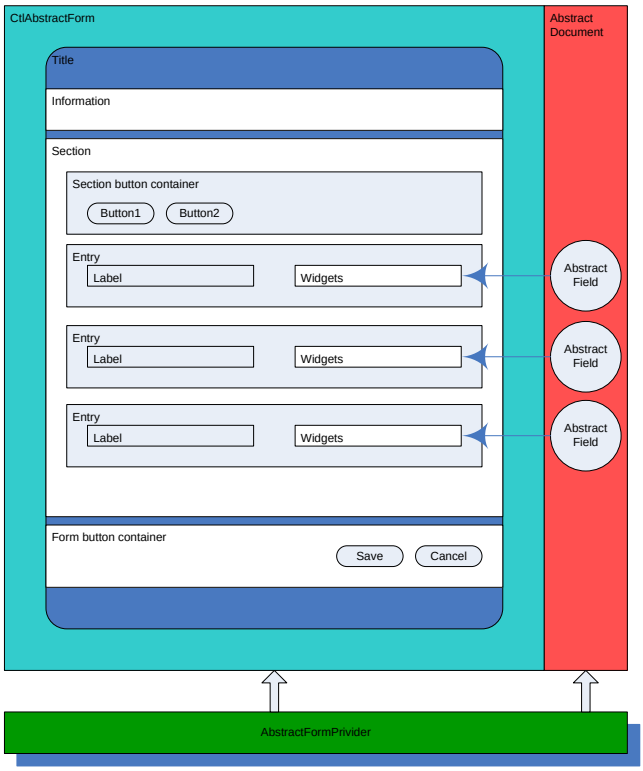
- **GenericDocument** : utilisée dans le cas des écrans de l'application, de l'administration et des assistants autour du document de processus.
- **CoreDocument** : utilisée exclusivement dans le cas des documents de processus.

La classe fournisseur

Une classe permettant d'alimenter les données du formulaire et, éventuellement, de jouer sur les éléments graphiques (ajouter des boutons, des entrées de champ, masquer des éléments, etc.). Exemple : AbstractFormProvider.

Schéma de principe

Le schéma suivant s'appuie sur un formulaire simple pour présenter un exemple de mise en œuvre des divers éléments constituant un formulaire.



Le provider peut intervenir à la fois sur l'aspect graphique et sur le document. Chaque champ du document est associé à un élément graphique qui peut être une succession de composants graphiques (widget).

Exemple

Un champ de type date peut être représenté sous la forme d'une zone de saisie, d'un bouton d'accès à un calendrier, et d'un bouton pour vider la valeur.

Cycle de vie des providers de document

Le cycle de vie des providers de formulaire est quasi identique pour la plupart des types de formulaires.

Construction du provider

Au chargement d'un formulaire, la classe provider associée est instanciée et reçoit trois paramètres :

- **INavigateContext** : contexte de navigation permettant de récupérer les paramètres envoyés ;
- **AbstractDocument** : le document dans lequel seront déversées les valeurs de champs ;
- Et la classe modèle du type d'écran sélectionné (CtlAbstractForm, CtlAbstractWizard, CtlAbstractSheet, CtlAbstractDocument).

Exemple de constructeur d'un provider de formulaire

```
public DemoFormProvider( INavigateContext context, AbstractDocument document, CtlAbstractForm
abstractForm )
{
    super( context, document, abstractForm );
}
```

Alimentation des données

Avant que le formulaire ne soit chargé, une première méthode est appelée pour aller chercher de la donnée afin d'alimenter les champs du formulaire. C'est la méthode prepareData(). Ensuite, la méthode load() est appelée pour chacun des champs construits du formulaire (définis dans le document de définition XML). C'est à ce moment que les objets « AbstractField » sont créés et accessibles.

Extrait des méthodes appelées avant le chargement du formulaire

```
public boolean prepareData()
{
    // get some data here…
    return true;
}

public boolean load( AbstractField value, Element element )
{
    // set the value here…
    return true;
}
```

Chargement du formulaire

Une fois le formulaire chargé, c'est-à-dire juste avant qu'il n'apparaisse à l'écran, la méthode readyState() est appelée. C'est l'endroit indiqué pour modifier l'interface.

Extrait d'un code mettant en œuvre des modifications telles que l'assignation de champs, l'ajout d'entrée et de bouton en dynamique juste avant le rendu du formulaire

```
public void readyState()
{
    // set field value
    AbstractField field1 = this.getDocument().getAbstractFieldByName("fldLabel1");
    field1.setValue( "Valeur du jour" );

    // add entry
    this.getForm().addEntry("fldDynamicEntry",new CtlText("Dynamic"),
      new CtlButton("myButton",new CtlText("Mon bouton")));

    // add button
    CtlButton button=this.getForm().createButton("message",new CtlLocalizedText("LG_MESSAGE"),
    false, null);

    this.getForm().getButtonsContainer().addFirst( button );
    button.addActionListener( new ActionListener()
    {
            public void onClick( ActionEvent event )
            {
                    getNavigator().showAlertBox( "Bonjour le Monde !!!" );
            }
    } );

    super.readyState();
}
```

Chargement d'une section

Pour les formulaires de type « sheet » ou « wizard », une autre méthode peut être surchargée pour intervenir lors de la construction de la page.

Extrait d'un code mettant en œuvre l'activation d'une page

Sur cet exemple, le provider vérifie, pour les pages nommées « page3 » et « page4 » que le contenu de la section soit une vue. Si tel est le cas, le provider associé à cette vue est sollicité pour alimenter le contenu.

```
public void activate( ISection section )
{
    if ( "page3".equals( section.getName() ) || "page4".equals( section.getName() ) )
    {
            if ( section.getContent() instanceof CtlAbstractView )
            {
                    CtlAbstractView view = (CtlAbstractView)section.getContent();
                    IViewProvider provider = view.getProvider();
                    provider.getColumns();
                    provider.getItems();
            }
    }

    super.activate( section );
}
```

Les éléments communs du document XML de définition

Dans cette section sont présentés les différents éléments qui peuvent être utilisés dans les écrans. Pour plus de précision sur la description de ces champs, reportez-vous à la section « Les champs du formulaire ».

L'élément « field »

L'élément « field » est une balise fille qui peut être utilisée sur tous les écrans de type formulaire. Selon la nature ou le type de champ souhaité, des classes spécialisées doivent être utilisées. Egalement, pour chaque type de champ un paramétrage spécifique est défini.

Les champs "numériques"

```
<field name="fldInteger" label="Integer" integer-value="7"
ctrl="com.axemble.vdp.ui.core.document.fields.IntegerField" mode="write" throw-events="true" />
<field name="fldInteger" label="Integer"
ctrl="com.axemble.vdp.ui.core.document.fields.IntegerField" mode="read" />
<field name="fldLong" label="Long" long-value="8"
ctrl="com.axemble.vdp.ui.core.document.fields.LongField" mode="write" />
<field name="fldLong" label="Long" ctrl="com.axemble.vdp.ui.core.document.fields.LongField"
mode="read" />
<field name="fldFloat" label="Float" float-value="9.5"
ctrl="com.axemble.vdp.ui.core.document.fields.FloatField" mode="write" />
<field name="fldFloat" label="Float" ctrl="com.axemble.vdp.ui.core.document.fields.FloatField"
mode="read" />
<field name="fldDouble" label="Double" double-value="10.5"
ctrl="com.axemble.vdp.ui.core.document.fields.DoubleField" mode="write" />
<field name="fldDouble" label="Double" ctrl="com.axemble.vdp.ui.core.document.fields.DoubleField"
mode="read" />
```

Les champs "date"

```
<field name="fldDate" label="Date" ctrl="com.axemble.vdp.ui.core.document.fields.DateField"
mode="write" throw-events="true" />
<field name="fldDate" label="Date" ctrl="com.axemble.vdp.ui.core.document.fields.DateField"
mode="read" />
<field name="fldTime" label="Time" long-value="43200000" hour-format="12"
ctrl="com.axemble.vdp.ui.core.document.fields.TimeField" mode="write" />
<field name="fldTime" label="Time" ctrl="com.axemble.vdp.ui.core.document.fields.TimeField"
mode="read" />
<field name="fldDateTime" label="DateTime"
ctrl="com.axemble.vdp.ui.core.document.fields.DateTimeField" mode="write" />
<field name="fldDateTime" label="DateTime"
ctrl="com.axemble.vdp.ui.core.document.fields.DateTimeField" mode="read" />
<field name="fldPeriod" label="LG_PERIOD"
ctrl="com.axemble.vdp.ui.core.document.fields.PeriodField" mode="write" />
<field name="fldPeriod" label="LG PERIOD"
ctrl="com.axemble.vdp.ui.core.document.fields.PeriodField" mode="read" />
<field name="fldPeriodTime" label="LG_PERIOD"
ctrl="com.axemble.vdp.ui.core.document.fields.PeriodTimeField" mode="write" />
<field name="fldPeriodTime" label="LG PERIOD"
ctrl="com.axemble.vdp.ui.core.document.fields.PeriodTimeField" mode="read" />
```

Les champs "texte"

```
<field name="fldTextBox" label="TextBox"
ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" throw-events="true" />
```

```
<field name="fldTextBox" label="TextBox"
ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mode="read" />
<field name="fldPassword" label="Password"
ctrl="com.axemble.vdp.ui.core.document.fields.PasswordField" throw-events="true" />
<field name="fldPassword" label="Password"
ctrl="com.axemble.vdp.ui.core.document.fields.PasswordField" mode="read" />
```

Les champs “liste”

```
<field name="fldRadioGroup" label="RadioGroup" string-value="un"
list="un;deux;trois;quatre;cinq;six;sept" nbofcolumns="3" fillingmode="horizontal"
ctrl="com.axemble.vdp.ui.core.document.fields.RadioGroupField" />
<field name="fldCheckBoxGroup" label="CheckBoxGroup" throw-events="true" string-values="un;trois"
list="un;deux;trois;quatre;cinq;six;sept" nbofcolumns="3" fillingmode="vertical" allowreset="true"
allowselectall="true" ctrl="com.axemble.vdp.ui.core.document.fields.CheckBoxGroupField" />
<field name="fldCheckBox" label="CheckBox" throw-events="true" boolean-value="true"
ctrl="com.axemble.vdp.ui.core.document.fields.CheckBoxField" />
<field name="fldComboBox" label="ComboBox" string-value="A" list="A;B;C;D"
ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField" mode="write" />
<field name="fldComboBox" label="ComboBox"
ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField" mode="read" />
```

Les champs “pièces jointes”

```
<!-- <field name="fldFiles" label="MultipleFile" value="files"
ctrl="com.axemble.vdp.ui.core.document.fields.MultipleFileField" min-file-number="2" max-file-
number="3" min-file-size="1024" max-file-size="71680" max-file-name-length="128" min-total-
size="2048" max-total-size="143360" supported-extensions=".txt;.xml"/> -->
<field name="fldFiles" label="MultipleFile" value="files"
ctrl="com.axemble.vdp.ui.core.document.fields.MultipleFileField" mandatory="true" max-file-
number="1" supported-extensions="*" />
```

Les champs “sélecteur JSP”

```
<field name="fldJSP" label="jsp" type="jspsample"
ctrl="com.axemble.vdp.ui.core.document.fields.JSPBrowserField" />
<field name="fldJSP" label="MAP" ctrl="com.axemble.vdp.ui.core.document.fields.MapTableField"
separator="|" columns="Col1|Col2|Col3" />
<field name="fldJSP" label="MAP" ctrl="com.axemble.vdp.ui.core.document.fields.MapTableField"
separator="|">
  <subfield mode="read" required="no" property="Col1" ctrl="text" labelid="LG COL1" />
  <subfield mode="read" required="no" property="Col2" ctrl="text" labelid="LG COL2" />
  <subfield mode="read" required="no" property="Col3" ctrl="text" labelid="LG COL3" />
</field>
```

Les champs “sélecteur d’annuaire”

```
<field name="fldDirectories" label="MultipleDirectoryUser" type="user"
ctrl="com.axemble.vdp.ui.core.document.fields.MultipleDirectoryField" />
<field name="fldDirectory" label="SingleDirectoryGroup" type="group"
ctrl="com.axemble.vdp.ui.core.document.fields.SingleDirectoryField" />
<field name="fldFilecenter" label="SingleFilecenter" type="filecenter"
ctrl="com.axemble.vdp.ui.core.document.fields.SingleDirectoryField" />
```

Les champs “sélecteur d’écran”

```
<field name="fldEmbedder" label="fldEmbedder"
ctrl="com.axemble.vdp.ui.core.document.fields.ScreenEmbedderField" mode="write"
screen="jsp_selector" method="select" />
```

Les attributs communs à tous les écrans

Le tableau suivant recense les attributs communs à tous les écrans.

Attribut	***Description***
Name	Nom système de l’écran
Action	Nom système de l’action associée à l’objet représenté par l’écran. Exemple de couples de valeurs (nom, action) : • (treatment, edit) ; • (treatment, abort) ; • (treatment, remind).
Label	Identifiant de traduction correspondant au titre de l’écran
Information	Identifiant de traduction correspondant à la zone d’information de l’écran

provider	Provider associé à l'écran.

Les balises d'écran du document XML de définition

Dans cette section sont présentés les différents modèles d'écran disponibles en standard dans la suite VDoc et, pour chacun d'eux, la définition des attributs et des balises filles.

L'élément « form »

L'interface graphique de la balise « form » est construite par la classe modèle suivante : **com.axemble.vdp.ui.framework.composites.xml.XMLForm**.

La classe provider à utiliser doit étendre **com.axemble.vdoc.sdk.providers.BaseFormProvider**.

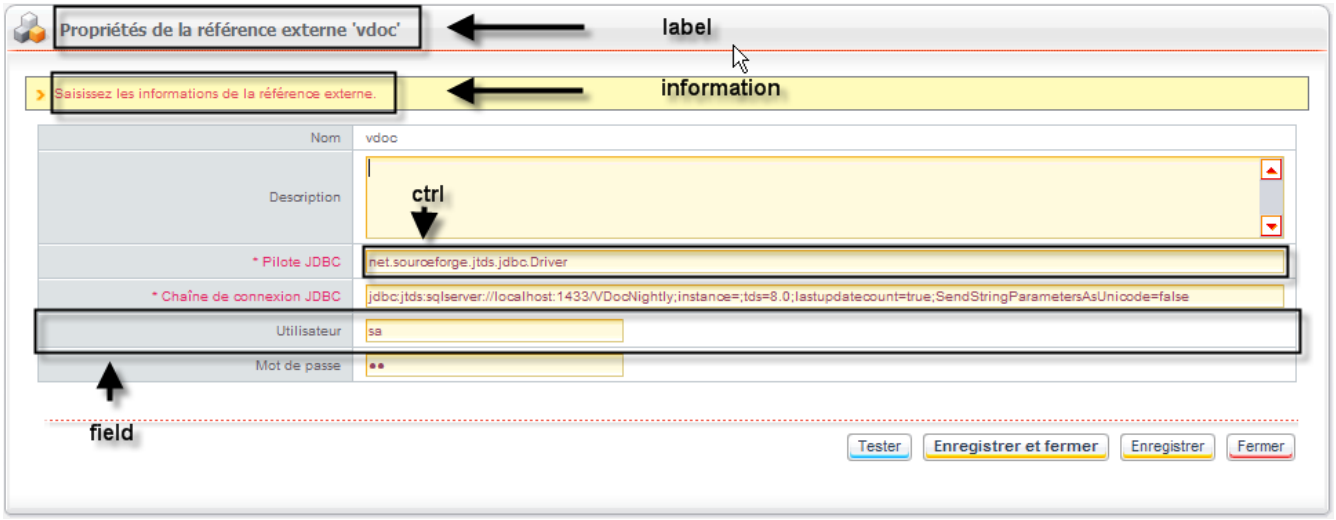
<fields>

Container d'éléments « field »

Exemple

```
<form name="DEMO" action="action1" label="LG VIEW PROP" information="LG PUT VIEW INFO"
    provider="com.axemble.education.providers.demo.DemoFormProvider">
  <fields>
    <field name="fldLabel1" label="LG_LABEL1"
           ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="true" />
    <field name="fldLabel2" label="LG LABEL2"
           ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="false" />
  </fields>
</form>
```

Aperçu d'un formulaire simple



L'élément « sheet »

L'interface graphique de la balise « sheet » est construite par la classe modèle suivante : **com.axemble.vdp.ui.framework.composites.xml.XMLSheet**.

La classe provider à utiliser doit étendre **com.axemble.vdoc.sdk.providers.BaseSheetProvider**.

<pages>

Container d'éléments « page ». Une des pages doit avoir l'attribut « default » positionné à la valeur « true ».

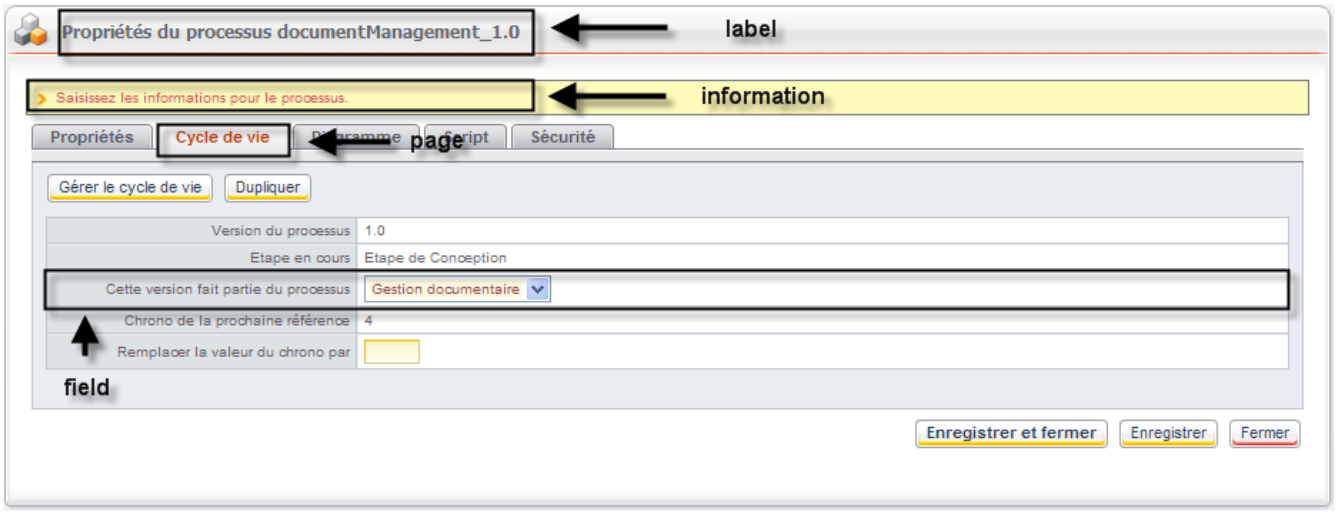
<fields>

Container d'éléments « field »

Exemple

```
<sheet name="treatment_class" label="LG_PROCESS_PROP_1" information="LG_PUT_PROCESSUS_INFO"
action="edit" provider="com.axemble.vdp.ui.core.providers.sheets.TreatmentClassProvider">
<pages>
<page name="properties" label="LG_PROPERTIES" default="true">
     <fields>
       <field name="fldLabel" label="LG_LABEL"
ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField"/>
       <field name="fldSystemName" label="LG SYST NAME"
ctrl="com.axemble.vdp.ui.core.document.fields.TextAreaField" mode="read"/>
     </fields>
</page>
<page name="lifecycle" label="LG LIFE CYCLE">
     <fields>
       <field name="fldProcessVersion" label="LG PROCESS VERSION"
ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mode="read"/>
       <field name="fldCurrentStage" label="LG_CURRENT_STAGE"
ctrl="com.axemble.vdp.ui.core.document.fields.TextAreaField" mode="read"/>
       <field name="fldProcessGroup" label="LG VERSION GROUP" throw-events="true"
ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField"/>
      <field name="fldNextChrono" label="LG NEXT CHRONO"
ctrl="com.axemble.vdp.ui.core.document.fields.IntegerField" mode="read"/>
       <field name="fldChangeChrono" label="LG_CHANGE_CHRONO"
ctrl="com.axemble.vdp.ui.core.document.fields.IntegerField" mode="write" min="0"
max="2147483647"/>
     </fields>
</page>
…
</pages>
</sheet>
```

Aperçu d'un formulaire multi-onglets



L'élément « wizard »

L'interface graphique de la balise « wizard » est construite par la classe modèle suivante : **com.axemble.vdp.ui.framework.composites.xml.XMLWizard**.

La classe provider à utiliser doit étendre **com.axemble.vdoc.sdk.providers.BaseWizardProvider**.

<pages>

Container d'éléments « page ». Une des pages doit avoir l'attribut « default » positionné à la valeur « true ».

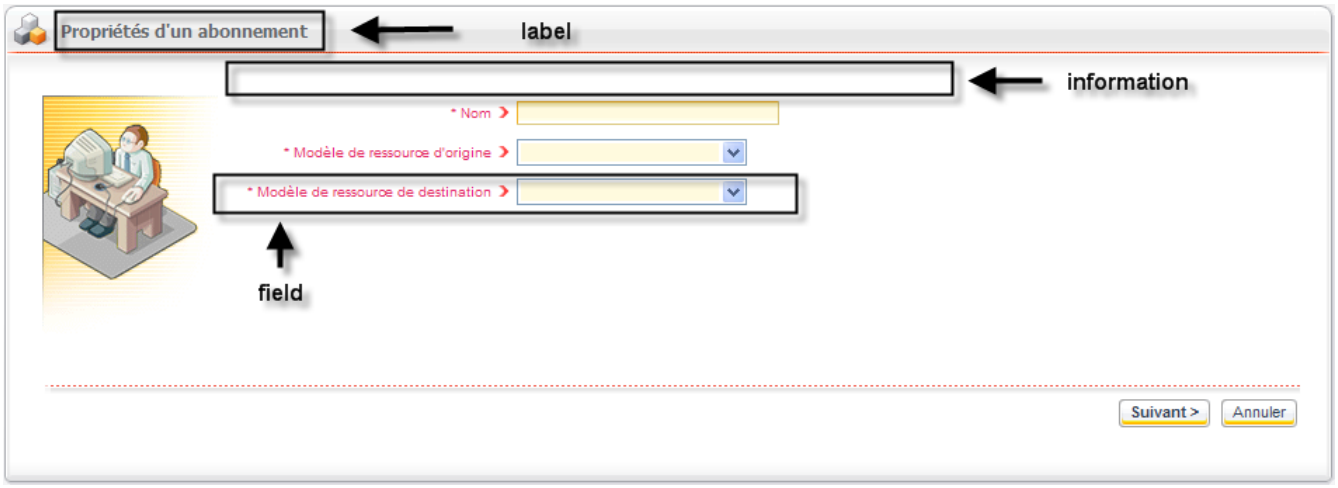
<fields>

Container d'éléments « field »

Exemple

```
<wizard name="resource subscriptions" action="create"
provider="com.axemble.vdp.ui.core.providers.wizards.SubscriptionProvider">
<pages>
<page name="page1" label="LG_SUBSCRIPTION_PROP" information="" default="true">
    <fields>
      <field name="fldName" label="LG NAME"
ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="true"/>
      <field name="fldSrcResourceTemplate" label="LG_RESOURCE_TEMPLATE_SRC"
ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField" mandatory="true"/>
      <field name="fldDstResourceTemplate" label="LG_RESOURCE_TEMPLATE_DST"
ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField" mandatory="true"/>
    </fields>
</page>
<page name="page2" label="LG_SUBSCRIPTION_PROP">
    <content>
      <view name="rulesView" label=""
provider="com.axemble.vdp.ui.core.providers.views.SubscriptionRuleProvider" >
      </view>
    </content>
</page>
…
</pages>
</wizard>
```

Aperçu d'un assistant



L'élément « view »

L'élément « view » peut être utilisée sous deux formes :

- balise fille des écrans de type formulaire ou explorateur ;
- balise parent dans le cas des Portlets ou des contenus Easysite.

L'interface graphique de la balise « view » est construite par la classe modèle suivante : **com.axemble.vdp.ui.framework.composites.xml.XMLView**.

Pour le cas des vues, deux classes provider peuvent être utilisées :

- vue standard : **com.axemble.vdoc.sdk.providers.BaseViewProvider** ;
- vue navigable : **com.axemble.vdoc.sdk.providers.BaseBrowsableViewProvider** ;

La pagination

Pour gérer la pagination dans les providers de vue le Framework UI propose deux interfaces à implémenter :

- **ICollectionModelViewProvider** : toutes les données sont gérées en mémoire ;
- **IBasicModelViewProvider** : seules les données relatives à la page courante sont gérer en mémoire.

Si le système interrogé gère la pagination il faut alors implémenter l'interface **IBasicModelViewProvider**. Dans ce cas, la méthode getModelItems() devra renvoyer uniquement les données relatives à la page courante. Pour récupérer les informations de contexte de la vue il suffit de passer par l'objet CtlAbstractView via la méthode getView().

Provider de vue standard

Le concept de modèle permet de faire abstraction des éléments graphiques pour la construction de la vue. En Il met en œuvre les éléments ViewModel, ViewModelColumn, et ViewModelItem qui simplifient considérablement le développement.

Exemple de provider de vue standard

L'exemple suivant montre comment utiliser le concept de « modèle de vue ». Il illustre à la fois la création des colonnes et la création des lignes. Notez l'utilisation de l'interface **ICollectionModelViewProvider**.

```
public class DemoViewProvider extends BaseViewProvider implements ICollectionModelViewProvider
{
 public DemoViewProvider( INavigateContext context, CtlAbstractView view )
 {
  super( context, view );
 }
 public void init()
 {
  super.init();

  CollectionViewModel viewModel = (CollectionViewModel)this.getModel();

  // construction des colonnes
  ViewModelColumn modelColumn = new ViewModelColumn( "firstName",
    new CtlLocalizedText( "LG FIRSTNAME" ).getText(), ViewModelColumn.TYPE STRING );
  viewModel.addColumn( modelColumn );
  modelColumn = new ViewModelColumn( "lastName",
    new CtlLocalizedText( "LG_LASTNAME" ).getText(), ViewModelColumn.TYPE_STRING );
  viewModel.addColumn( modelColumn );
  modelColumn = new ViewModelColumn( "email",
    new CtlLocalizedText( "LG_EMAIL" ).getText(), ViewModelColumn.TYPE_STRING );
  viewModel.addColumn( modelColumn );
 }
 public List getModelItems()
 {
  List items = new ArrayList();
  for ( int i = 0 ; i < 40 ; i++ )
  {
   // construction d'un élément par ligne affichée dans la vue
   ViewModelItem line = new ViewModelItem();
   line.setKey( "u" + i );
   line.setValue( "firstName", "Utilisateur" + i );
   line.setValue( "lastName", "U" + i );
   line.setValue( "email", "u" + i + "@vdocsoftware.com" );
   items.add( line );
  }
  return items;
 }
}
```

Provider de vue navigable

Le concept de vue navigable est très proche de celui d'une vue standard, avec pour avantage, la manipulation d'éléments arborescents.

Exemple de description XML d'une vue navigable

La description XML reste identique à celle d'une vue standard.

```
<view name="browsableView" action="display"
  provider="com.axemble.education.providers.demo.advanced.BrowsableView">
  <column name="NAME" label="LG NAME" />
  <column name="COMMANDS" label="LG_COMMANDS" />
</view>
```

Exemple de code d'une classe associée à une vue navigable

Voici un exemple de mise en œuvre d'une vue navigable. L'exemple se base sur une structure arborescente fictive.

```
public class BrowsableView extends BaseBrowsableViewProvider
  implements ICollectionModelViewProvider
{
 private Object parent;
 public HashMap childrenMap = new HashMap();

 public class Tree...
 Tree root = new Tree( "root", "Root" );

 public BrowsableView( INavigateContext context, CtlAbstractView view )
 {
  super( context, view );

  // build tree structure...
  root.children.add( firstRow );
  root.children.add( secondRow );
 }
 protected Collection getBrowsableOptions()
 {
  return null;
 }
 protected Collection getRootOptions()
 {
  if ( parent == null )
  {
   Collection ret = new ArrayList();
   ret.add( new Option( "", "ROOT" ) );
   return ret;
  }
  else
  {
   return getBrowsableOptions();
  }
 }
 public List getModelItems()
 {
  Tree tree = null;
  if ( parent == null )
   tree = root;
  else tree = (Tree)parent;

  ArrayList lines = new ArrayList();
  for ( Iterator iterator = tree.children.iterator() ; iterator.hasNext() ; )
  {
   Tree child = (Tree)iterator.next();
   ViewModelItem line = new ViewModelItem( child.key );
   line.setValue( "NAME", child.name );
   line.setBrowsableColumn( "NAME" );

   LinkedList l = new LinkedList();
   l.add( new CtlButton( "ok", new CtlText( "OK" ) ) );
   line.setValue( "COMMANDS", l );
```

```
   lines.add( line );
  }
  return lines;
 }
 protected void onBrowse( Object key )
 {
  try
  {
   if ( StringUtils.isEmpty( (String)key ) )
    parent = null;
   else
   {
    Tree tree = (Tree)childrenMap.get( key );
    parent = tree;
   }
   super.onBrowse( key );
  }
  catch( Exception e )
  {
   getNavigator().processErrors( e );
  }
 }
}
```

La balise <view> peut être complétée par les attributs présentés dans le tableau suivant :

Attributs	Description
selectable	Indique si la vue présente des éléments « case à cocher » permettant de sélectionner les lignes.
filterable	Indique si les filtres doivent être positionnés.
paginable	Indique si le page par page est supporté.
rowsperpage	Indique le nombre de lignes affichées par page.

<button>

Permet d'ajouter un bouton de navigation dans l'interface.

<image>

Permet de positionner des boutons image associés à des commandes ou de la navigation.

<link>

Permet de spécifier un lien de navigation.

<column>

Permet de spécifier les colonnes de la vue affichée.

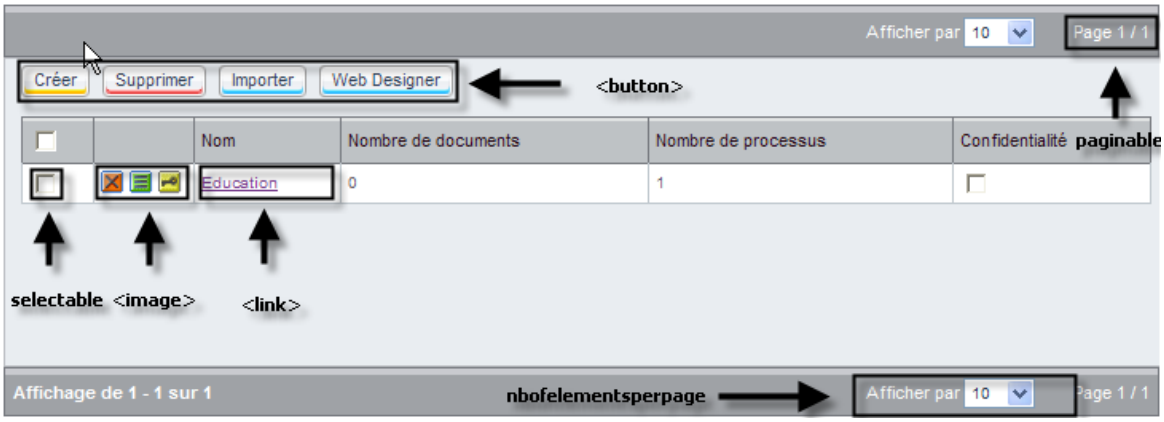
Description XML de l'écran permettant d'afficher la liste des applications

```
<view name="catalogs" label="LG APPLICATIONS" selectable="true"
  provider="com.axemble.vdp.ui.core.providers.views.CatalogProvider" paginable="true"
  filterable="true">
    <button name="delete" label="LG_DELETE" style="style2">
           <action class="catalog" name="delete"/>
    </button>
    <button name="import" label="LG_IMPORT" style="style3">
           <action class="catalog" name="import"/>
    </button>
    <link name="browse">
           <action class="catalog" name="browse"/>
    </link>
    <image name="delete">
           <action class="catalog" name="delete"/>
    </image>
    <image name="properties">
           <action class="catalog" name="edit"/>
    </image>
    <image name="security">
           <action class="catalog" name="security"/>
```

```
    </image>
    <column name="label" label="LG NAME" zone="title"/>
    <column name="doc number" label="LG DOC NUMBER"/>
    <column name="process number" label="LG PROCESS NUMBER"/>
    <column name="confidential" label="LG_CONFIDENTIAL"/>
</view>
```

Aperçu de la liste des applications

Sur cet écran, la classe fournisseur associée (CatalogProvider) a ajouté le bouton « web Designer » en dynamique.



L'élément « group »

L'élément « group » permet de grouper plusieurs modèles d'écrans. Par exemple, un formulaire de saisie simple peut être groupé avec une vue pour afficher un résultat de recherche.

L'interface graphique de la balise « group » est construite par la classe modèle suivante : **com.axemble.vdp.ui.framework.composites.xml.XMLGroup**.

La classe provider à utiliser doit étendre **com.axemble.vdoc.sdk.providers.BaseGroupProvider**.

Exemple

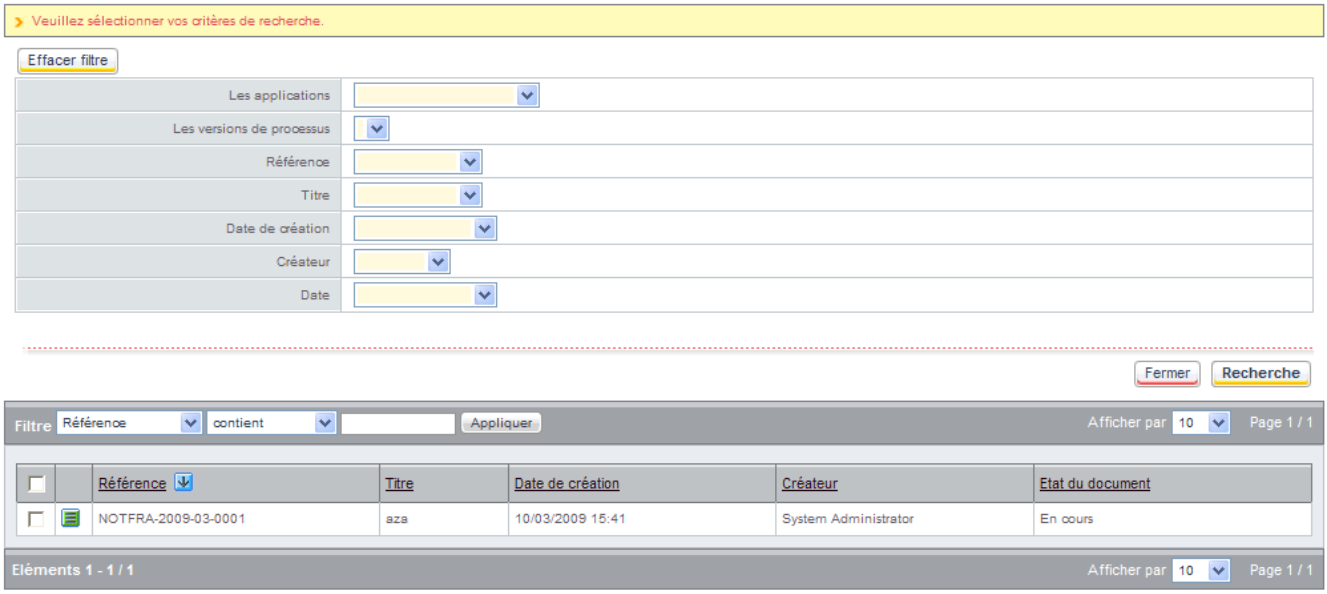
Dans cet exemple sont utilisés deux modèles d'écrans :

- un formulaire permettant d'afficher le formulaire de saisie d'une recherche ;
- une vue permettant de visualiser le résultat de la recherche.

```
<group name="generic" action="search" label="LG SEARCH MULTICRITERIA"
  provider="com.axemble.vdoc.sdk.providers.search.FormGroup">
    <form name="search" label="LG_SEARCH" information="LG_SEARCH_MULTICRITERIA_INFO"
      provider="com.axemble.vdoc.sdk.providers.search.IncludedForm">
          <buttons>
                <button name="reset" label="LG RESET" type="section">
                      <action name="reset"/>
                </button>
          </buttons>
          <fields>
                <field name="fldCatalogs" label="LG SEARCH CATALOGS"
                  ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField"
                  mode="write" throw-events="true" />
                <field name="fldWorkflows" label="LG_SEARCH_WORKFLOW_VERSIONS"
                  ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField"
                  mode="write" throw-events="true" />
          </fields>
          <filters>
                <field name="sys_Reference" label="LG_SEARCH_REFERENCE"
                  ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" />
                <field name="sys Title" label="LG SEARCH TITLE"
                  ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" />
                <field name="sys CreationDate" label="LG SEARCH CREATION DATE"
                  ctrl="com.axemble.vdp.ui.core.document.fields.DateField" />
                <field name="sys_Creator" label="LG_SEARCH_CREATOR" type="user"
                  ctrl="com.axemble.vdp.ui.core.document.fields.SingleDirectoryField" />
          </filters>
    </form>
```

```
    <view name="result" label="LG_RESULT"
      provider="com.axemble.vdoc.sdk.providers.search.IncludedView" paginable="true"
      selectable="true" filterable="true">
            <image name="properties">
                  <action class="treatment" name="edit" />
            </image>
            <column name="sys Reference" />
            <column name="sys Title" />
            <column name="sys CreationDate" />
            <column name="sys_Creator" />
            <column name="DocumentState" />
    </view>
</group>
```

Aperçu d'un groupe



L'élément « singleselector »

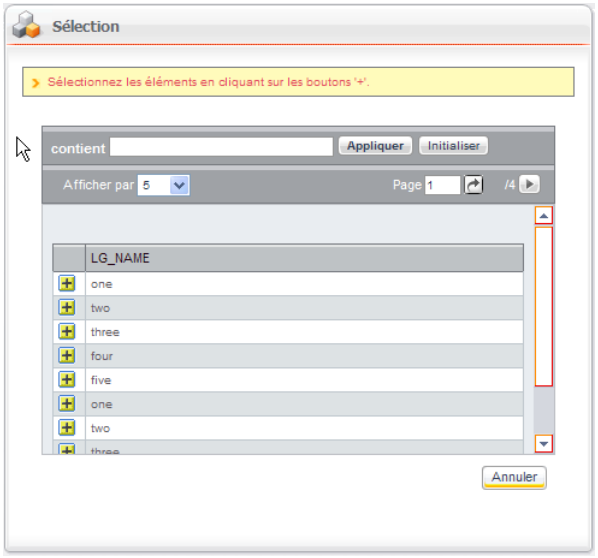
Le sélecteur simple peut facilement être remplacé par un élément « view ». Il suffit ensuite de déclarer un élément type « field » au sein d'un écran qui fait référence à la vue.

Exemple de mise en place d'un sélecteur simple basé sur une vue

L'exemple suivant montre comment faire référence à une vue depuis un champ.

```
<field
  name="DEMO"
  label="LG_SINGLE_SELECTION"
  ctrl="com.axemble.vdp.ui.core.document.fields.SelectorField"
  mode="write"
  multiple="false"
  ordered="true"
  screen="users"
  method="select" />
```

Aperçu d'un sélecteur simple



Il est possible aussi de directement contruire au sein d'un élément « field » une vue représentant les éléments sélectionnables du sélecteur.

```
<field
  name="DEMO"
  label="LG SINGLE SELECTION"
  ctrl="com.axemble.vdoc.sdk.document.fields.SelectorField"
  mode="write"
  multiple="false"
  ordered="true">
  <view name="catalogs" label="LG_APPLICATIONS" selectable="true"
    provider="com.axemble.education.providers.views.UsersProvider" paginable="true"
    filterable="true">
    <column name="firstname" label="LG_FIRSTNAME" zone="title"/>
    <column name="lastname" label="LG LASTNAME"/>
    <column name="email" label="LG EMAIL"/>
    <column name="active" label="LG_aCTIVE"/>
  </view>
</field>
```

L'élément « multipleselector »

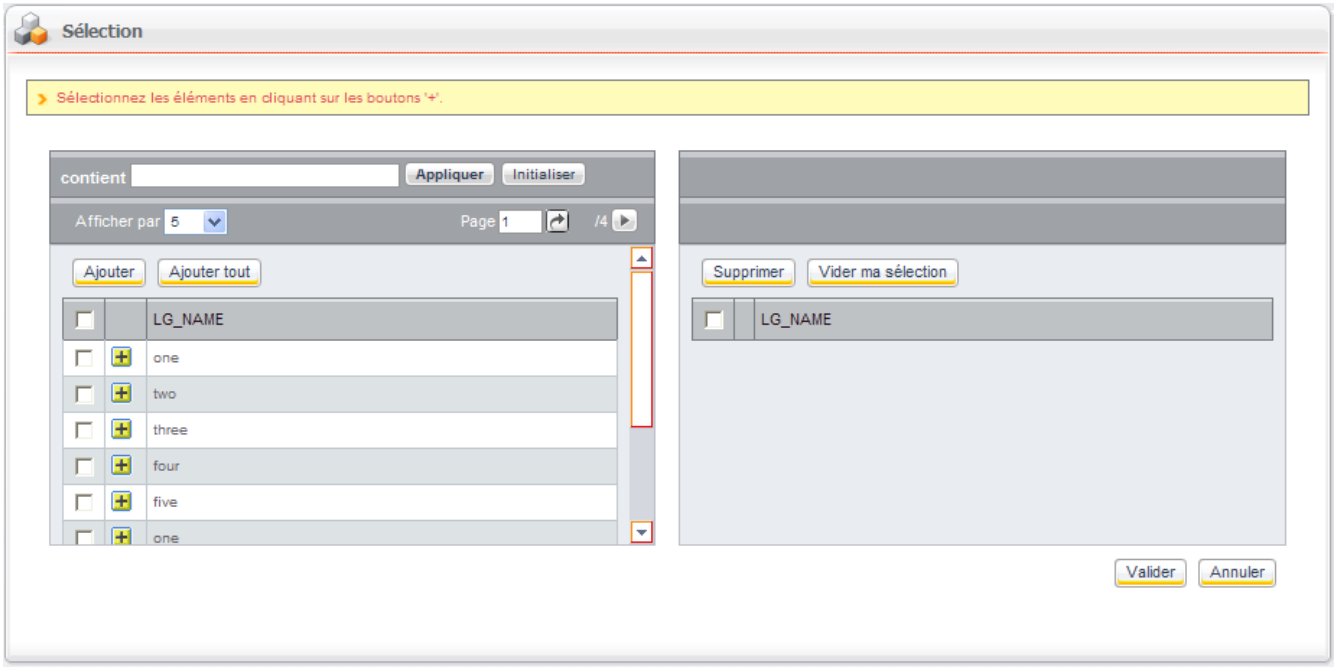
Le sélecteur multiple peut facilement être remplacé par un élément « view ». Il suffit ensuite de déclarer un élément type « field » au sein d'un écran qui fait référence à la vue.

Exemple de mise en place d'un sélecteur multiple basé sur une vue

L'exemple suivant montre comment faire référence à une vue depuis un champ.

```
<field
  name="DEMO"
  label="LG_MULTIPLE_SELECTION"
  ctrl="com.axemble.vdp.ui.core.document.fields.SelectorField"
  mode="write"
  multiple="true"
  ordered="true"
  screen="view"
  method="select" />
```

Aperçu d'un sélecteur multiple



Il est possible aussi de directement contruire au sein d'un élément « field » une vue représentant les éléments sélectionnables du sélecteur.

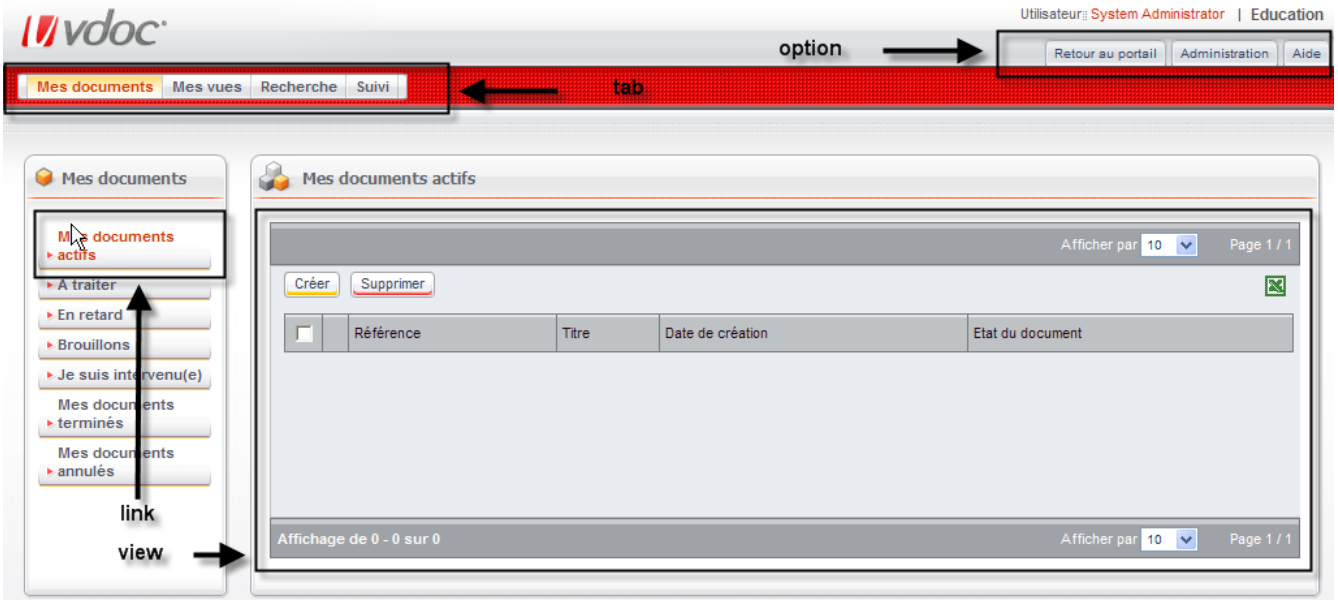
```
<field
  name="DEMO"
  label="LG_SINGLE_SELECTION"
  ctrl="com.axemble.vdoc.sdk.document.fields.SelectorField"
  mode="write"
  multiple="false"
  ordered="true">
  <view name="catalogs" label="LG_APPLICATIONS" selectable="true"
    provider="com.axemble.education.providers.views.UsersProvider" paginable="true"
    filterable="true">
    <column name="firstname" label="LG FIRSTNAME" zone="title"/>
    <column name="lastname" label="LG LASTNAME"/>
    <column name="email" label="LG_EMAIL"/>
    <column name="active" label="LG_aCTIVE"/>
  </view>
</field>
```

L'élément « explorer »

Exemple

```
<explorer name="catalog" action="access" showhistory="false"
provider="com.axemble.vdp.ui.core.providers.explorers.CatalogProvider">
<options>
<option label="LG BACKTOPORTAL">
    <action class="external" name="browse"/>
</option>
<option name="administration" label="LG_ADMINISTRATION">
    <action class="catalog" name="browse"/>
</option>
</options>
<tabs>
  <tab name="MYDOCS" label="LG_MYDOCS" default="true" >
    <links>
            <link name="myActiveDocs" label="LG MY ACTIVE DOCS" default="true">
              <view name="myActiveDocs" label="LG MY ACTIVE DOCS" selectable="true"
                provider="com.axemble.vdp.ui.core.providers.views.MyActiveDocumentsProvider"
                paginable="true" >
                    <button name="create" label="LG_CREATE">
                            <action class="catalog" name="create treatment"/>
                    </button>
                    <button name="delete" label="LG DELETE" style="style2">
                            <action name="delete"/>
                    </button>
                    <image name="delete">
                            <action name="delete"/>
                    </image>
                    <image name="properties">
                            <action class="treatment" name="edit"/>
                    </image>
              </view>
            </link>
            <link name="todo" label="LG TODO">
            ...
            </link>
...
    </links>
  </tab>
</tabs>
</explorer>
```

Aperçu d'un écran explorateur



Ajout de nouveaux écrans

Pour ajouter un écran dans la navigation VDoc, il suffit de créer un fichier XML de définition dans le répertoire **WEB-INF/storage/custom/navigation**. Ce fichier doit avoir la même structure que le fichier XML de définition des écrans standard.

Exemple de structure du fichier de personnalisation des écrans

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definition name="VDoc - Custom">
  <screens>

  </screens>
</definition>
```

Une fois la structure posée, il suffit de déclarer les nouveaux écrans en les plaçant sous la balise **<screens>**.

Exemple de déclaration de deux nouveaux écrans

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<definition name="VDoc - Custom">
  <screens>

    <form name="DEMO" action="action1" label="LG_VIEW_PROP" information="LG_PUT_VIEW_INFO"
      provider="com.axemble.education.providers.demo.DemoFormProvider">
      <fields>
        <field name="fldLabel1" label="LG LABEL1"
          ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="true" />
        <field name="fldLabel2" label="LG_LABEL2"
          ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="false" />
      </fields>
    </form>

    <sheet name="DEMO" action="action2" label="LG_VIEW_PROP" information="LG_PUT_VIEW_INFO"
      provider="com.axemble.education.providers.demo.DemoSheetProvider">
      <pages>
        <page name="page1" label="LG PAGE1" information="*** information 1 ***" default="true">
          <fields>
            <field name="fldLabel1" label="LG_LABEL1"
              ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="false" />
          </fields>
        </page>
        <page name="page2" label="LG PAGE2" information="*** information 2 ***">
          <fields>
            <field name="fldLabel2" label="LG_LABEL2"
              ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="true" />
          </fields>
        </page>
      </pages>
    </sheet>

  </screens>
</definition>
```

Les composants graphiques de la navigation

La couche graphique de VDoc dispose d'un grand nombre de composants graphiques permettant de construire des formulaires adaptés rapidement.

Liste non exhaustive des composants graphiques disponibles

Catégorie	Classes	Description
Nombres	CtlNumber	Gérer des éléments nombre
Dates	CtlDate CtlDateTime	Gérer des éléments date

<table>
<tr><td></td><td>CtlPeriodDate
CtlPeriodDateTime</td><td>Gérer des éléments période</td></tr>
<tr><td rowspan="4">Textes</td><td>CtlText
CtlIconText</td><td>Afficher du texte simple et avec une icône</td></tr>
<tr><td>CtlMandatoryText</td><td>Permet de gérer les éléments obligatoires</td></tr>
<tr><td>CtlLocalizedText</td><td>Permet de gérer l'internationalisation des chaînes affichées</td></tr>
<tr><td>CtlTextBox
CtlTextArea</td><td>Permet la saisie de texte</td></tr>
<tr><td rowspan="2">Listes</td><td>CtlCheckBox</td><td>Permet de gérer les valeurs booléennes</td></tr>
<tr><td>CtlComboBox
CtlSelectList</td><td>Permet de gérer des éléments listes

▫ Exemple de code
<pre>CtlComboBox selectList = new CtlComboBox();

selectList.addOption(new Option("key1" ,getStaticString(
"LG VALUE1")));

selectList.addOption(new Option("key2", getStaticString(
"LG_VALUE2")));

selectList.setSelectedKey("key1");</pre></td></tr>
<tr><td></td><td>CtlCheckBoxGroup
CtlRadioGroup</td><td>Groupe d'éléments</td></tr>
<tr><td>Boutons</td><td>CtlButton
CtlImageButton
CtlHyperLink</td><td>▫ Exemple de code utilisant un ActionListener
<pre>CtlButton demoAlertBtn = new CtlButton("alert", new
CtlLocalizedText("LG ALERT"));
demoAlertBtn.addActionListener(new ActionListener()
{
  public void onClick(ActionEvent event)
  {
    Navigator.getNavigator().showAlertBox("Message Alert !");
  }
});
getResourceController().getButtonContainer(
IResourceController.TOP_CONTAINER).addLast(demoAlertBtn);</pre>▫ Exemple de code utilisant un ConfirmBoxListener
<pre>CtlButton demoConfirmBtn = new CtlButton("confirm", new
CtlLocalizedText("LG CONFIRM"));
demoConfirmBtn.addActionListener(new ActionListener()
{
  public void onClick(ActionEvent event)
  {
    Navigator.getNavigator().showConfirmBox("Message Confirm !",
new ConfirmBoxListener()
    {
      public void onCancel(ActionEvent event)
      {
        Navigator.getNavigator().showAlertBox("onCancel()");
      }
      public void onOk(ActionEvent event)
      {
        Navigator.getNavigator().showAlertBox("onOk()");
      };
    });
    }
});
getResourceController().getButtonContainer(
IResourceController.TOP_CONTAINER).addLast(demoConfirmBtn);</pre></td></tr>
<tr><td>Containers</td><td>CtlContainer</td><td>Permet de gérer des éléments composites

▫ Exemple de code
<pre>CtlText text = new CtlLocalizedText("LG_TEXTE");</pre></td></tr>
</table>

		`` // créer le container Container container = new Container(new FreeLayout()); // ajouter du texte container.add(text); // ajouter une liste container.add(selectList); // ajouter du texte container.add(text); // ajouter une vue container.add(listView); ```
Vues	CtlListView	Gérer les tableaux (colonnes, lignes, cellules) **Exemple de code** ``` // ajouter un tableau CtlListView listView = new CtlListView(); // créer les colonnes du tableau listView.createColumn("col1",new CtlLocalizedText("LG COL1")); listView.createColumn("col2",new CtlLocalizedText("LG_COL2")); // créer des éléments dans le tableau CtlListView.Item item = listView.createItem(new CtlText("1")); item.setParam("key1"); listView.createSubitem(new CtlText("A"), item); item = listView.createItem(new CtlText("2")); item.setParam("key2"); listView.createSubitem(new CtlText("B"), item); ```

Le paquetage regroupant tous les composants CtlXXX est **com.axemble.vdp.ui.framework.widgets**. Ces composants peuvent être utilisés pour créer des interfaces graphiques dynamiques.

Il faut toutefois noter l'absence de document pour la manipulation des données. En effet, ces composants graphiques n'ont aucune notion de document (CoreDocument ou GenericDocument).

Extrait de code de la classe DemoWizardProvider permettant l'ajout de page dans un assistant en dynamique

```
public void readyState()
{
    super.readyState();

    // récupération de la position de la 'page1'
    int pos = this.abstractWizard.getPageIndex( "page1" ) + 1;

    // création d'une nouvelle page nommée 'newPage1'
    CtlWizardPage wizardPage = new CtlWizardPage( this.abstractWizard, "newPage1" );
    this.abstractWizard.addPage( wizardPage, pos );
    pos++;
    wizardPage.setLabel( new CtlLocalizedText("LG_LABEL_PAGE_1") );
    wizardPage.setInformation( new CtlLocalizedText("LG_INFORMATION_1") );

    // ajout de nouvelles entrées
    wizardPage.addEntry( "fldText", new CtlLocalizedText( "LG TEXTE" ), new CtlTextBox( "valeur
texte" ) );
    wizardPage.addEntry( "fldNumber", new CtlLocalizedText( "LG_NUMBER" ), new CtlNumber() );
    wizardPage.addEntry( "fldDate", new CtlLocalizedText( "LG_DATE" ), new CtlDate() );

    // création d'une nouvelle page nommée 'newPage2'
    wizardPage = new CtlWizardPage( this.abstractWizard, "newPage2" );
    this.abstractWizard.addPage( wizardPage, pos );
    pos++;
    wizardPage.setLabel( new CtlLocalizedText("LG_LABEL_PAGE_2") );
    wizardPage.setInformation( new CtlLocalizedText("LG_INFORMATION_2") );

    // ajouter une liste
    CtlComboBox selectList = new CtlComboBox();
    selectList.addOption( new Option( "key1", getStaticString( "LG_VALUE1" ) ) );
    selectList.addOption( new Option( "key2", getStaticString( "LG_VALUE2" ) ) );
    selectList.setSelectedKey( "key1" );
    wizardPage.addEntry( "fldList", new CtlLocalizedText( "LG LISTE" ), selectList );
```

```
    // ajouter un tableau
    CtlListView listView = new CtlListView();
    listView.createColumn( "col1", new CtlLocalizedText( "LG COL1" ) );
    listView.createColumn( "col2", new CtlLocalizedText( "LG COL2" ) );

    CtlListView.Item item = listView.createItem( new CtlText( "1" ) );
    item.setParam( "key1" );
    listView.createSubitem( new CtlText( "A" ), item );
    item = listView.createItem( new CtlText( "2" ) );
    item.setParam( "key2" );
    listView.createSubitem( new CtlText( "B" ), item );

    wizardPage.addEntry( "fldTable", new CtlLocalizedText( "LG TABLEAU" ), listView );

    // création d'une nouvelle page nommée 'newPage3'
    wizardPage = new CtlWizardPage( this.abstractWizard, "newPage3" );
    this.abstractWizard.addPage( wizardPage, pos );
    pos++;
    wizardPage.setLabel( new CtlLocalizedText("LG LABEL PAGE 3") );
    wizardPage.setInformation( new CtlLocalizedText("LG INFORMATION 3") );

    Container container = new Container(new FreeLayout());

    CtlText textBidon = new CtlText(getStaticString( "LG BIDON" ));
    container.add( textBidon );
    container.add( listView );
    container.add( textBidon );
    container.add( listView );

    wizardPage.setContent( container );
}
```

Extrait de code de la classe SimpleTableField qui permet de créer un tableau en fonction de type d'attribut manipulé

```
for ( Iterator iter = attributes.iterator() ; iter.hasNext() ; )
{
    String attributeValues = (String)iter.next();

    String[] extractedValues = StringUtils.split( attributeValues, SEPARATOR );
    String key = extractedValues[0];
    String type = extractedValues[1];
    String label = extractedValues[2];
    String value = extractedValues[3];

    CtlListView.Item item = this.listView.createItem( new CtlText( label ) );
    item.setParam( key );

    Widget widget = null;

    if ( "string".equals( type ) )
    {
        if ( NULL_VALUE.equals( value ) )
            widget = new CtlTextBox();
        else widget = new CtlTextBox( value );
    }
    else if ( "number".equals( type ) )
    {
        if ( NULL_VALUE.equals( value ) )
            widget = new CtlNumber();
        else
        {
            CtlNumber number = new CtlNumber();
            number.setIntegerOnly( false );
            number.setFloatValue( Float.valueOf( value ) );
            widget = number;
        }
    }
    else if ( "date".equals( type ) )
    {
        if ( NULL_VALUE.equals( value ) )
            widget = new CtlDate();
        else widget = new CtlDate( new Timestamp( Long.valueOf( value ).longValue() ) );
    }

    // ajout de l'évènement en cas de modification de la valeur
    widget.addChangeListener( new ChangeListener()
    {
```

```
            public void onChange( ChangeEvent event )
            {
                    SimpleTableField.this.aggregate.safeUpdateValue();
            }
    } );

    CtlListView.Subitem subitem = this.listView.createSubitem( widget, item );
    subitem.setParam( type );
}
```

Les champs des formulaires

En complément des composants graphiques, VDoc offre un ensemble d'objets nommés « champs » permettant de développer rapidement, via une description XML, des écrans.

Ces champs sont disponibles pour la construction des formulaires de documents processus (personnalisation) et les écrans déclarés dans le fichier XML de définition des écrans.

Deux remarques toutefois sur la description de ces champs :

- Les raccourcis ne sont valables que dans le contexte des formulaires de personnalisation.
- L'attribut « name » qui identifie un champ d'un écran s'appelle « property » dans le contexte des formulaires de personnalisation.

La plupart des classes Java présentées dans cette section respectent la même convention. Pour une classe Java nommée XXXField (champ), il existe une classe Java de base nommée CtlXXX (composant graphique : widget).

Une classe XXXField est en réalité un élément CtlXXX relié par un mécanisme d'évènements internes à un document.

Rappel

Dans le cas des écrans déclarés dans le fichier de définition, les champs sont rattachés à un document de type GenericDocument. Dans le cas des formulaires de personnalisation, les champs sont rattachés au document processus de type CoreDocument.

Quelques exemples de relation entre un champ et un composant graphique

Champ	Composant graphique
TextBoxField	CtlTextBox
PasswordField	CtlPassword
DateField	CtlDate
DateTimeField	CtlDateTime
PeriodDateField	CtlPeriodDate
PeriodDateTimeField	CtlPeriodTimeDate
CheckBoxField	CtlCheckBox
CheckBoxGroupField	CtlCheckBoxGroup
ComboBoxField	CtlComboBox
RadioGroupField	CtlGroupField
TextAreaFieldField	CtlTextArea

IntegerField

Nom	IntegerField
Description	Permet la saisie d'une valeur numérique entière
Composant	com.axemble.vdp.ui.core.document.fields.IntegerField

Raccourci	integer
Type Java	java.lang.Integer
Table de stockage	-
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldInteger` `  label = "?"   - label of the control, ex: LG TEXT` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.IntegerField"` `  mode  = "write" | "read" - set editable or not` `  mandatory  = "false" | "true" - set mandatory or not` `  throw-events  = "false" | "true" - throw events from the webbrowser` `  size = "medium" | "veryshort" | "short" | "long" | "auto"` `  integer-value="7"` `/>`

LongField

Nom	LongField
Description	Permet la saisie d'une valeur numérique entière
Composant	Com.axemble.vdp.ui.core.document.fields.LongField
Raccourci	Long
Type Java	java.lang.Long
Table de stockage	-
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldLong` `  label = "?"   - label of the control, ex: LG TEXT` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.LongField"` `  mode  = "write" | "read" - set editable or not` `  mandatory  = "false" | "true" - set mandatory or not` `  throw-events  = "false" | "true" - throw events from the webbrowser` `  size = "medium" | "veryshort" | "short" | "long" | "auto"` `  long-value="7"` `/>`

FloatField

Nom	FloatField
Description	Permet la saisie d'une valeur numérique à virgule
Composant	com.axemble.vdp.ui.core.document.fields.FloatField
Raccourci	float
Type Java	java.lang.Float
Table de stockage	vdp_float_values
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldFloat` `  label = "?"   - label of the control, ex: LG TEXT` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.LongField"` `  mode  = "write" | "read" - set editable or not` `  mandatory  = "false" | "true" - set mandatory or not` `  throw-events  = "false" | "true" - throw events from the webbrowser` `  size = "medium" | "veryshort" | "short" | "long" | "auto"` `  float-value="7.86"` `/>`

DoubleField

Nom	DoubleField
Description	Permet la saisie d'une valeur numérique à virgule
Composant	com.axemble.vdp.ui.core.document.fields.DoubleField
Raccourci	double
Type Java	java.lang.Double
Table de stockage	-
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldDouble`

```
  label = "?"   - label of the control, ex: LG TEXT
  ctrl  = "com.axemble.vdp.ui.core.document.fields.DoubleField"
  mode           = "write" | "read" - set editable or not
  mandatory      = "false" | "true" - set mandatory or not
  throw-events   = "false" | "true" - throw events from the webbrowser
  size = "medium" | "veryshort" | "short" | "long" | "auto"
/>
```

DateField

Nom	DateField
Description	Permet la saisie d'une date
Composant	com.axemble.vdp.ui.core.document.fields.DateField
Raccourci	date
Type Java	java.util.Date
Type stockage	java.sql.Timestamp
Table de stockage	vdp_timestamp_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldDate label = "?" - label of the control, ex: LG_TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.DateField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser />

DateTimeField

Nom	DateTimeField
Description	Permet la saisie d'une date avec l'heure
Composant	com.axemble.vdp.ui.core.document.fields.DateTimeField
Raccourci	datetime
Type Java	java.util.Date
Type stockage	java.sql.Timestamp
Table de stockage	vdp_timestamp_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldDateTime label = "?" - label of the control, ex: LG TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.DateTimeField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser hour-format = "24" | "12" - display hours in 24 or 12 hours format />

TimeField

Nom	TimeField
Description	Permet la saisie d'une heure
Composant	com.axemble.vdp.ui.core.document.fields.TimeField
Raccourci	time
Type Java	java.lang.Long
Type stockage	java.lang.Long
Table de stockage	vdp_long_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldTime label = "?" - label of the control, ex: LG TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.TimeField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser hour-format = "24" | "12" - display hours in 24 or 12 hours format

	/>

PeriodDateField

Nom	PeriodDateField
Description	Permet la saisie d'une période
Composant	com.axemble.vdp.ui.core.document.fields.PeriodDateField
Raccourci	dateperiod
Type Java	com.axemble.vdp.information.structures.Period
Type stockage	com.axemble.vdp.information.structures.Period
Table de stockage	vdp_period_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldPeriodDate label = "?" - label of the control, ex: LG_TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.PeriodDateField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser />

PeriodTimeField

Nom	PeriodTimeField
Description	Permet la saisie d'une plage horaire
Composant	com.axemble.vdp.ui.core.document.fields.PeriodTimeField
Raccourci	datetimeperiod
Type Java	com.axemble.vdp.information.structures.Period
Type stockage	com.axemble.vdp.information.structures.Period
Table de stockage	vdp_period_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldPeriodTime label = "?" - label of the control, ex: LG TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.PeriodTimeField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser />

TextBoxField

Nom	TextBoxField
Description	Permet la saisie d'une chaîne de caractères
Classe	com.axemble.vdp.ui.core.document.fields.TextBoxField
Raccourci	text
Type Java	java.lang.String
Table de stockage	vdp_string_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG_TEXT ctrl = "com.axemble.vdp.ui.core.document.fields.TextBoxField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser size = "medium" | "veryshort" | "short" | "long" | "auto" />

PasswordField

Nom	PasswordField
Description	Permet la saisie d'un mot de passe avec validation
Classe	com.axemble.vdp.ui.core.document.fields.PasswordField

Raccourci	password
Type Java	java.lang.String
Table de stockage	-
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG PASSWORD ctrl = "com.axemble.vdp.ui.core.document.fields.PasswordField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change size= "medium" | "veryshort" | "short" | "long" | "auto" - set the size authentication-class = "?" - the class used to validate field input (implements com.axemble.vdp.ui.core.document.fields.IDirectoryAuthentication). com.axemble.vdp.ui.core.document.fields.classes.DefaultAuthentication is the default authentication class for both Websphere and Portal environments. />

RadioGroupField

Nom	RadioGroupField
Description	Permet la sélection d'un élément contenu dans un ensemble
Classe	com.axemble.vdp.ui.core.document.fields.RadioGroupField
Raccourci	radiogroup
Type Java	java.lang.String
Table de stockage	-
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG MODE ctrl = "com.axemble.vdp.ui.core.document.fields.RadioGroupField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events= "false" | "true" - throw events from the webbrowser when the control change nbofcolumns = "1" | "2" | "3" | ... - number of column used to display the options fillingmode = "vertical" | "horizontal" - arrange options (left-right and top-bottom) or (top-bottom and left-right) list-from-list = "?" - name of an alternative property to get the list with property.getList() list-from-value= "?" - name of an alternative property to get the list with property.getValue() allowreset = "true" | "false" - display a button to clear the selection, so the user will be able to select null summaryMode = "false" | "true" - if true, in reading mode only the selected value are displayed and the list is not requested advance usage for definition.xml: force-get-list = "false" | "true" - in reading mode force the getList to take advantage of key label difference. />

CheckBoxGroupField

Nom	CheckBoxGroupField
Description	Permet la sélection de plusieurs éléments d'un ensemble
Classe	com.axemble.vdp.ui.core.document.fields.ChechBoxGroupField
Raccourci	checkboxgroup
Type Java	java.util.Collection
Table de stockage	vdp_collection_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG MODE ctrl = "com.axemble.vdp.ui.core.document.fields.CheckBoxGroupField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change

	nbofcolumns = "1" | "2" | "3" | ... - number of column used to display the options fillingmode = "vertical" | "horizontal" - arrange options (left-right and top-bottom) or (top-bottom and left-right) list-from-list = "?" - name of an alternative property to get the list with property.getList() list-from-value= "?" - name of an alternative property to get the list with property.getValue() allowreset = "false" | "true" display a button to clear the selection allowselectall = "false" | "true" display a button to select all options summaryMode = "false" | "true" - if true, in reading mode only the selected value are displayed and the list is not requested advance usage for definition.xml: force-get-list = "false" | "true" - in reading mode force the getList to take advantage of key label difference. />

CheckBoxField

Nom	CheckBoxField
Description	Permet la sélection d'un élément
Classe	com.axemble.vdp.ui.core.document.fields.ChechBoxField
Raccourci	checkbox
Type Java	java.lang.Long
Table de stockage	vdp_long_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldCheckBox label = "?" - label of the control, ex: LG_MODE ctrl = "com.axemble.vdp.ui.core.document.fields.CheckBoxField" mode = "write" | "read" - set editable or not throw-events = "false" | "true" - throw events from the webbrowser />

ComboBoxField

Nom	ComboBoxField
Description	Permet la sélection d'un élément
Classe	com.axemble.vdp.ui.core.document.fields.ComboBoxField
Raccourci	text
Type Java	java.lang.String
Table de stockage	vdp_string_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG_MODE ctrl = "com.axemble.vdp.ui.core.document.fields.ComboBoxField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change list-from-list = "?" - name of an alternative property to get the list with property.getList() list-from-value= "?" - name of an alternative property to get the list with property.getValue() allowreset = "true" | "false" - if (getValue()!=null && allowreset==false) the first null value will not be present, so the user will not be able to select null. advance usage for definition.xml: force-get-list = "false" | "true" - in reading mode force the getList to take advantage of key label difference. />

SelectListField

Nom	SelectListField
Description	Permet la sélection de plusieurs éléments d'une liste

Classe	com.axemble.vdp.ui.core.document.fields.SelectListField
Raccourci	textselectlistmultiple
Type Java	java.util.Collection
Table de stockage	vdp_collection_values
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldSelectList` `  label = "?"   - label of the control, ex: LG MODE` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.SelectListField"` `  mode        = "write" | "read" - set editable or not` `  mandatory   = "false" | "true" - set mandatory or not` `  throw-events = "false" | "true" - throw events from the webbrowser when the control change` `  list-from-list = "?" - name of an alternative property to get the list with property.getList()` `  list-from-value= "?" - name of an alternative property to get the list with property.getValue()` `  advance usage for definition.xml:` `  force-get-list = "false" | "true" - in reading mode force the getList to take advantage of key` `  label difference.` `  />`

MultipleFileField

Nom	MultipleFileField
Description	Permet d'ajouter des pièces jointes
Classe	com.axemble.vdp.ui.core.document.fields.MultipleFileField
Raccourci	file_multiple
Type Java	java.util.Collection
Table de stockage	vdp_collection_values
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldMode` `  label = "?"   - label of the control, ex: LG MODE` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.MultipleFileField"` `  mode          = "write" | "read" - set editable or not` `  mandatory     = "false" | "true" - set mandatory or not` `  throw-events   = "false" | "true" - throw events from the webbrowser when the control change` `  min-file-number = ? default 0` `  max-file-number = ? default Integer.MAX_VALUE` `  min-file-size   = ? default 0` `  max-file-size    = ? default read the max file size in the Organization configuration or 2 Go` `  min-total-size  = ? default 0` `  max-total-size  = ? default Long.MAX VALUE` `  max-file-name-length = ? default 250` `  supported-extensions = ? default read in the Organization configuration` `  "*" mean all extensions` `  ".txt;.xml;.doc" mean accept only .txt .xml .doc` `  />`

SingleDirectoryField

Nom	SingleDirectoryField
Description	Permet de sélectionner un élément dans l'annuaire
Classe	com.axemble.vdp.ui.core.document.fields.SingleDirectoryField
Raccourci	user_browser, group_browser, localization_browser, organization_browser Si le raccourci est utilize, l'attribut type de sera pas nécessaire.
Type Java	com.axemble.vdp.information.structures.DirectoryElement
Table de stockage	vdp_directory_element_values
Format d'utilisation	`<field` `  name  = "?"   - name of the IAbstractField, ex: fldSingleDirectory` `  label = "?"   - label of the control, ex: LG_MODE` `  type  = "user | group | localization | organization | filecenter"` `  ctrl  = "com.axemble.vdp.ui.core.document.fields.SingleDirectoryField"` `  mode          = "write" | "read" - set editable or not`

	mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change /> Exemple pour un sélecteur d'élément FileCenter <field name="fldFilecenter" label="SingleFilecenter" type="filecenter" ctrl="com.axemble.vdp.ui.core.document.fields.SingleDirectoryField" />

MultipleDirectoryField

Nom	MultipleDirectoryField
Description	Permet de sélectionner plusieurs éléments de l'annuaire
Classe	com.axemble.vdp.ui.core.document.fields.MultipleDirectoryField
Raccourci	user_browser_multiple, group_browser_multiple, localization_browser_multiple, organization_browser_multiple
Type Java	java.util.Collection
Table de stockage	vdp_collection_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMultipleDirectory label = "?" - label of the control, ex: LG_MODE type = "user | group | localization | organization | filecenter" ctrl = "com.axemble.vdp.ui.core.document.fields.MultipleDirectoryField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change />

ScreenEmbedderField

Nom	ScreenEmbedderField
Description	Permet de lancer de nouveaux écrans
Classe	com.axemble.vdp.ui.core.document.fields.ScreenEmbedderField
Raccourci	screen_embedder
Type Java	java.util.Map
Table de stockage	vdp_byte_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG_MODE ctrl = "com.axemble.vdp.ui.core.document.fields.ScreenEmbedderField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change screen = "?" - name of the screen method = "?" - name of the method shared-document-key = "?" – key of the shared document entry shared-field-key = "?" – key of the shared field entry />

SelectorField

Nom	SelectorField
Description	Permet de faire appel à un sélecteur simple ou multiple
Classe	com.axemble.vdoc.sdk.document.fields.SelectorField
Raccourci	screen_embedder
Type Java	java.lang.String (simple) ou java.util.Collection (multiple)
Table de stockage	vdp_string_values ou vdp_collection_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldMode

	label = "?" - label of the control, ex: LG MODE ctrl = "com.axemble.vdoc.sdk.document.fields.SelectorField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change screen = "?" - name of the screen method = "?" - name of the method />

FCKEditorField

Nom	FCKEditorField
Description	Permet de saisir du texte dans l'éditeur WYSIWYG (FCK Editor)
Classe	com.axemble.vdp.ui.core.document.fields.FCKEditorField
Raccourci	fckeditor
Type Java	Byte[]
Table de stockage	vdp_byte_values
Format d'utilisation	<field name = "?" - name of the IAbstractField, ex: fldFCKEditor label = "?" - label of the control, ex: LG MODE ctrl = "com.axemble.vdp.ui.core.document.fields.FCKEditorField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser />

EnhancedUrlField

Nom	EnhancedUrlField
Description	Permet de présenter un champ URL
Classe	com.axemble.vdoc.sdk.document.fields.EnhancedUrlField
Raccourci	
Type Java	java.lang.String (ei. url¤title¤params)
Table de stockage	vdp_string_values
Format d'utilisation	La valeur d'un tel champ est une concaténation de trois ou quatre valeurs texte : url¤title¤params(¤description). Le caractère séparateur est '¤'. <field name = "?" - name of the IAbstractField, ex: fldMode label = "?" - label of the control, ex: LG MODE ctrl = " com.axemble.vdoc.sdk.document.fields.EnhancedUrlField" mode = "write" | "read" - set editable or not mandatory = "false" | "true" - set mandatory or not throw-events = "false" | "true" - throw events from the webbrowser when the control change title = "?" - set the field title params = "width, height, resizable, scrollbars, status, toolbar" ("0" | "1") - set the field parameters show-url = "false" | "true" – display the url text box. show-title = "false" | "true" – display the title text box. show-description = "false" | "true" – display the description textBox. show-params = "false" | "true" – display the parameter textBox. />

Intégration des formulaires

VDoc offre un système de navigation simple qui permet de naviguer aisément d'un écran vers un autre. Il existe plusieurs façons d'intégrer un nouvel écran dans un écran existant :

- Via le document de définition (mode statique)
- Via un champ spécial « déclaré » dans le web Designer (mode statique)
- Par programmation (mode dynamique)

Via le document de définition

Dans le document XML de définition des écrans, il est possible d'ajouter des écrans et de spécifier des boutons images, des boutons simples, ou hyperliens qui permettent de naviguer vers d'autres écrans.

Dans l'exemple suivant, l'action du bouton nommé « demo1 » permet de naviguer vers l'écran [DEMO, action1]. Egalement, le bouton image nommé « demo2 » permet de naviguer vers l'écran [DEMO, action2].

Exemple d'écran appelant

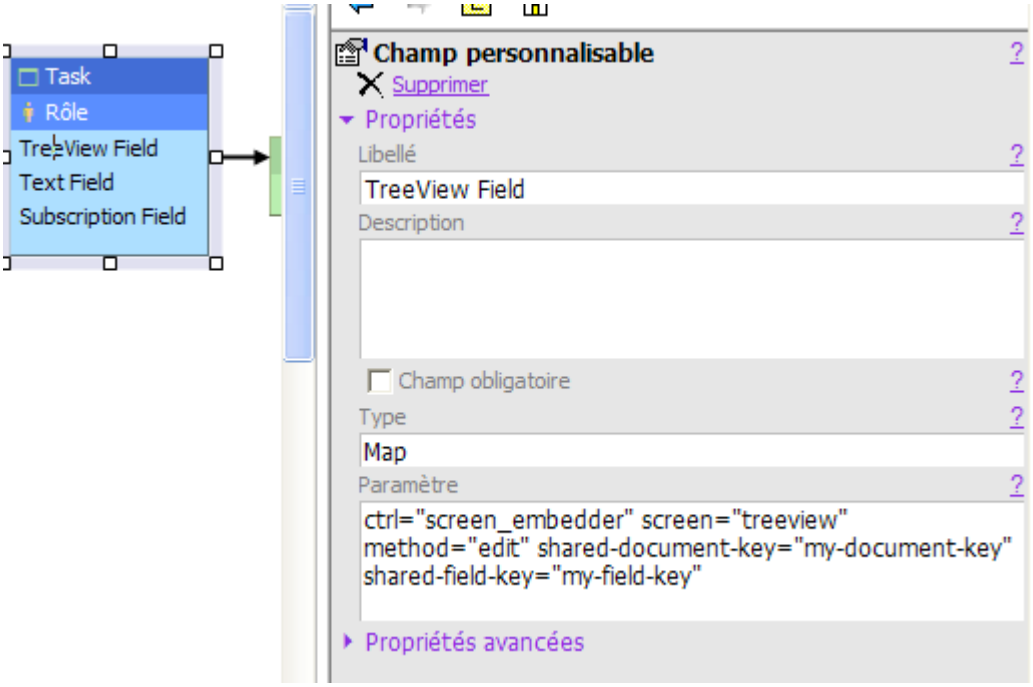
```
<!-- ajout d'un bouton dans la vue mes documents -->
<explorer override="catalog.access">
  <tabs>
    <tab name="MYDOCS">
      <links>
        <link name="myActiveDocs">
          <view name="myActiveDocs">
            <button name="demo1" label="LG_DEMO_LABEL">
              <action class="DEMO" name="action1"/>
            </button>
           <image name="demo2">
             <action class="DEMO" name="action2" />
           </image>
          </view>
        </link>
      </links>
    </tab>
  </tabs>
</explorer>
```

Exemple d'écran appelé

```
<form name="DEMO" action="action1" label="LG_VIEW_PROP" information="LG_PUT_VIEW_INFO"
  provider="com.axemble.education.providers.demo.DemoFormProvider">
  <fields>
    <field name="fldLabel1" label="LG_LABEL1"
      ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="true" />
    <field name="fldLabel2" label="LG LABEL2"
      ctrl="com.axemble.vdp.ui.core.document.fields.TextBoxField" mandatory="false" />
  </fields>
</form>
```

Via un champ du document

Depuis le web Designer, il est possible de déclarer un champ personnalisable de type « screen_embedder » qui permet de décrire l'appel à un nouvel écran. Ce champ est présenté sous la forme habituelle d'un bouton faisant appel à un sélecteur.



Attributs du champ

Attribut	Description
screen	Nom système de l'écran
method	Nom de l'action liée à l'écran
shared-document-key (optionnel)	Nom de clé permettant de retrouver, depuis le contexte de navigation reçu dans le provider, le document père. Si cet attribut n'est pas spécifié, les deux écrans partageront le même document.
shared-field-key (optionnel)	Nom de clé permettant de retrouver, depuis le contexte de navigation reçu dans le provider, le champ (screen_embedder).

Les attributs « shared-document-key » et « shared-field-key » sont optionnels. Si ces attributs ne sont pas utilisés, l'écran appelé reçoit en tant que paramètre AbstractDocument du constructeur l'objet CoreDocument de l'écran appelant.

Dans le cas contraire, l'objet CoreDocument sera placé en tant que paramètre de contexte de navigation de l'écran appelé. Il pourra être récupéré à l'aide de la clé indiquée dans l'attribut « shared-document-key ». Il en va de même pour l'objet CoreField.

Extrait de code de la classe NotSharedDocumentSheetProvider : les clés posées dans la personnalisation du champ « screen_embedder » sont sharedDocument et sharedField.

```
public NotSharedDocumentSheetProvider( INavigateContext context, AbstractDocument document,
CtlAbstractSheet abstractSheet )
{
    super( context, document, abstractSheet );
    this.sharedDocument = (CoreDocument)this.context.getParameterMap().get( "sharedDocument" );
    this.sharedField = (CoreField)this.context.getParameterMap().get( "sharedField" );
}
```

Via programmation

Il est possible depuis les providers ou les classes d'extension de document de créer des éléments tels que les boutons, et d'associer à l'évènement « onClick » une navigation.

Dans les exemples ci-dessous deux méthodes de navigation sont utilisées :

- **Navigate()** : permet de naviguer vers un écran tout en préservant l'enchaînement des écrans, si bien qu'à la fermeture de l'écran appelé, le système affiche l'écran précédent.
- **Forward()** : permet de naviguer vers un écran tout en précisant que l'écran appelant ne sera pas préservé dans la chaîne des écrans appelés. Ainsi, lorsque l'écran est fermé, le système de navigation affiche l'écran précédant l'écran appelant.

Exemple de navigation

```
CtlButton demoBtn = new CtlButton( "demo", new CtlLocalizedText( "LG DEMO" ) );
demoBtn.addActionListener( new ActionListener()
{
    public void onClick( ActionEvent event )
    {
            NavigateContext ctx = new NavigateContext();
            ctx.setClassName( "DEMO" );
            ctx.setMethodName( "action1" );
            Navigator.getNavigator().navigate( ctx );
    }
} );
getResourceController().getButtonContainer( IResourceController.TOP_CONTAINER ).addLast(demoBtn);
```

Exemple de remplacement d'écran (forward)

```
CtlButton simpleBtn = new CtlButton( "simple", new CtlText( "Simple" ) );
simpleBtn.addActionListener( new ActionListener()
{
    public void onClick( ActionEvent event )
    {
            NavigateContext ctx = new NavigateContext();
            ctx.setClassName( "DEMO" );
            ctx.setMethodName( "action1" );
            Navigator.getNavigator().forward( ctx );
    }
} );
controller().getButtonContainer(IResourceController.BOTTOM_CONTAINER).addLast(simpleBtn);
```

Passer des paramètres à l'écran appelé

Il existe deux façons de passer des paramètres à l'écran appelé. La première est figée et statique, et implique que les paramètres à passer sont connus à l'avance (au moment de la construction d'un bouton, par exemple). La deuxième consiste à passer des paramètres qui ne peuvent être connus avant que l'action de naviguer soit effective.

Passer des paramètres pré-définis

Pour passer des paramètres pré-définis, il suffit de créer un contexte de navigation, puis poser des paramètres à l'aide des méthodes setParameter(…). Si vous maîtrisez la création du bouton, il est aussi possible de créer un « bouton de navigation ». Ce bouton de navigation implémente l'interface INavigateContext qui supporte le passage de paramètres.

Extrait de code de la classe DMProvider

Au moment où l'utilisateur cliquera sur le bouton, l'objet **workflowInstance** sera passé dans le contexte de navigation du provider associé à l'écran appelé.

```
CtlNavigationButton btn = getProcessDocument().createNavigationButton( "new", new
  CtlLocalizedText( "LG NEW VERSION" ), false,
  getProcessDocument().getNavigationActionListener());
```

```
btn.setClassName( "treatment" );
btn.setMethodName( "create" );
btn.setParameter( DMProvider.ORIGINAL_WORKFLOWINSTANCE, workflowInstance );
```

Passer des paramètres dynamiques

Dans le cas où la création du bouton est maîtrisée, il est possible de compléter les paramètres au moment où l'utilisateur clique sur le bouton.

Extrait de code de la classe PlayWithButtonsExtension

```
CtlButton demoBtn = new CtlButton( "demo", new CtlLocalizedText( "LG_DEMO" ) );
demoBtn.addActionListener( new ActionListener()
{
    public void onClick( ActionEvent event )
    {
            NavigateContext ctx = new NavigateContext();
            ctx.setClassName( "DEMO" );
            ctx.setMethodName( "action1" );

            Object obj1 = new Float(1);
            String stringValue = null;
            String[] stringsValue = null;

            ctx.setParameter( "objectParam", obj1 );
            ctx.setParameter( "textParam", stringValue );
            ctx.setParameter( "stringsParam", stringsValue );

            Navigator.getNavigator().navigate( ctx );
    }
} );
```

Dans le cas où un écran est surchargé, il est difficile de maîtriser la création d'un élément et le passage de paramètre. Il existe alors une méthode qui peut être surchargée dans le provider appelant : onNavigate(). Celle-ci prend en paramètre un contexte de navigation. A partir de ce contexte, les méthodes **contex.getClassName()** et **context.getMethodName()** permettent de retrouver vers quel écran le système navigue. Si bien qu'en fonction de l'écran cible, il est possible de spécifier des paramètres adaptés.

Extrait de code de la classe DemoFormProvider

```
public boolean onNavigate( INavigateContext context )
{
  super.onNavigate( context );

  if ("DEMO".equals(context.getClassName()) && "action1".equals(context.getMethodName()))
    context.getParameterMap().put( "param1", obj1 );

  return true;
}
```

Cas des sélecteurs

Le Kit de développement rend possible la création de sélecteurs par programmation. Il permet également de spécifier le composant graphique à utiliser pour réaliser la sélection.

Exemple de création de sélecteur par programmation

Le code suivant s'exécute dans le contexte d'une classe d'extension de document. Il permet de créer un bouton nommé « select » et de lui associer le comportement du sélecteur. Notez la variable membre « selector ».

```
public class UseSelector extends BaseDocumentExtension
{
 // keep the selector as a member
 private SelectorField selector = null;

 public boolean onAfterLoad()
 {
  // create a button
  CtlButton selectorButton = (CtlButton)new CtlButton( "select",
    new CtlLocalizedText( "LG SELECT" ) );
  getResourceController().getButtonContainer( IResourceController.TOP_CONTAINER )
    .addLast( selectorButton );

  // create the selector field
  selector = new SelectorField();
  selector.addParameter( "screen", "users" );
  selector.addParameter( "method", "select" );
  selector.addParameter( "multiple", "true" );

  // delegate the action to the button
  selector.delegate( selectorButton );

  // associate a ChangeListener to get informed when the value has changed
  selector.addChangeListener( new ChangeListener()
  {
   public void onChange( ChangeEvent event )
   {
    getWorkflowInstance().setValue( "fldOtherField", selector.getValue() );
   }
  } );

  return super.onAfterLoad();
 }
}
```

Rafraîchir l'écran appelant

A la fermeture d'un écran appelé, le système de navigation demande au provider associé à cet écran de déterminer si l'écran appelant doit être rafraîchi ou non. Le système de navigation prévoit une méthode mustRefresh() qui permet d'indiquer ce qu'il faut faire en fonction du traitement réalisé.

Exemple : si l'écran appelé permet de renseigner les informations d'un nouvel utilisateur créé, l'écran appelant peut nécessiter, dans le cas d'une sauvegarde, un rafraîchissement pour mettre à jour la liste des utilisateurs (écran appelant).

Extrait de code de la classe AcknowledgementSheetProvider

```
    public boolean mustRefresh()
    {
            // ne pas rafraichir le document principal car le document sera chargé
            // à nouveau et la sélection sera perdue.
            return false;
    }
```

Accéder aux boutons des formulaires

Plusieurs cas d'utilisation peuvent se présenter quant à l'utilisation des boutons :

- ajouter un listener ;
- simuler un clic ;
- cacher un bouton ;
- modifier le libellé.

Pour toutes ces opérations, il est nécessaire de retrouver le bouton. Les boutons, comme tous les éléments qui implémentent l'interface INamedWidget, possèdent un nom système interne qui les identifie.

Tableau récapitulatif des boutons disponibles dans les écrans

Libellé	Nom système
Enregistrer et fermer	saveAndClose
Enregistrer	save
Fermer	close
Annuler	cancel
Terminer	terminate
Précédent	previous
Suivant	next
Information	header
Historique	history
Annuler ma demande	abort
Relancer	reminder
Envoyer un message d'information	sendinformationmail
Déléguer le document	delegate_treatment
Déléguer la tâche	delegate_task
Réfuser la délégation	refuse_delegation
Annuler la délégation	cancel_delegation

Extrait de code de la classe TreatmentSearchSheetProvider

Dans cet exemple, les deux premiers boutons sont récupérés puis cachés et le troisième a son libellé modifié.

```
public void readyState()
{
  CtlButton saveButton = (CtlButton)getSheet().getButtonsContainer().get("save");
  saveButton.setHidden(true);
  CtlButton saveAndCloseButton = (CtlButton)getSheet().getButtonsContainer().get("saveandclose");
  saveAndCloseButton.setHidden(true);

  CtlButton closeButton = (CtlButton)getSheet().getButtonsContainer().get("close");
  closeButton.setLabel(new CtlLocalizedText("LG_OK"));

  super.readyState();
}
```

La surcharge des écrans

VDoc propose un mécanisme de surcharge des écrans standard. Il est en effet possible d'utiliser des mots-clés, tels que « extends », « override », pour étendre ou modifier des écrans sans perturber pour autant le bon fonctionnement du produit.

Tous les écrans standard sont identifiés par leur « nom » et leur « nom d'action ».

Tableau récapitulatif des écrans VDoc standards

Catégorie	Nom	Méthode	Description
explorer	server	browse	Administration niveau serveur
explorer	organization	browse	Administration niveau organisation
explorer	catalog	browse	Administration niveau application
explorer	version_group	browse	Administration niveau processus
explorer	treatment_class	browse	Administration niveau version de processus
explorer	resource_template	browse	Administration niveau version de modèle de resource
explorer	catalog	access	Affichage d'une application
form	subscription_rule	edit	Gestion des règles de souscription
form	external_references	edit	Propriétés d'une référence externe
form	external_references	create	Création d'une référece
form	role	edit	Propriété d'un role
form	role	create	Création d'un role
form	security	add	Edition d'une entrée de sécurité
form	security	edit_by_user	Présentation de la sécurité par utilisateur, groupe ou rôle
form	security	edit_by_type	Présentation de la sécurité par type
form	document	edit	Affichage du document
form	information_mail	create	Mail d'information
sheet	organization	edit	Propriétés d'une organisation
sheet	catalog	edit	Propriétés d'une application
sheet	catalog	security	Sécurité d'une application
sheet	catalog	create	Création d'une application
sheet	configuration	edit	Configuration du serveur
sheet	configuration	diagnose	Diagnostic du serveur
sheet	resource_subscriptions	edit	Abonnements inter-ressources
sheet	agent	edit	Propriétés d'un agent
sheet	agent	create	Création d'un agent
sheet	version_group	edit	Propriétés d'un procesus
sheet	version_group	security	Sécurité d'un processus
sheet	version_group	create	Création d'un processus
sheet	list	edit	Propriétés d'une liste
sheet	portlet	edit	Propriétés des portlets
sheet	portlet	create	Création d'une portlet
sheet	treatment_class	edit	Propriétés d'une version de processus
sheet	resource_template	edit	Propriétés d'une version de modèle de ressources
sheet	treatment	security	Sécurité d'un document
sheet	view	edit	Propriétés d'une vue
sheet	view	security	Sécurité d'une vue
sheet	stage_form	edit	Propriétés d'un formulaire d'étape
sheet	action_form	edit	Propriétés d'un formulaire d'action
sheet	process_form	edit	Propriétés d'un formulaire de version de processus
sheet	mailtemplate_form	edit	Propriétés d'un modèle de formulaire de mail
sheet	mail_form	edit	Propriétés d'un formulaire de mail
sheet	proceduresubform_form	edit	Propriétés d'un sous-formulaire
sheet	resourcetemplate_form	edit	Propriétés d'un formulaire de ressource
wizard	list	create	Création d'une liste
wizard	catalog	create_treatment	Création d'un document depuis l'administration d'une application
wizard	version_group	create_treatment	Création d'un document depuis l'administration d'un processus
wizard	treatment_class	create_treatment	Création d'un document depuis l'administration d'une version de processus
wizard	activity	transition	Changement d'étape sur un document processus
wizard	document	abort	Annulation d'un document
wizard	document	remind	Relance d'un document

wizard	process	generate	Génération d'une version de processus
wizard	process	duplicate	Duplication d'un processus
wizard	catalog	delete	Suppression d'une application
wizard	catalog	export	Export d'une application
wizard	catalog	import	Import d'une application
wizard	catalog	replacement	Remplacement de personnes
wizard	resource_subscriptions	create	Création d'un abonnement inter-ressources
wizard	process	delete	Suppression d'un processus
document	treatment	edit	Edition d'un document processus
screen	child-resource	edit	Edition d'une ligne de tableau dynamique
screen	help-document	read	Affichage de l'aide sur un document processus
singleselector	property	select	Sélecteur de propriétés
singleselector	actor	select	Sélecteur d'élément pour un champ rôle
singleselector	role	select	Sélecteur de rôle
multipleselector	properties	select	Sélecteur multiple de propriétés
multipleselector	ordered_properties	select	Sélecteur multiple trié de propriétés
multipleselector	actors	select	Sélecteur mutiple d'éléments pour un champ rôle
multipleselector	ordered_actors	select	Sélecteur multiple trié d'éléments pour un champ rôle
multipleselector	roles	select	Sélecteur multiple de rôles

Exemples de surcharge d'écran

Dans cette section une série d'exemples sont présentés pour vous aider à construire des écrans basés sur les écrans standard de VDoc.

L'écran de base utilisé pour tous les exemples est le suivant :

```
<test name="element" action="edit">
  <fields>
    <field name="A" param1="value1" />
    <field name="B" param2="value2">
      <subfield name="B1" param21="value21" />
    </field>
    <field name="C" param3="value3" />
  </fields>
</test>
```

Héritage simple

Dans cet exemple, l'écran nommé [element2, edit] sera strictement identique à l'écran de base nommé [element,edit].

```
<test name="element2" action="edit" extends="element.edit"></test>
```

Ajout d'un élément

Dans cet exemple, l'écran nommé [element3, edit] possédera un élément supplémentaire « buttons ».

```
<test name="element3" action="edit" extends="element.edit">
  <buttons/>
</test>
```

Ajout d'un attribut

Dans cet exemple, l'écran nommé [element4, edit] possédera un attribut supplémentaire « ctrl ».

```
<test name="element4" action="edit" extends="element.edit" ctrl="my.lovely.widget"></test>
```

Ajout d'un sous-attribut

Dans cet exemple, l'écran nommé [element5, edit] possédera un sous-attribut supplémentaire « param4 » sous l'élément « fields ».

```
<test name="element5" action="edit" extends="field.edit">
  <fields param4="value4" />
</test>
```

Suppression d'un élément

Dans cet exemple, l'écran nommé [element6, edit] ne contiendra que deux éléments « field A » et « field C ». L'élément « field B » sera supprimé.

```
<test name="element6" action="edit" extends="field.edit">
  <fields>
    <field name="B" override="delete" />
  </fields>
</test>
```

Surcharge d'un écran

Dans cet exemple, l'écran nommé [element, edit] est surchargé. Désormais, il contiendra un un élément « field D » supplémentaire.

```
<test name="element.edit">
  <fields>
    <field name="D" param4="value4" />
  </fields>
</test>
```

Extrait de surcharge du projet Educaction

Dans cet exemple, l'écran nommé [catalog, access], qui correspond à l'application, est surchargé. Deux éléments sont ajoutés :

- une option donnant accès à l'aide de la Gestion des processus ;
- un lien donnant accès à la vue des accusés de réception.

```
<explorer override="catalog.access">
  <options>
    <option name="help" label="LG_HELP">
      <action class="help" name="browse" />
    </option>
  </options>
  <tabs>
    <tab name="follow-up" label="LG FOLLOW UP">
      <links>
        <link name="follow-up" label="LG_FOLLOW_UP" default="true">
          <view name="follow-up" label="LG_FOLLOW_UP_VIEW"
            provider=
            "com.axemble.education.providers.acknowledgement.AcknowledgementViewProvider">
          </view>
        </link>
      </links>
    </tab>
  </tabs>
</explorer>
```

Chapitre 4 : Le document processus

Introduction

Le terme « document processus » se rapporte spécifiquement à un document issu d'un modèle de processus (workflow) qui se construit à partir des éléments générés présents dans la Gestion des processus.

Le document processus ne sort pas du cadre de la navigation présenté dans le chapitre 3. Il est donc déclaré dans le fichier XML de définition des écrans mais sous une forme la plus simple qu'il soit.

Déclaration du document dans le fichier de définition

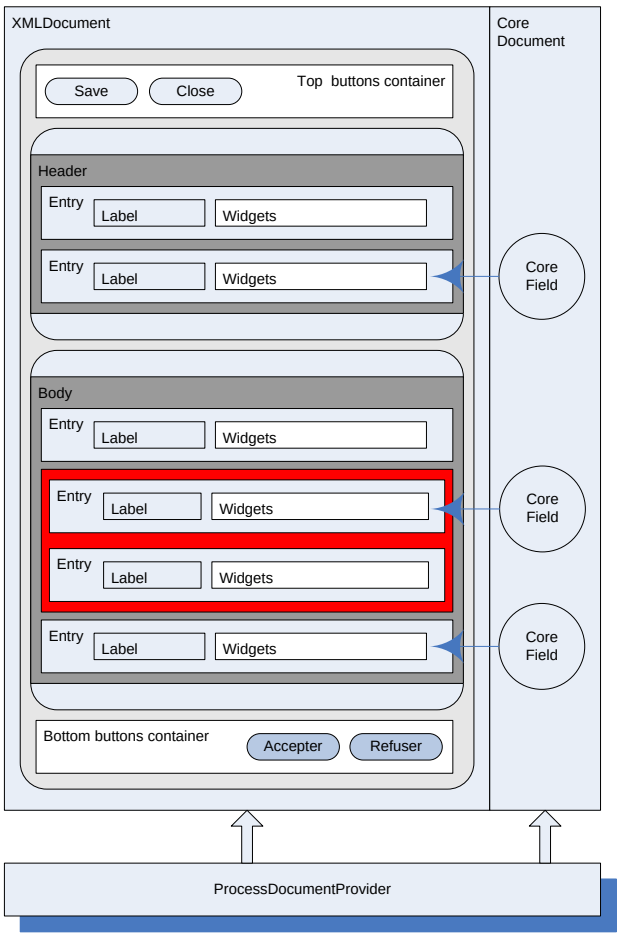
```
<document name="treatment" action="edit">
</document>
```

Une telle déclaration implique qu'il est possible de surcharger l'écran du document processus de manière standard et globale au serveur : intervention sur les boutons (ajout / masquage), sur les sections d'en-tête ou d'historique, et les règles d'affichage des boutons d'annulation, de relance.

De la même façon que les écrans (type formulaire), un document processus manipule en interne une classe modèle de représentation graphique, une classe document et un fournisseur.

Schéma de principe

Le schéma suivant présente les objets mis en œuvre dans un document processus.



La classe modèle

La classe modèle de représentation graphique utilisée dans les documents processus est CtlAbstractDocument. Son nom complet est **com.axemble.vdp.ui.framework.composites.base.CtlAbstractDocument**.

Pour tout document processus, cette classe modèle manipule les objets suivants :

- Deux containers de formulaires : l'en-tête (HEADER_DOCUMENT) et le corps du document composé d'un formulaire de processus (EDIT_DOCUMENT), ou d'un formulaire d'étape dans le cas d'une intervention. Ces deux containers utilisent un objet unique CoreDocument ;
- Deux containers de boutons : l'ensemble des boutons placés en haut du formulaire TOP_CONTAINER et ceux présents dans la partie basse du formulaire BOTTOM_CONTAINER.

La classe de document

La classe document utilisée dans les documents processus est CoreDocument. Son nom complet est **com.axemble.vdp.ui.core.document.CoreDocument**.

Tout comme la classe AbstractDocument, celle-ci maintient en mémoire une liste de champs correspondant aux « field » décrits dans les formulaires d'administration. C'est le représentant Java des données des formulaires présentés à l'utilisateur final.

Cette classe, notion abstraite et dynamique représentant les données stockées en base, permet principalement de modifier les valeurs des champs du « document processus » en dynamique sans forcer le stockage physique des données en base. En effet, une série d'actions peut être exécutée sur cette classe sans pour autant que les données en base ne soient altérées.

Elle permet également de retrouver des informations dynamiques telles que l'étape ou l'activité en cours, l'utilisateur connecté, le document père (cas des tableaux dynamiques), et aussi les objets back-office. Toutefois, ces informations sont accessibles depuis les API SDK et la navigation.

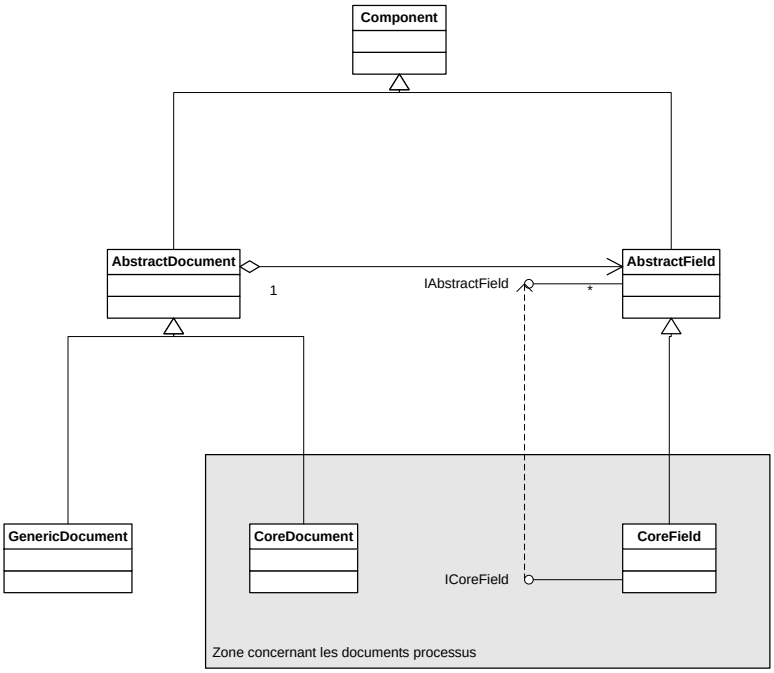
Méthodes principales de la classe CoreDocument

Méthode	Description
public boolean isCreationMode()	Grâce à cet état, il est possible de savoir si le document en cours de chargement vient d'être créé ou non.
public boolean isLoading()	Cet état indique si le document est en cours de chargement ou non.
public boolean isModified()	Cet état indique si le document a été modifié.
public IFieldControl getMainControlByPropertyName(String name)	Retrouve le contrôle graphique principal (souvent en mode édition) correspondant à une propriété donnée.
public ICoreField getFieldByName(String name)	Retrouve l'objet CoreField via le nom système de la propriété.
public Collection getFields()	Retrouve l'ensemble des objets CoreField actifs sur le document.
public void saveToResource()	Sauvegarde le document. Les données seront stockées en base.
public CoreDocument getParentDocument()	Retrouve le document parent dans le cas des tableaux dynamiques de type « documents liés ».
public ServiceManager getServiceManager()	Retrouve l'objet service permettant d'accéder à tous les autres gestionnaires et les services.
public User getUser()	Retrouve l'utilisateur connecté.
public Catalog getCatalog()	Retrouve le catalogue courant.
public ResourceTemplate getResourceTemplate()	Retrouve le modèle de ressource associé avec le document.

public Treatment getTreatment()	Retrouve l'objet back-office représentant le document en base.
public Resource getResource()	Retrouve l'objet back-office représentant l'ensemble des valeurs du document.
public ManualActivity getActivity()	Retrouve l'étape active du document (accès en mode intervention)
public String getStageName()	Retrouve le nom système de l'étape active du document.
public String getStoragePath()	Renvoie le répertoire racine des descripteurs de composants et de leur interface graphique associée.

Diagramme simplifié de la classe CoreDocument

Le diagramme suivant présente les relations qui existent entre les couples (AbstractDocument, AbstractField) et (CoreDocument, CoreField).



Ce diagramme identifie deux catégories de document :

- **CoreDocument** : pour les documents processus ;
- **GenericDocument** : pour tous les écrans réalisés pour l'administration, l'application ou ceux autour du document, tels que l'assistant de création, de changement d'étape, d'annulation, de relance, de suppression, ou encore celui de l'envoi de mail d'information.

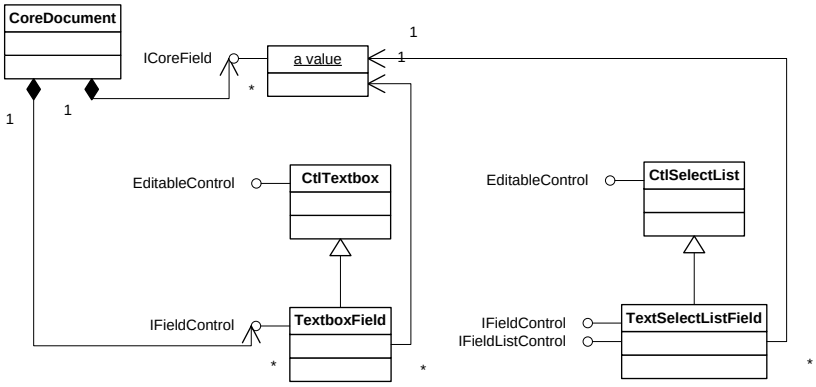
VDoc utilise pleinement cette classe de document pour gérer les abonnements entre champs et toutes les ouvertures de code Java (extensions) ou Java Script.

Les objets liés à la classe CoreDocument

La classe CoreDocument est liée à des champs qui sont constitués de composants graphiques et d'une classe CoreField qui maintient la valeur manipulée.

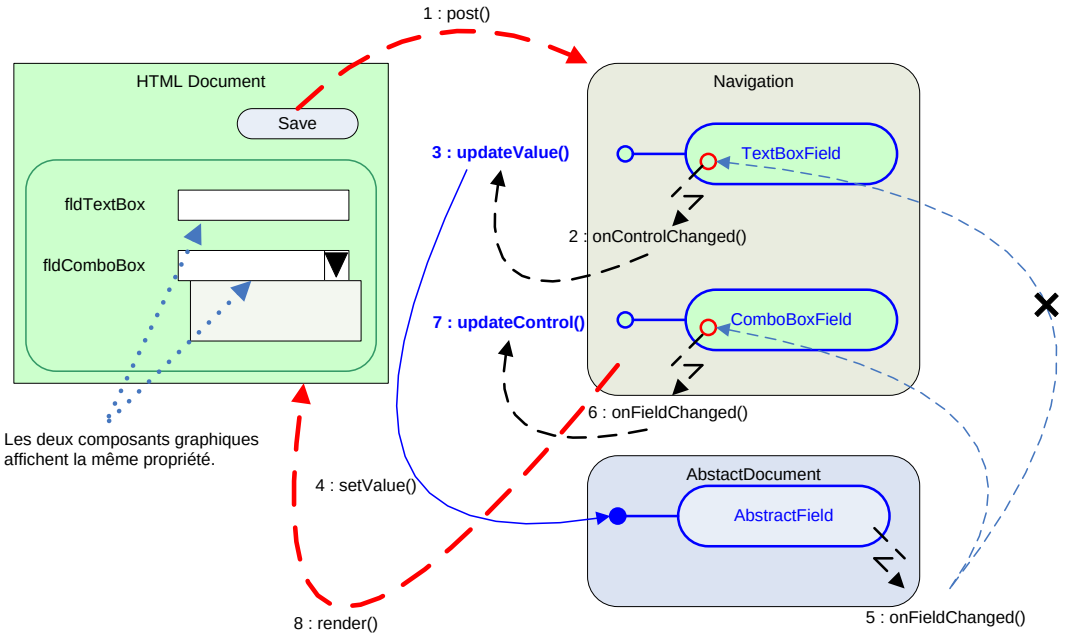
Les champs du document implémentent tous l'interface IFieldControl. Pour ceux qui gèrent plusieurs valeurs (ex. sélecteur d'annuaire multiple), ils implémentent l'interface IFieldListControl.

Schéma de principe entre les objets liés



Il existe un câblage d'évènements entre le CoreField (la valeur) et les composants graphiques associés. En effet, si la valeur d'un champ est modifiée via l'objet CoreDocument, tous les composants graphiques seront rafraîchis. A l'inverse, si un utilisateur modifie la valeur d'un champ (depuis l'interface) ce dernier déclenchera un évènement qui mettra à jour le CoreField (la valeur), et éventuellement les autres composants graphiques associés à cette même valeur.

Mécanisme d'évènements entre les objets liés



Tout ce mécanisme est entièrement géré dans VDoc. Il est applicable dans le cas d'abonnement sur les champs mais également au cours de l'exécution des extensions. Ce mécanisme est placé sur les éléments de base : AbstractDocument et AbstractField, si bien que la classe GenericDocument bénéficie également de ce mécanisme évènementiel.

Classe CoreField

La classe CoreField est une représentation abstraite de la valeur d'un champ du document. Elle implémente l'interface ICoreField. Le but de cette classe est de découpler la valeur de chaque champ du document de son aspect graphique. En effet, selon la conception réalisée au niveau du web Designer, un champ peut apparaître plusieurs fois dans le document (sur plusieurs sections). La valeur pourra de ce fait être représentée différemment (ex. sous forme de texte simple sur une section en mode lecture, sous la forme d'une liste sur la section d'intervention).

L'interface ICoreField

```
public interface ICoreField
{
public boolean loadFromResource();
public boolean loadFromAnotherDocument(CoreDocument srcDocument);
```

```
public boolean saveToResource();
public boolean isModified();
public Property getProperty();
public String getPropertyName();
public boolean setValue(Object value);
public boolean setValue(Object value, Object changeSource);
public Object getValue();
public boolean equalsValue(Object valueToCompare);
public CoreDocument getDocument();
public void release();
public void setList(Collection values, Object changeSource);
public Collection getList();
}
```

La classe fournisseur

La classe fournisseur associée par défaut aux documents processus est : DefaultProcessDocumentProvider. Son nom complet est **com.axemble.vdp.ui.core.providers.documents.DefaultProcessDocumentProvider**.

Il est possible de déclarer sa propre classe fournisseur à l'aide du fichier de définition des écrans (comme pour les autres écrans).

Exemple de surcharge de l'écran « document processus »

Dans cet exemple, plusieurs configurations standard sont réalisées :

- L'écran nommé [treatment, edit] correspondant à celui du document processus est surchargé ;
- L'en-tête du document est affiché et figé ;
- L'historique du document est masqué mais peut être activé grâce au bouton « Historique » ;
- Les règles d'affichage des boutons d'annulation, de relance, ainsi que de délégation sont figées respectivement à seul le créateur peut activer le bouton « Annuler ma demande », seuls les intervenants passés peuvent activer le bouton « Relancer ».
- Trois boutons sont ajoutés au document. Le premier est un bouton de navigation permettant d'accéder à la Gestion des processus. Le second permet d'afficher l'aide. Il pointe sur un lien local au serveur. Le dernier permet d'afficher une page web externe.
- Le bouton « Enregistrer » est masqué pour tous les documents ;
- Enfin, un fournisseur de document processus est associé à cet écran.

```
<document override="treatment.edit" provider="com.axemble.education.providers.DemoProvider">
  <header display="true" pin="true" />
  <history display="false" pin="false" />
  <rules>
    <rule name="abort" use="checkCreator" />
    <rule name="reminder" use="checkPastActors" />
  </rules>
  <buttons>
    <button name="admin" label="LG_ADMINISTRATION" style="style3">
      <action class="server" name="browse" />
    </button>
    <button name="help" label="LG_HELP" style="style1">
      <action url="html/help/vdoc Process Java user/fr/index.html" scope="relative" params="896,
        704, 1, 1, 0, 0" />
    </button>
    <button name="google" label="LG GOOGLE" style="style1">
      <action url="http://www.google.com" scope="absolute" params="896, 704, 1, 1, 0, 0" />
    </button>
    <button name="save" override="delete"></button>
  </buttons>
  <workflow-buttons />
</document>
```

Exemple d'une classe fournisseur associée au document processus

```
public class DemoProvider extends AbstractProcessDocumentProvider
{
  public DemoProvider(INavigateContext context,AbstractDocument document,CtlAbstractDocument
    abstractDocument )
  {
    super( context, document, abstractDocument );
  }
```

```
  public boolean evaluateAbortRules()
  {
    // si la valeur renvoyée est "true", le bouton d'annulation s'affichera
    return false;
  }
  public boolean evaluateDelegateRules()
  {
    // si la valeur renvoyée est "true", le bouton de délégation s'affichera
    return false;
  }
  public boolean evaluateReminderRules()
  {
    // si la valeur renvoyée est "true", le bouton de relance s'affichera
    return false;
  }
  public void readyState()
  {
    super.readyState();

    // ici, il est possible d'ajouter, masquer des boutons,
    // d'affecter certaines valeurs du document.
  }
}
```

Les champs

Depuis le web Designer il est possible de créer un certain nombre de champs basés sur des types prédéfinis tels que les champs de type texte, liste, date, pièces jointes, etc. Pour ne pas être limités, VDoc offre un système ouvert qui permet l'ajout de nouveaux types de champs sans altération du web Designer, du générateur ou des APIs.

Les champs fournis en standard sont les mêmes que ceux présentés dans le chapitre 3 concernant la navigation.

Tableaux des champs disponibles directement depuis le menu du web Designer

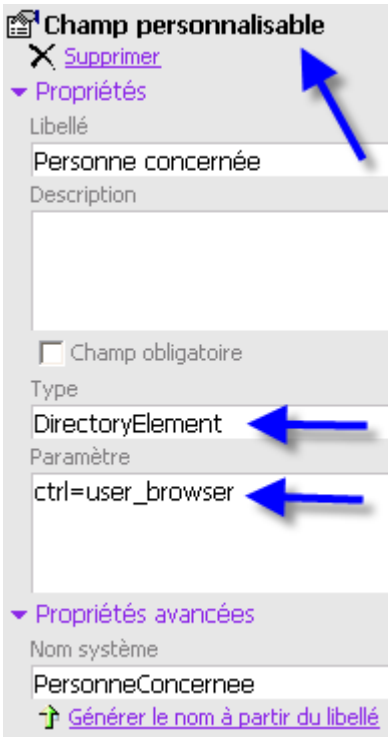
Catégorie	Champ	Description
Champs simples	Texte	Les valeurs de ce type de champ sont stockées dans la table « vdp_string_values »
	Nombre	Les valeurs de ce type de champ sont stockées dans la table « vdp_float_values »
	Date	Les valeurs de ce type de champ sont stockées dans la table « vdp_timestamp_values »
Champs composites	Période	Les valeurs de ce type de champ sont stockées dans la table « vdp_timestamp_values » (deux enregistrements par champ période)
	Liste	Les valeurs de ce type de champ sont stockées dans la table « vdp_string_values » liées à la table « vdp_collection_items »
	Pièces jointes	Les valeurs de ce type de champ sont stockées dans la table « vdp_file_values »
	Liste de personnes	Les valeurs de ce type de champ sont stockées dans la table « vdp_directory_elements » liées à la table « vdp_collection_items »
	Rôle	Les valeurs de ce type de champ sont stockées dans la table « vdp_user_values ». Chaque ligne correspond à un pointeur vers un utilisateur
	Tableau	Les valeurs de ce type de champ sont stockées dans la table « vdp_line_values ». Chaque ligne correspond à un pointeur vers

		une ressource
	Tableau de sous-processus	Les valeurs de ce type de champ sont stockées dans la table « vdp_treatment_values ». Chaque ligne correspond à un pointeur vers un sous-processus

Les champs personnalisables

Pour ajouter un champ personnalisable au processus :

- Dans le web Designer, sélectionner « **Insertion** > **Champ** > **Champ personnalisable** ;
- Dans le panneau de propriétés, saisissez le type et le paramétrage nécessaire respectivement dans les zones « Type » et « Paramètre ».



Les champs du document

Les sections suivantes présentent les différents types de champs personnalisables déployés et supportés par VDoc.

Nom	Type	Paramètres	Description
Tableau de documents liés	Collection	ctrl=”treatmenttable” catalogname=”APPLICATION_NAME” treatmentclassname=”PROCESS_NAME”	Tableau de documents liés instances du processus PROCESS_NAME de l’application APPLICATION_NAME
Groupe de cases à cocher	Collection	ctrl=”checkboxgroup” fillingmode=”vertical” nbofcolumns=”n” allowreset=”true” allowselectall=”true”	Ce type de champ s’applique sur les champs de type liste avec sélection multiple. Le paramètre ‘fillingmode’ peut prendre les valeurs vertical ou horizontal. Ce paramètre indique le sens de remplissage du groupe de cases à cocher. Le deuxième paramètre ‘nbofcolumns’ correspond au nombre de colonnes souhaitées. ‘allowreset’ et ‘allowselectall’ permettent respectivement de désélectionner tous les éléments et de tous les sélectionner.
Groupe de bouton radio	String	ctrl=”radiogroup” fillingmode=”vertical” nbofcolumns=”n” allowreset=”true”	Ce type de champ s’applique sur les champs de type liste avec sélection simple. Le paramètre ‘fillingmode’ peut prendre les valeurs vertical ou horizontal. Ce paramètre indique le sens de remplissage du groupe de boutons radio. Le deuxième paramètre ‘nbofcolumns’ correspond au nombre de colonnes souhaitées. Le paramètre ‘allowreset’ permet de remettre à vide la sélection.

Les sélecteurs d’annuaire simples

Nom	Type	Paramètres	Description
Sélecteur d’une personne	DirectoryElement	ctrl=”user_browser”	Sélecteur simple d’utilisateur
Sélecteur d’un groupe	DirectoryElement	ctrl=”group_browser”	Sélecteur simple de groupe
Sélecteur d’une organisation	DirectoryElement	ctrl=”organization_browser”	Sélecteur simple d’organisation
Sélecteur d’une localisation	DirectoryElement	ctrl=”localization_browser”	Sélecteur simple de localisation
Sélecteur d’une entrée dans FileCenter	DirectoryElement	ctrl=”com.axemble.vdp.ui.core.document.fields.SingleDirectoryField” type=”filecenter”	Sélecteur simple d’élément dans FileCenter

Les sélecteurs d’annuaire multiples

Nom	Type	Paramètres	Description
Sélecteur de personnes	Collection	ctrl=”user_browser_multiple”	Sélecteur multiple d’utilisateurs. Note : équivalent du sélecteur de personnes.
Sélecteur de groupes	Collection	ctrl=”group_browser_multiple”	Sélecteur multiple de groupes
Sélecteur de organisations	Collection	ctrl=”organization_browser_multiple”	Sélecteur multiple d’organisations
Sélecteur de localisations	Collection	ctrl=”localization_browser_multiple”	Sélecteur multiple de localisations

Les sélecteurs de page JSP

Nom	Type	Paramètres	Description
Exemple de sélecteur de page JSP	Map	ctrl=”jspsample_browser”	Ce sélecteur permet de réaliser des développements spécifiques autour du document dans une page JSP. Il intègre un mécanisme de paramétrage lors du lancement de la page JSP et un système de mises à jour du document lors de la validation de celle-ci.

Le champ FCK Editor

Nom	Type	Paramètres	Description
FCKEditor	[B	ctrl=”fckeditor” toolbar=”Basic”	Ce champ intègre le composant FCK Editor. Il permet de bénéficier des mêmes options via une personnalisation. L’attribut toolbar est optionnel. Toutefois, il est possible de bénéficier des barres d’outils standard. Il suffit de sélectionner l’une des entrées suivantes : Default | Basic | Basic01 | Custom01 | Custom02

Le champ EnhancedUrl

Nom	Type	Paramètres	Description
EnhancedUrl	String	ctrl="com.axemble.vdoc.sdk.document.fields.EnhancedUrlField" title="VDoc Software" description="Editeur de logiciel" url="http://www.vdocsoftware.com"	Ce champ permet de créer un lien URL en spécifiant le titre du lien, l’URL, la description et les paramètres d’ouverture. Il est possible de faire apparaître ou non les zones de saisie. En effet, les attributs suivants peuvent être utilisés : • **show-title** : affiche la zone de saisie du titre ; • **show-url** : affiche la zone de saisie de l’URL ; • **show-description** : affiche la zone de saisie de la description ; • **show-params** : affiche la zone de saisie des paramètres d’ouverture. Ce champ est principalement utilisé en mode lecture. Il est possible d’alimenter ce champ en abonnement JavaScript sur un événement onAfterLoad, par exemple. Exemple de l’abonnement : `<script>` `<![CDATA[` `function onAfterLoad()` `{` `  var url = iResourceController.getDefaultWidget( "fldURL" );` `  url.setTitle( "VDoc Software" );` `  url.setUrl( "http://www.vdocsoftware.com" );` `  url.setDescription( "Editeur de logiciel");` `}` `]]>` `</script>`

Implémentation d'un champ personnalisable

Le framework UI permet de créer de nouveaux qui pourront simplement être intégrés dans les doucments processus ou dans les écrans génériques.

Pour créer un champ personnalisable, il suffit de créer une classe Java qui étend la classe de base nommée **com.axemble.vdoc.sdk.document.fields.base.BaseField**.

Extrait de code

Le code suivant présente les méthodes nécessaries à implementer pour réaliser un champ.

```
public class SimpleField extends BaseField
{
     public void init( Element element )
     {
     }
     public void updateControl()
     {
     }
     public void updateValue()
     {
     }
     public boolean isEmpty()
     {
            return false;
     }
     public IWritable render() throws RenderException
     {
            return new CtlButton("okButton",new CtlText("Bouton OK"));
     }
}
```

La méthode **init**() reçoit en argument un objet org.w3c.dom.Element représentant la description XML du champs dans les formulaires de personnalisation ou les écrans du fichier de définition.

La méthode **updateControl**() est appelée à chaque fois que la valeur interne du champ est altérée. Le framework demande au champ développé de se mettre à jour en fonction de cette valeur.

La méthode **updateValue**() est appelée à chaque fois que l'utilisateur modifie les valeurs dans le document HTML.

La méthode **isEmpty**() doit être renseignée par le champ développé. Dans le cas où le champ est marqué obligatoire, si cette méthode renvoit la valeur « true », le framework indiquera que le champ doit être renseigné.

La méthode **render**() est appelée par le framework pour inscrire la partie graphique du champ dans le document HTML. Dans l'exemple précédent, le composant graphique affiché sera un bouton.

Définition des templates HTML

Les champs personnalisables peuvent être des composants composites des éléments graphiques proposés par le framework VDoc (utilisation du widget Container). Dans ce cas, ils n'ont pas besoin de définir de template HTML.

Cependant, dans de nombreux cas, il sera nécessaire de définir plusieurs tempates HTML en fonction d'un état interne du champ (exemple : mode écriture, mode lecture).

Le framework d'interface graphique de VDoc prévoit un dossier « custom » pour déposer les templates HTML spécifiques : **WEB-INF\storage\custom\controls**. En déposant les templates HTML dans ce dossier, il sera possible de les atteindre via la méthode getTemplateWriter().

Exemple d'un template HTML

Dans cet exemple sont présentés un ensemble de balises entourées du symbol « $ ».

```
<input type="text" name="$id$" value="$label$">
$error-message$
```

Exemple d'utilisation d'un template HTML

Cet exemple montre comment remonter un objet TemplateWriter à partir d'un fichier HTML (template HTML).

Une fois, le TemplateWriter remonté, il est possible de remplacer les balises xxx par des éléments de type IWritable grâce à la méthode setEntry() de la classe TemplateWriter. Il est également possible de récupérer un objet EntryWriter pour inscrire plusieurs éléments de type IWritable dans une seule balise : utilisation de la méthode getEntryWriter de la classe TemplateWriter.

```
public IWritable render() throws RenderException
{
    if ( isHidden() )
      return null;

    TemplateWriter tw = null;
    if ( this.isEditable() )
    {
      tw = this.getTemplateWriter( "CtlTextBox.edit.html" );
      tw.setEntry( "id", new TemplateToken( this.getNavigator().registerWidget( this ) ) );
      tw.setEntry( "label", new TemplateToken( HTMLUtils.getHTMLAttrString( this.textValue ) ) );

      if ( checkErrorMessage() )
        tw.setEntry( "error-message", this.renderErrorMessage() );
    }
    else
    {
      tw = this.getTemplateWriter( "CtlTextBox.read.html" );
      tw.setEntry( "label", new TemplateToken( HTMLUtils.getHTMLString( this.textValue ) ) );
    }

    return tw;
}
```

Implémentation d'un champ personnalisable dédié processus

Un champ dédié document processus ne peut être utilisé que dans le contexte d'un document processus. Il ne pourra être déclaré dans un écran générique.

Pour implémenter un tel champ, il suffit de créer une classe Java qui étend la classe nommée **com.axemble.vdoc.sdk.document.fields.base.BaseWorkflowField**.

Les documents liés

La notion de documents liés apparaît avec les tableaux. Elle apparaît sous deux grandes formes :

- **Tableau de sous-ressources** : tableau dynamique associé à un nouveau modèle créé lors de la génération. Chaque ligne du tableau correspond à une nouvelle ressource interne basée sur sa description associée.
- **Tableau de sous-processus** : tableau dynamique associé à un modèle de processus existant (déjà généré). Chaque ligne du tableau correspond à un nouveau document lié.

Tableau dynamique

Depuis le web Designer, il est possible d'associer un champ de type tableau à une tâche, puis de décrire ce tableau en ajoutant des « sous-champs » correspondant aux colonnes.

A la génération, pour chaque champ tableau défini dans le processus, un nouveau modèle de ressources est généré et placé en référence sur le champ lui-même. Un formulaire de ressources est également généré pour la saisie des champs d'une ligne du tableau.

A l'exécution, l'utilisateur peut ajouter une nouvelle ligne à l'aide du bouton « Créer ». A l'activation de ce bouton, une nouvelle ressource interne est créée.

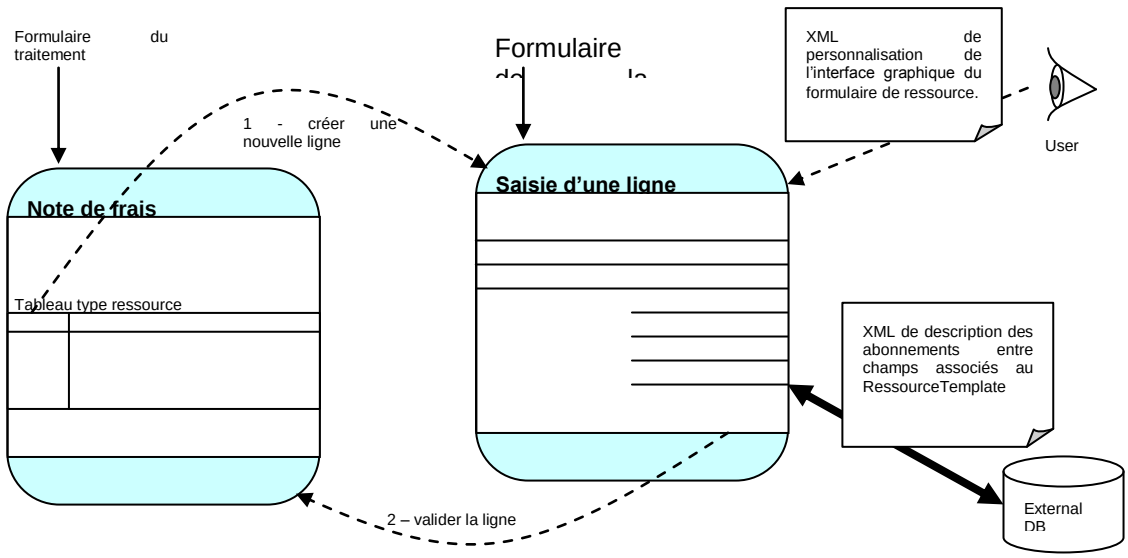
Définition d'une ligne de tableau

Une ligne d'un tableau dynamique est une ressource interne de type ILinkedResource. Un champ « tableau dynamique » est une collection d'objets ILinkedResource.

Schéma de principe

Ce schéma présente plusieurs informations :

- L'enchaînement des écrans ;
- La personnalisation de l'écran de saisie d'une ligne ;
- La possibilité de réaliser des abonnements comme pour le document principal.



Tableaux de sous-processus

Le web Designer permet de traiter le cas du tableau de type sous-processus en ajoutant à une tâche un champ de type « **tableau de sous-processus** ». Il faut renseigner :

- le nom système du processus fils à utiliser ;
- le nom système de l'application dans laquelle le processus fils a été généré ;
- le nom système de la vue associée.

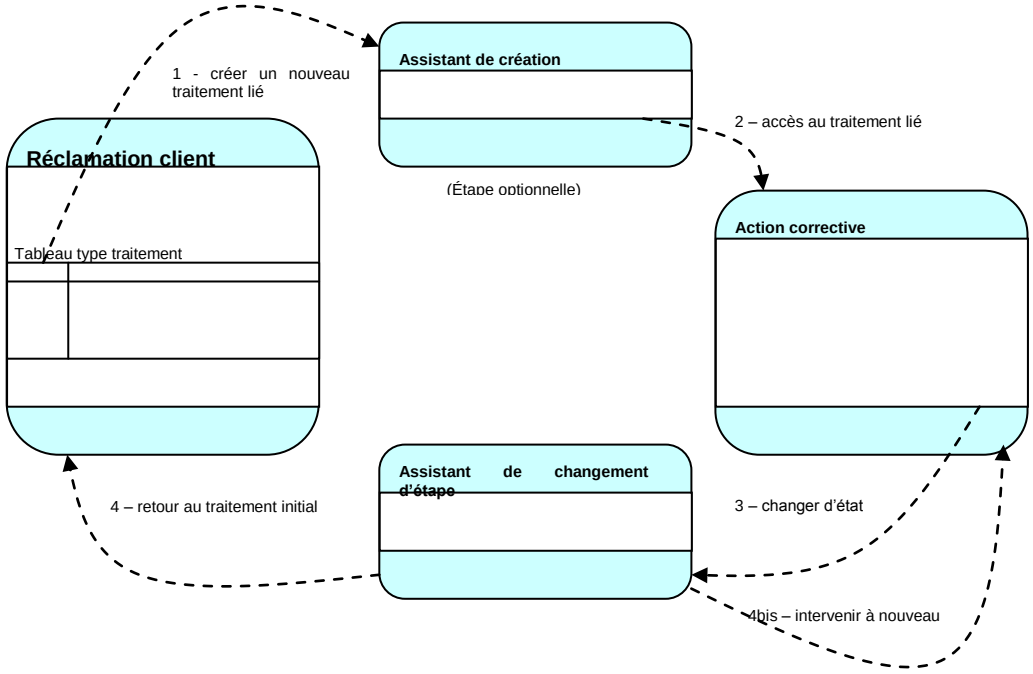
A l'utilisation, une nouvelle ligne peut être ajoutée à l'aide du bouton « Créer ». A l'activation de ce bouton, l'assistant de création de document est appelé. A la validation de ce dernier un « sous-processus lié » est créé et associé au document principal.

Définition d'une ligne de tableau de sous-processus

Une ligne de tableau de sous-processus est un traitement lié de type IWorkflowInstance. Un champ « tableau de sous-processus » est une collection d'objets IWorkflowInstance.

Schéma de principe

Ce schéma présente l'enchaînement des écrans. La personnalisation et les abonnements sont valables dans ce contexte.



Activer/Désactiver l'assistant de création de documents liés

VDoc vous permet d'activer ou non l'assistant de création du document lié.

La manière de gérer l'activation de l'assistant de création est identique quel que soit le contexte d'appel. Depuis un document père ou depuis le portail l'assistant réagira de la même façon : il sera activé s'il existe au moins une balise <field> dans le document de personnalisation du formulaire CREATE_DOCUMENT. A l'inverse, si aucun champ n'est spécifié, il ne sera pas affiché.

> *Toutes les extensions et les codes Java Script présents dans le formulaire seront pris en compte et exécutés même si l'assistant ne s'affiche pas.*
> *C'est aussi l'endroit indiqué pour définir des valeurs par défaut sur certains champs du document au moment de la création.*

Assistant de changement d'étape

De la même façon que pour l'assistant de création, si aucune balise <field> n'est présente dans le formulaire de personnalisation de l'action, l'assistant de changement d'étape ne sera pas affiché à l'utilisateur.

Les extensions et code Java Script seront exécutés, puis s'affichera la page de rapport.

Personnalisation de la page de rapport

La page de rapport permet d'afficher la liste des intervenants suivants par étape (en cas d'étapes en parallèle) et/ou la liste des étapes suivantes (sans les intervenants). Ces informations affichées étant parfois non souhaitées, VDoc offre une possibilité pour les intégrateurs de personnaliser ce rapport.

Il est possible de définir la propriété "TRANSITION_REPORT" sur l'objet coreDocument pour ne pas afficher le rapport en exécutant automatiquement une action.

Les options possibles

- "**autoTerminate**" Appuie sur le bouton Terminer, et ferme l'assistant ;
- "**autoContinue**" Appuie sur le bouton Continuer s'il est présent pour ré-ouvrir le document, sinon affiche le rapport ;
- "**hideContinue**" Cache le bouton Continuer si besoin, et le texte associé ;
- "**hideActors**" Cache la liste des intervenants pour chaque activité ;
- "**hideActivities**" Cache la liste des activités, donc également des utilisateurs.

Ces valeurs peuvent être cumulées séparées par des virgules.

Exemple : "autoContinue, autoTerminate" Appuie sur le bouton Continuer s'il est présent pour ré-ouvrir le document, sinon appuie sur le bouton Terminer, et ferme l'assistant.

Cette propriété peut-être affectée dans une classe d'extension java ou dans un Java Script de formulaire d'action ou sur le modèle de ressource.

Exemple de positionnement de valeur pour la personnalisation du rapport de l'assistant de changement d'étape

```
<script>
  coreDocument.setContext("TRANSITION_REPORT","autoContinue,autoTerminate");
</script>
```

Gestion des boutons du tableau

Utilisation des attributs de personnalisation

Plusieurs mots clés sont disponibles pour activer ou non les boutons des tableaux.

Mot clé	Description
allowcreate	Permet d'activer ou non le bouton de création.
allowremove	Permet d'activer ou non le bouton de suppression.
alloweditchild	Permet de définir si l'utilisateur peut éditer une ligne de tableau.
allowreadchild	Permet de définir si l'utilisateur peut afficher une ligne de tableau.
allowselectchild	Permet d'indiquer s'il est possible de sélectionner des éléments.
securecreation	Permet d'indiquer que le bouton de création ne s'affichera que si la personne connectée possède les droits de création d'éléments fils nécessaires du processus fils.
securedeletion	Permet d'indiquer que le bouton de suppression ne s'affichera que si la personne connectée possède les droits nécessaires de suppression des éléments fils.

elementsperpage	Nombre d'éléments par page affichée. La valeur par défaut est 50.

Exemple d'utilisation des attributs de tableaux

```
<section labelid="Property:DetailDeLaNote:label" display="" class="subsection"
descriptionid="Property:DetailDeLaNote:description">
  <line>
    <col>
      <field property="DetailDeLaNote" ctrl="resourcetable" resourcetemplate="DetailDeLaNote"
        mode="write"
        required="no"
        allowcreate="false"
        allowremove="false"
      >
        <subfield mode="write" required="yes" property="Date" ctrl="date"
          labelid="TableView:DetailDeLaNote:col#Date#label" />
        <subfield mode="write" required="yes" property="Client" ctrl="text"
          labelid="TableView:DetailDeLaNote:col#Client#label" />
        <subfield mode="write" required="no" property="Type" ctrl="textselectlist"
          labelid="TableView:DetailDeLaNote:col#Type#label" />
        <subfield mode="write" required="yes" property="Cout" ctrl="float"
          labelid="TableView:DetailDeLaNote:col#Cout#label" />
        <subfield mode="write" required="no" property="Commentaires2" ctrl="textarea"
          labelid="TableView:DetailDeLaNote:col#Commentaires2#label" />
      </field>
    </col>
  </line>
</section>
```

Interaction avec les boutons du tableau

VDoc permet de surcharger le comportement de ses champs. De ce fait, il est possible de réagir aux évènements déclenchés par un tableau. Tous les tableaux implémentent l'interface **ITableSupport**. Son nom complet est **com.axemble.vdp.ui.core.providers.ITableSupport**.

L'interface ITableSupport

Cette interface permet de réagir sur la création d'élément fils, de suppression unitaire ou multiple d'éléments, ou encore à l'ouverture d'une ligne.

```
public interface ITableSupport
{
  boolean onCreateChild( CtlInputTable table );

  boolean onDeleteChild( CtlInputTable table, AbstractDocument child );

  boolean onDeleteChildren( CtlInputTable table, Collection children );

  boolean onOpenChild( CtlInputTable table, AbstractDocument child );
}
```

Extrait de code de surchage de tableau dynamique (sous-ressources)

```
public class CustomResourceTableField extends ResourceTableField
{
  public boolean onCreateChild( CtlInputTable table )
  {
    return super.onCreateChild( table );
  }
  public boolean onDeleteChild( CtlInputTable table, AbstractDocument document )
  {
    return super.onDeleteChild( table, document );
  }
  public boolean onDeleteChildren( CtlInputTable table, Collection document )
  {
    return super.onDeleteChildren( table, document );
  }
  public boolean onOpenChild( CtlInputTable table, AbstractDocument document )
  {
    return super.onOpenChild( table, document );
  }
}
```

Une fois le code réalisé, il suffit de déployer la solution et de déclarer votre classe comme étant celle implémentant le comportement du tableau dans le formulaire correspondant.

Extrait du code standard

```
<field property="DetailDeLaNote" ctrl="resourcetable" resourcetemplate="DetailDeLaNote"
    mode="write" required="no">
  <subfield ...
</field>
```

Extrait du code personnalisé

```
<field property="DetailDeLaNote"
    ctrl="com.axemble.education.document.fields.demo.CustomResourceTableField"
    resourcetemplate="DetailDeLaNote" mode="write" required="no">
  <subfield ...
</field>
```

Les abonnements inter-ressources

Les abonnements inter-ressources sont mis en œuvre dans le cadre des tableaux. Ils vous permettent de positionner des valeurs de champs d'une ressource liée ou d'un document lié à partir des valeurs de champs issus du document principal (père).

Une extension Java est présentée dans le chapitre suivant. Elle permet de réagir au moment de la création d'un élément fils.

Chapitre 5 : Les extensions sur le document

VDoc permet aux intégrateurs de développer des classes d'extension Java qui pourront être appelée sur les événements déclenchés par des actions utilisateur sur le document (ex. **Enregistrer**, **Fermer**, **Annuler**, etc.).

Ce type d'extension doit être utilisé dans le cas où il est nécessaire de manipuler le document dynamiquement.

Ces classes d'extension s'exécutent dans le contexte d'utilisation des formulaires. De ce fait, elles peuvent être déclarées sur tous les types de formulaires suivants :

- de processus ;
- d'étape ;
- d'action ;
- ou de ligne de tableau.

Il est aussi possible de placer plusieurs extensions sur un même formulaire.

VDoc utilise, en interne, une classe d'extension pour gérer à la fois les abonnements inter-champs et inter-ressources. Cette classe implémente les interfaces IDocumentExtension3 et ILinkExtension.

Cycle de vie d'une classe d'extension de document

Avant que le document ne soit totalement chargé, plusieurs méthodes sont appelées :

isOnChangeSubscriptionOn() : cette méthode est appelée sur chacun des champs présents dans le formulaire en mode écriture. Elle permet de définir, pour chacun d'eux, si un aller-retour serveur doit se faire sur la modification de la valeur du champ.

Remarque : côté serveur, la méthode onPropertyChanged() sera appelée dans tous les cas, si le champ est modifié. Il n'y a pas de relation directe avec la valeur renvoyée par la méthode isOnChangeSubscriptionOn(). Par défaut, cette méthode doit renvoyer faux.

onBeforeLoad() : correspond à l'initialisation de certains champs (comme les listes) sans déclenchement d'évènement. Si vous souhaitez affecter certaines valeurs du document sans qu'elles ne déclenchent d'évènements, c'est l'emplacement idéal. Notez tout de même que tous les abonnements « onLoad » sont déclenchés avant que cette méthode ne soit appelée.

onAfterLoad() : à cette étape, tous les abonnements sont lus et prêts à être déclenchés en cas de modification de champ (« onChange »).

A chaque modification d'un champ, la méthode **onPropertyChanged()** sera appelée dans les cas suivants :

- si un abonnement lui a été associé ;
- si la valeur vraie a été renvoyée par la méthode isOnChangeSubscriptionOn ;
- si l'attribut « throw-events » a été placé à vrai.

Le paramètre passé à la méthode onPropertyChanged correspond à la propriété du champ : IProperty.

Sur la sauvegarde, deux évènements sont traités :

onBeforeSave() : Cette méthode est présente surtout pour des cas de vérification de valeur. Si des valeurs ne sont pas désirées il est possible d'annuler le processus de sauvegarde en retournant la valeur « false ».

onAfterSave() : Lors du changement d'étape du document, il peut être nécessaire d'effectuer quelques opérations de vérification avant de soumettre le document. Pour cela, la méthode **onBeforeSubmit()** permet de stopper le processus de changement d'étape en retournant « false ». La méthode **onAfterSubmit()** n'est fournie que pour indiquer que le changement d'étape s'est bien effectué.

onBeforeAbort() fonctionne de la même façon que la méthode onBeforeSubmit(). Le lancement de l'assistant d'annulation ne sera effectif que si la valeur « true » est renvoyée.

Création d'une classe d'extension sur le document processus

VDoc fournit une classe de base qui simplifie l'implémentation des classes d'extension. Cette classe de base possède des méthodes pour accéder directement aux éléments souvent utilisés, tels que le module de workflow, l'instance de workflow, ainsi que le contrôleur de ressources.

Pour implémenter une classe d'extension type IDocumentExtension4, il suffit de dériver de la classe BaseDocumentExtension. La classe d'extension devra être déclarée dans l'administration des formulaires.

Le nom complet est : **com.axemble.vdoc.sdk.document.extensions.BaseDocumentExtension**.

Méthodes de la classe BaseDocumentExtension

```
public abstract class BaseDocumentExtension implements IDocumentExtension4
{
     // helper methods
     public IResourceController getResourceController();
     public IWorkflowInstance getWorkflowInstance();
     public IWorkflowModule getWorkflowModule();

     // load
     public boolean onBeforeLoad();
     public boolean onAfterLoad();

     // subscription
     public boolean isOnChangeSubscriptionOn( IProperty property );
     public void onPropertyChanged( IProperty property );

     // save
     public boolean onBeforeSave();
     public boolean onAfterSave();

     // change stage
     public boolean onBeforeSubmit( IAction action );
     public boolean onAfterSubmit( IAction action );

     // abort
     public boolean onBeforeAbort();

     // remind
     public boolean onBeforeRemind();

     // close
     public boolean onBeforeClose();

     // delegation
     public boolean onBeforeDelegate();
     public boolean onAfterDelegate();
     public boolean onBeforeDelegateTaskOnly();
     public boolean onAfterDelegateTaskOnly();
     public boolean onBeforeRefuseDelegation();
     public boolean onAfterRefuseDelegation();
     public boolean onBeforeCancelDelegation();
     public boolean onAfterCancelDelegation();

     // send information
     public boolean onBeforeSendInformation();
     public boolean onAfterSendInformation();
}
```

Exemple d'implémentation d'une extension de document

Comme le montre l'exemple suivant, la classe est très réduite. Cet exemple montre comment récupérer l'utilisateur connecté, son supérieur, positionner ce dernier dans un champ sélecteur de personne et lui ajouter le droit de lecture sur le document courant.

```
public class GrantAccessToManager extends BaseDocumentExtension
{
     private static final long serialVersionUID = 3457373535964512940L;

     public boolean onAfterLoad()
     {
       try
```

```
        {
              // récupérer l'utilisateur connecté
              IUser user = this.getWorkflowModule().getLoggedOnUser();

              // récupérer le responsable de l'utilisateur connecté
              IUser manager = user.getManager();
              if ( manager == null )
              {
                     Navigator.getNavigator().
                            showAlertBox( "L'utilisateur connecté n'a pas de responsable." );
                     return false;
              }

              // positionnement de la valeur d'un sélecteur d'utilisateur simple
              this.getWorkflowModule().
                     setExternalUser(this.getWorkflowInstance(),
                     "VerificationChefDeService", manager );

              // utiliser le gestionnaire de sécurité de VDoc Process pour manipuler les droits et
              // ajouter le droit de lecture sur le document
              ISecurityController securityController =
                     getWorkflowModule().getSecurityController( getWorkflowInstance() );

              // positionner un droit de lecture
              securityController.addPermission( manager,
                     Rights.Treatment.TreatmentLevel.READ CONTENT );
        }
        catch( WorkflowModuleException e )
        {
              Navigator.getNavigator().processErrors( e, true );
        }
        return super.onAfterLoad();
     }
}
```

Classe d'extension pour les tableaux dynamiques

Les classes d'extension type IDocumentExtension3, peuvent aussi être utilisées dans le contexte d'exécution des tableaux dynamiques. Il suffit de dériver de la classe BaseResourceExtension. La classe d'extension devra être déclarée dans l'administration du formulaire correspondant à l'édition d'une ligne de tableau.

Le nom complet est : **com.axemble.vdoc.sdk.document.extensions.BaseResourceExtension**.

Méthodes de la classe BaseResourceExtension

```
public abstract class BaseResourceExtension implements IDocumentExtension3
{
     // helper methods
     public IResourceController getResourceController();
     public ILinkedResource getLinkedResource();
     public IWorkflowModule getWorkflowModule();

     // load
     public boolean onBeforeLoad()
     public boolean onAfterLoad()

     // subscription
     public boolean isOnChangeSubscriptionOn( IProperty property )
     public void onPropertyChanged( IProperty property )

     // close
     public boolean onBeforeClose()
}
```

Exemple d'implémentation d'une extension de document

Comme le montre l'exemple suivant, la classe reste très simple car il n'est plus nécessaire d'implémenter toutes les méthodes de l'interface IDocumentExtension3. De plus, certaines méthodes ont été traduites dans les objets d'API SDK.

L'exemple suivant montre comment, sur la modification d'un champ de la resource (ligne de tableau), modifier la valeur d'un champ du père.

```
public class ChildExtension extends BaseResourceExtension
{
    private static final long serialVersionUID = 93867211817252323L;

    public void onPropertyChanged( IProperty property )
    {
            // récupérer la nouvelle valeur
            Object newValue = this.getLinkedResource().getValue( property.getName() );

            // positionner un champ du parent avec la nouvelle valeur récupérée
            this.getLinkedResource().getParentInstance().setValue( "parentProperty", newValue );

            super.onPropertyChanged( property );
    }
}
```

Cycle de vie de l'extension de document sur la création d'un élément lié

VDoc vous permet de déclarer des extensions qui seront appelées lors de la création d'une ressource interne (ou d'un document processus lié).

L'intérêt d'une telle classe d'extension est de permettre des traitements sur les champs de l'élément lié en ayant accès au document père.

Pour créer une classe d'extension de lien, créez une classe Java qui étend la classe de base nommée : **com.axemble.vdoc.sdk.link.extensions.BaseLinkExtension**.

Pour que la classe déployée soit appelée, il suffit de la déclarer sur le formulaire contenant le tableau.

Méthodes de la classe BaseLinkExtension

```
public abstract class BaseLinkExtension implements ILinkExtension
{
     // helper method
     protected IWorkflowModule getWorkflowModule():

     // method to implement
     public abstract boolean onCreate( IWorkflowInstance workflowInstance, IResource resource );
}
```

Extrait de code de la classe CopyLinkExtension

Cet exemple illustre la copie de la valeur du document principal vers un champ du tableau dynamique.

```
public class CopyLinkExtension extends BaseLinkExtension
{
     private static final long serialVersionUID = 4291330280452507689L;

     public boolean onCreate( IWorkflowInstance workflowInstance, IResource resource )
     {

       // assignation de la valeur du champ 'Commentaires2' avec la valeur du champ 'Commentaires'
       //du document père
       resource.setValue( "Commentaires2", workflowInstance.getValue( "Commentaires" ) );

       return true;
     }
}
```

Chapitre 6 : Les abonnements inter champs

Les abonnements inter champs permettre de rendre les documents plus dynamiques et plus adaptés aux données de l'entreprise.

VDoc propose un certain nombre d'abonnements standard. Dans les sections suivantes nous allons décrire tous les éléments concernant les différents types d'abonnements que vous pouvez mettre en place.

Les six types d'abonnement standard

- Sur les données du document ;
- Sur les données de l'annuaire ;
- Sur des données externes (SQL) ;
- Sur les acteurs du document ;
- Sur tout à l'aide de code Java Script ;
- Sur les vues.

Abonnement sur les données du document

Ce type d'abonnement doit être utilisé si vous avez besoin de mettre à jour un ou plusieurs champs du document en fonction d'un champ de type VDoc.

Exemple de définition d'un abonnement se basant sur les données du document.

```
<subscription field-id="Field_1" name="onChange" prop-id="" subscription-
class="com.axemble.vdp.mapping.extensions.ResourceBrowserMappingExtension">
<action field-id="Field_2" prop-id="" throw-events="false" update-mode="both" action-key=""/>
</subscription>
```

Cet exemple indique que sur la modification de « Field_1 » le champ « Field_2 » prendra pour valeur celle de « Field_1 » car l'attribut « action-key » est vide.

<subscription>

L'élément « subscription » définit l'entrée de la souscription dans le XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	champ du document ayant été modifié et déclencheur de l'événement « onChange »	
name	nom de l'évènement levé	
prop-id	Non disponible	
subscription-class	Classe d'extension : "com.axemble.vdp.mapping.extensions.ResourceBrowserMappingExtension"	

<on-stage>

L'élément «on-stage» définit l'entrée d'une étape sur laquelle l'abonnement sera exécuté.

Elément	Description	Où trouver plus d'information ?
name	Nom système de l'étape concernée. Le mot-clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

<action>

L'élément « action » définit l'entrée d'une action à réaliser suite à un déclenchement d'événement.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ abonné	
prop-id	Non actif	
throw-events	Non actif	
update-mode	Mode de mise à jour du document: "frontend", "backend" ,"both"	
action-key	Propriété de l'objet déclencheur obtenue par introspection	

Les actions

Les actions définissent l'ensemble des champs du document qui seront mis à jour au moment de l'exécution de l'abonnement. Les actions peuvent être définies uniquement au niveau du document XML du modèle de ressource. Il est possible de définir 1 à n actions.

Les filtres

Des filtres sur les étapes peuvent être mises en place de façon à restreindre l'exécution de l'abonnement à certaines étapes. Si aucune étape n'est précisée, l'abonnement sera valide à toutes les étapes du document. Les filtres sont définis au niveau du document XML du modèle de ressource. La balise <on-stage> peut être présente 0 à n fois.

L'introspection

L'introspection va permettre d'accéder aux propriétés intrinsèques de l'objet. Ces propriétés doivent être accessibles par les méthodes de type « get ». L'introspection est possible au niveau de l'attribut « action-key » de la balise « action ».

Les fonctions

Les fonctions vont permettre d'effectuer des calculs sur les champs de type tableau, mais aussi d'accéder aux valeurs des attributs étendus dans le cadre de champs représentant les valeurs d'objets de l'annuaire.

Fonction	Paramètres	Classes concernées	Description
@total(column_name) @TOTAL est encore supporté	Nom de la colonne	Collection	Calcul le total d'une colonne nombre d'un tableau. S'applique sur un champ du document de type Collection représentant un tableau dynamique.
@average(column_name) @AVERAGE est encore supporté	Nom de la colonne	Tableau dynamique	Calcul la moyenne d'une colonne nombre d'un tableau. S'applique sur un champ du document de type Collection représentant un tableau dynamique.
@get(key)	Clé de l'élément du map	Map	S'applique sur un champ du document de type Map représentant les valeurs stockées d'un JSP Browser. Il permet de récupérer la valeur d'une clé.
@toDirectoryElement(field_name)	Nom du champ	User, Group, Organization	Construit un objet de type DirectoryElement à partir d'une chaîne stockée dans un autre champ. S'applique sur un champ du document de type DirectoryElement.
@getOrganizationAttribute(field_name)	Nom de l'attribut	String	Récupère un attribut étendu d'une organisation. S'applique sur un champ du document de type chaîne de caractères représentant le nom d'une organisation.
@getLocalizationAttribute(field_name)	Nom de l'attribut	String	Récupère un attribut étendu d'une localisation. S'applique sur un champ du document de type chaîne de caractères représentant le nom d'une localisation

@getUserAttribute(field_name)	Nom de l'attribut	String	Récupère un attribut étendu d'un utilisateur. S'applique sur un champ du document de type chaîne de caractères représentant le login d'un utilisateur
@trunc(value,endIndex)	Chaine	String	Tronque le contenu de l'élément au nombre de caractères spécifiés par 'endIndex'. S'applique sur un champ du document de type chaîne de caractères ou sur une autre fonction renvoyant une chaîne de caractères.
	Index de fin	Integer	
@copy(field_name)	Nom du champ	String	Récupère la propriété et copie son contenu dans le champ abonné. Les types doivent être identiques.
@toString(field_name)	Nom du champ	String	Récupère la propriété et converti son contenu en chaîne de caractères.
@concat(value1,value2,value3,….)	Liste de chaines	String	Effectue la concaténation des éléments donnés en paramètres. valueX peut être : -le nom d'un champ de type chaîne de caractères. -une chaîne de caractères entourée de « ' » par « ' » -un autre fonction renvoyant une chaîne (@toString(..) par exemple)

Exemple

```
<subscription field-id=”Field_0” name=”onChange” prop-id=””
subscription-class=”com.axemble.vdp.mapping.extensions.ResourceBrowserMappingExtension”>
<action field-id=”Field_1” prop-id=”” throw-events=”false” update-mode=”both” action-
key=”@copy(Field_2)”/>
<action field-id=”Field 3” prop-id=”” throw-events=”false” update-mode=”both” action-
key=”@toString(sys_CreationDate)”/>
<action field-id=”Field_4” prop-id=”” throw-events=”false” update-mode=”both” action-
key=”@concat(Field 5,’-‘,@toString(sys Reference))”/>
</subscription>
```

Sur la modification du champ « Field_0 » :

- Le champ « Field_1 » prendra le contenu du champ « Field_2 ».
- Le champ « Field_3 », qui doit être de type texte, prendra le contenu du champ « sys_CreationDate » (date de création du document) transformé en chaîne.
- Le champ « Field_4 », qui doit être de type texte, prendra le concaténation du contenu du champ « Field_5 », de type texte également, de la chaîne de caractère « - » (tiret) et du contenu transformé en chaîne du champ « sys_Reference» (référence du document).

Abonnement sur les données de l'annuaire

Voici un exemple de définition d'une souscription se basant sur les données de l'annuaire.

```
<subscription field-id="UserField" name="onChange" prop-id=""
subscription-class="com.axemble.vdp.mapping.extensions.DirectoryBrowserMappingExtension">
<action field-id="Field_1" prop-id="" throw-events="false" update-mode="both" action-
key="Organization.Name"/>
<action field-id="Field 2" prop-id="" throw-events="false" update-mode="both" action-
key="PhoneNumber"/>
<action field-id="Field_3" prop-id="" throw-events="false" update-mode="both" action-key="Email"/>
</subscription>
```

Sur la modification du champ « UserField » les champs « Field_1, Field_2, Field_3 » sont affectés et prennent pour valeurs respectives : le nom de l'organisation de rattachement, le numéro de téléphone, et l'email de l'utilisateur.

<subscription>

L'élément « subscription » définit l'entrée de la souscription. Utilisé dans le XML du modèle de ressource.

Elément	Description	Où trouver plus

		d'information ?
field-id	champ du document ayant été modifié et déclencheur de l'évènement « onChange ». Ce champ doit être de type Directory.	
name	nom de l'évènement levé	
prop-id	Non disponible	
subscription-class	Classe d'extension: "com.axemble.vdp.mapping.extensions.directoryBrowserMappingExtension"	

<action>

L'élément « action » définit l'entrée d'une action à réaliser suite à un déclenchement d'évènement. Utilisé dans XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ abonné	
prop-id	Non disponible	
throw-events		
update-mode		
action-key	Propriété du DirectoryElement que l'on souhaite récupérer	

<on-stage>

L'élément «on-stage» définit l'entrée d'une étape sur laquelle l'abonnement sera exécuté.

Elément	Description	Où trouver plus d'information ?
name	Nom système de l'étape concernée. Le mot-clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

Les actions

Les actions définissent l'ensemble des champs du document qui seront mis à jour au moment de l'exécution de l'abonnement. Les actions peuvent être définies uniquement au niveau du document XML du modèle de ressource. Il est possible de définir 1 à n actions.

Les filtres

Des filtres sur les étapes peuvent être mis en place de façon à restreindre l'exécution de l'abonnement à certaines étapes. Si aucune étape n'est précisée, l'abonnement sera valide à toutes les étapes du document. Les filtres sont définis au niveau du document XML du modèle de ressource. Les balises <on-stage> peuvent être présente 0 à n fois.

L'introspection

L'introspection vous permet d'accéder aux propriétés intrinsèques de l'objet. Ces propriétés doivent être accessibles par les méthodes de type « get » sur les objets de l'annuaire. L'introspection est possible au niveau de l'attribut « action-key » de la balise « action ».

Les fonctions

Les fonctions vont permettre d'accéder aux valeurs des attributs étendus dans le cadre de champs de l'annuaire.

Fonction	Classes concernées	Description
@getOrganizationAttribute(field_name)	Organisation de l'annuaire	S'applique sur un champ du document de type Organization

@getLocalizationAttribute(field_name)	Localisation de l'annuaire	S'applique sur un champ du document de type Localization
@getUserAttribute(field_name)	Utilisateur de l'annuaire	S'applique sur un champ du document de type User
@toDirectoryElement(field_name)	User, Group, Organization	S'applique sur un champ du document de type DirectoryElement.
@toDirectoryElement(method_name)	Méthode renvoyant un objet de type User, Group, Organization, Localization de l'annuaire	Permet, via l'introspection d'un object User (par exemple), de récupérer sa localisation, puis de la convertir en DirectoryElement afin de la stocker.

Abonnement sur les données externes

Voici un exemple de définition d'une souscription se basant sur les données externes.

```
<subscription name="onLoad" subscription-
class="com.axemble.vdp.mapping.extensions.SQLResourceMappingExtension">
<sql reference=" SQLReference 1" order="SELECT PRODUCTNAME FROM VDOC PRODUCTS">
</sql>
<action field-id="Field_1" prop-id="" throw-events="false" update-mode="both" sql-type="VARCHAR"
field-list="PRODUCTNAME"/>
</subscription>
```

Au chargement du document, la souscription de type « onLoad » sera traitée. Elle remplira la liste du champ « Field_1 » avec le nom du produit de chaque ligne ramenée en exécutant la requête SQL.

```
<subscription field-id="Field_1" name="onChange" subscription-
class="com.axemble.vdp.mapping.extensions.SQLResourceMappingExtension">
<on-stage name="Stage_1"/>
<on-stage name="Stage 2"/>
<sql reference="SQLReference 1" order="SELECT * FROM Table 1 WHERE Column 1 =?">
<bind field-id="Field_1" prop-id="" sql-type="VARCHAR"/>
</sql>
<action field-id="Field_2" prop-id="" throw-events="false" update-mode="both" sql-type="FLOAT"
field-list="column 2"/>
<action field-id="Field 3" prop-id="" throw-events="false" update-mode="both" sql-type="FLOAT"
field-value="column_3"/>
</subscription>
```

Ce deuxième exemple de souscription SQL va s'exécuter sur l'altération de la valeur du champ « Field_1 ». Les champs « Field_2 » et « Field_3 » prendront pour valeur respective la valeur de la colonne « column_2 » et celle de la colonne « column_3 ».

Les balises <on-stage> indiquent que cette souscription ne sera prise en compte que sur les étapes « Stage_1 » et « Stage_2 ». Si vous souhaitez qu'elle soit traitée chaque fois que la valeur du champ « Field_1 » est altérée, supprimez ces lignes.

La balise <bind> est le moyen de renseigner des contraintes sur l'ordre SQL dépendantes de valeurs d'autres champs.

<subscription>

L'élément « subscription » définit l'entrée de la souscription. Utilisé dans le XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	champ du document ayant été modifié et déclencheur de l'évènement « onChange ».	
name	nom de l'évènement levé	
prop-id	Non disponible	
subscription-class	Classe d'extension: "com.axemble.vdp.mapping.extensions.SQLResourceMappingExtension "	

<on-stage>

L'élément «on-stage» définit l'entrée d'une étape sur laquelle l'abonnement sera exécuté.

Elément	Description	Où trouver plus d'information ?
name	Nom système de l'étape concernée. Le mot clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

<sql >

L'élément «sql» définit l'entrée de la requête SQL vers la base de données externe qui sera exécutée.

Elément	Description	Où trouver plus d'information ?
reference	Nom de la référence externe définie sur l'organisation de rattachement et permettant la connexion à une base de données externe.	
ordre	Ordre SQL à exécuter.	

<bind >

L'élément «bind» associe un paramètre de la requête SQL à un champ du document. Vous devez avoir autant de « ? » dans la requête que de « bind » déclarés. L'ordre des « bind » est important.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ du document donc la valeur sera prise comme paramètre à la requête.	
prop_id	Nom de la propriété ou de la méthode de type « get » qui permettra l'introspection de l'objet bind.	
sql-type	Type SQL de stockage du champ paramètre.	

<action>

L'élément « action » définit l'entrée d'une action à réaliser suite à un déclenchement d'évènement. Utilisé dans XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ abonné.	
prop-id	Non disponible	
throw-events		
update-mode		
field-list / field-value	Champ de la requête qui servira à la mise à jour du champ du document selon s'il est de type liste (field-list) ou non (field-value).	
sql-type	Type SQL de stockage du champ à mettre à jour. (VARCHAR ou FLOAT)	

Les actions

Les actions définissent l'ensemble des champs du document qui seront mis à jour au moment de l'exécution de l'abonnement. Les actions peuvent être définies uniquement au niveau du document XML du modèle de ressource. Il est possible de définir 1 à n actions.

Les filtres

Des filtres sur les étapes peuvent être misen place de façon à restreindre l'exécution de l'abonnement à certaines étapes. Si aucune étape n'est précisée, l'abonnement sera valide à toutes les étapes du document. Les filtres sont définis au niveau du document XML du modèle de ressource. Les balises <on-stage> peuvent être présentes 0 à n fois.

Les contraintes

Les contraintes de la requête SQL sont définies en tant que binds au niveau du document XML. Chaque balise <bind> va donc représenter un « ? » de la requête. Attention, l'ordre doit être respecté.

Dans le cadre d'abonnement de type « OnLoad », les contraintes peuvent aussi être définies au niveau de l'Administration des listes. De la même façon, chaque champ contrainte ajouté doit l'être dans l'ordre des « ? » de la requête.

Abonnement sur les acteurs du document

Ce type d'abonnement doit être utilisé si vous avez besoin de mettre à jour un ou plusieurs champs rôle du document en fonction d'un champ de type VDoc.

Voici un exemple de définition d'une souscription se basant sur les acteurs du document.

```
<subscription name="onChange" field-id="Utilisateur"
subscription-class="com.axemble.vdp.mapping.extensions.ActorsMappingExtension">
<on-stage name="Tache"/>
<action field-id="ManagerRole" action-key="@toHierarchicalManager(Utilisateur)"/>
<action field-id="HierarchicalManagerRole" action-key="@toManager(Utilisateur)"/>
<action field-id="AssistantRole" action-key="@toAssistant(Utilisateur)"/>
<action field-id="Role" action-key="@toRole(Utilisateur)"/>
</subscription>
```

L'exemple précédent vous indique comment réaliser des abonnements qui vous permettent de récupérer des informations de l'annuaire, notamment le supérieur hiérarchique, le supérieur ou encore l'assistant d'une personne. Il vous montre également comment assigner un rôle avec un champ utilisateur.

Ce deuxième exemple vous montre comment assigner un rôle à partir d'une liste d'utilisateurs :

```
<subscription name="onChange" field-id="ListeDePersonnes"
subscription-class="com.axemble.vdp.mapping.extensions.ActorsMappingExtension">
<on-stage name="Tache"/>
<action field-id="Role" action-key="@toRole(ListeDePersonnes)"/>
</subscription>
```

<subscription>

L'élément « subscription » définit l'entrée de l'abonnement dans le XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	champ du document ayant été modifié et déclencheur de l'évènement « onChange »	
name	nom de l'évènement levé	
prop-id	Non disponible	
subscription-class	Classe d'extension : "com.axemble.vdp.mapping.extensions.ResourceBrowserMappingExtension"	

<on-stage>

L'élément «on-stage» définit l'entrée d'une étape sur laquelle l'abonnement sera exécuté.

Elément	Description	Où trouver plus d'information ?
name	Nom système de l'étape concernée. Le mot-clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

<action>

L'élément « action » définit l'entrée d'une action à réaliser suite à un déclenchement d'évènement.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ abonné	
prop-id	Non actif	
action-key	Propriété de l'objet déclencheur obtenue par introspection	

Les actions

Les actions définissent l'ensemble des champs du document qui seront mis à jour au moment de l'exécution de l'abonnement. Les actions peuvent être définies uniquement au niveau du document XML du modèle de ressource. Il est possible de définir 1 à n actions.

Les filtres

Des filtres sur les étapes peuvent être mis en place de façon à restreindre l'exécution de abonnement à certaines étapes. Si aucune étape n'est précisée, l'abonnement sera valide à toutes les étapes du document. Les filtres sont définis au niveau du document XML du modèle de ressource. Les balises <on-stage> peuvent être présentes 0 à n fois.

L'introspection

L'introspection va permettre d'accéder aux propriétés intrinsèques de l'objet. Ces propriétés doivent être accessibles au travers des méthodes de type « get ». L'introspection est possible au niveau de l'attribut « action-key » de la balise « action ».

Les fonctions

Fonction	Paramètres	Classes concernées	Description
@toHierarchicalManager(field_name)	Nom du champ	User ou DirectoryElement	Affectation d'un rôle avec le supérieur hiérarchique d'un utilisateur
@toManager(field_name)	Nom du champ	User ou DirectoryElement	Affectation d'un rôle avec le supérieur d'un utilisateur
@toAssistant(field_name)	Nom du champ	User ou DirectoryElement	Affectation d'un rôle avec l'assistant d'un utilisateur
@toRole(field_name)	Nom du champ	Collection d'utilisateurs DirectoryElement	Affectation d'un rôle avec un utilisateur ou une liste d'utilisateurs

Abonnement Java Script

Ce type d'abonnement est le plus générique qu'il soit. Il vous permet de réellement contrôler le document. C'est le type d'abonnement idéal pour réaliser des calculs entre les champs et affecter plusieurs champs du document.

Voici un exemple permettant de réaliser un total sur une colonne d'un tableau dynamique.

```
<script event="onChange" for="DetailDeLaNote" on-stage="DemandeDeRemboursement"><![CDATA[
  var total = 0;
  var resources = iWorkflowModule.getLinkedResources(iWorkflowInstance,"DetailDeLaNote");
  if( resources!=null )
  {
    for( var it=resources.iterator(); it.hasNext(); )
```

```
        {
          var value = it.next().getValue("Cout");
          if( value!=null )
            total += value.floatValue();
        }
      }
      iWorkflowInstance.setValue("MontantTotal",new java.lang.Float(total));
    ]]></script>
```

<script>

L'élément « script » définit l'entrée de la souscription dans XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
event	Nom de l'évènement levé. Les valeurs possibles sont : onLoad | onChange.	
for	Champs du document, séparés par des virgules, ayant été modifiés et déclencheur de l'évènement « onChange »	
on-stage	Noms des étapes sur lesquelles le traitement devra s'exécuter. Le mot-clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

.

Il est fortement recommandé d'utiliser les balises < ![CDATA[…]]> pour enrober le code Java Script et ainsi d'éviter de causer des erreurs de syntaxe XML.

Les API SDK sont disponibles lors de l'exécution des abonnements Java Script.

Les mots clés API disponibles en JavaScript

Mot clé	Description
iWorkflowModule	IWorkflowModule est le module de workflow de VDoc
iContext	IContext est le contexte de l'utilisateur connecté
iUser	IUser est l'utilisateur connecté
iWorkflowInstance	IWorkflowInstance représente le document en cours
iTaskInstance	ITaskInstance est la tâche active du document en cours
iAction	IAction correspond à l'action au moment du changement d'étape.
iResourceController	IResourceController est un contrôleur qui permet de modifier en dynamique l'aspect graphique d'une resource. Il permet de spécifier que les champs d'une ressource sont éditables, cachés, obligatoire, ou en erreur.

L'interface IDocumentExtension4 en Java Script

L'interface IDocumentExtension4 a été implémentée dans le Java Script de VDoc, si bien qu'il est possible d'utiliser toutes les fonctions de cette interface sous la forme de « callback ».

Pour utiliser ces « callback », il n'est pas nécessaire de les déclarer toutes dans l'administration du modèle de ressources. Il suffit de déclarer seulement la fonction que vous souhaitez utiliser.

Exemple d'implémentation des méthodes de l'interface IDocumentExtension4

Dans cet exemple, toutes les fonctions sont déclarées. Elles ont toutes une instruction de « log » qui permet de tracer l'ordre des appels.

```
<script>
log.info("script initialization");

function onBeforeLoad()
{
  log.info("onBeforeLoad()");
```

```
  return true;
}
function onAfterLoad()
{
  log.info("onAfterLoad()");
  return true;
}
function isChangeSubscriptionOn( field )
{
  log.info("isChangeSubscriptionOn("+field.getPropertyName()+")");
  return false;
}
function onFieldChanged( field )
{
  log.info("onFieldChanged("+field.getPropertyName()+")");
}
function onBeforeSave( checkMandatory )
{
  log.info("onBeforeSave("+checkMandatory+")");
  return true;
}
function onAfterSave()
{
  log.info("onAfterSave()");
  return true;
}
function onBeforeSubmit( actionKey )
{
  log.info("onBeforeSubmit()");
  return true;
}
function onAfterSubmit( actionKey )
{
  log.info("onAfterSubmit()");
  return true;
}
function onBeforeClose()
{
  log.info("onBeforeClose()");
  return true;
}
function onBeforeAbort()
{
  log.info("onBeforeAbort()");
  return true;
}
function onBeforeRemind()
{
  log.info("onBeforeRemind()");
  return true;
}
function onBeforeCancelDelegation()
{
  log.info("onBeforeCancelDelegation()");
  return true;
}
function onAfterCancelDelegation()
{
  log.info("onAfterCancelDelegation()");
  return true;
}
function onBeforeRefuseDelegation()
{
  log.info("onBeforeRefuseDelegation()");
  return true;
}
function onAfterRefuseDelegation()
{
  log.info("onAfterRefuseDelegation()");
  return true;
}
function onBeforeDelegate()
{
  log.info("onBeforeDelegate()");
  return true;
}
function onAfterDelegate()
{
```

```
  log.info("onAfterDelegate()");
  return true;
}
function onBeforeDelegateTaskOnly()
{
  log.info("onBeforeDelegateTaskOnly()");
  return true;
}
function onAfterDelegateTaskOnly()
{
  log.info("onAfterDelegateTaskOnly()");
  return true;
}
function onBeforeSendInformation()
{
  log.info("onBeforeSendInformation()");
  return true;
}
function onAfterSendInformation()
{
  log.info("onAfterSendInformation()");
  return true;
}
</script>
```

Exemple d'initialisation d'un champ utilisateur

Dans cet exemple, l'exécution du script est réalisée sur l'étape de création.

```
<script on-stage="sys_CreationWizard"><![CDATA[
  function onAfterLoad()
  {
    iWorkflowModule.setExternalUser(iWorkflowInstance, "Utilisateur", iUser);
  }
]]></script>
```

Abonnement sur les vues

Ce type d'abonnement permet d'alimenter des valeurs de champs de document processus avec des données provenant de documents VDoc tels que les instances de processus ou les données du Data Universe.

L'interrogation est entièrement basée sur les vues. La syntaxe est identique à la définition des vues mais peut être simplifiée dans le cas où une vue générée correspond exactement au besoin de la requête. Dans ce cas, il suffira de référencer la vue soit par son identifiant, soit par sa propriété « protocolURI ».

Exemple de souscription type « onLoad »

Voici un exemple de définition d'une souscription se basant sur les vues et exécutée lors du chargement du document.

Au chargement du document, la souscription de type « onLoad » sera exécutée. L'attribut « view » permet de spécifier sur quelle vue générée doit se baser l'interrogation. L'attribut « action-key » permet de définir quelle colonne de la vue sera utilisée pour alimenter le champ de la balise <action>. Par défault, si cet attribut est vide, la souscription remplira la liste du champ « Field_1 » avec la première colonne de la vue.

```
<subscription name="onLoad"
  subscription-class="com.axemble.vdoc.sdk.mapping.extensions.ViewMappingExtension"
  view="-bidzawpqyday3cew6fxys">
 <action field-list="Field 1" prop-id="" throw-events="false" action-key="" />
</subscription>
```

Exemple de souscription type « onChange »

Voici un exemple de définition d'une souscription exécutée lors de la modification de la valeur d'un champ.

```
<subscription field-id="Field_1" name="onChange" prop-id=" "
    subscription-class="com.axemble.vdoc.sdk.mapping.extensions.ViewMappingExtension">
  <definition catalog="CustomerSupport" catalogType="4" name="DEFAULT" type="resource">
    <filters>
      <fieldgroup operator="AND">
        <field name="sys_ResourceTemplate" value="Test"/>
        <field name="Column_1" value="$Document.Value('Field_1')"/>
      </fieldgroup>
    </filters>
    <view>
      <column name="Column_2"/>
    </view>
  </definition>
  <action field-list="Field 2"/>
</subscription>
```

Ce deuxième exemple de souscription type vue va s'exécuter sur l'altération de la valeur du champ « Field_1 ». Le champ « Field_2 » prendra la valeur de la colonne « column_2 ». La balise <definition> permet de déclarer une vue en dynamique. La syntaxe est la même que celle utilisée pour les vues générées.

<subscription>

L'élément « subscription » définit l'entrée de la souscription. Utilisé dans le XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	champ du document ayant été modifié et déclencheur de l'événement « onChange ».	
name	nom de l'événement levé	
prop-id	Non disponible	
subscription-class	Classe d'extension: " com.axemble.vdoc.sdk.mapping.extensions.ViewMappingExtension "	

<on-stage>

L'élément «on-stage» définit l'entrée d'une étape sur laquelle l'abonnement sera exécuté.

Elément	Description	Où trouver plus d'information ?
name	Nom système de l'étape concernée. Le mot clé « **sys_CreationWizard** » permet d'indiquer que le traitement sera exécuté uniquement sur l'action de création du document processus.	

<definition>

L'élément « definition » définit la vue dynamique qui sera exécutée. La syntaxe est identique aux vues générées.

<action>

L'élément « action » définit l'entrée d'une action à réaliser suite à un déclenchement d'événement. Utilisé dans le XML du modèle de ressource.

Elément	Description	Où trouver plus d'information ?
field-id	Nom système du champ abonné.	
prop-id	Non disponible	
throw-events		
update-mode		
field-list / field-value	Champ de la requête qui servira à la mise à jour du champ du document selon s'il est de type liste (field-list) ou non (field-value).	

Les actions

Les actions définissent l'ensemble des champs du document qui seront mis à jour au moment de l'exécution de l'abonnement. Les actions peuvent être définies uniquement au niveau du document XML du modèle de ressource. Il est possible de définir 1 à n actions.

Les filtres

Des filtres sur les étapes peuvent être misen place de façon à restreindre l'exécution de l'abonnement à certaines étapes. Si aucune étape n'est précisée, l'abonnement sera valide à toutes les étapes du document. Les filtres sont définis au niveau du document XML du modèle de ressource. Les balises <on-stage> peuvent être présentes 0 à n fois.

Cycle de vie des extensions sur les abonnements

VDoc permet d'étendre cette liste de types d'abonnement en vous donnant la possibilité de déclarer vos propres classes d'abonnement.

Une classe qui souhaite s'intégrer au mécanisme d'abonnement doit impérativement étendre la classe de base nommée **com.axemble.vdoc.sdk.mapping.extensions.BaseMappingExtension**.

Méthode	Description	Paramètres
onLoad	Cette méthode sera appelée au chargement du document	• Element : nœud XML correspondant à la description de l'abonnement.
onChange	Cette méthode sera appelée sur chaque modification effectuée sur la propriété	• Property : la propriété en cours de modification.

	spécifiée dans la balise field-id (ex. ChampEnCoursDeModification)	• Element : nœud XML correspondant à la description de l'abonnement.

La déclaration de la classe d'abonnement s'effectue directement dans le bloc XML des abonnements.

Syntaxe générique

La syntaxe générique des abonnements est la suivante :

```
<subscription name="onChange" field-id="ChampEnCoursDeModification"
    subscription-class="NomCompletDeVotreClasse">
<!--.set some parameters here...-->
</subscription>
```

Exemple

L'exemple suivant présente une classe qui derive de la classe BaseMappingExtension.

```
public class SimpleMappingExtension extends BaseMappingExtension
{
  public void onLoad( Element element )
  {
    // méthode déclenché sur le chargement du document
  }

  public void onChange( IProperty property, Element element )
  {
    // methode déclenché lors de la modification d'une propriété
  }
}
```

Chapitre 7 : Personnalisation de l'interface du document processus

Introduction

VDoc offre une grande souplesse dans la personnalisation de ses formulaires. En effet, vous pouvez spécifier la présentation de chaque formulaire grâce à une description XML disponible depuis l'administration.

Les formulaires que vous pouvez adapter sont :

- les formulaires d'étapes en modification ;
- les formulaires d'étapes en impression ;
- le formulaire principal lorsque la personne connectée n'a pas à intervenir ;
- l'en-tête des formulaires en lecture/modification ;
- l'en-tête des formulaires en impression ;
- le formulaire de création ;
- le formulaire de relance ;
- le formulaire de changement d'étape ;
- les modèles de mail pour
  - le retard sur une étape
  - l'intervention à une étape
  - le mail d'information qui peut être envoyé depuis une étape
  - le mail d'information complémentaire à une personne tierce
- et enfin les formulaires de mail qui peuvent être personnalisés, ou utiliser la description du modèle de mail.

Tous les formulaires utilisent le même formalisme de description de l'interface.

Définition de la structure XML

Les descriptions de formulaires possèdent un élément racine **page**.

Légende : Ci-dessous, une indication **tag**{**x**,**y**} signifie que l'élément possède le fils **tag** au minimum **x** fois et au maximum **y** fois.

abstract <cell>

Un élément **cell** est un élément abstrait. Il n'existe pas en tant que tel mais décrit une structure commune à d'autres éléments. Il permet de décrire le style d'arrière-plan et la formule d'affichage.

attributs

display (optionnel)

Formule d'affichage de la forme [type]:[param]. Si l'évaluation de la formule d'affichage ne permet pas l'affichage d'un bloc, celui-ci ne sera pas visible et non présent dans le code source de la page HTML.

Les types supportés sont :

stageused - stagenotused

Correspond à une formule d'affichage évaluant si le document est déjà passé par l'étape spécifiée dont le nom système est utilisé pour l'élément [param].

Exemple : display="stageused:demande"

fieldused - fieldnotused

Correspond à une formule d'affichage évaluant si un champ a déjà été proposé en modification à un utilisateur. [param] correspond au nom système du champ.

Exemple : display="fieldused:DateDebut"

variable

Correspond à une formule d'affichage évaluant la valeur booléenne d'une variable globale Java Script du bloc **<script>**. Si la valeur de la variable est 'true', l'élément sera affiché, sinon l'élément complet ne sera pas affiché.

Si la valeur n'est pas booléenne, la valeur 'false' sera utilisée (donc l'élément sera caché).

[param] correspond au nom de la variable Java Script.

Exemple : Dans le bloc script "var display_demande = …" => display="variable:display_demande"

variableisfalse

Fonctionne de la même manière que l'option "variable", sauf qu'elle permet d'afficher l'élément uniquement si la variable est "false".

Il est donc possible de réutiliser la même variable pour afficher des éléments différents, ou un champ en lecture seule ou en modification selon certains droits.

enfants

<aucun>

Exemple d'affichage d'éléments selon l'appartenance de l'utilisateur connecté à un rôle

```
<page>
 <script><![CDATA[
  var role = iWorkflowModule.getRole( iContext, iWorkflowInstance.getCatalog(), "Vérificateur" );
  var user_has_right = false;

  if ( iUser.isMemberOf(role,true) )
  {
    user_has_right = true;
  }
]]></script>
  <body>
    <section ...>
      <line display="variable:user_has_right">

        <field display="variable:user_has_right" property="HL_Contact" ctrl="text" mode="write"
          required="true"/>
        <field display="variableisfalse:user has right" property="HL Contact" mode="read"
          ctrl="text" required="true"/>

      </line>
    </section>
  </body>
</page>
```

abstract <container>

Un élément **container** est un élément abstrait. Il n'existe pas en tant que tel, mais décrit une structure commune à d'autres éléments. Il permet d'imbriquer les éléments entre eux, et peut contenir :

- Soit une liste d'éléments **line**.
- Soit un des éléments suivants : **section**, **text**, **html**, **field** ou **actionhistory**.

Dans certains cas, il pourra contenir une liste de plusieurs éléments **field** et/ou **text** et/ou **html** dans le but de concaténer en une seule phrase différents éléments. Mais tous les types d'élément **field** ne seront pas forcément supportés car certains entraînent un retour chariot.

attributs

innerClass (optionnel)

Personnalisation du style CSS d'arrière-plan.

cellspacing (optionnel)

Espacement entre cellules (identique à l'attribut HTML du même nom).
Exemple : cellspacing="5px"

cellpadding (optionnel)

Marges intérieures des cellules (identique à l'attribut HTML du même nom).
Exemple : cellpadding="10px"

enfants

line {1,n}
ou
section {0,n} et **actionhistory** {0,n} et **text** {0,n} et **field** {0,n} et **html** {0,n}

<page>

L'élément **page** est l'élément racine d'un formulaire. Il peut contenir les éléments **script**, **header**, **body** et **footer**. Mais comme pour cette version l'en-tête et le pied de page ne sont pas modifiables, les éléments **header** et **footer** ne sont pas supportés.

attributs

<aucun>

enfants

script {0,1}

body {0,1}

Exemple d'incrustation de script

```
<?xml version="1.0" encoding="UTF-8"?>
<page>
    <script>
            ...
    </script>
    <body>
            ...
    </body>
</page>
```

<script>

L'élément **script** contient le code Java Script à évaluer côté serveur lors de l'affichage du formulaire. Ce code est évalué uniquement lors du premier affichage du formulaire et les valeurs des variables pourront être utilisées par tous les éléments du formulaire. Par exemple, cela peut être une formule d'affichage qui définit si un élément est visible ou non.

Les variables disponibles dans le bloc <script>

Mots clés API SDK

Mot clé	Classe	Description
iWorkflowModule	com.axemble.vdoc.sdk.modules.IWorkflowModule	Module de workflow
iUser	com.axemble.vdoc.sdk.interfaces.IUser	Utilisateur connecté
iContext	com.axemble.vdoc.sdk.interfaces.IContext	Contexte de l'utilisateur connecté
iWorkflowInstance	com.axemble.vdoc.sdk.interfaces.IWorkflowInstance	Document actif
iResource	com.axemble.vdoc.sdk.interfaces.IResource	Ressource active. Valable principalement dans le contexte

		des tableaux dynamiques.
iTaskInstance	com.axemble.vdoc.sdk.interfaces.ITaskInstance	Tâche active
iAction	com.axemble.vdoc.sdk.interfaces.IAction	Action en cours. Valable lors du changement d'étape.

Mots-clés de navigation

Mot-clé	Classe	Description
navigator	com.axemble.vdp.ui.framework.controls.Navigator	Objet de navigation. Disponible uniquement dans les formulaires web. Depuis cet objet, il est possible de récupérer le gestionnaire des services (ServiceManager), le contexte d'exécution IExecutionContext.
skinUrl	java.lang.String	Url de la skin. Disponible uniquement dans les formulaires web.
showHelpButton	java.lang.Boolean	Affichage du bouton image d'aide. Disponible uniquement dans les formulaires web.
converter	com.axemble.vdp.services.tools.ConverterService	Permet de convertir des éléments de différentes natures en chaîne.
format	com.axemble.vdp.localization.interfaces.IFormatService	Permet de définir le format d'affichage des éléments
log	com.axemble.vdp.utils.Logger	Permet d'insérer des messages logs dans les scripts. Nécessaire pour les développements spécifiques.
language	java.lang.String	Langue courante utilisée.
baseUrl	java.lang.String	URL de base
applicationUrl	java.lang.String	URL de l'application

Quelques exemples d'utilisation des mots-clés

```
// récupérer la précédente valeur d'un champ
var oldValue = iWorkflowInstance.getValue("Total");

// positionner la nouvelle valeur d'un champ
iWorkflowInstance.setValue("Total", newValue);
```

Les fonctions disponibles

Méthode	Description
function alert(msg);	Affiche un message d'alerte.
function setValue(String propertyName, Object value);	Positionne la valeur d'un champ du document courant.
function getValue(String propertyName);	Retrouve la valeur d'un champ du document courant.
function setValue(CoreDocument document, String propertyName, Object value);	Positionne la valeur d'un champ du document spécifié. Cas des tableaux dynamiques.
function getValue(CoreDocument document, String propertyName);	Retrouver la valeur d'un champ du document spécifié.
function getStringValue(String propertyName);	Renvoie une valeur chaîne de caractères.
function getHtmlValue(String propertyName);	Renvoie une valeur HTML.
function show(String id, boolean visible);	Affiche ou masque un élément identifié. Cas également des fragments.
function setHtml(String id, String html);	Positionne le contenu d'un élément HTML.
function toHtml(String text);	Convertit une chaîne de caractères au format HTML.
function toAttribute(String text);	Convertit une chaîne de caractères au format d'un attribute HTML.
function toTextArea(String text);	Convertit une chaîne de caractères au format d'un texte area.

function toJSString(String text);	Convertit une chaîne de caractères au format Java Script.

Les fonctions avancées disponibles

Méthode	Description
function getConnection(String reference);	Récupère une connexion JDBC à partir d'une référence donnée. Si la référence est null, la connexion correspondra à la base VDoc.
function sqlQuery(String reference, String order, Object[] paramerters);	Exécute une requête SQL paramétrable dans une base de données référencée.
function sqlUpdate(String reference, String order, Object[] paramerters);	Exécute un traitement SQL (insert, update, delete paramétrable dans une base de données référencée.

La classe de conversion

La classe de conversion **« converter »** permet de convertir un objet en une chaîne de caractères affichable en fonction du contexte de l'utilisateur (langue, format de date et nombre).

Les méthodes disponibles

```
public String object(Object obj);
public String object(Object obj,int size);
public String onlyDate(Date date,int size);
public String onlyTime(Date date,int size);
public String resourceValue(Resource resource, String propertyName);
public String resourceValue(Resource resource, String propertyName, int size);
```

Les paramètres passés

- **obj** : l'objet à afficher de type String, Integer, Long, Float, Double, Date, Period, FileInfo, Collection, DirectoryElement ;
- **date**: la date à afficher ;
- **size**: la taille d'affichage des informations.

Valeurs de taille possibles

Mot clé	Valeur	Rendu
SHORT	1	21/07/2004 15:50
MEDIUM	2	21 juil 2004 15:50
LONG	3	21 juillet 2004 15:50:42
FULL	4	mercredi 21 juillet 2004 15:50:42 CET

Quelques exemples

```
// conversion d'une date en chaîne de caractères
var /*String*/ displayDate = converter.object(date,1/*SHORT*/);

// récupérer une chaîne de caractères traduite dans le fichier strings global.xml à partir d'un
// identifiant
var /*String*/ msg = converter.sysString("LG_DECISION");
```

La classe de format

La classe de format **« format »** permet d'obtenir le contexte de conversion de l'utilisateur, comme la langue active (ex. format.getLanguage()).

Méthodes disponibles

Méthode	Description

public String getLanguage();	Récupération de la langue
public Locale getLocale();	Récupération de la locale
public Calendar getCalendar();	Récupération d'un calendrier
public DateFormat getShortDateFormat();	Récupération d'un format de date
public DateFormat getMediumDateFormat();	
public DateFormat getLongDateFormat();	
public DateFormat getFullDateFormat();	
public DateFormat getShortTimeFormat();	
public DateFormat getMediumTimeFormat();	
public DateFormat getLongTimeFormat();	
public DateFormat getFullTimeFormat();	
public DateFormat getShortDateTimeFormat();	
public DateFormat getMediumDateTimeFormat();	
public DateFormat getLongDateTimeFormat();	
public DateFormat getFullDateTimeFormat();	
public DecimalFormat getIntegerNumberFormat();	Récupération d'un format numérique entier
public DecimalFormat getDecimalNumberFormat();	Récupération d'un format numérique à virgule
public DecimalFormat getCurrencyNumberFormat();	Récupération d'un format monétaire

attributs

<aucun>

enfants

text() {0,1}

Code source JavaScript du formulaire, souvent entouré par **<![CDATA[** et **]]>** pour éviter l'interprétation XML.

Exemple

```
<script><![CDATA[
    var isExtern = iWorkflowInstance.getValue("isExtern");
    var display_address = (isExtern == "yes");
]]></script>
```

<body>

En prévisualisation, l'élément **body** n'est pas visible, c'est juste une zone entre l'en-tête et le pied de page. L'élément **body** étend **container**.

attributs

<aucun>

enfants

Ceux du **<container>**.

Exemple

```
<body>
  <line>
    <section>
      ...
    </section>
  </line>
  <line>
    <section>
      ...
    </section>
```

```
  </line>
</body>
```

<line>

L'élément **line** n'a pas de représentation. Il n'existe que pour séparer les éléments par un retour chariot. Sa hauteur n'est pas modifiable, elle sera définie automatiquement en fonction de son contenu. L'élément **line** étend **container** et peut aussi contenir une liste d'éléments **col**. Il étend également **cell**.

attributs

Ceux de **<cell>**

enfants

Ceux de **<container>**

ou

col {0,n}

Exemple

```
<line height="50%">
  <section>
    ...
  </section>
</line>
<line height="30px">
  <col width="50">
    ...
  </col>
  <col>
    ...
  </col>
</line>
```

<col>

L'élément **col** n'a pas de représentation. Il est utile pour découper une ligne en plusieurs morceaux. Chaque élément **col** possède une largeur exprimée en pourcentage et la somme des largeurs des éléments **col** d'un élément **line** doit être égale à 100.

L'élément **col** étend **container** et **cell**.

attributs

Ceux de **<cell>**

et

class (optionnel)

Définit le nom du style CSS à utiliser.

Les différents styles standard pour les colonnes sont :

- "**cell-left**" : c'est une cellule gauche, utilisé pour la représentation du libellé d'un champ. Par défaut, la couleur de fond de la cellule est plus contrastée, l'alignement du texte se fait à droite et au centre verticalement.
- " **cell-right** " : c'est une cellule droite, utilisée pour la représentation du libellé d'un champ. Par défaut, la couleur de fond de la cellule est plus claire, l'alignement du texte se fait à gauche et au centre verticalement.

width (optionnel)

Largeur de la colonne comprise entre 1 et 100. Le défaut de valeur, comme la valeur *, représente le complément jusqu'à 100. Si plusieurs éléments **col** n'ont pas de valeur, elles se partagent équitablement le complément jusqu'à 100. ex : 30

enfants

Ceux de **<container>**

Exemple

```
<line>
  <col width="30" class="cell-right" >
    ...
  </col>
  <col width="70" class="cell-left" >
    ...
  </col>
</line>
<line>
  <col width="50">
    ...
  </col>
  <col width="*">
    ...
  </col>
</line>
<line>
  <col width="80">
    ...
  </col>
  <col>
    ...
  </col>
</line>
```

<section>

L'élément **section** permet de regrouper de l'information thématiquement. Visuellement il entoure son contenu par un cadre et possède une icône et un titre. Les styles de ces trois éléments pourront être modifiés.

L'élément **section** étend **container** et **cell**.

attributs

Ceux de **<cell>**

et

title

Titre de la section. Exemple : Demande

class (optionnel)

Personnalisation du style de la section.

Les différents styles standard pour les sections sont :

- "**section**" : c'est la représentation d'une section inactive
- "**section-highlighted**" : c'est la représentation d'une section active
- "**subsection**" : c'est la représentation d'une sous-section inactive
- "**subsection-highlighted**" : c'est la représentation d'une sous-section active

enfants

Ceux de **<container>**

Exemple

```
<section title="Demande" cellpadding="5px">
  ...
</section>
```

<field>

L'élément **field** affiche la valeur d'un champ du document. Il définit si la valeur peut être modifiée, indique le contrôle qui prendra en charge ce champ et permet d'en modifier le style.

L'élément **field** étend **cell** et ne contient rien.

attributs

Ceux de **<cell>**

et

property

Nom système du champ du document à connecter.

Exemple : DateDebut

ctrl

Nom du contrôle utilisé.

Exemple : text, textarea, date, …

Le nom doit correspondre à l'une des valeurs de l'énumération com.axemble.vdp.ui.interfaces.UIControlsEnum

mode (optionnel)

Mode d'exécution du contrôle. Les valeurs possibles sont '**read**' ou '**write**'. Si le mode n'est pas précisé, il est '**read**'.

Depuis la version 2005 Service Pack 1, l'attribut "mode" peut prendre la valeur d'une variable selon la syntaxe suivante :

```
<field … mode="variable:var_etat" … >
```

De la même manière que "variable:" pour l'affichage ou non d'un élément, il est possible de pré-calculer dans une variable l'état d'un champ.

La valeur de var_etat peut être :

"write" ou "true" ou "1" pour être modifiable

Dans tous les autres cas, le champ sera en lecture seule.

required (optionnel)

Indique si le champ est obligatoire ou pas. ("**yes**" ou "**no**")

L'attribut "required" peut prendre la valeur d'une variable selon la syntaxe suivante :

```
<field … required="variable:var_required" … >
```

De la même manière que "variable:" pour l'affichage ou non d'un élément, il est possible de pré-calculer dans une variable l'état d'un champ.

La valeur de var_required peut être :

"yes" ou "true" ou "1" pour être modifiable

Dans tous les autres cas, le champ sera en lecture seule.

enfants

subfield

- **Exemple**

```
<field property="sys_Title" ctrl="text"/>
<field property="DateDebut" ctrl="date" mode="write"/>
```

<subfield>

Balise enfant de **field** (uniquement disponible pour les champs de type tableaux).

Détermine les champs du tableau dynamique à afficher lors de l'affichage en liste.

attributs

property

Identifie le nom du champ à afficher dans la colonne.

label

Indique le nom de la colonne à afficher.

ctrl

Indique le type de contrôle à afficher.

Exemple

```
<field property="Tableau" ctrl="resourcetable" resourcetemplate="Tableau" mode="write">
  <subfield property="Date" label="Date" ctrl="date"/>
  <subfield property="Type" label="Type" ctrl="textselectlist"/>
  <subfield property="Cout" label="Coût" ctrl="float"/>
  <subfield property="Client" label="Client" ctrl="text"/>
  <subfield property="Commentaire" label="Commentaire" ctrl="text"/>
</field>
```

<html>

L'élément **html** est utilisé pour afficher une portion d'HTML, il est utilisé pour afficher les informations statiques du formulaire (aide, lien…). Son contenu est placé sans transformation dans le formulaire. L'élément **html** étend **cell**.

attributs

Ceux de **<cell>**

et

eval (optionnel)

Nom d'une variable Java Script déclarée dans le bloc **script**. La valeur string de cette variable sera utilisée comme contenu.

enfants

text() {0,n}

Source html, souvent entouré par **<![CDATA[** et **]]>** pour éviter les conflits.

On peut également imbriquer des éléments de type **section, actionhistory, text, field, html attribute**.

Exemple

```
<html><![CDATA[
    <a href="http://www.vdocsoftware.com">VDoc Software </a>
]]></html>
```

```
<html eval="field_Reference"/>
```

<text>

L'élément **text** est utilisé pour afficher un texte (libellés des champs, aide…). Son contenu est placé dans le formulaire en veillant à remplacer certains caractères html réservés par leurs codes. L'élément **text** étend **cell**.

attributs

ceux de **cell**

et

eval (optionnel)

Nom d'une variable Java Script déclarée dans le bloc **script**. La valeur string de cette variable sera utilisée comme contenu.

enfants

text() {0,1}

Texte libre, quelquefois entouré par **<![CDATA[** et **]]>** pour éviter les conflits.

On peut également imbriquer des éléments de type **section, actionhistory, text, field, html attribute**.

Exemples

```
<text>Demande</text>
<text><![CDATA[ Pierre & Paul ]]></text>
<text eval="field_Nom"/>
```

<attribute>

L'élément **attribute** est utilisé pour afficher un texte. Son contenu est placé dans le formulaire en veillant à remplacer certains caractères réservés d'un attribut html par leurs codes. L'élément **attribute** étend **cell**.

attributs

ceux de **cell**

et

eval (optionnel)

Nom d'une variable Java Script déclarée dans le bloc **script**. La valeur string de cette variable sera utilisée comme contenu.

enfants

text() {0,1}

Texte libre, quelques fois entouré par **<![CDATA[** et **]]>** pour éviter les conflits.

On peut également imbriquer des éléments de type **section, actionhistory, text, field, html attribute**.

Exemple

```
<html>&gt;input type="text" value="<attribute eval="field_Nom">"&lt;</html>
```

<highlighted>

L'élément **highlighted** marche exactement comme **text**, sauf qu'il entoure le texte d'une mise en forme pour le mettre en valeur.

L'élément **highlighted** étend **cell**.

attributs

ceux de **cell**

et

eval (optionnel)

Nom d'une variable Java Script déclarée dans le bloc **script**. La valeur string de cette variable sera utilisée comme contenu.

enfants

text() {0,1} : Texte libre, quelquefois entouré par **<![CDATA[** et **]]>** pour éviter les conflits.

On peut également imbriquer des éléments de type **section, actionhistory, text, field, html attribute**.

Exemples

```
<highlighted>Demande</highlighted>
<highlighted><![CDATA[ Pierre & Paul ]]></highlighted>
<highlighted eval="field_Nom"/>
```

<get>

L'élément **get** affiche la valeur d'un champ système.

attributs

name

Nom système disponible parmi la liste ci-dessous :

procedure.label

Retourne le libellé du processus.

stage.label

Retourne le libellé de l'étape en cours.

treatment.openurl

Retourne l'url d'accès au document. Utilisé dans les formulaires de mails.

comment

Disponible lors de l'envoi de mails d'information.

Retourne le commentaire saisi par l'utilisateur.

enfants

<aucun>

Exemples

```
<get name="procedure.label"/>
<get name="stage.label"/>
```

<actionhistory>

L'élément **actionhistory** représente un contrôle d'affichage de l'historique des actions effectuées à une étape donnée et des champs d'actions associés. Pour cette version, il est limité à une seule ligne décrivant, si elle existe, la dernière action effectuée à cette étape. On peut paramétrer l'affichage : de la décision, de la date, de l'intervenant, de son rôle, du commentaire d'action et des champs d'action. Par défaut, tous ces champs seront affichés.

Exemple :

Décision **Accepter** prise le **22/10/2003 13:34** par **Zorgly** en tant que **Responsable**

Motif: **Tout est possible !**

L'élément **actionhistory** étend **cell**.

attributs

ceux de **cell**

et

stage

Nom système de l'étape en question. ex : demande

action (optionnel)

Affichage de la décision, 1 ou 0 (affiché ou pas), par défaut 1.

date (optionnel)
Affichage de la date, 1 ou 0 (affiché ou pas), par défaut 1.

fulfiller (optionnel)
Affichage de l'intervenant, 1 ou 0 (affiché ou pas), par défaut 1.

role (optionnel)
Affichage de son rôle, 1 ou 0 (affiché ou pas), par défaut 1.

comment (optionnel)
Affichage du commentaire d'action, 1 ou 0 (affiché ou pas) par défaut 1.

fields (optionnel)
Affichage des champs d'action, 1 ou 0 (affiché ou pas) par défaut 1.

enfants
<aucun>

Exemples

```
<actionhistory stage="demande"/>
```

<include>
L'élément **include** n'a pas de représentation. Il est utile pour inclure des sous-formulaires.
L'élément **include** étend **cell**.

attributs
Ceux de **<cell>**
Et

subform
Nom système du sous-formulaire à inclure.

enfants
<aucun>

Deux exemples d'inclusion de sous-formulaires

```
<!--
  Formulaire d'étape stgCREATION :
  inclusion du sous-formulaire nommé sys stgCREATION Edit
-->
<page>
  <body>
    <include subform="sys_stgCREATION_Edit" />
  </body>
</page>

<!--
  Sous-formulaire d'étape sys_stgCREATION_Edit :
  inclusion des sous-formulaires nommés dm.edit et dm.edit
-->
<section labelid="documentManagement 1.0:ActionStage:stgCREATION:label" class="section-
highlighted" helpid="stgCREATION"
descriptionid="documentManagement_1.0:ActionStage:stgCREATION:description">
  <include subform="dm.script" />
  <include subform="dm.edit" />
  <actionhistory stage="stgCREATION" fields="1" />
</section>
```

<fragment>

L'élément **fragment** n'a pas de représentation. Il est utile pour enrober des éléments.
L'élément **col** étend **container** et **cell**.

attributs

Ceux de **<cell>**
et

id (optionnel)
Définit l'identifiant permettant de retrouver le fragment pour l'afficher ou le masquer.

enfants

Ceux de **<container>**

- **Exemple d'utilisation des fragments pour enrober des éléments**

```
<line>
  <fragment id="marker.DelaiConfime.Required">
    <col width="25" class="cell-left">
      <highlighted>* <text labelid="Property:DelaiConfime:label"
        descriptionid="Property:DelaiConfime:description" /></highlighted>
    </col>
    <col class="cell-right">
      <field property="DelaiConfime" ctrl="date" mode="write" required="yes" />
    </col>
  </fragment>
  <fragment id="marker.DelaiConfime.NotRequired">
    <col width="25" class="cell-left">
      <text labelid="Property:DelaiConfime:label"
        descriptionid="Property:DelaiConfime:description" />
    </col>
    <col class="cell-right">
      <field property="DelaiConfime" ctrl="date" mode="write" required="no" />
    </col>
  </fragment>
</line>
```

- **Exemple d'utilisation d'un controller pour afficher et masquer des fragments**

Dans cet exemple est également présentée la manière de rendre un champ obligatoire.

```
<script>
    function onAfterLoad()
    {
      var controller = iWorkflowModule.getController( iWorkflowInstance );

      if ( "UneEtapeParticuliere" == iTaskInstance.getTask().getName() )
      {
        controller.showBodyBlock("marker.DelaiConfime.Required",false);
        controller.showBodyBlock("marker.DelaiConfime.NotRequired",true);
        controller.setMandatory("DelaiConfime",false);
      }
      else
      {
        controller.showBodyBlock("marker.DelaiConfime.Required",true);
        controller.showBodyBlock("marker.DelaiConfime.NotRequired",false);
        controller.setMandatory("DelaiConfime,true);
      }
    }
</script>
```

<button>

L'élément **button** permet de positionner un bouton graphique au sein du formulaire tout comme un élément <text>.

attributs

ceux de **cell**

et

onclick (peu conseillé)

Code Java Script qui sera exécuté.

enfants

text() {1,1}

Il est recommandé d'utiliser **<![CDATA[** et **]]>** pour éviter les conflits.

On peut également imbriquer des éléments de type **section, actionhistory, text, field, html attribute**.

Quelques exemples simples

```
<button label="Hello VDoc" onclick="alert('Hello VDoc !!!')"/>

<button label="Hello VDoc"><![CDATA[
  log.info("Hello VDoc");
  alert("Hello VDoc !");
]]></button>
```

Afficher et masquer des éléments à l'aide de boutons

Dans cet exemple il est montré comment afficher et masquer une zone identifiée à l'aide de boutons.

```
<line>
  <col>
    <button label="Cacher" onclick="show('marker.Text',false)"/>
    <button label="Montrer" onclick="show('marker.Text',true)"/><br/>
    <text id="marker.Text">Ceci est un bloc de texte</text>
  </col>
</line>
```

Augmentation du délai d'une tâche depuis le document

Dans cet exemple, plusieurs traitements sont effectués :

- augmentation du délai de la tâche active ;
- ajout d'un évènement dans l'historique ;
- envoi de mail d'information ;
- fermeture du document.

```
<button label="Increase task delay"><![CDATA[
  // retrouver la définition de la tâche à partir de la tâche active
  var iTask = iTaskInstance.getTask();

  // calcul de la nouvelle date en millisecondes (2j supplémentaires)
  var delayMillis = (2) * 24 * 60 * 60 * 1000;
  var maxEndedDateMillis = iTaskInstance.getCreatedDate().getTime() + delayMillis;

  // transformation en Date
  var maxEndedDate = new java.util.Date( maxEndedDateMillis );
  iTaskInstance.setMaxEndedDate( maxEndedDate );

  // ce commentaire sera placé dans l'historique
  var delayComment = "Délai repoussé" ;

  // ajout de l'évènement dans l'historique
  iWorkflowInstance.getHistory().addEvent( "Augmenter le délai", iTask.getLabel(), iUser, iUser,
    delayComment, iTask.getRole().getLabel() );

  // envoie du message d'information
  var iMailForm = iTask.getMailForm( 3 ); // IMailForm.INFORMATION = 3
  if ( iMailForm.isActive() )
    iWorkflowModule.send(iContext, iTaskInstance, iMailForm, delayComment );

  // Récupérer un controller pour manipuler les boutons du formulaire
```

```
  var resourceController = iWorkflowModule.getController(iWorkflowInstance);

  // Retrouver le bouton de fermeture du document et simuler la fermeture du document
  var button = resourceController.getTopButton( "close" );
  resourceController.execute( button );
]]></button>
```

Chapitre 8 : Personnalisation des pages de site

La plateforme Easysite vous permet de construire des sites complets. Elle vous offre la possibilité de créer des pages rapidement basées sur les éléments tels que les modèles de pages, les modèles de blocs, les blocs ou les composants.

Cette section vous présente l'ensemble des éléments présents dans le Kit de développement VDoc qui vous permettent de rendre vos pages dynamiques.

Extension de page

VDoc fournit une classe de base qui simplifie l'implémentation des classes d'extension. Cette classe de base possède des méthodes pour accéder directement aux éléments souvent utilisés, tels que le module de site, le contexte d'exécution ainsi que le contrôleur de messages.

Les extensions de page s'exécutent dans le contexte de construction et d'affichage d'une page. Le système permet de distinguer les extensions de page sur les pages publiées et celle en cours de développement sur les pages brouillon.

Une extension de page vous permet de réaliser des traitements suivants :

- l'ajout de composants ou blocs au moment du chargement de la page ;
- le masquage de certains éléments de la page par leur nom système ;
- le positionnement des signets ;
- l'évaluation de signets non connus par le produit (signets personnalisés) ;
- la réception des messages liés à un récipient.

Cycle de vie d'une extension de page

Avant que la page ne soit affichée à l'utilisateur, le système déclenche un certain nombre d'événements :

onBeforeLoad() :cette méthode est appelée lors de l'initialisation de la page avant sa reconstruction complète. C'est sur cet événement qu'il est possible de :

- ajouter de nouveaux éléments (composants ou blocs) ;
- remplacer des éléments ou modifier le modèle de bloc utilisé ;
- déclarer des récipients universels.

onAfterLoad() : cette méthode est appelée une fois que tous les composants ou blocs sont évalués. Sur cet événement il est possible :

- de positionner la valeur des signets ;
- de retrouver les éléments constituant la page pour éventuellement les masquer, les remplacer.

onEvaluateBookmarks() : cette méthode est appelée pour chaque signet posé sur la page qui ne seraient pas interprété par le système ;

onMessage() : cette méthode est appelée à la réception d'un message destiné à un récipient non associé avec une classe récipient. En d'autres termes, le récipient est l'extension de page elle-même.

Implémentation d'une classe d'extension de page

Pour implémenter une classe d'extension de page, il suffit de dériver de la classe BasePageExtension. La classe d'extension devra être déclarée dans l'animation de la page concernée (onglet Développement).

Le nom complet est : **com.axemble.vdoc.sdk.site.extensions.BasePageExtension**.

Méthode de la classe de base BasePageExtension

Cette classe de base donne accès au module de site, au contexte d'exécution, ainsi qu'au contrôleur de messages.

```
public abstract class BasePageExtension implements Serializable
{
     final protected ISiteModule getSiteModule();
     final protected ISiteExecutionContext getExecutionContext();
     final protected IMessageController getMessageController();

     public boolean onBeforeLoad();
     public boolean onAfterLoad( IBlock rebuiltBlock );
     public String onEvaluateBookmarks( String expression );
     public boolean onMessage( IMessage message );
}
```

Exemple d'implémentation d'une extension de page

L'exemple suivant montre une extension de page très simple qui surcharge l'ensemble des événements qui peuvent être déclenchés par le système.

```
public class SimplePageExtension extends BasePageExtension
{
     public boolean onBeforeLoad()
     {
            return super.onBeforeLoad();
     }
     public boolean onAfterLoad( IBlock rebuiltBlock )
     {
            return super.onAfterLoad( rebuiltBlock );
     }
     public String onEvaluateBookmarks( String expression )
     {
            return super.onEvaluateBookmarks( expression );
     }
     public boolean onMessage( IMessage message )
     {
            return super.onMessage( message );
     }
}
```

Utilisation des récipients

La notion de récipient représente un objet en mémoire maintenu tout au long de la session HTTP. Ce récipient peut recevoir des messages et tenir des structures d'objets qui seront altérés selon la nature des messages reçus.

Exemple d'utilisation d'un récipient au sein d'une extension de page

L'exemple suivant montre une extension de page qui déclare un récipient et qui traite les messages envoyés à ce récipient.

```
public class RecipientPageExtension extends BasePageExtension
{
 public boolean onBeforeLoad()
 {
  // déclarer un récipient
  getMessageController().registerRecipient( "a-recipient" );

  // envoyer un message au récipient
  getMessageController().sendMessage( "a-recipient", 1, "Bonjour le Monde !!!" );

  return super.onBeforeLoad();
 }
 public boolean onMessage( IMessage message )
 {
  // réceptionner le message
  if ( message.getEventType() == 1 )
  {
   // positionner la valeur du signet ${message} avec le corps du message reçu.
   this.getExecutionContext().getMarkerModel().setValue( "message", message.getBody() );
  }

  return super.onMessage( message );
 }
}
```

Pour plus d'information sur les récipients référez-vous au chapitre « Référence de gestion de site ».

Extension de page Java Script

Le Kit de développement VDoc offre également la possibilité de répondre aux événements déclenchés par le système en JavaScript.

Le code Java Script peut être déposé dans l'animation de la page concernée (onglet Développement).

Exemple de code d'une extension Java Script

L'exemple suivant montre comment répondre aux événements en Java Script (équivalents de ceux présents sur les extensions de page).

Cet exemple vous montre comment récupérer le contexte d'exécution et la page courante (IContent) pour ajouter un composant « Titre » de niveau 1.

Le module de site donne accès à deux usines :

- l'usine à composants : création d'instances de composants ;
- l'usine à blocs : création d'instances de blocs de différentes natures.

```
/**
* This method is called just before building the tree of content components (blocks and
components).
*
* @param siteModule The site module object
*/
function onBeforeLoad(siteModule)
{
  var headingComponent = siteModule.getComponentsFactory().newHeadingComponent();
  headingComponent.setContent( "Hello World!!!" );
  headingComponent.setLevel( "1" );
  siteModule.getExecutionContext().getContent().getBlock().addComponent( headingComponent );
  return true;
}
/**
* This method is called each time the block is about to be rendered.
*
* @param siteModule The site module object
* @param rebuiltBlock The rebuilt block
*/
function onAfterLoad(siteModule, rebuiltBlock)
{
  return true;
}
/**
* This method is called each time an unknown expression has been encountered.
*
* @param siteModule The site module object
* @param expression The expression to evaluate
*/
function onEvaluateBookmarks(siteModule, expression)
{
  return expression;
}
```

Extension de bloc

Le Kit de développement VDoc permet de déclarer au sein d'une page un bloc dont le contenu sera construit à partir d'une extension de bloc.

La classe de base fournit simplifie l'implémentation des classes d'extension de blocs. De la même façon que pour la classe d'extension de page, cette classe possède des méthodes pour accéder directement au module de site, et au contexte d'exécution.

Les extensions de bloc s'exécutent dans le contexte de construction et d'affichage d'une page mais ne concerne que le bloc lui-même.

Une extension de bloc vous permet de réaliser des traitements suivants :

- l'ajout de composants ou blocs au moment du chargement de la page ;
- le positionnement des signets ;

Cycle de vie d'une extension de bloc

Avant que la page ne soit affichée à l'utilisateur, le système déclenche un certain nombre d'événements sur le bloc en cours de construction :

onPrepareBlock() :cette méthode est appelée avec le bloc reconstruit en paramètre. C'est sur ce bloc que vos composants, et sous-blocs pourront être ajoutés.

onPrepareModel() :cette méthode est appelée pour vous permettre de positionner des valeurs aux éventuels signets posés lors de la construction du bloc.

Implémentation d'une extension de bloc

Pour implémenter une classe d'extension de bloc, il suffit de dériver de la classe BaseBlockExtension. La classe d'extension devra être déclarée dans les propriétés du bloc concerné au travers de l'outil de conception WYSIWYG de la page.

Le nom complet est : **com.axemble.vdoc.sdk.site.extensions.BaseBlockExtension**.

Méthode de la classe de base BaseBlockExtension

Cette classe de base donne accès au module de site et au contexte d'exécution.

```
public abstract class BaseBlockExtension implements Serializable
{
    final protected ISiteModule getSiteModule();
    final protected ISiteExecutionContext getExecutionContext();

    public int onPrepareBlock( IBlock rebuiltBlock );
    public void onPrepareModel( IRenderModel renderModel ) throws Exception;
}
```

Exemple d'implémentation d'une extension de bloc

L'exemple suivant montre une extension de bloc très simple qui surcharge les événements déclenchés par le système.

Cet exemple fournit un cas d'utilisation des deux usines qui simplifient la manipulation des composants et des blocs :

- **usine de composants** : regroupe l'ensemble des composants fournis par le produit ;
- **usine de blocs** : regroupe l'ensemble des blocs fournis par le produit.

Il montre comment créer dynamiquement un tableau, une ligne et des cellules et positionner la valeur d'un signet également créé dynamiquement.

```
public class TableBlockExtension extends BaseBlockExtension
{
 public int onPrepareBlock( IBlock rebuiltBlock )
 {
  try
  {
   // création du tableau
   TableBlock tableBlock = getSiteModule().getBlocksFactory().newTableBlock();

   //création d'une ligne de tableau
   TableRowBlock tableRowBlock = getSiteModule().getBlocksFactory().newTableRowBlock();

   // création d'une cellule
   TextComponent textComponent = getSiteModule().getComponentsFactory().newTextComponent();
   textComponent.setContent( "cellule 01" );
   tableRowBlock.addComponent( textComponent );

   // création d'une deuxième cellule
   TextComponent textComponent2 = getSiteModule().getComponentsFactory().newTextComponent();
   textComponent2.setContent( "cellule 0${marker.number}" );
   tableRowBlock.addComponent( textComponent2 );

   tableBlock.addBlock( tableRowBlock );

   rebuiltBlock.addBlock( tableBlock );

   // n'afficher que les éléments ajoutés au bloc reconstruit
   return IRenderOptions.ONLY_CHILDREN;
  }
  catch( Exception e )
  {}
  return super.onPrepareBlock( rebuiltBlock );
 }
 public void onPrepareModel( IRenderModel renderModel ) throws Exception
 {
  renderModel.setValue( "marker.number", "2" );
  super.onPrepareModel( renderModel );
 }
}
```

Extension de bloc Java Script

Le Kit de développement VDoc offre également la possibilité de répondre aux événements déclenchés par le système en JavaScript.

Le code Java Script peut être déposé dans les propriétés du bloc concerné (outil de conception WYSIWYG).

Exemple de code d'une extension de bloc Java Script

L'exemple suivant montre comment répondre aux événements en Java Script (équivalents de ceux présents sur les extensions de bloc).

Cet exemple vous montre comment ajouter un composant « Image » en utilisant la notion de « ProtocolURI ».

```
/**
* This method is called just before building the children (blocks and components).
*
* @param siteModule The site module object
* @param block The current block
* @param renderModel The render model to complete
*/
function onPrepareBlock(siteModule, block)
{
  var protocolUri = siteModule.createProtocolSupport(
"http://www.vdocsoftware.com/vdoc/resource/filecenter/document/02m-00000e-003/schema-offre-vdoc"
);
  var imageComponent = siteModule.getComponentsFactory().newImageComponent();
  imageComponent.setHRef( protocolUri );
  imageComponent.setSourceRef( protocolUri );
  block.addComponent( imageComponent );
  return 2;
}
/**
* This method is called each time the block is about to be rendered.
*
* @param siteModule The site module object
* @param block The current block
* @param renderModel The render model to complete
*/
function onPrepareModel(siteModule, block, renderModel)
{
}
```

Création de composants

Le Kit de développement VDoc vous permet d'enrichir Easysite en vous donnant la possibilité de développer vos propres composants.

Les composants sont des éléments unitaires qui peuvent être ajoutés à une page, un bloc, un modèle de page ou un modèle de bloc. Ils correspondent au plus petit élément manipulé dans Easysite.

Un composant est au minimum constitué de cet ensemble d'éléments :

- **Définition** : description XML du composant ;
- **Fichier de traduction** : fichier XML de traduction des chaînes de caractères ;
- **Contrôleur** : gestion du rendu des événenements ;
- **Extension d'édition des propriétés** : gestion du paramétrage ;
- **Mode de rendu** : les types d'affichage supportés sont les écrans VDoc et les pages JSP.

Tous les fichiers relatifs à un composant devront être placés sous la racine WEB-INF/storage/custom/components/<**project**>/<**package**>. Les fichiers JSP devront être placés dans un sous-niveau « **views** ».

Définition XML d'un composant

Le fichier de définition de composants doit être nommé « components.xml ». Ce fichier permet en réalité de déclarer plusieurs composants. Tous les composants déclarés dans le fichier XML seront rassemblés au sein d'un même paquetage (« package ») et seront affichés sous la même option de menu présente dans l'outil de conception WYSIWYG.

Dans ce fichier XML, chaque composant est représenté par une balise <**element**>. Sous la balise « element » se trouve la definition du composant.

<controller>

Classe Java dérivant de BaseComponentController permettant de positionner des valeurs de signets accessibles pour les rendus JSP.

<extension>

Classe Java dérivant de BaseEditorExtension permettant de manipuler l'ensemble des propriétés présentes dans la balise <section>.

<depends>

Permet de spécifier des dépendances à des fichiers Java Script ou CSS.

<navigation> ou <view>

La balise « navigation » permet d'associer au composant développé un écran VDoc. Utilisez l'attribut « screen » avec pour valeur la concaténation du nom de l'écran et de l'action séparer par un point.

La balise <view> permet de déclarer une page JSP qui affichera le rendu graphique du composant.

<alternateView>

Cette balise permet de définir le rendu intermédiaire lors de l'affichage de la page en mode « conception ».

Vous avez la possibilité de déclarer soit une page JSP de rendu, soit une succession de noms de champs entourés par le caractère $.

<configuration>

Cette balise tient deux sections, « properties » et « advancedProperties » permettant de configurer le composant développé. Chaque section tient des balises <field> dont la description est très similaire à celle que vous pouvez trouver dans les écrans VDoc. L'attribut « preview » permet d'indiquer que la valeur du champ sera affichée en « info-bulle » sur le rendu intermédiaire du composant.

Exemple de fichier XML

Cet exemple montre comment déclarer un composant au sein d'un paquetage. Notez les balises <project>, <package>, <component> qui devront être remplacées respectivement par le nom système du projet, le nom système du paquetage et le nom système du composant. Ces noms correspondant également aux dossiers créés sous la racine « /custom/components ».

```
<?xml version="1.0" encoding="UTF-8"?>
<package name="<project>.<package>" provider="education.com" version="1.0">
 <runtime />
 <depends />
 <components>
  <element name="<project>.<package>.dummy">
   <def>
    <controller
      className="com.vdocsoftware.<project>.components.<package>.dummy.SimpleController" />
    <extension type="editor"
      className="com.vdocsoftware.<project>.components.<package>.dummy.SimpleEditorExtension" />
    <navigation screen="dummy.edit" />
    <alternateView>$fldTextBox$</alternateView>
    <depends>
     <ref type="javascript" value="/resources/external/js/beautiful.js" />
     <ref type="stylesheet" value="/resources/external/css/lovely.css" />
    </depends>
    <configuration>
     <section name="properties">
      <fields>
       <field name="fldTextBox" ctrl="text" width="long" mandatory="true" />
       <field name="fldTextArea" ctrl="textarea" width="long" cssclass="textarea"
         preview="true" />
       <field name="fldCheckBox" ctrl="checkbox" boolean-value="true" />
      </fields>
     </section>
     <section name="advancedProperties">
      <fields />
     </section>
    </configuration>
   </def>
  </element>
 </components>
</package>
```

Fichier de traduction

Le fichier XML de traduction relatif aux composants développés doit être nommé « strings.xml » et placé sous la racine du paquetage (/custom/components/<**project**>/<**package**>).

Ce fichier devra contenir l'ensemble des chaînes utiles pour tous les composants déclarés dans le fichier XML de définition (components.xml).

Exemple de fichier XML de traduction

Dans cet exemple, le nom système utilisé pour l'attribut « value » est important. Il est constitué des éléments suivants séparés par des points :

- du préfixe système « ezs.component » obligatoire ;
- du nom du projet ;
- du nom du paquetage ;
- du nom du composant.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<res>
    <id value="ezs.component.<project>.<package>.label">
           <lang value="fr">Mon composant</lang>
           <lang value="en">My component</lang>
    </id>
    <id value="ezs.component.<project>.<package>.description">
           <lang value="fr">Description de mon composant</lang>
           <lang value="en">My component description</lang>
    </id>
    <id value="ezs.component.<project>.<package>.<component>.label">
           <lang value="fr">Mon élément</lang>
           <lang value="en">My element</lang>
```

```
    </id>
    <id value="ezs.component.<project>.<package>.<component>.description">
           <lang value="fr">Description de mon élément</lang>
           <lang value="en">My element description</lang>
    </id>
</res>
```

Dans le cas de l'utilisation de champs de section les chaînes devront suivre la norme suivante :

```
    <id value="ezs.component.<project>.<package>.<component>.<field>.label">
           <lang value="fr">Mon champ</lang>
           <lang value="en">My field</lang>
    </id>
    <id value="ezs.component.<project>.<package>.<component>.<field>.description">
           <lang value="fr">Description de mon champ</lang>
           <lang value="en">My field description</lang>
    </id>
```

Contrôleur de composant

A chaque composant peut être associé un contrôleur qui a pour but de réceptionner et traiter les événements et positionner les valeurs de signets.

Pour implémenter un contrôleur de composant, il suffit de créer une classe Java qui dérive de la classe BaseComponentController. Cette classe devra être déclarée dans la balise <controller> du fichier XML de definition du composant (components.xml).

Le nom complet de la classe de base est : **com.axemble.vdoc.sdk.impl.base.BaseComponentController**.

Méthode de la classe de base BaseComponentController

Cette classe de base donne accès au module de site qui lui-même permet de manipuler l'ensemble de l'API de site et le contexte d'exécution.

```
public abstract class BaseComponentController implements IComponentController
{
 final protected ISiteModule getSiteModule();
 final public IComponent getComponent();
 final public Map getComponentContext();

 protected abstract void prepareModel( IRenderModel model ) throws Exception;
 protected abstract void processEvent( String eventName, String eventValue ) throws Exception;
}
```

Exemple d'implémentation d'un contrôleur de composant

L'exemple suivant montre comment positionner et récupérer des valeurs à partir du contexte courant du composant et comment manipuler le modèle de rendu.

```
public class SimpleController extends BaseComponentController
{
 protected void processEvent( String eventName, String eventValue ) throws Exception
 {
  // check the incoming event
  if ( eventName.equals( "refresh" ) && eventValue.equals( "true" ) )
  {
   // store an information into the component context
   this.getComponentContext().put( "forceRefresh", "true" );
  }
 }
 protected void prepareModel( IRenderModel model ) throws Exception
 {
  // retrieve the information from the component context
  if ( this.getComponentContext().get( "forceRefresh" ).equals( "true" ) )
  {
   // set a value into the render model
   model.setValue( "executeRefresh", Boolean.TRUE );
  }
  else model.setValue( "executeRefresh", Boolean.FALSE );
```

```
 }
}
```

Extension d'édition des propriétés

Une classe d'extension de propriété d'un composant permet de traiter l'aspect dynamique de l'affichage des propriétés présentes dans le fichier XML de définition du composant.

Pour implémenter une classe d'extension de propriétés d'un composant, il suffit de dériver de la classe BaseEditorExtension. Cette classe devra être déclarée dans le fichier XML de definition du composant (components.xml).

Le nom complet de la classe est : **com.axemble.vdoc.sdk.document.extensions.BaseEditorExtension**.

Méthode de la classe de base BaseEditorExtension

Cette classe de base reprend les concepts des classes de base documentaires : BaseDocumentExtension et BaseResourceExtension. Elle donne accès aux modules de site et de workflow, fournit des méthodes d'accès aux informations relatives à la page courante, permet de manipuler le document associé aux propriétés ainsi qu'au composant graphique représentant le formulaire d'édition des propriétés (CtlAbstractForm).

```
public abstract class BaseEditorExtension implements IEditorExtension
{
 // initialization
 public void init()

 // events
 public boolean onBeforeLoad()
 public boolean onAfterLoad()
 public boolean onBeforeSave()
 public boolean onAfterSave()
 public boolean onBeforeRemove()
 public boolean onAfterRemove()

 // validate event
 public boolean validate()

 // document access
 final public AbstractDocument getDocument()
 final public boolean isCreation()

 // graphical elements access
 final public CtlAbstractForm getForm()
 final public IResourceController getResourceController()

 // site elements
 final public IContentContainer getContentContainer()
 final public IContent getContent()
 final public IContentComponent getContentComponent()
 final public IBlock getRootBlock()
 final public IBlock getParentBlock()
 final public String getComponentName()

 // modules
 final public IBaseWorkflowModule getWorkflowModule()
 final public IBaseSiteModule getSiteModule()
}
```

Mode de rendu

Le Kit de développement VDoc propose deux modes de rendu des composants Easysite :

- sous forme d'écrans VDoc : déclarer un écran VDoc comme étant associé au composant ;
- à l'aide de pages JSP : déclarer un ensemble de pages JSP qui devront être présentes dans le dossier « views » du composant.

Nous orientons toutefois les développements basés sur les écrans VDoc.

Mode « Ecrans VDoc »

Le mode écrans VDoc est activé si dans la description XML de définition du composant vous utilisez la balise « navigation ». Il suffit alors de renseigner l'attribut « screen » avec la concaténation du nom et de l'action de l'écran séparés par un point.

```
<navigation screen="dummy.edit" />
```

Mode « Page JSP »

Le mode pages JSP est activé si dans la description XML de définition du composant vous utilisez la balise « view ». Celle-ci permet de déclarer une page JSP qui réalisera le rendu graphique.

```
<view>default.jsp</view>
```

Exemple de page JSP

Cet exemple présente l'utilisation des nouveaux tags disponibles dans le Kit de développement VDoc :

1. **recipientMessage** : permet d'envoyer des messages à un récipient ;
2. **event** : permet de déclencher un événement ;
3. **resource** : permet de traduire un identifiant de chaîne.

```
<%@ taglib uri='http://www.vdocsoftware.com/enhancement' prefix='sdk' %>

<button type="button" name="sendAMessage"
  onclick="<sdk:recipientMessage recipient="basket"
  eventType="2"
  eventBody="-1" />" class="button2"><span><sdk:resource id="send.a.message" /></span></button>

<button type="button" name="fireAnEvent" onclick="<sdk:event
  name="basket"
  value="2" />" class="button2"><span><sdk:resource id="fire.an.event" /></span></button>
```

Chapitre 9 : Framework de Plugin

Le Kit de développement VDoc vous propose un framework d'enrichissement du fonctionnel d'un site : le framework de Plugin.

Les Plugins sont des composants logiciels unitaires qui s'intègrent dans l'architecture d'un site. Ils peuvent être activés dans l'administration et configurés dans l'espace d'animation du site.

En utilisant pleinement le framework de Navigation et les API, les Plugins vous permettent d'implémenter de nouvelles fonctionnalités au-delà du fonctionnel proposé par le produit VDoc.

Introduction

Le framework de Plugins vous permet :

- de remonter des données issues de systèmes externes ;
- de classer et de présenter des informations dans divers contenus : liste, vue, menu, masque de rendu ;
  - création de masque de rendu sous forme de page dynamique ;
  - positionnement de signets sur les masques de rendu ;
- d'intégrer des informations accessibles depuis le module de recherche du site ;
- et de packager votre plugin constitué d'un certain nombre d'éléments : écrans VDoc, configuration, règles de gestion sur les pages, classes utilisées.

Tout comme les composants, un Plugin est constitué au minimum des éléments suivants :

- **Définition** : description XML du Plugin ;
- **Fichier de traduction** : fichier XML de traduction des chaînes de caractères ;
- **Contrôleur** : gestion du rendu des événenements ;
- **Fournisseur de section** : gestion de la portée d'utilisation du plugin ;
- **Mode de rendu** : masque de rendu créé sous forme d'une page.

Tous les fichiers relatifs à un plugin devront être placés dans le répertoire racine nommé WEB-INF/storage/custom/plugins/<**project**>/<**plugin**>.

Définition XML d'un Plugin

Le fichier de définition de plugin doit être nommé « plugin.xml ». Ce fichier XML permet à la fois de fournir une description du plugin et de référencer tous les éléments externes nécessaires à son exécution (configuration, écrans).

Dans ce fichier XML, la balise racine est <**plugin**>. Sous cette balise se trouve la definition du plugin avec pour balises filles.

<controller>

Classe Java dérivant de BasePluginController permettant de positionner des valeurs de signets accessibles pour les rendus JSP, ou d'autres composants tels que les composants « iterator », et « conditionnel ».

<scope>

Cette balise tient une section associée à un « provider de section » qui permet de définir les champs nécessaires à la construction de la portée du plugin. La notion de portée de plugin correspond au domaine d'exécution du plugin.

Exemple de portée de plugin

Un plugin ayant une portée de plugin limitée à un processus particulier n'afficherait, ou ne traiterait, que les documents relatifs à ce processus.

Chaque section tient des balises <field> dont la description est très similaire à celle que vous pouvez trouver dans les écrans VDoc. Le « provider » associé à cette section permet, comme dans les écrans VDoc, de gérer l'aspect dynamique de l'affichage. La classe de base à utiliser pour le « provider » de section est BaseSectionProvider.

Nom complet : **com.axemble.vdoc.sdk.providers.BaseSectionProvider**.

<root-plugin>

La balise « root-plugin » permet de spécifier une configuration globale du plugin. Cette configuration sera valable pour toutes les instances de plugin créées sur le site.

Exemple de configuration racine d'un plugin

Dans l'exemple suivant deux instances de plugin sont utilisées : Articles_A et Articles_B. La configuration racine indique que si la langue de l'utilisateur est égale à la valeur « en » l'instance de plugin utilisée sera celle placée dans l'arborescence de rubriques « /en/catalog/ » et nommée « Articles_B ». Dans tous les autres cas, l'instance de plugin « Articles_A » sera utilisée.

```
<configuration>
 <plugin uri="/fr/catalog/Articles_A">
  <rule uri="/en/catalog/Articles_B">
   <fieldgroup>
     <field name="${iSiteExecutionContext.Locale.Language}" operator="equals" value="en" />
   </fieldgroup>
  </rule>
 </plugin>
 <custom />
</configuration>
```

Notez également la présence de la balise <custom> qui est entièrement réservée aux développements spécifiques. C'est l'endroit indiqué pour déposer votre paramétrage global du plugin.

<plugin-instance>

La balise « plugin-instance » permet de spécifier la configuration d'une instance de plugin. La configuration par défaut d'une instance peut être personnalisée sur les quatre entrées filles suivantes :

- <**pages**> : ensemble des pages gérées par le plugin. L'attribut « enable-stats » permet d'indiquer que la page sera prise en compte dans les statistiques ;

```
<page id="content" uri="/article" enable-stats="true" />
```

- <**search**> : configuration des colonnes de résultat de recherche ;

```
<search>
 <columns>
  <column name="fldName" label="ezs.plugin.<project>.<plugin>.column.name" zone="title" />
  <column name="fldCreatedDate" label="ezs.plugin.<project>.<plugin>.column.createddate"
    zone="date" displayLabel="true" />
  <column name="fldSize" label="ezs.plugin.<project>.<plugin>.column.size" zone="author"
    displayLabel="true" />
 </columns>
</search>
```

- <**components**> : ensemble des écrans VDoc utilisés et mis à disposition par le plugin. Les écrans VDoc déclarés doivent bien sûr être présents dans un fichier de navigation (ex : process.xml) ;

```
<components>
 <view-component name="view.list">
  <label>ezs.component.<project>.<plugin>.view.label</label>
  <navigation screen="articles.view" />
 </view-component>
 <form-component name="form.edit">
  <label>LG BASKET EDIT</label>
  <navigation screen="basket.edit" />
 </form-component>
 <sheet-component name="sheet.edit">
  <label>LG_SHEET_EDIT</label>
```

```
  <navigation screen="configuration.edit" />
 </sheet-component>
 <wizard-component name="wizard.edit">
  <label>LG WIZARD EDIT</label>
  <navigation screen="process.generate" />
 </wizard-component>
 <screen-component name="screen.edit">
  <label>LG SCREEN EDIT</label>
  <navigation screen="article.edit" />
 </screen-component>
</components>
```

- **<custom>** : configuration libre disponibles pour les développements spécifiques.

Exemple de fichier XML

Cet exemple montre comment déclarer un plugin. Notez les balises <project>, <plugin> qui devront être remplacées respectivement par le nom système du projet, et le nom système du plugin. Ces noms correspondant également aux dossiers créés sous la racine « /custom/plugins ».

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin name="<project>.<plugin>">
 <def>
  <controller>com.vdocsoftware.<project>.plugins.<plugin>.SimplePluginController</controller>
  <scope>
   <section provider="com.vdocsoftware.<project>.plugins.<plugin>.SimpleSectionProvider">
    <fields>
     <field name="fldNavigationScope" label="ezs.plugin.<project>.<plugin>.navigation.scope.label"
       ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField" mandatory="true" />
    </fields>
   </section>
  </scope>
  <root-plugin>
   <configuration />
  </root-plugin>
  <plugin-instance>
   <configuration>
    <search>
     <columns>
      <column name="fldName" label="ezs.plugin.<project>.<plugin>.column.name" zone="title" />
      <column name="fldCreatedDate" label="ezs.plugin.<project>.<plugin>.column.createddate"
        zone="date" displayLabel="true" />
      <column name="fldSize" label="ezs.plugin.education.fs.column.size" zone="author"
        displayLabel="true" />
     </columns>
    </search>
    <pages>
     <page id="index" uri="/index" />
     <page id="content" uri="/content" />
    </pages>
    <!--
    <components>
     <view-component name="articles.list">
      <label>ezs.component.<project>.<plugin>.articles.label</label>
      <navigation screen="articles.view" />
     </view-component>
     <form-component name="form.edit">
      <label>LG FORM EDIT</label>
      <navigation screen="basket.edit" />
     </form-component>
     <sheet-component name="sheet.edit">
      <label>LG SHEET EDIT</label>
      <navigation screen="configuration.edit" />
     </sheet-component>
     <wizard-component name="wizard.edit">
      <label>LG_WIZARD_EDIT</label>
      <navigation screen="process.generate" />
     </wizard-component>
     <screen-component name="screen.edit">
      <label>LG_SCREEN_EDIT</label>
      <navigation screen="article.edit" />
     </screen-component>
    </components>
    -->
    <custom />
```

```
   </configuration>
  </plugin-instance>
 </def>
</plugin>
```

Fichier de traduction

Le fichier XML de traduction relatif au plugin développé doit être nommé « strings.xml » et placé sous la racine du plugin (/custom/plugins/<**project**>/<**plugin**>).

Ce fichier devra contenir l'ensemble des chaînes utiles pour le bon fonctionnement du plugin et la configuration de ses éléments.

Exemple de fichier XML de traduction

Dans cet exemple, le nom système utilisé pour l'attribut « value » est important. Il est constitué des éléments suivants séparés par des points :

- du préfixe système « ezs.plugin » obligatoire ;
- du nom du projet ;
- du nom du plugin.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<res>
 <id value="ezs.plugin.<project><plugin>.label">
  <lang value="fr">Système de fichiers</lang>
  <lang value="en">File system</lang>
 </id>
 <id value="ezs.plugin.<project><plugin>.description">
  <lang value="fr">Permet de naviguer dans le système de fichiers</lang>
  <lang value="en">Allows browsing the file system</lang>
 </id>
 <id value="ezs.plugin.<project><plugin>.navigation.scope.label">
  <lang value="fr">Répertoire de navigation</lang>
  <lang value="en">Navigation directory</lang>
 </id>
 <id value="ezs.plugin.<project><plugin>.column.name">
  <lang value="fr">Nom</lang>
  <lang value="en">Name</lang>
 </id>
 <id value="ezs.plugin.<project><plugin>.column.createddate">
  <lang value="fr">Date de création</lang>
  <lang value="en">Creation date</lang>
 </id>
 <id value="ezs.plugin.<project><plugin>.column.size">
  <lang value="fr">Taille</lang>
  <lang value="en">Size</lang>
 </id>
</res>
```

Contrôleur de Plugin

Au plugin est associé un contrôleur qui constitue le cœur du plugin. Toutes les actions utilisateurs seront orientées vers ce contrôleur. Ce dernier exécutera les traitements associés et mettra à jour, si nécessaire, le modèle permettant d'alimenter les signets proposés sur les divers masques de rendu.

Pour implémenter un contrôleur de plugin, il suffit de créer une classe Java qui dérive de la classe BasePluginController. Cette classe devra être déclarée dans la balise <controller> du fichier XML de definition du plugin (plugin.xml).

Le nom complet de la classe de base est : **com.axemble.vdoc.sdk.impl.base.BasePluginController**.

Méthode de la classe de base BasePluginController

Cette classe de base donne accès au module de site, aux informations du plugin et au contexte d'exécution.

```
public abstract class BasePluginController
{
 // accès au module de site
 final protected ISiteModule getSiteModule()
```

```
// création d'une requête de plugin
final protected IPluginRequest newPluginRequest( String path )

// création d'éléments relatifs au référencement
final protected SEO newSEO()
final protected Meta newMeta( String name, String content )

// accès aux informations du plugin
final public IPlugin getPlugin()
final public Map getPluginContext()
final public IRootPluginConfiguration getRootConfiguration()
final public String getPluginBaseUrl()

// retrouver un objet à partir des informations de la requête du plugin
protected abstract Object getObject( IPluginRequest pluginRequest ) throws Exception;

// construction du modèle à partir de l'objet retrouvé
protected abstract void buildModel( Object object, IRenderModel model ) throws Exception;

// construire l'identifiant unique à partir de la requête du plugin
protected abstract String getObjectUniqueIdentifier( IPluginRequest pluginRequest )
  throws Exception;

// retrouver la requête à partir de l'objet manipulé
protected abstract IPluginRequest getPluginRequest( Object object ) throws Exception;

// retrouver la requête à partir de l'identifiant unique
protected abstract IPluginRequest getPluginRequest( String uniqueIdentifier )
  throws Exception;

// déterminer en fonction de l'état des données internes le masque de rendu à utiliser
protected abstract String preparePage( IPluginRequest pluginRequest, IRenderModel renderModel )
  throws Exception;
final protected boolean pageExists( String pageId )

// positionner un certain nombre d'attribut nécessaires
protected abstract void completePage( IVirtualPage virtualPage ) throws Exception;

// exécution de la recherche
protected abstract Collection search( String keywords, int maxResults ) throws Exception;
}
```

Cycle de vie du contrôleur de plugin

Le framework de Plugin invoque les méthodes exposées par le contrôleur du plugin dès qu'une URL correspondant au format suivant est appelée :

```
<protocol>://<server>:<port>/<application>/easysite/<site>/<plugin-uri>
```

Exemple d'URL destinée à un plugin

Tous les éléments d'URL placés derrière <plugin-uri> sont à destination du contrôleur de plugin. L'exemple suivant montre comment utiliser l'objet IPluginRequest pour récupérer la « request URI » et les paramètres associés à une URL.

```
Exemple d'URL à destination d'un plugin
<protocol>://<server>:<port>/<application>/easysite/<site>/<plugin-uri>/shop/article/xyz?arg1=X

IPluginRequest pluginRequest…
pluginRequest.getRequestURI(); // Renvoie /shop/article/xyz
pluginRequest.getParameterValue("arg1"); // Renvoie la valeur "X"
```

Lorsqu'une URL est appelée et correspond au format permettant d'atteindre un plugin, le framework de plugin est sollicité. Celui-ci décompose l'URL pour déterminer l'instance de plugin concernée. Depuis l'instance de plugin trouvée, le framework de Plugin récupère le contrôleur de plugin associé et invoque les méthodes exposées.

La première méthode appelée est **getObject**() qui permet de traduire une URL en un objet fonctionnel.

Exemple : dans le cas d'un catalogue d'articles en ligne, l'objet fonctionnel serait un article.

Une fois l'objet fonctionnel récupéré, le framework appelle la méthode **buildModel**(). Cette fonction permet de renseigner les valeurs de signets proposés par le plugin.

Le framework de plugin appelle ensuite la méthode **preparePage**() qui devra retourner l'identifiant de page (masque de rendu) à utiliser pour présenter les informations.

La méthode **completePage**() est appelée pour traiter les informations relatives aux statistiques. Pour cela une structure est passée en argument vous permettant de spécifier certaines valeurs de propriétés, de l'objet fonctionnel manipulé, que vous souhaiteriez rendre disponibles pour les statistiques. Parmi ces propriétés, vous sont proposées le titre, l'auteur, la description et le résumé. Cette structure est l'interface **IVirtualPage**.

Par ailleurs, le plugin peut être appelé sur la méthode **search**() à l'exécution d'une recherche sur le site. L'appel à la méthode search() sera effective que lorsque le plugin sera ajouté aux domaines de recherche de l' « élément de recherche ».

Enfin, plusieurs autres méthodes peuvent être appelées par le framework de plugin pour réaliser des conversions :

- à partir de l'objet récupérer la IPluginRequest ;
- à partir de l'identifiant unique récupérer la IPluginRequest ;
- à partir de la IPluginRequest récupérer l'identifiant unique ;
- à partir de la IPluginRequest récupérer l'objet fonctionnel ;

Exemple d'implémentation d'un contrôleur de plugin

L'exemple suivant montre comment réaliser un contrôleur de plugin qui permet de remonter des informations du système de fichier à partir d'un dossier racine défini.

Cet exemple illustre plusieurs notions tels que :

- la portée : représentée ici par un sous-dossier ;
- le positionnement des valeurs de signets ;
- le choix du masque de rendu ;
- la construction du modèle de recherche ;
- et la manipulation de la requête de plugin.

```
public class FSPluginController extends BasePluginController
{
 private static final long serialVersionUID = -6864905233548315067L;

 // répertoire racine
 private static final String ROOT_DIRECTORY = "D:/tmp/dummies";

 // noms des signets
 public final static String FIELD_NAME = "fldName";
 public final static String FIELD_CREATEDDATE = "fldCreatedDate";
 public final static String FIELD_SIZE = "fldSize";

 protected Object getObject( IPluginRequest pluginRequest ) throws Exception
 {
  // récupération de l'URI à partir de la requête du plugin
  String path = pluginRequest.getRequestURI();

  // récupération du dossier racine
  File rootDirectory = new File( ROOT_DIRECTORY );

  // construction de l'objet java.io.File à partir des paramètres
  // - dossier racine ;
  // - portée du plugin (scope) ;
  // - URI du plugin.
  return new File( rootDirectory.getPath() + File.separator +
this.getPlugin().getScope().getValue( FSSectionProvider.FIELD_SCOPE ) + path );
 }
 protected IPluginRequest getPluginRequest( Object object ) throws Exception
 {
  // vérification de la validité de l'objet passé en argument.
  if ( !( object instanceof File ) )
```

```
   throw new IllegalArgumentException( "The 'object' argument sould be a java.io.File type." );

  // Construction d'une request à partir de l'objet passé en argument.
  return newPluginRequest( "/" + ( (File)object ).getName() );
 }
 protected String getObjectUniqueIdentifier( IPluginRequest pluginRequest ) throws Exception
 {
  // construction d'une chaîne unique à partir de la requête du plugin.
  return pluginRequest.getRequestURL();
 }
 protected IPluginRequest getPluginRequest( String uniqueIdentifier ) throws Exception
 {
  // construction d'une requête de plugin à partir de l'identifiant unique.
  return newPluginRequest( uniqueIdentifier );
 }
 protected String preparePage( IPluginRequest pluginRequest, IRenderModel renderModel )
  throws Exception
 {
  // positionnement du masque de rendu.
  return ( "content" );
 }
 protected void buildModel( Object object, IRenderModel model ) throws Exception
 {
  // mise à jour du modèle pour alimenter les signets.
  updateModel( object, model );
 }
 protected Collection search( String keywords, int maxResults ) throws Exception
 {
  ArrayList list = new ArrayList();

  File rootDirectory = new File( ROOT DIRECTORY );

  // respecter la portée de recherche
  rootDirectory = new File( rootDirectory.getPath() + File.separator +
this.getPlugin().getScope().getValue( FSSectionProvider.FIELD SCOPE ) );

  File[] directories = rootDirectory.listFiles();

  int index = 0;
  for ( int indexDirectory = 0 ; indexDirectory < directories.length ; indexDirectory++ )
  {
   File directory = directories[indexDirectory];
   if ( directory.isDirectory() )
   {
    if ( !directory.getName().equals( getInnerPluginScope().getPropertyValue(
FSSectionProvider.FIELD_SCOPE ) ) )
     continue;

    File[] files = directory.listFiles();
    for ( int indexFile = 0 ; indexFile < files.length ; indexFile++ )
    {
     File file = files[indexFile];
     if ( !file.getName().contains( keywords ) )
      continue;

     // pour chaque fichier trouvé, contruire un élément modèle typé recherche
     SearchResultModelItem searchResult = new SearchResultModelItem();

     // mise à jour du modèle à partir du fichier
     updateModel( file, searchResult );
     list.add( searchResult );

     index++;
     if ( index > maxResults )
      return list;
    }
   }
   else
   {
    File file = directory;
    if ( !file.getName().contains( keywords ) )
     continue;

    // contruire un élément modèle typé recherche
    SearchResultModelItem searchResult = new SearchResultModelItem();

    // mise à jour du modèle à partir du fichier
    updateModel( file, searchResult );
```

```
    list.add( searchResult );

    index++;
    if ( index > maxResults )
     return list;
   }
  }
  return list;
 }
 protected void completePage( IVirtualPage virtualPage ) throws Exception
 {
  IPluginRequest pluginRequest = virtualPage.getPluginRequest();

  File file = (File)this.getObject( pluginRequest );

  // positionnement de quelques propriétés de la page affichée
  virtualPage.setLabel( file.getName() );
 }
 private void updateModel( Object object, IModel model ) throws Exception
 {
  if ( !( object instanceof File ) )
   throw new IllegalArgumentException( "The 'object' argument sould be a java.io.File type." );

  File file = (File)object;

  IPluginRequest pluginRequest = getPluginRequest( file );

  // positionnement du lien cliquable
  model.setLinkUrl( pluginRequest.getRequestURL() );

  // positionnement de la clé
  model.setKey( pluginRequest.getRequestURL() );

  // positionnement de certaines valeurs correspondant à des signets
  model.setValue( FIELD NAME, file.getName() );
  model.setValue( FIELD CREATEDDATE, new Timestamp( file.lastModified() ) );
  model.setValue( FIELD SIZE, new Long( file.length() ) );
 }
}
```

Fournisseur de section : notion de portée

Une classe « provider de section » d'un plugin permet de définir sa portée d'exécution. Cette classe permet également de traiter l'aspect dynamique de l'affichage des propriétés présentes au sein de la balise <scope> du fichier XML de définition du plugin.

Ayez en tête qu'une portée peut être déterminée à partir plusieurs propriétés.

```
<scope>
   <section provider="com.vdocsoftware.<project>.plugins.<plugin>.SimpleSectionProvider">
    <fields>
     <field
       name="fldNavigationScope"
       label="ezs.plugin.<project>.<plugin>.navigation.scope.label"
       ctrl="com.axemble.vdp.ui.core.document.fields.ComboBoxField"
       mandatory="true" />
    </fields>
   </section>
</scope>
```

Pour implémenter une classe « provider de section », il suffit de dériver de la classe BaseSectionProvider. Cette classe devra être déclarée dans le fichier XML de definition du plugin (plugin.xml).

Le nom complet de la classe est : **com.axemble.vdoc.sdk.providers.BaseSectionProvider**.

Méthode de la classe de base BaseSectionProvider

Cette classe de base reprend les concepts des classes de base de fournisseur : BaseFormProvider, BaseSheetProvider, BaseWizardProvider. Elle donne accès aux modules de site et de workflow, fournit des méthodes d'accès aux informations relatives à la section courante, permet de manipuler le document associé aux propriétés ainsi qu'au composant graphique représentant le formulaire d'édition des propriétés (CtlSection).

```
public abstract class BaseSectionProvider
{
  // initialisation methodes
  public boolean prepareData()
  public void activate()
  public boolean load( AbstractField value, Element element )

  // modules accessors
  final protected IWorkflowModule getWorkflowModule()
  final protected ISiteModule getSiteModule()

  // context accessors
  final public IContext getLoggedOnContext()
  final public IResourceController getResourceController()
  final protected IPlugin getPlugin()
}
```

Mode de rendu

Le mode de rendu des plugins s'effectue à l'aide de pages référencées dans la configuration du plugin. Ces pages ne doivent pas être accessibles par lien URL direct car elles doivent nécessairement être évaluées par le contrôleur de plugin avant le rendu définitif.

Le framework de Plugin vous propose un plugin system qui vous permet d'utiliser ou personnaliser les pages systèmes suivantes ;

- **404NotFound** : page non trouvée ;
- **410Gone** : page expirée ;
- **401Unauthorized** : accès refusé ;
- **500InternalServerError** : erreur interne.

Exemple de personnalisation d'une page système

Pour personnaliser une page prédéfinie, il suffit de suivre la procédure suivante :

- ajouter, depuis l'administration, le plugin « système » ;
- créer une instance de ce plugin depuis l'animation ;
- créer une page nommée « 404NotFound », par exemple ;
- ajouter un élément « texte » en positionnant le signet ${exceptionMessage}.

Déploiement et configuration d'un plugin

Déploiement

Le déploiement d'un plugin consiste simplement à déposer :

- la bibliothèque de classes (JAR) dans « default/lib » du serveur JBOSS ;
- et les fichiers de définition et de traduction dans le dossier storage/custom/plugins/<project>/<plugin>.

Configuration

Les plugins sont configurables à deux endroits :

- l'administration du site : configuration du plugin racine ;
- et dans l'espace animation : configuration de l'instance de plugin.

Configuration depuis l'administration du site

Un nouveau plugin doit être ajouté dans l'administration du site pour être configuré et également rendu visible dans l'espace d'animation.

Pour cela, allez dans l'onglet « Plugins » et cliquez sur le bouton **Ajouter**. Un assistant de création de nouveau plugin présente la liste des plugins déployés sur le serveur. Sélectionnez votre plugin et suivez les étapes de l'assistant.

Une fois le plugin référencé sur le site, il peut être configuré en éditant ses propriétés. Cette configuration sera globale à toutes les instances de plugin (administrable depuis l'espace d'animation).

Configuration depuis l'espace d'animation du site

Pour bénéficier du fonctionnel d'un plugin et le configurer il est nécessaire de créer une instance de plugin dans l'espace d'animation. L'instance de plugin peut être placée à la racine du site ou sous une rubrique souhaitée.

Pour créer une instance de plugin, il suffit de cliquer sur le bouton **Ajouter**. Un assistant affiche la liste des plugins qui ont été rendus disponibles depuis l'administration. Sélectionnez votre plugin et suivez les instructions de l'assistant.

Une fois l'instance de plugin créée, elle peut être configurée en éditant ses propriétés.

Sur un même site, plusieurs instances de plugin peuvent être ajoutées. Chaque instance de plugin pourra être définie différement avec des noms de page différents et des rendus graphiques différents.

Chapitre 10 : La messagerie

Utilisation des formulaires de mail

Le module de workflow permet d'utiliser des formulaires de mail en dehors du contexte d'un document processus. Pour cela, une ressource générique est utilisée. Celle-ci se manipule de la même façon que les interfaces IWorkflowInstance et ILinkedResource.

Exemple de code

Cet exemple présente l'envoi d'un message électronique utilisant une ressource générique (fictive) en dehors du contexte d'un document processus.

```
public void general_sendMail( IContext context, IWorkflowModule workflowModule ) throws Exception
{
    // création d'une liste d'utilisateurs
    ArrayList arrUsers = new ArrayList();
    arrUsers.add( workflowModule.getUserByLogin( "froggy" ) );
    arrUsers.add( workflowModule.getUserByLogin( "zorgly" ) );

    // sélectionner le nom du formulaire à utiliser
    String formName = "MY_CUSTOM_MAIL_FORM";

    // créer une resource fictive
    IResource resource = workflowModule.createGenericResource();

    // alimenter avec quelques champs présents dans le formulaire de mail sélectionné
    resource.setValue( IProperty.System.REFERENCE, "REF-ZRG-0001" );
    resource.setValue( IProperty.System.TITLE, "My lovely document" );
    resource.setValue( "any-property-name", "any-value" );

    // récupérer la version de processus qui contient le formulaire de mail sélectionné
    IWorkflow workflow = workflowModule.getWorkflow( context, "DocumentManagement_1.0" );

    // envoyer le mail
    workflowModule.send( context, workflow, resource, formName, arrUsers );
}
```

Si une extension de messagerie est déclarée dans le formulaire de mail utilisé, alors elle sera appelée lors de l'envoi du message électronique sur les événements beforeSend() et afterSend().

Les extensions de messagerie

VDoc permet aux intégrateurs de développer des classes Java qui pourront réagir sur des évènements déclenchés lors de l'envoi de messages électroniques.

Cette notion d'extension doit être utilisée dans le cas où vous souhaitez, par exemple, compléter la liste des destinataires par défaut qui recevront le message.

Ces classes d'extension peuvent être déclarées sur les formulaires de message.

Principe

Lors d'un changement d'étape, par exemple, si la messagerie est bien configurée et que le formulaire de messages associé à l'action est actif, un mail sera envoyé par VDoc.

Cycle de vie

Avant que cet envoi de mail ne soit effectif, les classes d'extension de message sont appelées sur la méthode **beforeSend**(). Sur cet évènement il est alors possible d'inhiber ou de compléter l'envoi de message.

Après l'envoi de mail, VDoc rappelle la classe d'extension sur la méthode **afterSend**(). Sur cet évènement il est possible de réaliser des tâches annexes, mais en gardant à l'esprit qu'il n'est plus possible d'intervenir sur l'envoi du message.

Développer une classe d'extension

- Pour développer une classe d'extension de mail, il suffit d'étendre la classe nommée BaseMailExtension.

La classe BaseMailExtension

La classe **com.axemble.vdoc.sdk.mail.extensions.BaseMailExtension** fournit un accès direct aux méthodes des APIs et propose des raccourcis pour les éléments suivants :

- le module de Workflow ;
- le dernier operator ;
- le document actif ;
- la tâche active ;

Méthodes de la classe BaseMailExtension

```
public abstract class BaseMailExtension implements IMailExtensionSupport
{
    // helper methods
    protected final IWorkflowModule getWorkflowModule();
    protected final IWorkflowInstance getWorkflowInstance();
    protected final ITaskInstance getTaskInstance();
    protected final IUser getPreviousTaskInstanceOperator();
    protected final MimeMessage getMessage();

    // gestion dynamique du format xml de personnalisation du formulaire
    public void onPrepare( Element documentElement );

    // mail events
    protected abstract boolean beforeSend();
    protected abstract void afterSend();
}
```

La méthode onPrepare() permet de modifier en dynamique le flux XML de présentation du mail avant qu'il ne soit interprété par le moteur de rendu.

Extrait du code de la classe NewMailExtension

Le code suivant complète la liste des destinataires du message en ajoutant tous les membres du rôle « Responsable RH ».

```
public class NewMailExtension extends BaseMailExtension
{

    private final static Logger LOGGER = Logger.getLogger(NewMailExtension.class);

    /* (non-Javadoc)
     * @see com.axemble.vdoc.sdk.mail.BaseMailExtension#beforeSend()
     */
    protected boolean beforeSend()
    {
            try
            {
                    // Récupération de l'application dans laquelle se situe le document
                    ICatalog catalog = this.getWorkflowInstance().getCatalog();

                    // Récupération du rôle contenant les utilisateurs à qui nous souhaitons
                    // envoyer un message
                    IRole role = catalog.getRole( "ResponsableRH" );
```

```
                    // Extraction de tous les utilisateurs membres du rôle
                    Collection users = role.getAllMembers();

                    ArrayList newAdresses = new ArrayList();

                    for ( Iterator iter = users.iterator() ; iter.hasNext() ; )
                    {
                            IUser user = (IUser)iter.next();
                            String email = user.getEmail();
                            if ( email != null && !email.equals( "" ) )
                              newAdresses.add( new InternetAddress( email, user.getFullName() ) );
                    }

                    // Transformation en tableau
                    InternetAddress[] iadresses = new InternetAddress[newAdresses.size()];
                    newAdresses.toArray( iadresses );

                    // Ajout des destinataires complémentaires
                    this.getMessage().addRecipients( Message.RecipientType.CC, iadresses );

                    return true;
            }
            catch( UnsupportedEncodingException e )
            {
                    LOGGER.error( e.toString() );
            }
            catch( MessagingException e )
            {
                    LOGGER.error( e.toString() );
            }

            return false;
    }
    /* (non-Javadoc)
     * @see com.axemble.vdoc.sdk.mail.BaseMailExtension#afterSend()
     */
    protected void afterSend()
    {
    }
}
```

Chapitre 11 : Interaction avec le back-office

Introduction

Les entreprises souhaitent pouvoir inclure leur système d'information lorsqu'elles automatisent leurs processus. En effet, les informations stratégiques résident généralement dans des systèmes métier de back-office extrêmement hétéroclites.

VDoc utilise à la fois des extensions Java et des scripts d'exécution back-office pour permettre aux entreprises d'associer ces systèmes disparates aux processus à automatiser, et d'optimiser leurs performances et leur productivité.

Le système VDoc propose les types d'extension Java suivants :

- extensions sur transactions (uniquement sur l'annuaire) ;
- modules et extensions d'authentification ;
- les ouvertures de code JavaScript.

Les extensions sur les transactions de l'annuaire

La classe BaseDirectoryTransactionExtension

La **classe com.axemble.vdoc.sdk.transaction.extensions.BaseDirectoryTransactionExtension** fournit un accès direct aux méthodes des APIs de l'annuaire. Elle propose des raccourcis pour les opérations suivantes :

- les éléments créés ;
- les éléments mis à jour ;
- les éléments supprimés ;
- les relations entre éléments ;

Méthodes de la classe BaseDirectoryTransactionExtension

```
public abstract class BaseDirectoryTransactionExtension extends BaseTransactionExtension
{
    // helper methods
    protected final IDirectoryModule getDirectoryModule()
    protected final Collection getCreatedOrganizations()
    protected final Collection getUpdatedOrganizations()
    protected final Collection getDeletedOrganizations()
    protected final IOrganization getPreviousOrganization( IStorageKey id )
      throws DirectoryModuleException
    protected final Collection getCreatedLocalizations()
    protected final Collection getUpdatedLocalizations()
    protected final Collection getDeletedLocalizations()
    protected final ILocalization getPreviousLocalization( IStorageKey id )
      throws DirectoryModuleException
    protected final Collection getCreatedGroups()
    protected final Collection getUpdatedGroups()
    protected final Collection getDeletedGroups()
    protected final IGroup getPreviousGroup( IStorageKey id ) throws DirectoryModuleException
    protected final Collection getCreatedUsers()
    protected final Collection getUpdatedUsers()
    protected final Collection getDeletedUsers()
    protected final IUser getPreviousUser( IStorageKey id ) throws DirectoryModuleException
    protected final Collection getCreatedUserParents( IUser user )
    protected final Collection getDeletedUserParents( IUser user )
    protected final Collection getCreatedGroupParents( IGroup group )
    protected final Collection getDeletedGroupParents( IGroup group )
    protected final Collection getCreatedGroupChildren( IGroup group )
```

```
    protected final Collection getDeletedGroupChildren( IGroup group )
    protected final Collection getCreatedGroupMembers( IGroup group )
    protected final Collection getDeletedGroupMembers( IGroup group )

    // methods to implement
    public boolean onBeforeCommit()
    public void onAfterCommit()
    public void onAfterRollback()
}
```

Les méthodes dites « helper » permettent de récupérer les éléments impliqués lors de la transaction.

Important
La récupération des anciennes valeurs via les méthodes getPreviousXXX() se limitent aux propriétés directes de l'objet manipulé. La récupération d'objets liés n'est pas conseillée car elle risque de provoquer des locks en base (exemple de ce qu'il ne faudrait pas faire : previousUser.getOrganization().getParent()).

Les méthodes onBeforeCommit(), onAfterCommit(), et onAfterRollback() permettent de placer du code sur les événements de la transaction.

La déclaration d'une classe d'extension sur les transactions de l'annuaire peut être faite dans le fichier CustomResource.properties en spécifiant une valeur pour la clé com.axemble.directory.transaction.extensions. La valeur doit être un nom de classe étendant BaseDirectoryExtension.

Exemple de déclaration d'une extension de transaction de l'annuaire

```
com.axemble.vdoc.sdk.transaction.DirectoryTransactionExtensions=com.axemble.education.directory.ex
tensions.DemoTransactionExtension
```

Les modules et extensions d'authentification

Configuration des modules d'authentification

Le fichier **login-modules.xml** présent sur la distribution VDoc permet de déclarer différents modules d'authentification.

Extrait du fichier de configuration

```
<application-policy name="VDoc">
  <!-- JAAS configuration for VDoc-->
  <authentication>
    <login-module code="com.axemble.security.loginmodules.KerberosAutoLoginModule"
      flag="sufficient">
      <module-option name="domains">lyon,axemble</module-option>
    </login-module>
    <login-module code="com.axemble.security.loginmodules.CookiesAutoLoginModule"
      flag="sufficient" />
    <!-- <login-module code="com.axemble.security.loginmodules.LDAPLoginModule" flag="sufficient"
    /> -->
    <login-module code="com.axemble.security.loginmodules.ForceUserAutoLoginModule"
      flag="sufficient">
      <module-option name="user">sysadmin</module-option>
      <module-option name="address">192.168.1.2</module-option>
    </login-module>
    <login-module code="com.axemble.security.loginmodules.ForceUserAutoLoginModule"
      flag="sufficient">
      <module-option name="user">user1</module-option>
      <module-option name="address">192.168.1.*</module-option>
    </login-module>
    <login-module code="com.axemble.security.loginmodules.DirectoryLoginModule" flag="sufficient"
    />
  </authentication>
</application-policy>
```

La classe BaseAutoLoginModule

La **classe com.axemble.vdoc.sdk.authentication.base.BaseAutoLoginModule** simplifit la mise en place d'un module d'authentification automatique. Cette classe n'est appelée que si les informations d'identifiant et mot de passe utilisateur ne sont pas présentes dans la requête HTTP.

Méthodes de la classe BaseAutoLoginModule

```
public class BaseAutoLoginModule extends AbstractAutoLoginModule
{
    // helper methods
    final protected Object getOption( String key );
    final protected Map getOptions();

    // method to implement
    public String doAutoLogin() throws LoginException;
}
```

La méthode doAutoLogin() doit renvoyer le login de l'utilisateur authentifié.

La méthode getOption() permet de récupérer la valeur associée à une clé posée dans le fichier de configuration de l'authentification JAAS.

La classe BasePasswordLoginModule

La **classe com.axemble.vdoc.sdk.authentication.base.BasePasswordLoginModule** simplifit la mise en place d'un module d'authentification basé sur les information d'identifiant et mot de passe utilisateur contenues dans la requête HTTP.

Méthodes de la classe BasePasswordLoginModule

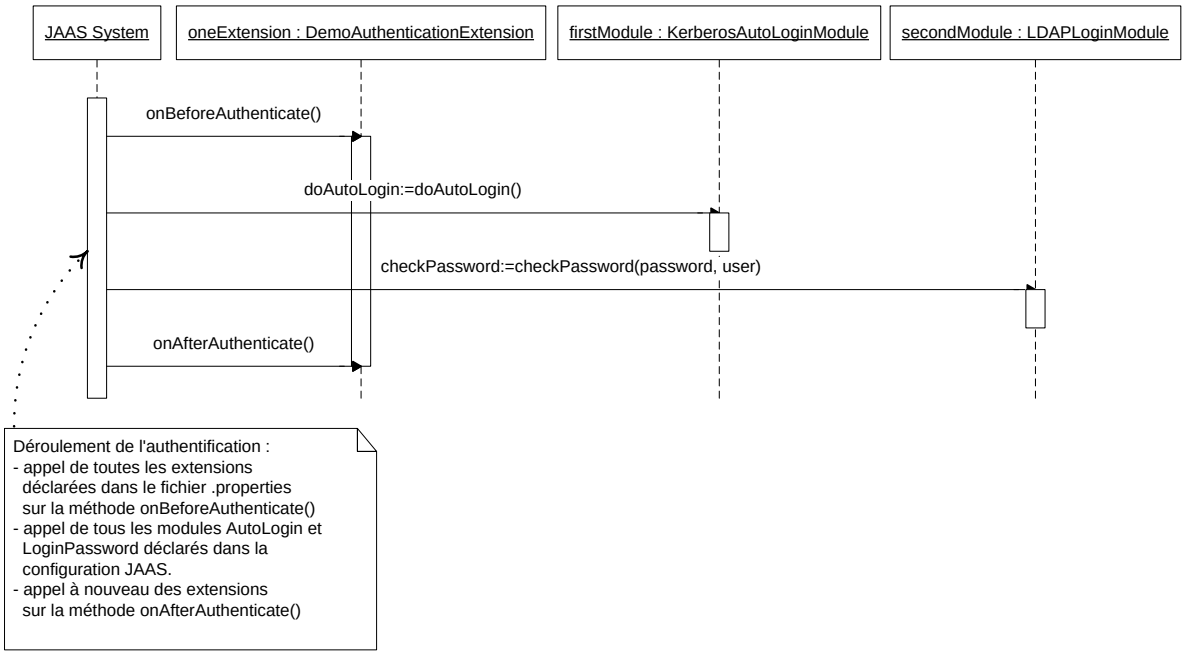
```
public abstract class BasePasswordLoginModule extends AbstractLoginPasswordModule
{
     // helper methods
     final protected Object getOption( String key );
     final protected Map getOptions();

     // method to implement
     public abstract boolean checkPassword( String password, IUser user )
       throws AuthenticationException, LoginException;
}
```

La méthode checkPassword() doit renvoyer « true » ou « false ».

La méthode getOption() permet de récupérer la valeur associée à une clé posée dans le fichier de configuration de l'authentification JAAS.

Diagramme de séquence du système d'authentification

Le schéma suivant montre la séquence d'appel des différents modules ainsi que des classes d'extension d'authentification



La classe BaseAuthenticationExtension

La **classe com.axemble.vdoc.sdk.authentication.base.BaseAuthenticationExtension** est appelée sur deux évènements : onBeforeAuthenticate() et onAfterAuthenticate().

Méthodes de la classe BaseAuthenticationExtension

```
public class BaseAuthenticationExtension implements Serializable
{
     // methods to implement
     public boolean onBeforeAuthenticate()
     public boolean onAfterAuthenticate()
}
```

L'événement onBeforeAuthenticate() est appelé avant l'évaluation de tous les modules d'authentification.

L'événement onAfterAuthenticate() est appelé après l'évaluation.

La déclaration d'une classe d'authentification peut être faite dans le fichier CustomResource.properties en spécifiant une valeur pour la clé com.axemble.vdoc.sdk.security.AuthenticationExtensions. La valeur doit être un nom de classe étendant BaseAuthenticationExtension.

Exemple de déclaration d'une extension d'authentification

```
com.axemble.vdoc.sdk.security.AuthenticationExtensions=com.axemble.education.authentication.extens
ions.DemoAuthenticationExtension
```

Les ouvertures de code Java Script

VDoc met à disposition un certain nombre d'ouvertures Java Script qui sont activées au moment où l'information est en train d'être inscrite en base. Ces ouvertures script peuvent être comparées à des triggers classiques de base de données.

Tableau des ouvertures de code

Nom de fonction	Description	Utilisation
onEvaluateOperators	Cette méthode est appelée lors de l'évaluation des prochains intervenants sur l'étape.	`function onEvaluateOperators(workflowModule, taskInstance, list)` `{` `}`
onCreateTaskInstance	Cette méthode est appelée sur chaque création d'activité manuelle (ActionActivity ou DecisionActivity).	`function onCreateTaskInstance(workflowModule, taskInstance)` `{` `}`
onAbortWorkflowInstance	Cette méthode est appelée sur l'annulation d'une demande.	`function onAbortWorkflowInstance(workflowModule, workflowInstance, result)` `{` `}`
onRemindWorkflowInstance	Cette méthode est appelée sur la relance d'une demande.	`function onRemindWorkflowInstance( workflowModule, workflowInstance, fulfiller, result)` `{` `}`
onRemoveWorkflowInstance	Cette méthode est appelée à chaque fois qu'un document est supprimé.	`function onRemoveWorkflowInstance( storageKey )` `{` `}`
onStartRemovingWorkflowInstance	Cette méthode est appelée au moment ou la suppression est sur le point de démarrer. L'instance de workflow est encore présente à l'appel de cette fonction.	`function onStartRemovingWorkflowInstance( workflowModule, workflowInstance, fulfiller )` `{` `}`

Exemple d'appel à une classe Java depuis une ouverture JavaScript

Ce code instancie et appelle une classe Java pour exécuter un traitement.

```
function onCreateTaskInstance(workflowModule, taskInstance)
{
  var object = new Packages.com.axemble.education.samples.AssignRole();
  object.onCreateTaskInstanceHook(workflowModule, taskInstance);
}
```

Extrait de code de la classe AssignRole

Ce code récupère l'ensemble des membres d'un rôle A et les affecte au champ rôle A de l'instance courante.

```
public void onCreateTaskInstanceHook( IWorkflowModule workflowModule, ITaskInstance taskInstance )
{
  try
  {
    if ( "S1".equals( taskInstance.getTask().getName() ) )
    {
      IWorkflowInstance workflowInstance = taskInstance.getWorkflowInstance();

      IRole roleA = workflowInstance.getCatalog().getRole( "RoleA" );

      workflowInstance.setValue( "RoleA", ( (Role)roleA.getNativeObject() ).getAllUsers() );
    }
  }
```

```
  catch( Exception e )
  {
    throw CastException.toRuntimeException( e );
  }
}
```

Chapitre 12 : Les agents

VDoc prévoit un système permettant de planifier l'exécution de tâches. Ce sont les classes « d'Agent ». Ces classes d'agent sont des classes Java très simples qui, depuis l'administration, peuvent être planifiées.

Pour créer un agent, il suffit de créer une classe qui étend BaseAgent (la classe RunnableAgent est rendue obsolète). En étendant la classe BaseAgent, il est nécessaire d'implémenter la méthode execute().

La classe « BaseAgent »

La classe **com.axemble.vdoc.sdk.agent.base.BaseAgent** est la classe de base de tout agent au sein du système VDoc.

En dérivant de cette classe de base, votre classe sera automatiquement reconnue en tant qu'agent et pourra être planifiée ou manuellement exécutée via l'interface d'administration.

Cette classe d'agent donne un accès direct aux APIs. Elle permet également d'inscrire des messages d'information ou d'erreur dans le rapport d'exécution de l'agent.

Méthodes de la classe BaseAgent

```
public abstract class BaseAgent extends ...
{
     // method to implement
     protected abstract void execute();

     // helper methods
     final protected IWorkflowModule getWorkflowModule()
     final protected void printInfo( String message )
     final protected void printError( String message )
     final protected void printSeparator()
     final protected void printLine()
}
```

Extrait de code de la classe LogAgent

Le code suivant retrouve tous les documents d'une application, et inscrit leur référence et titre dans le rapport d'exécution de l'agent.

```
public class LogAgent extends BaseAgent
{
  public void execute()
  {
    IContext processContext = getWorkflowModule().getContextByLogin( "sysadmin" );

    ICatalog catalog = getWorkflowModule().getCatalog( processContext, "Education" );

    // récupérer une référence externe (il faut la déclarer !)
    IJdbcReference jdbcReference =
      getWorkflowModule().getJdbcReference( processContext, "vdocDatabase" );

    // récupérer un contrôleur pour effectuer des recherches
    ISearchController searchController = getWorkflowModule().getSearchController( jdbcReference );

    // récupérer l'ensemble des documents
    this.printSeparator();
    this.printInfo( "Récupération de l'ensemble des documents de la base spécifiée..." );
    this.printSeparator();

    this.printLine();

    Collection worflowInstances = searchController.findElements(
      IWorkflowInstance.class,
      "select id from vdp treatments where ref catalog=?",
      new Object[] { catalog.getId().toString() } );
```

```
    for ( Iterator iter = worflowInstances.iterator() ; iter.hasNext() ; )
    {
      IWorkflowInstance workflowInstance = (IWorkflowInstance)iter.next();

      this.printInfo( "REFERENCE   : " + (String)workflowInstance.getValue( "sys_Reference" ) );
      this.printInfo( "TITRE       : " + (String)workflowInstance.getValue( "sys_Title" ) );
      this.printInfo( "IDENTIFIANT : " + workflowInstance.getId().toString() );

      this.printLine();
    }

    this.printInfo( "Opération terminée avec succès" );
    this.printSeparator();
  }
}
```

Gestion des transactions

Les agents s'exécutent, par défaut, dans un contexte non-transactionnel. Si vous souhaitez exécuter des traitements au sein d'une transaction, vous devez utiliser les méthodes beginTransaction, commitTransaction et rollbackTransaction du module IWorkflowModule.

Extrait de code utilisant les transactions

Le code suivant montre la mise en place d'une transaction au sein du module de Workflow.

```
try
{
    // démarrage d'une transaction
    getWorkflowModule().beginTransaction( this );

    ...

    // validation de la transaction
    getWorkflowModule().commitTransaction( this );
}
catch( Exception e )
{
    // en cas d'exception, annuler le traitement
    getWorkflowModule().rollbackTransaction( this );
}
```

Piloter les agents par programmation

Depuis les APIs VDoc il est possible de déclencher le traitement d'un agent declaré dans la Gestion des processus.

Exemple d'exécution d'un agent par API

Cet extrait de code permet de montrer comment récupérer un agent par son nom et l'exécuter.

```
// récupération d'un agent par son nom
IAgent agent = getWorkflowModule().getAgent( getWorkflowModule().getLoggedOnUserContext(),
  "Nom de l'agent" );

// exécution de l'agent récupéré
getWorkflowModule().execute( agent );
```

Chapitre 13 : Les systèmes d'interrogation

Système basé sur les vues

VDoc propose un système d'interrogation basé sur le moteur d'évaluation des vues.

L'objectif de ce mode d'interrogation du serveur est de permettre des développements spécifiques éloignés des détails d'implémentation des produits VDoc (ex. schéma de la base de données).

Ce système d'interrogation introduit une classe **com.axemble.vdoc.sdk.interfaces.IViewController**.

L'interface IViewController

L'interface IViewController permet de construire simplement des requêtes SQL basées sur des éléments fonctionnels.

Les méthodes de l'interface IViewController

```
public interface IViewController extends IController
{
    // constraints
    void addEqualsConstraint( String name, Object value );
    void addNotEqualsConstraint( String name, Object value );
    void addGreaterConstraint( String name, Object value );
    void addGreaterOrEqualConstraint( String name, Object value );
    void addLessConstraint( String name, Object value );
    void addLessOrEqualsConstraint( String name, Object value );
    void addLikeConstraint( String name, Object value );
    void addNotLikeConstraint( String name, Object value );
    void addInConstraint( String name, Object value );
    void addNotInConstraint( String name, Object value );

    // parameters
    void setStart( int start );
    void setCount( int count );
    void setOrderBy( String orderBy, Class orderByType, boolean ascendant );
    void setStatus( int status );

    // evaluation
    Collection evaluate();
    Collection evaluate( ICatalog catalog );
    Collection evaluate( IWorkflowContainer workflowContainer );
    Collection evaluate( IWorkflow workflow );
    Collection evaluate( ITask task );

    // size evaluation
    int evaluateSize();
    int evaluateSize( ICatalog catalog );
    int evaluateSize( IWorkflowContainer workflowContainer );
    int evaluateSize( IWorkflow workflow );
    int evaluateSize( ITask task );

    // set secured mode
    void setSecurityApplied( boolean securityApplied );

    // allow to search within the system catalogs
    void setAllowSystemCatalogs( boolean allowSystemCatalogs );
}
```

Construction d'une requête « fonctionnelle »

Exemple de code utilisant la classe IViewController pour construire et exécuter une requête.

```
// récupération d'une application en tant que scope d'interrogation
ICatalog catalog = workflowModule.getCatalog( context, "Education" );

IViewController viewController = workflowModule.getViewController();
```

```
viewController.setOrderBy( "sys Reference", String.class, true );

// positionner certaines contraintes.
viewController.addEqualsConstraint( "sys_Creator", context.getUser() );
viewController.addLessConstraint( "sys_CreationDate", new Date());
viewController.addEqualsConstraint( "fldDocumentVersion", "2.0" );
viewController.addEqualsConstraint( "DocumentState", "Applicable" );

// évaluation de la vue et récupération des documents processus.
Collection instances = viewController.evaluate(catalog);
for ( Iterator iterator = instances.iterator() ; iterator.hasNext() ; )
{
    IWorkflowInstance workflowInstance = (IWorkflowInstance)iterator.next();
    LOGGER.info( workflowInstance.getName() + " : " +
      workflowInstance.getValue( "DocumentState" ) );
}
```

Classe d'extension de vue

Les classes d'extension sur les vues permettent de personnaliser à la fois, les vues générées, et les vues développées en étendant la classe BaseViewProvider et implémentant l'interface ICollectionViewModelProvider.

Ces classes d'extension de vue peuvent être déclarées soit dans l'administration des vues générées, soit directement dans la description XML d'un écran de type vue.

Exemple de déclaration d'une classe d'extension de vue générée

```
<definition catalog="EDUCATION" catalogType="0"
extensionClasses="com.vdocsoftware.education.extensions.SimpleViewExtension" type="treatment">
  <filters>
[…]
```

Pour développer une classe d'extension de vue, il suffit de créer une classe qui étende BaseViewExtension. Le nom complet de la classe est : **com.axemble.vdoc.sdk.view.extensions.BaseViewExtension**.

Méthode de la classe de base BaseViewExtension

```
public abstract class BaseViewExtension implements Serializable
{
  // initialisation
  public void init()

  // accès à l'élément graphique et au model de vue
  protected final CtlAbstractView getView()
  protected final ICollectionViewModel getModel()

  // accès aux modules
  final public IWorkflowModule getWorkflowModule()
  final protected ISiteModule getSiteModule()

  // événements liés aux vues générées
  public boolean onPrepareView( Definition viewDefinition )
  public String onPrepareSQL( String sqlQuery )

  // événements liés à la construction de la vue
  public void onPrepareColumns( List viewModelColumns )
  public abstract void onPrepareItem( ViewItem item );

  public boolean onReady()
}
```

Exemple d'implémentation d'une extension de vue

Cet exemple montre comment ajouter des colonnes de différentes natures à la vue d'origine, et comment alimenter les éléments de la vue en travaillant sur les modèles de vue. Ce type de développement peut être mis en place aussi bien pour les vues générées que pour les vues développées (basées sur des « provider » de vue).

Il montre aussi comment utiliser les zones d'affichage au travers des modèles. En effet, pour chaque colonne ajoutée, il est possible de spécifier une zone d'affichage qui sera interprétée par le framework de Navigation en fonction de l'environnement d'exécution utilisé (Easysite, Portal ou autonome).

Les valeurs correspondant aux zones d'affichage sont proposées dans le framework de Navigation via la classe ViewModelColumn :

- ZONE_PROPERTIES
- ZONE_EXTENDED_PROPERTIES
- ZONE_ADDITIONAL_PROPERTIES
- ZONE_TITLE
- ZONE_DATE
- ZONE_ACTOR
- ZONE_ICON
- ZONE_DESCRIPTION
- ZONE_THUMBNAIL
- ZONE_CONTENT
- ZONE_ATTACHMENTS
- ZONE_PATH

Tableau récapitulatif des modes d'affichage en fonction de la zone spécifiée

Zone / Mode	Lien	Liste	Tableau	Miniatures
Titre				
Icône				
Description				
Miniature				
Date				
Acteur				
Documents attachés				
Chemin				
Propriétés principales				
Propriétés secondaires				
Propriétés annexes				

Legende

Affiché
Tooltip
Non affiché
Libellé du lien
A gauche du lien

```
public class SimpleViewExtension extends BaseViewExtension
{
 public void onPrepareColumns( List viewModelColumns )
 {
  // construction de colonnes
  ViewModelColumn propertyColumn = new ViewModelColumn( "actions", "LG_ACTIONS" );
  propertyColumn.setZone( ViewModelColumn.ZONE_TITLE );
  viewModelColumns.add( propertyColumn );

  propertyColumn = new ViewModelColumn( "text1", "LG TEXT1" );
  propertyColumn.setZone( ViewModelColumn.ZONE TITLE );
  viewModelColumns.add( propertyColumn );

  propertyColumn = new ViewModelColumn( "text2", "LG_TEXT2" );
  propertyColumn.setZone( ViewModelColumn.ZONE TITLE );
  viewModelColumns.add( propertyColumn );

  propertyColumn = new ViewModelColumn( "text3", "LG_TEXT2" );
  propertyColumn.setZone( ViewModelColumn.ZONE_TITLE );
  viewModelColumns.add( propertyColumn );

  super.onPrepareColumns( viewModelColumns );
 }
 public void onPrepareItem( ViewItem item )
 {
  final ViewModelItem viewModelItem = item.getViewModelItem();

  CtlButton addButton = new CtlButton( "add", new CtlText( "Ajouter au panier" ) );
  addButton.addActionListener( new ActionListener()
  {
   public void onClick( ActionEvent event )
   {
    try
    {
     getSiteModule().getMessageController().
       sendMessage( "basket", BasketHandler.IEventTypes.ADD,
         ( (IStorageKey)viewModelItem.getKey() ).toString() );
    }
    catch( SiteModuleException e )
    {
     Navigator.getNavigator().processErrors( e, true );
    }
   };
  } );

  // positionnement des widgets pour chaque colonne
  viewModelItem.setValue( "actions", addButton );
  viewModelItem.setValue( "text1", "Bonjour" );
  viewModelItem.setValue( "text2", "Salut" );
  viewModelItem.setValue( "text3", new CtlButton( "add", new CtlText( "Autre" ) ) );
 }
}
```

Notez dans l'extrait de code précédent la possibilité d'ajouter des éléments graphiques directement en tant que valeur d'une propriété d'un objet ViewModelItem.

Si plusieurs éléments graphiques doivent être ajoutés sur une même cellule, il faut passer par un objet intermédiaire de type ***java.util.LinkedList****.*

Pour le cas des vues générées (modélisées avec le web Designer), il est possible de modifier la définition de la vue d'origine ainsi que les contraintes d'ordre SQL (mode avancé).

Exemple de modification de la définition d'origine d'une vue

Dans cet exemple, une nouvelle contrainte est créée. Elle contraint le champ « fldUser » à la valeur du protocolURI de l'utilisateur connecté.

```
public boolean onPrepareView( Definition viewDefinition )
{
 try
 {
  // récupération des contraintes actuelles
  Fieldgroup fieldGroup = (Fieldgroup)viewDefinition.getFilters().getFieldgroup();

  ObjectFactory objFactory = new ObjectFactory();

  // création d'une nouvelle contraintes
  Field fieldNo = (Field)objFactory.createField();
  fieldNo.setName( "fldUser" );
  fieldNo.setValue( getWorkflowModule().getLoggedOnUser().getProtocolURI() );

  // ajout de la contrainte
  fieldGroup.getFieldgroupOrField().add( fieldNo );
 }
 catch( Exception e )
 {
  Navigator.getNavigator().processErrors( e, true );
  return false;
 }
 return super.onPrepareView( viewDefinition );
}
```

Système basé sur les flux XML

VDoc propose un nouveau système d'interrogation du serveur basé sur les classes « transformer ».

L'API XML permet, tout d'abord, la récupération des éléments suivants :

- mes tâches à traiter ;
- mes documents ;
- mes applications ;
- mes processus ;
- et mes versions de processus.

Elle permet également de réaliser certains traitements sur les documents :

- exécution et résultat d'une recherche indexée ;
- création, altération de documents processus ;
- et changement d'étape.

Les classes « transformer » disponibles

Le tableau suivant rassemble l'ensemble des « transformer » actuellement disponibles permettant de générer des flux XML.

Classe	Description
ICatalogViewTransformer	Récupération de l'ensemble des applications du serveur.
IWorkflowContainerViewTransformer	Récupération de l'ensemble des processus toutes applications confondues.
IWorkflowViewTransformer	Récupération de l'ensemble des versions de processus tous processus confondus.
ITaskViewTransformer	Récupération des tâches à traiter, des tâches en retard, des tâches déléguées de l'utilisateur

	connecté.
IResourceViewTransformer	Récupération des documents actifs, brouillons, terminés, annulés, ou, dont l'utilisateur connecté est intervenu.
ISearchViewTransformer	Récupération d'un résultat de recherche.
IDirectoryTransformer	Récupération des informations d'un utilisateur.
ITransformer	Transformation de document processus.

Utilisation des flux XML

Le mode d'interrogation du serveur VDoc est toujours le même. Il suffit de créer une commande XML et d'utiliser l'API java.net pour l'envoyer sur le serveur.

Dans tous les cas, le serveur répond un flux XML. La réponse peut soit contenir le résultat de l'exécution de la commande soit l'erreur éventuelle produite.

Commande XML

Exemple d'une commande d'export des propriétés d'une instance de workflow.

```
<?xml version="1.0" encoding="UTF-8"?>
<export-workflow-instance xmlns:d1="http://www.axemble.com/process">
    <header login="__CONTEXT_LOGIN__" /> <!-- à remplacer -->
    <body>
        <resource class="com.axemble.vdoc.sdk.interfaces.IWorkflowInstance">
            <header id="__ID__"> <!-- à remplacer -->
                <column name="" + propertyName + ""/>
            </header>
        </resource>
    </body>
</export-workflow-instance>
```

Exemple de code

Exemple de code permettant d'utiliser la commande ci-dessus pour récupérer une propriété particulière de l'instance de workflow (document processus).

Cet exemple présente l'utilisation de la classe **com.axemble.vdoc.sdk.utils.HttpUtils** permettant de réaliser l'envoi de la commande XML et de renvoyer un resultat sous la forme d'un élément DOM W3C.

Dans la plupart des cas, l'URL d'appel est serverURL + /navigation/sdk ?Controller=xml sauf pour l'import et l'export des documents où l'attribut 'Controller' doit être affecté, respectivement à 'import' et 'export'.

```
public String getProperty( String serverURL, String documentId, String propertyName )
{
    String serverUrl = serverURL + "/navigation/sdk?Controller=xml";

    // création de la commande XML en chaîne de caractères
    String stringCommand = createCommand( documentId, propertyName );

    Element rootElement = HttpUtils.sendXml( new URL( serverUrl ), stringCommand );
    if ( "response".equals( rootElement.getNodeName() ) )
        return rootElement.getAttribute( "message" );
    else
    {
        Element propertyElement = XMLUtils.selectSingleElement( rootElement,
          "body/string[@name='" + propertyName + "']" );

        return propertyElement.getAttribute( "value" );
    }
}
```

Les commandes sur les documents

Mes documents

Permet de récupérer les documents initiés par l'utilisateur placé dans l'attribut « login »

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="ACTIVE_DOCUMENTS" login="__CONTEXT_LOGIN__">
           <column name="sys Reference"/>
           <column name="sys Title"/>
           <column name="sys_Creator"/>
           <column name="sys_CreationDate"/>
           <column name="sys_CurrentSteps"/>
           <column name="DocumentState"/>
    </header>
</view>
```

Exécuter une recherche indexée

Permet d'exécuter une recheche basée sur le moteur d'indexation. Le résultat renvoit l'ensemble des documents que l'utilisateur placé dans l'attribut « login » peut voir.

```
<view xmlns:vw1="http://www.axemble.com/process/search">
    <header name="SEARCH" login="  CONTEXT LOGIN  " text="  TEXT  ">
           <column name="sys_Reference"/>
           <column name="sys_Title"/>
           <column name="sys_Creator"/>
           <column name="sys CreationDate"/>
           <column name="sys CurrentSteps"/>
           <column name="DocumentState"/>
    </header>
</view>
```

Export d'un document

Permet d'exporter un document du système de gestion des processus.

```
<export xmlns:d1="http://www.axemble.com/process/document">
  <resource class="com.axemble.vdoc.sdk.interfaces.IWorkflowInstance">
    <header id="jee5ta0j1zh1xwqu11fu" reference="DM-07-11-1.0-0011">
    <resource-definition name="documentManagement 1.0" />
    </header>
  </resource>
</export>
```

Import d'un document

Permet d'importer un document dans le système de gestion des processus. Il est possible de préciser les attributs à renseigner, les étapes à traverser, ainsi que les événements personnalisés de l'historique.

Exemple de commande d'import

Cet exemple montre comment importer une demande de congés dans la gestion de processus.

```
<resource xmlns:d1="http://www.axemble.com/process/document">
    <header reference="CONGES-2006-07-0001" creator="sysadmin"
      created-date="2006-07-08T11:30:00.000Z" modified-date="2006-07-08T11:30:00.000Z">
           <resource-definition id="" name="DemandeDeConges_1.0"/>
    </header>
    <history>
           <history-entry customized="false" name="Envoyer" label="Envoyer"
             source="SAISIE DE LA DEMANDE" date="2006-07-08T11:30:00.000Z" doneBy="Demandeur"
             description="" fulfiller="sysadmin" addressee="sysadmin"/>
    </history>
    <body>
           <date name="sys CreationDate" label="Date de création"
             value="2006-07-08T11:30:00.000Z"/>
           <user name="sys_Creator" label="Créateur" id="-21qajfa1k4mes5eon44kc9"
             external-id="387028092977152" login="sysadmin" first-name="System"
             last-name="Administrator" email=""/>
           <string name="sys Title" label="Titre" value="doc1"/>
           <string name="sys Reference" label="Référence" value="CONGES-2006-07-0001"/>
           <string name="Commentaires" label="Commentaires" value="DEMO"/>
           <period name="Periode" label="Période">
                  <date value="2006-07-08T11:00:00.000Z"/>
                  <date value="2006-07-08T23:00:00.000Z"/>
           </period>
           <float name="NombreDeJours" label="Nombre de jours" value="1"/>
           <string name="TypeDeConges" label="Type de congés" value="Congés Payés"/>
           <string name="Service" label="Service" value="Recherche et développement"/>
           <collection name="Drh" label="DRH"/>
           <collection name="ChefDeService" label="Chef de Service"/>
           <collection name="Demandeur" label="Demandeur"/>
           <string name="ProcessState" label="Etat du processus" value="Initialisé"/>
           <string name="DocumentState" label="Etat du document" value="Vérification"/>
    </body>
</resource>
```

Certains attributs ont été rendu optionnels pour simplifier la syntaxe du flux XML à produire.

Exemple de commande d'import simplifiée

```
<resource xmlns:d1="http://www.axemble.com/process/document">
    <header reference="CONGES-2006-08-0001"
           <resource-definition name="DemandeDeConges_1.0"/>
    </header>
    <history>
           <history-entry name="Envoyer"/>
    </history>
    <body>
           <date name="sys_CreationDate" value="2006-07-08T11:30:00.000Z"/>
           <string name="sys_Title" value="doc1"/>
           <string name="Commentaires" value="DEMO"/>
           <period name="Periode">
                  <date value="2006-07-08T11:00:00.000Z"/>
                  <date value="2006-07-08T23:00:00.000Z"/>
           </period>
           <float name="NombreDeJours" value="1"/>
           <string name="TypeDeConges" value="Congés Payés"/>
           <string name="Service" value="Recherche et développement"/>
           <collection name="Drh"/>
           <collection name="ChefDeService"/>
           <collection name="Demandeur"/>
           <string name="ProcessState" value="Initialisé"/>
           <string name="DocumentState" value="Vérification"/>
    </body>
</resource>
```

Les commandes sur les tâches

Mes tâches à traiter

Permet de récupérer les tâches à traiter de l'utilisateur placé dans l'attribut « login »

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="ASSIGNED_TASKS" login="__CONTEXT_LOGIN__">
           <column name="sys Reference"/>
           <column name="sys Title"/>
           <column name="sys_Creator"/>
           <column name="sys_CreationDate"/>
           <column name="sys_CurrentSteps"/>
           <column name="DocumentState"/>
    </header>
</view>
```

Mes tâches en retard

Permet de récupérer les tâches en retard de l'utilisateur placé dans l'attribut « login »

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="DELAYED_TASKS" login="__CONTEXT_LOGIN__">
           <column name="sys Reference"/>
           <column name="sys Title"/>
           <column name="sys Creator"/>
           <column name="sys CreationDate"/>
           <column name="sys_CurrentSteps"/>
           <column name="DocumentState"/>
    </header>
</view>
```

Les commandes sur les éléments de définition

Récupérer l'ensemble des applications

Permet de récupérer les applications que l'utilisateur placé dans l'attribut « login » peut voir.

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="CATALOGS" login="  CONTEXT LOGIN  ">
    </header>
</view>
```

Récupérer l'ensemble des processus

Permet de récupérer les processus que l'utilisateur placé dans l'attribut « login » peut voir.

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="WORKFLOW CONTAINERS" login="  CONTEXT LOGIN  ">
    </header>
</view>
```

Récupérer l'ensemble des versions de processus

Permet de récupérer les versions de processus que l'utilisateur placé dans l'attribut « login » peut voir.

```
<view xmlns:vw1="http://www.axemble.com/process/view">
    <header name="WORKFLOWS" login="__CONTEXT_LOGIN__">
    </header>
</view>
```

Les commandes sur l'annuaire

Récupérer un utilisateur

Permet de récupérer les informations relatives à un utilisateur placé dans l'attribut « login » du nœud « header ».

```
<export-user xmlns:d1="http://www.axemble.com/directory">
    <header login="__CONTEXT_LOGIN__" />
    <body>
           <user class="com.axemble.vdoc.sdk.interfaces.IUser">
                  <header login="sboirol" />
           </user>
    </body>
</export-user>
```

Les commandes sur les données du Data Universe

Le système d'interrogation basé sur les flux XML a été revu pour permettre l'intégration d'autres éléments que les documents processus ou les tâches. Les données du Data Universe peuvent être manipulées de la même façon.

Récupérer les données d'une table

Cet exemple permet de récupérer depuis la base « SDK » les données de la table « PRODUCTS ». Seules les colonnes « Name » et « Description » seront remontées.

L'attribut « catalogType » indique la nature des éléments interrogés :

- Document processus et tâches (valeur : 0) ;
- Données des formulaires de site (valeur : 2) ;
- Données des formulaires partagées de site (valeur : 3) ;
- Données du Data Universe (valeur : 4).

```
<view xmlns:vw1="http://www.axemble.com/vdoc/view">
 <header login="sysadmin">
  <include name="labels" value="false"/>
  <definition catalog="SDK" catalogType="4" class="com.axemble.vdoc.sdk.interfaces.IResource">
   <filters>
    <fieldgroup operator="AND">
     <field name="sys_ResourceDefinition" value="PRODUCTS" />
    </fieldgroup>
   </filters>
   <view sortAscending="true" sortBy="Name">
    <column name="Name" />
    <column name="Description "/>
   </view>
  </definition>
 </header>
</view>
```

Cet exemple utilise le moteur des vues. L'URL d'appel d'une telle vue est la suivante :

```
/navigation/sdk ?Controller=view.
```

Voici un autre exemple qui utilise le système d'interrogation des vues et qui permet de ramener les tâches à traiter de l'ulisateur spécifié dans l'attribut « login » de la balise « header » :

```
<view xmlns:vw1="http://www.axemble.com/vdoc/view">
 <header login="sysadmin">
  <definition catalog="Education">
   <filters>
    <fieldgroup operator="AND">
     <field name="$activity.fulfiller" operator="equals" value="$User" />
     <field name="dyn_Status" operator="equals" value="STARTED" />
```

```
    </fieldgroup>
   </filters>
   <view sortAscending="true" sortBy="sys Title">
    <column name="sys Title" />
   </view>
  </definition>
 </header>
</view>
```

Récupérer d'une donnée d'une table

Cet exemple permet de récupérer une donnée particulière par son identifiant.

```
<export xmlns:d1="http://www.axemble.com/vdoc/resource">
 <header login="sysadmin">
  <include name="labels" value="false"/>
  <include name="linkedResources" value="false"/>
  <include name="linkedWorkflowInstances" value="false"/>
  <include name="attachmentsContent" value="false"/>
 </header>
 <body>
  <resource class="com.axemble.vdoc.sdk.interfaces.IStorageResource">
  <header id="-27mm34m5qjeve9uq06c2kn">
   <column name="Name" />
   <column name="Description" />
  </header>
  </resource>
 </body>
</export>
```

L'URL d'appel d'une telle commande est la suivante :

```
/navigation/sdk ?Controller=xml.
```

Mettre à jour une donnée d'une table

Cet exemple permet de modifier une donnée particulière par son identifiant.

```
<update xmlns:d1="http://www.axemble.com/vdoc/resource">
 <header login="sysadmin" />
 <body>
  <resource class="com.axemble.vdoc.sdk.interfaces.IStorageResource">
    <header id="1pd35jxgkvnnjv2xpz6x07" />
    <history />
    <body>
      <string name="Name" value="VDoc Easysite (version 2)" />
      <string name="Description" value="Gestion de contenu de la gamme VDoc" />
    </body>
  </resource>
 </body>
</update>
```

L'URL d'appel d'une telle commande est la suivante :

```
/navigation/sdk ?Controller=xml.
```

Créer une donnée dans une table

Cet exemple permet de monter coment créer une donnée dans une table du Data Universe.

```
<import xmlns:d1="http://www.axemble.com/vdoc/resource">
 <header login="sysadmin" />
 <body>
  <resource xmlns:d1="http://www.axemble.com/process"
    class="com.axemble.vdoc.sdk.interfaces.IResource">
   <header id="1pd35jxgkvnnjv2xpz6x07" reference="" creator="sysadmin"
     created-date="2009-08-24T17:32:37.577Z" modified-date="2009-08-24T17:32:37.687Z">
    <resource-definition id="1pd35jxgkvnnjv2xpz6wx2" name="PRODUCTS" />
    <catalog id="1pd35jxgkvnnjv2xpz6ww6" name="SDK" type="4" />
   </header>
   <history />
   <body>
    <string name="Name" value="VDoc Document Management" />
    <string name="Description" value="Gestion documentaire" />
    <date name="DateDeCreation" value="2009-09-03T10:00:00.000Z" display-value="03/09/2009" />
    <float name="NombreDUtilisateurs" value="1000" />
    <period name="PeriodeDeDisponibilite">
     <date value="2009-09-03T10:00:00.000Z" display-value="03/09/2009" />
     <date value="2009-09-03T10:00:00.000Z" display-value="03/09/2009" />
    </period>
    <long name="Anonymous" value="1" />
    <collection name="Versions">
     <string name="Versions" value="2007" />
    </collection>
   </body>
  </resource>
 </body>
</import>
```

L'URL d'appel d'une telle commande est la suivante :

```
/navigation/sdk ?Controller=xml.
```

Système des contrôleurs

Le système d'interrogation par flux XML proposé par VDoc utilise en interne la notion de « contrôleur ». VDoc Software vous propose ce mécanisme pour réaliser des traitements en mode authentifié.

Les contrôleurs s'exécutent dans un contexte VDoc. Il est possible de récupérer, par exemple, des informations relatives à l'utilisateur connecté ou toute autre information des objets de l'API.

Implémentation d'une classe contrôleur

Pour développer un contrôleur, il suffit de créer une classe Java qui étend la classe de base **com.axemble.vdoc.sdk.controllers.BaseController**.

Chaque contrôleur est appelé par le Framework de Navigation sur les deux méthodes suivantes :

- **parseRequest** : méthode recevant en paramètre l'objet IRequest (représentant la requête HTTP) ;
- **doProcess** : méthode permettant d'exécuter un traitement et de fournir une réponse. Cette deuxième reçoit en paramètre l'objet IExecutionContext qui tient toutes les informations relatives au contexte d'exécution de la requête.

Exemple de code d'implémentation d'un contrôleur

Dans cet exemple sont présentés l'accès au module de processus (WorkflowModule) et la récupération du contexte de l'utilisateur connecté.

```
public class SimpleController extends BaseController
{
    public void parseRequest( IRequest request ) throws IOException
    {}
    public IExecutionContext doProcess( IExecutionContext ec ) throws IOException
    {
            // retrieve the logged on user context
            IContext context = getWorkflowModule().getLoggedOnUserContext();

            return super.doProcess( ec );
    }
}
```

Chapitre 14 : La délégation

Piloter la délégation

La délégation peut entièrement être pilotée par les APIs proposées dans le Kit de développement VDoc. Une nouvelle classe « Controller » a été mise en œuvre pour manipuler la délégation. Cette classe se nomme **com.axemble.vdoc.sdk.interfaces.IDelegationController**.

Depuis cette classe il est possible de déléguer des éléments dits de définition ou dynamiques. Toutes les interfaces implémentant l'interface **com.axemble.vdoc.sdk.supports.IDelegationSupport** sont susceptibles d'être concernées par le module de délégation.

Méthodes de l'interface IDelegationController

L'extrait de code suivant présente l'ensemble des méthodes disponibles sur l'interface IDelegationController.

```
public interface IDelegationController extends IController
{
    // délégation générale
    void addDelegation( IUser user, IUser substituteUser );
    void removeDelegation( IUser user );

    // délégation sur les éléments de définition
    void addDelegation( IUser user, IUser substituteUser, IDelegationSupport delegationSupport );
    void addDelegation( IUser user, IUser substituteUser, IDelegationSupport delegationSupport,
      String comment );
    void removeDelegation( IUser user, IDelegationSupport delegationSupport );
    void breakDelegation( IUser user, IDelegationSupport delegationSupport );

    // délégation sur les éléments dynamiques
    void addDelegation( IUser user, IUser substituteUser, IWorkflowInstance workflowInstance,
      ITask task );
    void removeDelegation( IUser user, IWorkflowInstance workflowInstance, ITask task );
    void breakDelegation( IUser user, IWorkflowInstance workflowInstance, ITask task );

    // délégation définitive (remplacement)
    void delegate( IUser user, IUser substituteUser, ITaskInstance taskInstance );
    IReport delegateResources( ICatalog catalog, Collection roles, Collection resources, IUser
      user, IUser substituteUser );
    IReport delegateRoles( ICatalog catalog, Collection roles, Collection resources, IUser user,
      IUser substituteUser );
}
```

Délégation des éléments de définition

La délégation peut être appliquée sur des éléments de définition suivants :

- ICatalog : les applications ;
- IWorkflowContainer : les processus ;
- Version de processus : les versions de processus ;

Exemple de délégation

Cet exemple présente la méthode **breakDelegation**() qui permet de gérer des cas d'exception sur une délégation.

Dans cet exemple, tous les documents de l'utilisateur connecté sont délégués à l'utilisateur « sboirol » sauf les documents issus de l'application nommée « adminitrative ».

```
// récupération de l'utilisateur connecté
IUser user = workflowModule.getLoggedOnUser();

// récupération de son contexte d'exécution
IContext context = workflowModule.getContext( user );

// récupération d'un autre utilisateur à qui déléguer les documents
IUser sboirol = workflowModule.getUserByLogin( "sboirol" );

// récupération du contrôleur de délégation
IDelegationController delegationController = workflowModule.getDelegationController();

// déléguer tous les éléments à l'utilisateur 'sboirol'
delegationController.addDelegation( user, sboirol );

// créer une exception de délégation pour l'application nommée 'administrative'
ICatalog catalog = workflowModule.getCatalog( context, "administrative" );
delegationController.breakDelegation( user, catalog );
```

Délégation des éléments dynamiques

La délégation peut également être appliquée sur des éléments dynamiques tels que les documents unitaires et les tâches à traiter.

Intervention sur le document processus

Le Kit de développement VDoc propose, au travers de la classe d'extension du document processus, des événements sur la délégation.

En déclarant une classe d'extension type documentaire il est possible de réagir sur les événements suivants :

Evénement	Description
onBeforeDelegate	Permet de controller l'apparition de l'assistant de délégation de document processus.
onAfterDelegate	Permet de réaliser du code une fois la délégation du document effectuée.
onBeforeDelegateTaskOnly	Permet de controller l'apparition de l'assistant de délégation d'une tâche.
onAfterDelegateTaskOnly	Permet de réaliser du code une fois la délégation de la tâche effectuée.
onBeforeRefuseDelegation	Permet de controller l'apparition de l'assistant de refus de la délégation.
onAfterRefuseDelegation	Permet de réaliser du code une fois le refus effectué.
onBeforeCancelDelegation	Permet de controller l'apparition de l'assistant d'annulation de la délégation.
onAfterCancelDelegation	Permet de réaliser du code une fois l'annulation effectuée.

Chapitre 15 : Référence de l'annuaire

L'API SDK intègre un nouveau module qui permet aux développeurs de manipuler, directement depuis les extensions Process (ou toute autre point d'accès au sein du Portail), les éléments de l'annuaire et les relations définies entre eux.
La communication entre Process et l'annuaire est transparente via le SDK.

Cette couche simplifie la réalisation de tâches administratives telles que :

- la création d'utilisateurs, de groupes, d'organisations ;
- l'association entre les groupes et les utilisateurs ;
- la mise en place de sécurité sur les éléments créés.

Cette API s'intègre pleinement avec celle du module de Workflow.

Quelques définitions

Diagramme de classes

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes du système d'annuaire.

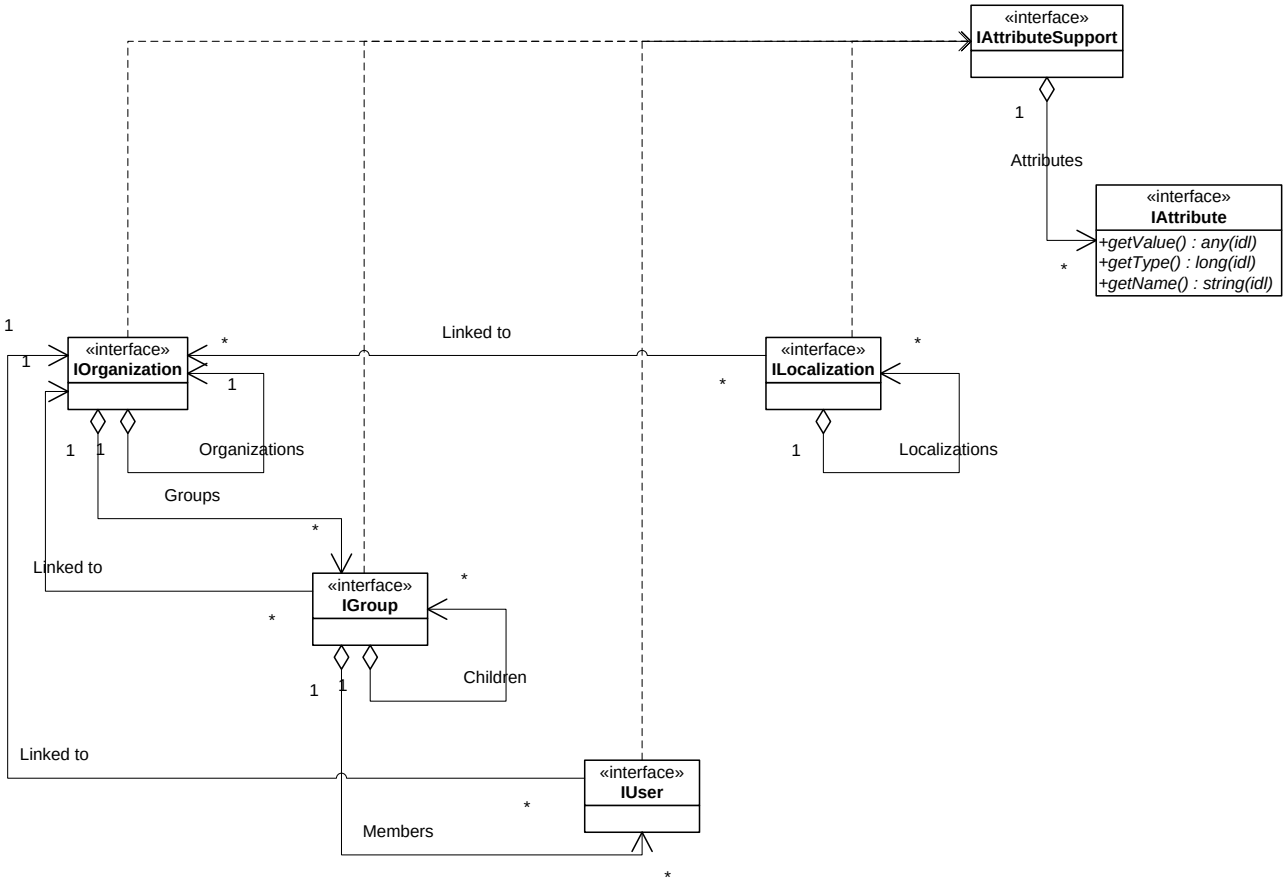


Tableau de correspondance des éléments

Terme fonctionnel	Terme technique	Terme SDK	Terme JDO	Description
Organisation / Structure	Organisation	IOrganization	Organization	Organisation / Structure
Groupe	Groupe	IGroup	Group	Groupe d'individus
Utilisateur	Utilisateur	IUser	User	Utilisateur du portail
Localisation / Structure	Localisation	ILocalization	Localization	Lieu ou situation géographique. Exemples : immeuble, bâtiment, étage, bureau.
Attributs étendus	Attributs étendus	IAttribute	Attribute	Attribut étendus de l'annuaire

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc sur le module d'annuaire.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module d'annuaire des API SDK Récupérer un contexte pour utiliser les API SDK Utiliser les transactions Requêter le système
Serveur	Récupérer les organisations Récupérer une organisation Créer une organisation Récupérer les localisations Récupérer une localisation Créer une localisation
Organisation	Créer un utilisateur Créer un groupe Ajouter un utilisateur ou un groupe dans un groupe
Element	Récupération d'un attribut étendu Création et ajout d'un attribut étendu Recherche d'éléments par attribut
Sécurité	Vérification des permissions Ajout d'une permission Suppression de permission

Tâches générales

Utiliser le module d'annuaire des API SDK

Le code suivant indique comment utiliser l'API directory. La méthode initialize() est utilisée pour positionner un certain nombre de paramètres. Dans la plupart des cas, cette méthode prendra en argument la valeur null, car au sein du système VDoc toutes les propriétés nécessaires sont positionnées.

```
public void general useModule()
{
     // création d'un object de module de Directory
     IDirectoryModule directoryModule = new PortalDirectoryModule();
     try
     {
            // initialisation du module
            directoryModule.initialize( null );
     }
     catch( Exception e )
     {
            // si une erreur s'est produite
            getNavigator().processErrors( e );
     }
     finally
     {
            directoryModule.unInitialize();
     }
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general_getContext( IDirectoryModule directoryModule ) throws Exception
{
    // 1er cas : récupération du contexte à partir d'un objet utilisateur
    IUser user = directoryModule.getUserByLogin( "sysadmin" );
    IContext userContext = directoryModule.getContext( user );

    // 2ème cas : récupération du contexte à partir d'un ID externe
    IContext externalIDContext = directoryModule.getContext(
           getNavigator().getLoggedOnUser().getExternId() );

    // 3ème cas : récupération du contexte à partir d'un login
    IContext loginContext = directoryModule.getContextByLogin( "sysadmin" );
}
```

Utiliser des transactions

Dans de nombreux cas, il est nécessaire de rassembler plusieurs traitements dans une même transaction. Chaque module possède son propre mécanisme transactionnel.

L'exemple suivant montre comment utiliser ce mécanisme transactionnel depuis le module directory.

```
public void general useTransaction( IContext context, IDirectoryModule directoryModule )
{
    try
    {
           // démarrage d'une transaction
           directoryModule.beginTransaction( this );

           // réaliser quelque chose
           // ...

           // validation de la transaction
           directoryModule.commitTransaction( this );
    }
    catch( Exception e )
    {
           // si une erreur s'est produite, annuler le traitement précédent
           directoryModule.rollbackTransaction( this );
    }
}
```

Requêter le système

Le module d'annuaire offre un système d'interrogation basé sur JDO. Dans l'exemple suivant nous utilisons un contrôler de type recherche (SearchController) pour interroger la base de données et remonter des objets IUser.

```
public void general_useSearch( IDirectoryModule directoryModule ) throws DirectoryModuleException
{
    String filter = "login==pLogin && (status==1 || status==99 || status==-1 || status==2)";

    ISearchController searchController = directoryModule.getSearchController();
    Collection users = searchController.findElements(
      IUser.class, filter, "pLogin", new Object[] { "sysadmin" }, null );

    for ( Iterator iteratorUser = users.iterator() ; iteratorUser.hasNext() ; )
    {
           IUser user = (IUser)iteratorUser.next();
           LOGGER.error( "User '" + user.getLogin() + "' found!" );
    }
}
```

Tâches serveur

Récupérer les organisations

Depuis le module d'annuaire, il est possible de récupérer toutes les organisations racine.

```
public void server getOrganizations( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
{
    // récupération de toutes les organisations
    Collection organizations = directoryModule.getOrganizations( context );
    for ( Iterator iter = organizations.iterator() ; iter.hasNext() ; )
    {
           IOrganization organization = (IOrganization)iter.next();
           System.out.println(organization.getName());
    }
}
```

Récupérer une organisation

Depuis le module d'annuaire, il est possible de récupérer une organisation racine par son nom.

```
public void server getOrganization( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
{
    // récupération d'une organisation
    IOrganization organization = directoryModule.getOrganization( context, "test-organization" );
    System.out.println(organization.getName());

    // récupération d'une filiale
    IOrganization childOrganization =
           directoryModule.getOrganization( context, organization, "test-child-organization" );
    System.out.println(childOrganization.getName());
}
```

Créer une organisation

Depuis le module d'annuaire, il est possible de créer de nouvelles organisations. Dans l'exemple suivant deux organisations sont créées.

```
public void server_createOrganization( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
{
  IOrganization organization = directoryModule.createOrganization(context, "test-organization");
  organization.setSkin( "skin-name" );

  IOrganization childOrganization =
    directoryModule.createOrganization(context, organization, "test-child-organization");
}
```

Récupérer les localisations

Depuis le module d'annuaire, il est possible de récupérer toutes les localisations racine.

```
public void server getLocalizations( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
{
    // récupération de toutes les localisations
    Collection localizations = directoryModule.getLocalizations( context );
    for ( Iterator iter = localizations.iterator() ; iter.hasNext() ; )
    {
           ILocalization localization = (ILocalization)iter.next();
           System.out.println(localization.getName());
    }
}
```

Récupérer une localisation

Depuis le module d'annuaire, il est possible de récupérer une localisation racine par son nom.

```
public void server_getLocalization( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
```

```
{
    // récupération d'une localisation
    ILocalization localization = directoryModule.getLocalization( context, "test-localization" );
    System.out.println(localization.getName());

    // récupération d'une localisation fille
    ILocalization childLocalization =
           directoryModule.getLocalization( context, localization, "test-child-localization" );
    System.out.println(childLocalization.getName());
}
```

Créer une localisation

Depuis le module d'annuaire, il est possible de créer de nouvelles localisations. Dans l'exemple suivant deux localisations sont créées.

```
public void server_createLocalization( IDirectoryModule directoryModule, IContext context )
  throws DirectoryModuleException
{
    // création d'une localisation
    ILocalization localization = directoryModule.createLocalization( context, "test-localization"
);

    // création d'une localisation fille
    ILocalization childLocalization =
           directoryModule.createLocalization(context, localization, "test-child-localization");
}
```

Tâches sur l'organisation

Créer un utilisateur

Dans l'exemple suivant un utilisateur est créé. Le code vérifie tout d'abord l'existence de l'utilisateur dans l'annuaire VDoc.

```
public void organization createUser(
  IDirectoryModule directoryModule,
  IContext context,
  IOrganization organization ) throws DirectoryModuleException
{
    // vérification de l'existence d'un utilisateur
    IUser user = directoryModule.getUserByLogin( "jbon" );
    if ( user == null )
    {
           // création de l'utilisateur rattaché à une organisation
           user = directoryModule.createUser( context, "jbon", "pwd", organization );
           user.setFirstName( "Jean" );
           user.setLastName( "Bon" );
           user.setEmail( "jbon@charcuterie.com" );
    }
}
```

Créer un groupe

Dans l'exemple suivant un groupe est créé. Le code utilise la fonction getGroup pour vérifier l'existence du groupe avant création.

```
public void organization createGroup( IDirectoryModule directoryModule, IContext context,
IOrganization organization ) throws DirectoryModuleException
{
    // vérification de l'existence du groupe
    IGroup managers = directoryModule.getGroup( context, organization, "Managers" );
    if ( managers == null )
           // création du groupe au sein de l'organisation
           managers = directoryModule.createGroup( context, organization, "Managers" );
}
```

Ajouter un utilisateur ou un groupe dans un groupe

Le code suivant montre les possibilités d'association entre les utilisateurs et les groupes. Il montre également comment vérifier l'appartenance d'un utilisateur ou un groupe à un autre groupe.

```
public void organization_addGroupAndMember(
  IDirectoryModule directoryModule,
  IContext context,
  IOrganization organization ) throws DirectoryModuleException
{
    // récupération d'un groupe
    IGroup group = directoryModule.getGroup( context, organization, "Managers" );

    // récuperation d'un utilisateur
    IUser user = directoryModule.getUserByLogin( "jbon" );

    // vérification de l'appartenance d'un utilisateur à un groupe
    if ( !user.isMemberOf( group, false ) )
           // ajout de l'utilisateur dans les membres du groupe
           group.addMember( user );

    // récupération d'un groupe dans l'annuaire
    IGroup nextGroup = directoryModule.getGroup( context, organization, "Managers" );

    // vérification de l'appartenance d'un groupe à un autre groupe
    if ( !nextGroup.isChildOf(  group, false ) )
           // ajout de l'utilisateur dans les fils du groupe
           group.addChild( nextGroup );
}
```

Tâches sur l'élément

Récupération d'un attribut

Le code suivant fournit un exemple de récupération d'un attribut à partir d'un utilisateur.

```
public void element getAttribute( IDirectoryModule directoryModule, IUser user )
{
    // récupération d'un attribut d'un élément de l'annuaire (ex. user)
    IAttribute attribute = user.getAttribute( "attr-1" );

    // retrouver le nom
    attribute.getName();

    // retrouver le type
    int type = attribute.getType();
    switch ( type )
    {
           case IAttribute.IType.STRING :
                  break;
           case IAttribute.IType.INTEGER :
                  break;
           case IAttribute.IType.FLOAT :
                  break;
           case IAttribute.IType.DATE :
                  break;
           case IAttribute.IType.UNKNOWN :
                  break;
    }

    // retrouver la valeur
    attribute.getValue();
}
```

Création et ajout d'un attribut

Le code suivant montre comment créer et associer un attribut étendu à un utilisateur.

```
public void element_addAttribute( IDirectoryModule directoryModule, IUser user )
{
```

```
    String attributeKey = "attr-1";
    try
    {
            // démarrage d'un transaction
            directoryModule.beginTransaction( this );

            // création d'un atribut
            IAttribute attribute = directoryModule.
              createAttribute( attributeKey, IAttribute.IType.STRING, "value-1" );

            // association de l'attribut à l'élément de l'annuaire
            user.addAttribute( attribute );

            // finalisation de la transaction
            directoryModule.commitTransaction( this );
    }
    catch( Exception e )
    {
            // annulation du traitement en cas d'erreur
            directoryModule.rollbackTransaction( this );
    }
}
```

Recherche d'éléments par attribut

Le code suivant utilise le SearchController pour effectuer une recherche de tous les utilisateurs qui ont un attribut étendu « attr-1 ».

```
public void element_useSearch( IDirectoryModule directoryModule ) throws DirectoryModuleException
{
    String attributeKey = "attr-1";

    ISearchController searchController = directoryModule.getSearchController();

    Collection users = searchController.
      findElements(
            IUser.class,
            "attributes.contains(vAttribute) && vAttribute.key == pKey",
            "vAttribute,
            pKey",
            new Object[] { "var:" + Attribute.class.getName(), attributeKey }, null );

    for ( Iterator iter = users.iterator() ; iter.hasNext() ; )
    {
            IUser aUser = (IUser)iter.next();
            System.err.println( aUser.getLogin() );
    }
    System.err.println( "Thanks to Franck Salque!" );
}
```

Tâches sur la sécurité

Vérification des permissions

Pour vérifier les droits sur un objet, il suffit de récupérer le contrôleur de sécurité (SecurityController), puis de tester le retour de la méthode checkPermission en fonction d'un utilisateur, d'un groupe, d'un rôle ou d'un joker (EVERYONE).

```
public void security checkRights( IDirectoryModule directoryModule, IOrganization entity, IGroup
managers ) throws DirectoryModuleException
{
    // Ajustement des droits
    ISecurityController entitySecurity = directoryModule.getSecurityController( entity );

    // Le groupe 'Managers' a le droit de gestion des organisations de l'entité créée
    if ( entitySecurity.checkPermission( managers, new Object[]
    { null, "read,write" } ) )
            System.out.println( "yes" );
    else System.out.println( "no" );
}
```

Ajout d'une permission

Depuis le module d'annuaire il est possible d'ajouter des permissions. L'exemple suivant montre comment donner le droit de modification d'une organisation « entity » à un groupe « Managers ».

Il montre également comment poser les droits sur les éléments gérés par cette organisation.

```
public void security_AddRights( IDirectoryModule directoryModule, IOrganization entity, IGroup
managers ) throws DirectoryModuleException
{
    // Ajustement des droits
    ISecurityController entitySecurity = directoryModule.getSecurityController( entity );

    // Le groupe 'Managers' a le droit de gestion des organisations de l'entité créée
    entitySecurity.addPermission( managers, new Object[]
    { null, "read,write" } );
    // Le groupe 'Managers' a le droit de gestion des groupes de l'entité créée
    entitySecurity.addPermission( managers, new Object[]
    { directoryModule.findNativeClass( IGroup.class ), "read,write" } );
}
```

Suppression de permission

En utilisant le contrôleur de sécurité il est possible de retirer des droits d'un utilisateur, d'un groupe sur un objet. L'exemple suivant montre comment enlever les droits de modifications sur l'objet « entity » au groupe « users ».

```
public void security removeRights( IDirectoryModule directoryModule, IOrganization entity, IGroup
users ) throws DirectoryModuleException
{
    // Ajustement des droits
    ISecurityController entitySecurity = directoryModule.getSecurityController( entity );

    // Le groupe 'Users' ne devrait plus avoir le droit de gestion des organisations de l'entité
    entitySecurity.removePermission( users, new Object[]
    { null, "read,write" } );
}
```

Chapitre 16 : Référence de la publication

L'API SDK intègre un module de publication documents qui permet de manipuler, directement depuis les extensions Process (ou toute autre système au sein du Portail), les éléments de FileCenter et les relations définies entre eux.
La communication entre Process et Filecenter est transparente via le SDK.

Cette couche simplifie la réalisation de tâches administratives telles que :

- la création de bibliothèques, de dossiers, de documents ;
- l'association entre les dossiers et les documents ;
- l'association entre les éléments créés et les catégories
- la mise en place de sécurité sur les éléments créés.

Cette API s'intègre pleinement avec celle du module de Workflow et de l'annuaire.

Quelques définitions

Diagramme de classes

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes du système de publication.

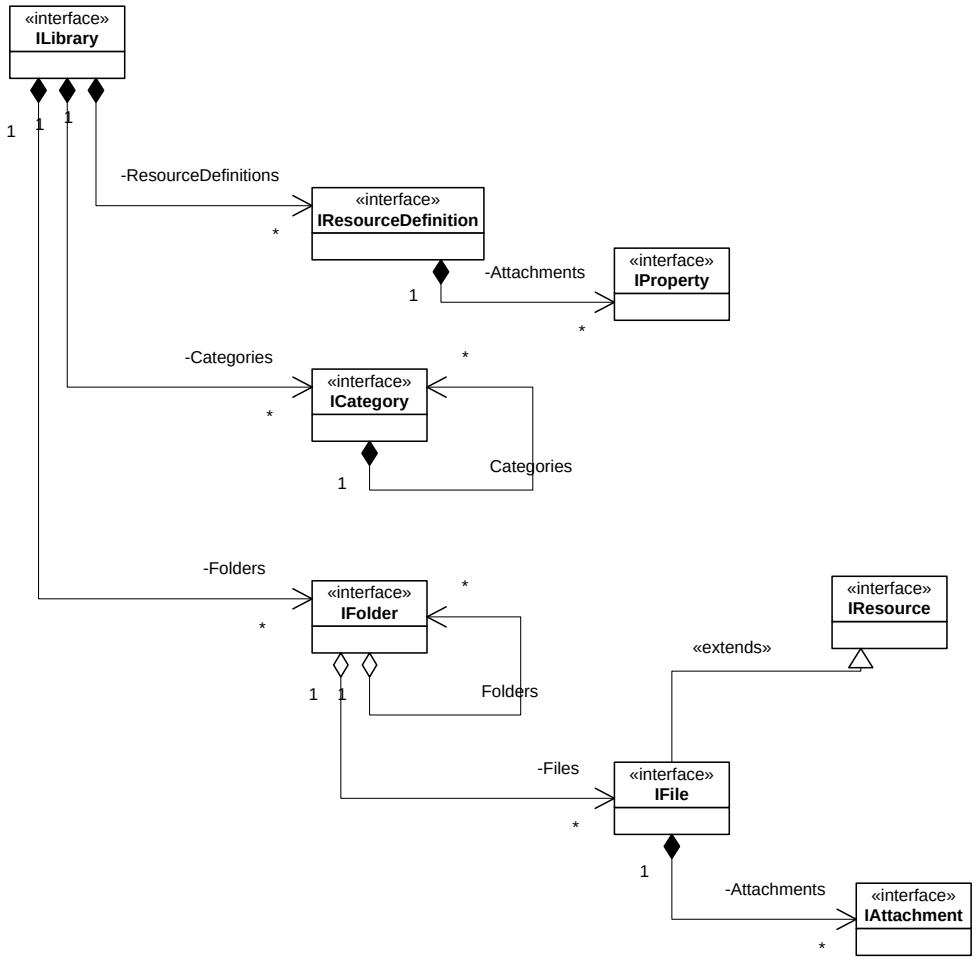


Tableau de correspondance des éléments

Terme fonctionnel	Terme technique	Terme SDK	Terme JDO	Description
Bibliothèque de documents	ContentStore	ILibrary	DataStore	Espace présent sur un disque permettant de stocker l'ensemble des documents publiés dans FileCenter. Plusieurs bibliothèques peuvent exister sur un même serveur.
Type de document	Type de document	IResourceDefinition	DocumentDefinition	Elément de définition d'un document FileCenter. Il décrit, via ses propriétés, ce que le document pourra tenir comme valeurs dans sa fiche documentaire.

Propriétés, Attributs de document	Propriétés	IProperty	PropertyDefinition	Elément de définition d'une propriété (type, valeur par défaut, liste de valeurs)
Catégorie	Catégorie	ICategory	CategoryNode	Elément récursif relié à des dossiers ou des documents.
Dossier, Classement	Dossier	IFolder	FolderNode	Container de documents FileCenter. Elément contrôlé par la sécurité.
Document	Fichier	IFile	DocumentNode	Document comprenant la fiche documentaire et le ou les fichiers joints.
Fichier joint	Fichier joint	IAttachment	ContentNode	Le fichier joint des documents.

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc sur le module de publication.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module de publication des API SDK Récupérer un contexte pour utiliser les API SDK Utiliser les transactions Requêter le système
Serveur	Récupérer les bibliothèques Récupérer une bibliothèque Créer une bibliothèque
Bibliothèque	Récupérer les catégories Récupérer des sous catégories Créer des catégories Créer un dossier Créer un sous-dossier Récupérer les définitions Récupérer une définition Récupérer les propriétés
Fichier	Créer un fichier Récupérer un fichier Récupérer le contenu

Tâches générales

Utiliser le module d'annuaire des API SDK

Le code suivant indique comment utiliser l'API de publication. La méthode initialize() est utilisée pour positionner un certain nombre de paramètres. Dans la plupart des cas, cette méthode prendra en argument la valeur null, car au sein du système VDoc toutes les propriétés nécessaires sont positionnées.

```
public void general_useModule()
{
     // création d'un object de module de publication
```

```
    ILibraryModule libraryModule = new PortalLibraryModule();
    try
    {
            // initialisation du module
            libraryModule.initialize( null );
    }
    catch( Exception e )
    {
            // si une erreur s'est produite
            getNavigator().processErrors( e );
    }
    finally
    {
            libraryModule.unInitialize();
    }
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general_getContext( ILibraryModule libraryModule ) throws Exception
{
    // 1er cas : récupération du contexte à partir d'un objet utilisateur
    IUser user = libraryModule.getUserByLogin( "sysadmin" );
    IContext userContext = libraryModule.getContext( user );

    // 2ème cas : récupération du contexte à partir d'un ID externe
    IContext externalIDContext = libraryModule.getContext( libraryModule.getLoggedOnUser() );

    // 3ème cas : récupération du contexte à partir d'un login
    IContext loginContext = libraryModule.getContextByLogin( "sysadmin" );
}
```

Utiliser des transactions

Dans de nombreux cas, il est nécessaire de rassembler plusieurs traitements dans une même transaction. Chaque module possède son propre mécanisme transactionnel.

L'exemple suivant montre comment utiliser ce mécanisme transactionnel depuis le module directory.

```
public void general useTransaction( IContext context, ILibraryModule libraryModule )
{
    try
    {
            // démarrage d'une transaction
            libraryModule.beginTransaction( this );

            // réaliser quelque chose
            // ...

            // validation de la transaction
            libraryModule.commitTransaction( this );
    }
    catch( Exception e )
    {
            // si une erreur s'est produite, annuler le traitement précédent
            libraryModule.rollbackTransaction( this );
    }
}
```

Requêter le système

Le module de publication offre un système d'interrogation basé sur JDO. Dans l'exemple suivant nous utilisons un contrôler de type recherche (SearchController) pour interroger la base de données et remonter des objets IFile.

```
public void general useSearch( ILibraryModule libraryModule ) throws LibraryModuleException
{
    try
    {
            ISearchController searchController = libraryModule.getSearchController();
```

```
            Collection files = searchController.findElements(IFile.class,"(status == 0)",null);
            for ( Iterator iter = files.iterator() ; iter.hasNext() ; )
            {
                    IFile iFile = (IFile)iter.next();
                    System.out.println( iFile.getName() );
            }
    }
    catch( Exception e )
    {
            getNavigator().processErrors( e, true );
    }
}
```

Tâches serveur

Récupérer les bibliothèques

Depuis le module de publication, il est possible de récupérer toutes les bibliothèques présentes sur le serveur.

```
public void server getLibraries( ILibraryModule libraryModule, IContext context ) throws
LibraryModuleException
{
    // récupération de toutes les bibliothèques
    Collection libraries = libraryModule.getLibraries( context );
    for ( Iterator iter = libraries.iterator() ; iter.hasNext() ; )
    {
            ILibrary library = (ILibrary)iter.next();
            System.out.println( library.getName() );
    }
}
```

Récupérer une bibliothèque

Depuis le module publication, il est possible de récupérer une bibliothèque par son nom.

```
public void server getLibrary( ILibraryModule libraryModule, IContext context ) throws
LibraryModuleException
{
    // récupération d'une bibliothèque
    ILibrary library = libraryModule.getLibrary( context, "test-library" );
    System.out.println( library.getName() );
}
```

Créer une bibliothèque

Depuis le module publication, il est possible de créer de nouvelles bibliothèques.

```
public void server_createLibrary( ILibraryModule libraryModule, IContext context ) throws
LibraryModuleException
{
    ILibrary library = libraryModule.createLibrary( context, "test-library" );

    // positionner quelques propriétés
    library.setForbiddenExtensions( "bat, js" );
}
```

Tâches sur la bibliothèque

Récupérer les catégories

Cet exemple montre comment récupérer les catégories d'une bibliothèque.

```
public void server_getCategories( ILibraryModule libraryModule, IContext context, ILibrary library
) throws LibraryModuleException
{
    // récupération des catégories placées à la racine de la bibliothèque
    Collection categories = libraryModule.getCategories( context, library, false );

    for ( Iterator iterator = categories.iterator() ; iterator.hasNext() ; )
    {
        ICategory category = (ICategory)iterator.next();
        System.out.println( category.getName() );
    }
}
```

Récupérer des sous catégories

Cet exemple montre comment récupérer des sous catégories.

```
public void server_getSubCategories( ILibraryModule libraryModule, IContext context, ICategory
category ) throws LibraryModuleException
{
    // récupération des catégories d'une catégorie parent
    Collection categories = libraryModule.getCategories( context, category, false );
}
```

Créer des catégories

Cet exemple montre comment créer des nouvelles catégories et sous-catégories.

```
public void server createCategory( ILibraryModule libraryModule, IContext context, ILibrary
library ) throws LibraryModuleException
{
    // création d'une catégorie
    ICategory category = libraryModule.createCategory( context, library, "test-category" );

    // création d'une sous-catégorie
    ICategory subCategory = libraryModule.createCategory( context, category, "test-sub-category"
);
}
```

Créer un dossier

Cet exemple montre comment créer des dossiers au niveau d'une bibliothèque.

```
public void library_createFolder( ILibraryModule libraryModule, IContext context, ILibrary library
) throws LibraryModuleException
{
    IFolder folder = libraryModule.createFolder( context, library, "test-folder" );
    folder.setDescription( "test-folder description" );
}
```

Créer un sous-dossier

Cet exemple montre comment créer des sous-dossiers.

```
public void library_createFolder( ILibraryModule libraryModule, IContext context, IFolder parent )
throws LibraryModuleException
{
    IFolder folder = libraryModule.createFolder( context, parent, "test-sub-folder" );
    folder.setDescription( "test-sub-folder description" );
}
```

Récupérer les définitions

Cet exemple permet à partir d'une bibliothèque de récupérer les définitions.

```
public void library getDefinitions( ILibraryModule libraryModule, IContext context, ILibrary
library ) throws LibraryModuleException
{
```

```
    // retrouver toutes les définitions
    Collection definitions = libraryModule.getResourceDefinitions( context, library );
    for ( Iterator iterator = definitions.iterator() ; iterator.hasNext() ; )
    {
            IResourceDefinition definition = (IResourceDefinition)iterator.next();
            System.out.println( definition.getLabel() + " [" + definition.getName() + "]" );
    }
}
```

Récupérer une définition

Cet exemple montre comment récupérer une définition

```
public void library_getDefinition( ILibraryModule libraryModule, IContext context, ILibrary
library ) throws LibraryModuleException
{
    // retrouver une définition
    IResourceDefinition definition = libraryModule.
      getResourceDefinition( context, library, "test-definition" );
    System.out.println( definition.getLabel() + " [" + definition.getName() + "]" );
}
```

Récupérer les propriétés

Cet exemple montre comment récupérer les propriétés d'une définition

```
public void library_getProperties( ILibraryModule libraryModule, IContext context,
IResourceDefinition definition ) throws LibraryModuleException
{
    // retrouver l'ensemble des propriétés
    for ( Iterator iterator = definition.getProperties().iterator() ; iterator.hasNext() ; )
    {
            IProperty property = (IProperty)iterator.next();
            System.out.println( property.getLabel() + " [" + property.getName() + "]" );
    }
}
```

Tâches sur les documents

Créer un document

Cet exemple montre comment créer un document, positionner sa défition, renseigner les propriétés et associer des catégories.

```
public void library_createFile( ILibraryModule libraryModule, IContext context, IFolder folder )
throws LibraryModuleException
{
    String contentAsBytes = "This is a dymmy content";
    IFile file = libraryModule.createFile( context, folder, "test-file", "test-file description",
contentAsBytes.getBytes() );

    IResourceDefinition definition = null; //...

    // associer le fichier à la définition
    file.setDefinition( definition );

    // positionner quelques propriétés
    file.setValue( "fldText", "dummy text" );
    file.setValue( "fldNumber", new Float(1) );
    file.setValue( "fldDate", new Date() );

    ICategory category = null; //...

    file.addCategory( category );
}
```

Récupérer un document

Exemple de récupération d'un document.

```
public void library_getFile( ILibraryModule libraryModule, IContext context, IFolder folder )
throws LibraryModuleException
{
    IFile file = libraryModule.getFile( context, folder, "test-file" );

    System.out.println(file.getValue( "fldText" ));
    System.out.println(file.getValue( "fldNumber" ));
    System.out.println(new SimpleDateFormat("yyyy-MM-dd").format( file.getValue( "fldDate" )));
}
```

Récupérer les pièces jointes d'un document

Exemple de récupération des pièces jointes d'un document.

```
public void library getAttachments( ILibraryModule libraryModule, IContext context, IFile file )
throws LibraryModuleException
{
    // retrouver toutes les pièces jointes d'un fichier
    Collection attachments = libraryModule.getAttachments( file );
    for ( Iterator iterator = attachments.iterator() ; iterator.hasNext() ; )
    {
            IAttachment attachment = (IAttachment)iterator.next();
            System.out.println(attachment.getShortName() + " : " + attachment.getPath());
    }

    // retrouver une pièce jointe particulière par son nom
    IAttachment attachment = libraryModule.getAttachment( file, "test-file" );
}
```

Récupérer le contenu d'une pièce jointe

Exemple de récupération du contenu d'une pièce jointe.

```
public void library getContent( ILibraryModule libraryModule, IContext context,
  IAttachment attachment ) throws LibraryModuleException
{
    try
    {
            // copier le contenu d'une pièce jointe sur un fichier du disque
            FileOutputStream outputStream = new FileOutputStream(new File("output.sdk"));
            attachment.getContent( outputStream );
    }
    catch( FileNotFoundException e )
    {
            // ...
    }
}
```

Chapitre 17 : Référence du workflow

Quelques définitions

Eléments de définition

A la génération d'un processus, plusieurs éléments sont implantés dans la base VDoc. Parmi ces éléments de définition, nous retiendrons les suivants :

- ICatalog : correspond à l'application ;
- IWorkflowContainer : constitue le processus. Il rassemble toutes les versions de processus ;
- IResourceDefinition : rassemble toutes les propriétés (champs sur les tâches et ceux du document) définies dans le processus via le web Designer ;
- IProperty : description d'un champ du processus ;
- IWorkflow : constitue la description du processus en termes de graphe. Rassemble les tâches, les actions et les rôles ;
- ITask : description d'une tâche manuelle ;
- IAction : description d'une action ;
- IRole : représentant du rôle.

Diagramme de classes des éléments de définition

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes descriptives du système de processus.

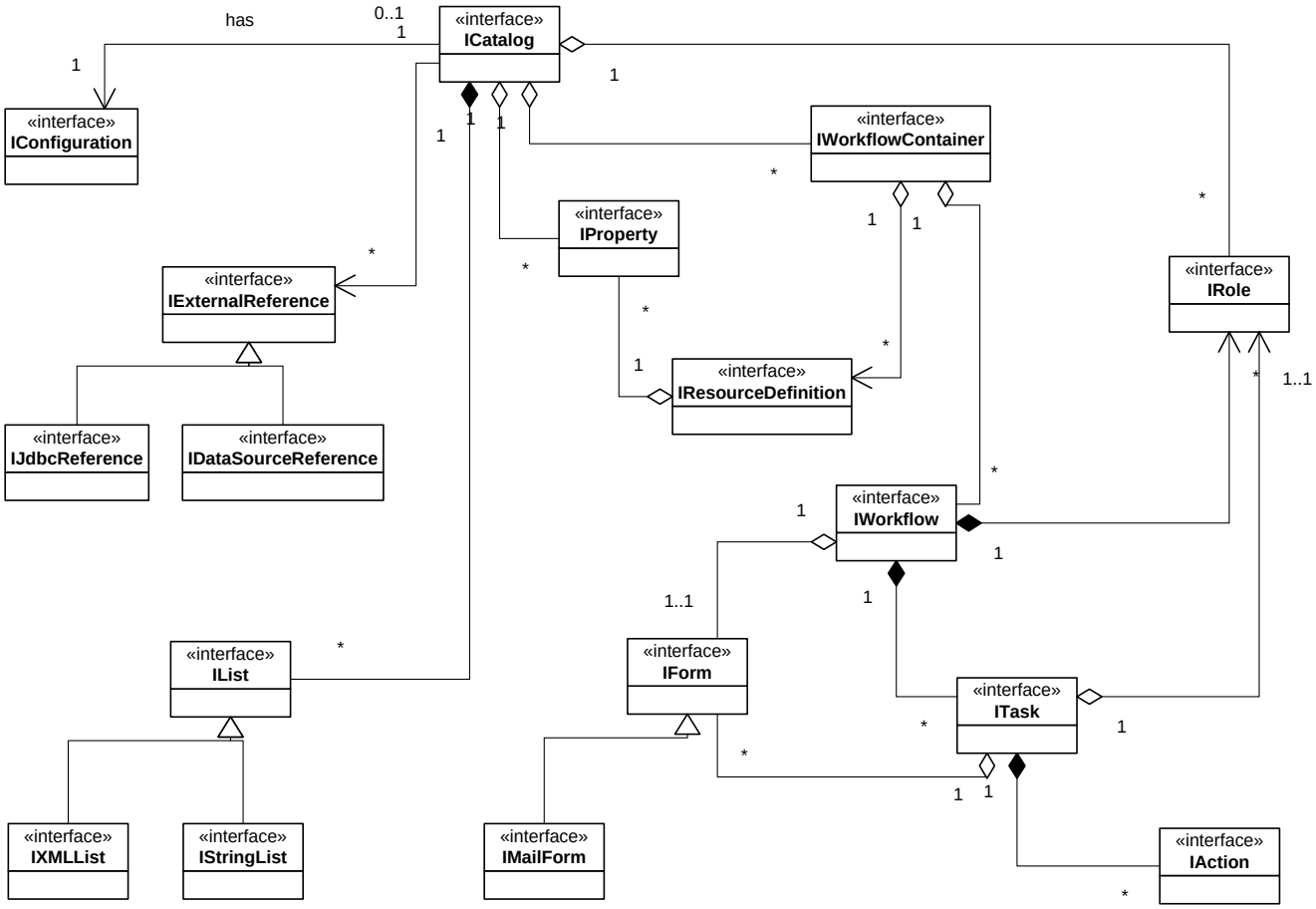


Tableau de correspondance des éléments de définition

Terme fonctionnel	Terme technique	Terme SDK	Terme EJB (préfixe : ejb/com/axemble/vdp/)	Description
Application	Catalogue	ICatalog	Catalog	Une application est un regroupement de processus et des documents associés.
Modèle de ressources	Modèle de ressources	IResourceDefinition	ResourceTemplate	
Champ, Propriété	Propriété	IProperty	Property	Elément constituant la description d'une valeur supportée par le document.
Processus	Groupe de version de processus	IWorkflowContainer	VersionGroup	Ensemble de versions de processus qui partagent des propriétés, des vues, des rôles.
Version de processus, workflow	Version de processus	IWorkflow	TreatmentClass	Un processus est la modélisation du cheminement d'un document à travers plusieurs étapes. Il peut également faire appel à des sous-processus.
Tâche, étape	Etape	ITask	Stage	Une tâche est l'objet modélisé identifiant une étape du processus.
Action de workflow, Action utilisateur, Action de routage	Action de routage	IAction	TransitionInlet	Action utilisateur permettant de sortir d'une étape.
Rôle	Rôle	IRole	Role	C'est un regroupement d'utilisateurs et de groupes. Il peut être associé à un ensemble de permissions ou droits communs pour une application, un processus, une version, une vue.

Eléments dynamiques

A la création d'un document, plusieurs éléments sont inscrits en base. Parmi ces éléments dynamiques, nous retiendrons les suivants :

- IResource : interface représentant les données du document et à partir de laquelle il est possible de modifier les valeurs de champs ;
- IWorkflowInstance : interface dérivant de l'interface IResource permettant de gérer le cycle de vie du document (changement d'état) ;
- ILinkedResource : interface représentant les données d'une ligne de tableau dynamique ;
- ITaskInstance : interface représentant une tâche active sur laquelle des opérateurs peuvent intervenir ;
- IResourceHistory : interface d'accès à l'historique d'un document ;
- IEvent : interface représentant un évènement dans l'historique ;
- IAttachment : interface permettant de manipuler des champs de type pièces jointes ;
- IOperator : interface représentant un utilisateur intervenant sur une étape.

Diagramme de classes des éléments dynamiques

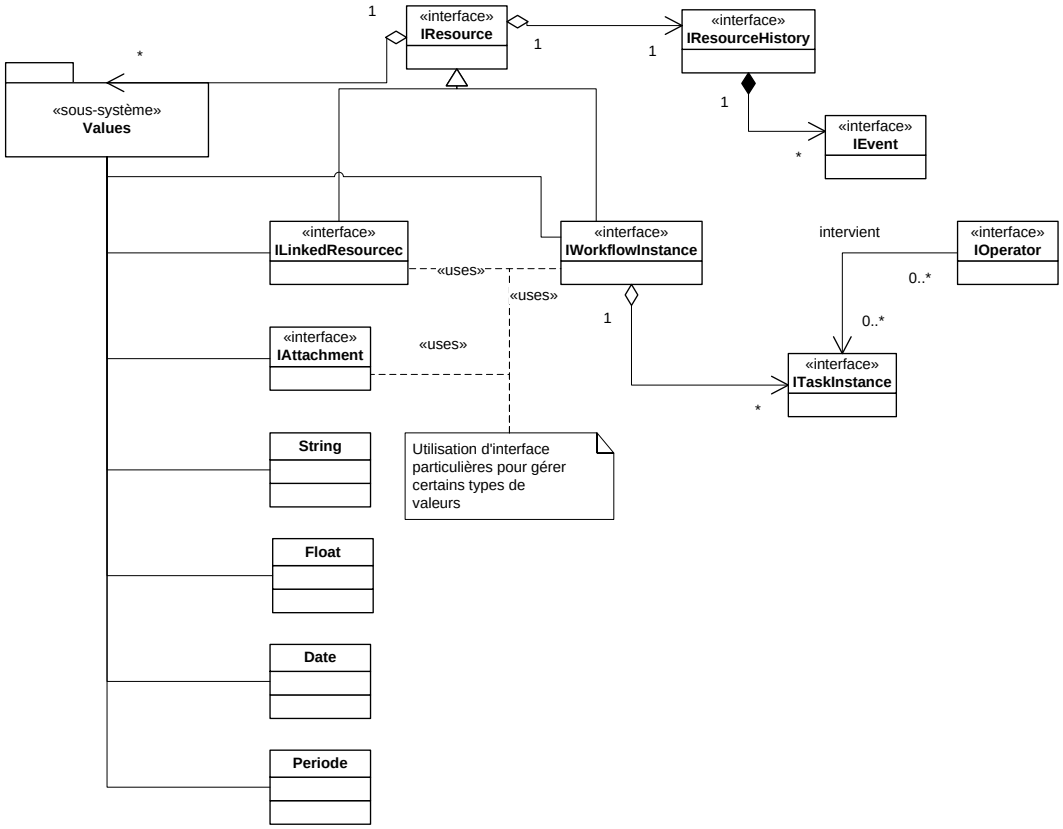


Tableau de correspondance des éléments dynamiques

Terme fonctionnel	Terme technique	Terme SDK	Terme EJB (préfix : ejb/com/axemble/vdp/)	Description
Ressource	Ressource	IResource	Resource	Le terme « Ressource » représente le conteneur de données. Elle permet de stocker l'ensemble des données utilisateur
Demande, Document	Traitement	IWorkflowInstance	Treatment	La notion de document a fortement évoluée dans VDoc. Elle définit une ressource et l'ensemble des traitements et sous-traitements associés.
Ligne de tableau	Ressource interne	ILinkedResource	Resource	Il s'agit de la ressource représentant une ligne d'un tableau dynamique type ressource.
Tâche à traiter	Activité, Activité manuelle	ITaskInstance	Activity, ManualActivity	Une activité représente une intervention à réaliser par une personne ou un programme. Une activité manuelle peut être de deux types : Action ou Décision
Historique	Historique	IResourceHistory	-	Historique d'un document.
Ligne d'historique	Evènement utilisateur	IEvent	-	Ligne d'historique du document correspondant à une action utilisateur.
Pièce jointe, fichier	Enrobeur de fichier	IAttachment	Structure FileInfo ou TempUploadFile	Description d'un fichier attaché dans le document via un champ pièces jointes.
Tâche à traiter	Activité utilisateur	IOperator	UsersActivity	Il s'agit de l'association entre une personne et une activité manuelle. Il y aura autant d'activités utilisateur créées que d'intervenants potentiels déterminés pour une tâche.
Demandes liées, documents liés, sous-processus	Traitement lié	IWorkflowInstance	Treatment	Il s'agit du traitement représentant une ligne d'un tableau dynamique type traitement.

API EJB vers API SDK

Pour basculer des API EJB vers les API SDK il suffit d'utiliser le module de workflow (IWorkflowModule). Cet objet possède un certain nombre de méthodes « getter » lui permettant de prendre en entrée diverses natures d'objets et de les convertir en objets API SDK.

Les objets de définition

```
// récupérer une application
public ICatalog getCatalog( Object object ) throws WorkflowModuleException;
```

```
// récupérer un modèle de ressources
public IResourceDefinition getResourceDefinition( Object object ) throws WorkflowModuleException;

// récupérer une propriété
public IProperty getProperty( Object object ) throws WorkflowModuleException;

// récupérer un processus
public IWorkflowContainer getWorkflowContainer( Object object ) throws WorkflowModuleException;

// récupérer une version de processus
public IWorkflow getWorkflow( Object object ) throws WorkflowModuleException;

// récupérer une tâche
public ITask getTask( Object object ) throws WorkflowModuleException;

// récupérer une action
public IAction getAction( Object object ) throws WorkflowModuleException;

// récupérer un rôle
public IRole getRole( Object object ) throws WorkflowModuleException;
```

Les objets dynamiques

```
// récupérer une ressource
public IResource getResource( Object object ) throws WorkflowModuleException;

// récupérer un document
public IWorkflowInstance getWorkflowInstance( Object object ) throws WorkflowModuleException;

// récupérer une tâche active
public ITaskInstance getTaskInstance( Object object ) throws WorkflowModuleException;

// récupérer un élément d'un champ pieces jointes
public IAttachment getAttachment( Object object ) throws WorkflowModuleException;

// récupérer l'utilisateur d'une activité utilisateur
public IOperator getOperator( Object object ) throws WorkflowModuleException;

// récupérer un utilisateur
public IUser getUser( Object object ) throws WorkflowModuleException;
```

Exemple de récupération d'un objet IWorkflowInstance à partir d'un objet Treatment

```
public IWorkflowInstance system_getWorkflowInstance( IWorkflowModule module, Treatment treatment )
  throws WorkflowModuleException
{
    return module.getWorkflowInstance( treatment );
}
```

API SDK vers API EJB

Pour basculer des API SDK vers les API EJB, il suffit d'utiliser chacun des objets SDK et de les appeler sur la méthode **getNativeObject()**. Celle-ci renvoie, selon le tableau présenté ci-dessus, l'objet natif correspondant.

Exemple de récupération d'un objet Treatment à partir d'une instance de workflow

```
public Treatment system_getTreatment( IWorkflowInstance workflowInstance )
  throws WorkflowModuleException
{
    return ((Treatment)workflowInstance.getNativeObject());
}
```

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc.

Pour des raisons de clarté, mais aussi pour marquer les différents niveaux d'API, nous avons décidé de classer ces développements en plusieurs sous-sections.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module workflow des API SDK Récupérer un contexte pour utiliser les API SDK Récupérer une référence externe Récupérer une DataSource Récupérer une connexion à partir d'une référence externe Récupérer une connexion à partir d'une DataSource Utiliser les transactions
Serveur	Récupérer une application
Application	Utiliser les configurations Récupérer les paramètres serveur Récupérer les rôles d'une application Récupérer les listes d'une application Récupérer une version de processus
Document	Créer un document Créer un document et personnaliser la référence Récupérer le prochain numéro de chrono de référence formatée Ajouter une ligne dans l'historique Ajouter le droit de lecture sur un document Positionner la valeur d'un sélecteur d'utilisateur simple Positionner la valeur d'un sélecteur d'utilisateurs multiples Utiliser une requête SQL pour remonter des documents Parcourir un tableau dynamique Ajouter une ligne à un tableau dynamique Ajouter des pièces jointes Récupérer les fichiers d'un champ pièces jointes Effectuer un changement d'étape Transformer un document au format XML
Personnalisation	Récupérer le Controller de ressource Rendre des champs du document processus obligatoires ou non Rendre des champs du document processus éditables ou non Masquer ou non des champs du document processus Masquer des éléments d'interface identifiés du document processus Masquer une action de workflow Ajouter un bouton sur un document processus
Script	Evaluer l'appartenance de l'utilisateur connecté à un rôle

Tâches générales

Utiliser le module workflow des API SDK

Le code suivant indique comment utiliser l'API workflow. La méthode initialize() est utilisée pour positionner un certain nombre de paramètres. Dans la plupart des cas, cette méthode prendra en argument la valeur null, car au sein du système VDoc toutes les propriétés nécessaires sont positionnées.

```
public void general useModule ()
{
     // création d'un object de module de workflow
     IWorkflowModule workflowModule = new ProcessWorkflowModule();
     try
     {
            // positionnement de certaines propriétés (optionel)
            Properties props = new Properties();
            props.setProperty( "language", "fr" );

            // initialisation du module
            workflowModule.initialize( props );
     }
     catch( Exception e )
     {
            // si une erreur s'est produite
            getNavigator().processErrors( e );
     }
     finally
     {
            // ne pas oublier de libérer le module utilisé
            workflowModule.unInitialize();
     }
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general_getContext( IWorkflowModule workflowModule ) throws Exception
{
     // 1er cas : récupération du contexte à partir d'un objet utilisateur
     IUser user = workflowModule.getUserByLogin( "sysadmin" );
     IContext userContext = workflowModule.getContext( user );

     // 2ème cas : récupération du contexte à partir d'un ID externe
     IContext extIDCtx= workflowModule.getContext(getNavigator().getLoggedOnUser().getExternId());

     // 3ème cas : récupération du contexte à partir d'un login
     IContext loginContext = workflowModule.getContextByLogin( "sysadmin" );
}
```

Récupérer une référence externe

Cet exemple montre comment récupérer l'objet Java correspondant à la référence externe déclarée dans l'administration.

```
public void general getExternalReference( IContext context, IWorkflowModule workflowModule )
  throws Exception
{
     // retrouver une référence externe avec son nom
     IJdbcReference jdbcReference = workflowModule.getJdbcReference( context, "vdocDatabase" );
     System.out.println( " référence : " + jdbcReference.getName() );
}
```

Récupérer une DataSource

Cet exemple montre comment récupérer une DataSource déclarée sur le serveur JNDI.

```
public void general_getDataSource( IWorkflowModule workflowModule ) throws Exception
{
  IDataSourceReference dataSourceReference=workflowModule.getDataSourceReference("education-ds");
}
```

Récupérer une connexion à partir d'une référence externe

Depuis un objet référence externe, il est possible de récupérer une connexion JDBC.

```
public void general getConnection( IJdbcReference jdbcReference )
{
    // récupérer une connexion
    Connection connection = jdbcReference.getConnection();
}
```

Récupérer une connexion à partir d'une DataSource

Depuis un objet data source, il est possible de récupérer une connexion JDBC.

```
public void general_getConnection( IDataSourceReference dataSourceReference )
{
    // récupérer une connection à partir d'une data source
    Connection connection = dataSourceReference.getConnection();
}
```

Utiliser des transactions

Dans de nombreux cas, il est nécessaire de rassembler plusieurs traitements dans une même transaction. Chaque module possède son propre mécanisme transactionnel.

L'exemple suivant montre comment utiliser ce mécanisme transactionnel depuis le module workflow.

```
public void general_useTransaction( IContext context, IWorkflowModule workflowModule )
{
    try
    {
            // démarrage d'une transaction
            workflowModule.beginTransaction( this );

            // réaliser quelque chose
            // ...

            // validation de la transaction
            workflowModule.commitTransaction( this );
    }
    catch( Exception e )
    {
            // si une erreur s'est produite, annuler le traitement précédent
            workflowModule.rollbackTransaction( this );
            getNavigator().processErrors( e );
    }
}
```

Tâches autour du serveur

Récupérer une application

Depuis le module de workflow, il est possible de récupérer toutes les applications créées. Dans l'exemple suivant, nous montrons comment en obtenir une à partir de son nom système.

```
public void server_getCatalog( IContext context, IWorkflowModule workflowModule )
  throws WorkflowModuleException
{
    // récupération d'une application à partir de son nom système
    ICatalog catalog = workflowModule.getCatalog( context, "Education" );
    System.out.println( "Application : " + catalog.getLabel() );
}
```

Tâches autour de l'application

Utiliser les configurations

VDoc permet aux intégrateurs la déclaration de paramètres pour le développement spécifique sur les niveaux serveur, organisation, ou application.

```
public void catalog useConfiguration( ICatalog catalog )
{
    // récupération de la configuration à partir d'une application
    IConfiguration configuration = catalog.getConfiguration();
}
```

Récupérer les paramètres serveur

A partir d'un objet configuration, il est possible de récupérer les paramètres standard en utilisant les méthodes getProperty() ou getProperties().

L'interface **com.axemble.vdp.configuration.interfaces.ConfigurationParameters** définit un certain nombre de mots-clés qui permettent de retrouver des informations serveur.

Liste des mots clés disponibles

Mot clé	Description
SMTP_SERVER	Adresse du serveur SMTP
SMTP_ENCODING	Encoding utilisé par le serveur SMTP
MAIL_BASE_URL	URL de base pour les liens contenus dans les messages électroniques.
MAX_FILE_SIZE	Taille maximum d'un fichier
SUPPORTED_LANGUAGES	Langues supportées
DEFAULT_LANGUAGE	Langue par défaut
ADMIN_EMAIL	Adresse de messagerie du compte administrateur
DEFAULT_EMAIL_SENDER	Aadresse de messagerie du compte expéditeur par défaut
SUPPORTED_FILE_EXTENSIONS	Extensions de fichier supportées pour le téléchargement
XLS_DATE_FORMAT	Format de la date pour les documents Excel
DEFAULT_MIN_DATE	Borne minimum des champs date
DEFAULT_MAX_DATE	Borne maximum des champs date

L'exemple suivant indique comment récupérer, depuis les paramètres serveur, l'adresse de messagerie de l'expéditeur par défaut.

```
public void catalog_getProperty( IConfiguration configuration )
{
    // récupération de l'expéditeur par défaut
    String emailSender = configuration.getProperty(ConfigurationParameters.DEFAULT EMAIL SENDER);

    System.out.println( "Default email sender" + defaultEmailSender );
}
```

Récupérer les paramètres utilisateur

A partir d'un objet configuration, il est possible de récupérer les paramètres « utilisateur » en utilisant les méthodes getUserProperty() ou getUserProperties().

```
public void catalog_getUserProperty( IConfiguration configuration )
{
    // récupération des paramètres utilisateur
    String anyParameterValue = configuration.getUserProperty( "any.parameter" );
    String anotherParameterValue = configuration.getUserProperty( "another.parameter" );

    System.out.println( "any.parameter : " + anyParameterValue );
    System.out.println( "another.parameter : " + anotherParameterValue );
}
```

Récupérer les rôles d'une application

Depuis un object ICatalog, il est possible de récupérer un rôle ou tous les rôles d'une application.

```
public void catalog_useRoles( ICatalog catalog )
{
    // récupération de l'ensemble des roles d'une application
    Collection roles = catalog.getRoles();
    for ( Iterator iterRole = roles.iterator() ; iterRole.hasNext() ; )
    {
            IRole role = (IRole)iterRole.next();
            System.out.println( role.getLabel() + " [" + role.getName() + "]" );
    }
}
```

Récupérer les listes d'une application

Depuis l'object ICatalog, il est possible de récupérer les listes d'une application.

```
public void catalog useLists( ICatalog catalog )
{
  // récupération de l'ensemble des roles d'une application
  Collection lists = catalog.getLists();
  for ( Iterator iterList = lists.iterator() ; iterList.hasNext() ; )
  {
    IList list = (IList)iterList.next();
    if ( list instanceof IStringList )
    {
      IStringList stringList = (IStringList)list;
      System.out.println( stringList.getLabel() + " [" + stringList.getName() + "]" );
      for ( Iterator iterValue = stringList.getValues().iterator() ; iterValue.hasNext() ; )
      {
        String stringListElement = (String)iterValue.next();
        System.out.println( stringListElement );
      }
    }
    else if ( list instanceof IXmlList )
    {}
  }
}
```

Récupérer une version de processus

Depuis l'object ICatalog, il est possible de récupérer toutes les versions de processus. L'exemple suivant montre comment récupérer une version particulière à partir de son nom système.

```
public void catalog getWorkflow(IContext context,IWorkflowModule workflowModule,ICatalog catalog)
  throws WorkflowModuleException
{
  IWorkflow workflow = workflowModule.getWorkflow( context, catalog, "documentManagement_1.0" );
}
```

Tâches autour du document

Créer un document

Depuis l'object IWorkflow, des documents peuvent être créés. Notez que le troisième argument de la méthode createWorkflowInstance est null. Ceci indique que le système déterminera la référence du document en fonction du format spécifié dans le web Designer. Toutefois, il est possible de forcer la référence à une valeur précise. Dans ce cas, le troisième argument devra être renseigné.

```
public void document create(IContext context, IWorkflowModule workflowModule, IWorkflow workflow)
  throws WorkflowModuleException
{
  try
  {
    // démarrage d'une transaction
    workflowModule.beginTransaction( this );

    // création du document
    IWorkflowInstance newInstance = workflowModule.createWorkflowInstance(context,workflow,null);

    // positionnement de quelques valeurs
    newInstance.setValue( "fldDocumentVersion", "5.0" );
    newInstance.setValue( "fldDocumentType", "Type simple" );
    newInstance.setValue( "fldNature", "Interne" );
    newInstance.setValue( "fldWritingRequired", "Non" );
    newInstance.setValue( "fldReadingRequired", "Non" );
    newInstance.setValue( "fldVerificationRequired", "Non" );
    newInstance.setValue( "fldApprovingRequired", "Non" );
    newInstance.setValue( "sys_Title", "Documentation SDK" );
    newInstance.setValue( "fldDocumentDescription", "Les possibilités d'intégration" );
    newInstance.setValue( "fldKeywords", "vdoc process sdk education" );

    // sauvegarde des modifications
    newInstance.save();

    // validation de la transaction
    workflowModule.commitTransaction( this );
  }
  catch( Exception e )
  {
    // si une erreur s'est produite, annuler le traitement précédent
    workflowModule.rollbackTransaction( this );
    getNavigator().processErrors( e );
  }
}
```

Créer un document et personnaliser la référence

L'exemple suivant montre comment personnaliser la référence avant qu'elle ne soit inscrite en base.

```
public void document_createWithCustomReference( IContext context, IWorkflowModule workflowModule,
IWorkflow workflow, IResourceDefinition definition ) throws WorkflowModuleException
{
    try
    {
            // démarrage d'une transaction
            workflowModule.beginTransaction( this );

            // 1ème forme d'utilisation : récupération de la référence calculée pour une
            // définition de resource
            String defaultReference = workflowModule.generateReference( definition );

            // 2ème forme d'utilisation : récupération de la référence calculée pour une
            // définition de resource et un format souhaité
            String myReference = workflowModule.generateReference( definition,
              "[MYTAG]-[site]-[company]-[jj]-[mm]-[aaaa]-[aa]-[dd]-[yyyy]-[yy]-[chrono]" );
            myReference = StringUtils.replaceAll( myReference, "[MYTAG]", "REF" );

            // création du document
            IWorkflowInstance newWorkflowInstance = workflowModule.createWorkflowInstance(
              context, workflow, myReference );

            // positionnement de quelques valeurs
            // ...

            // sauvegarde des modifications
            newWorkflowInstance.save();

            // validation de la transaction
            workflowModule.commitTransaction( this );
    }
    catch( Exception e )
    {
            // si une erreur s'est produite, annuler le traitement précédent
            workflowModule.rollbackTransaction( this );
            getNavigator().processErrors( e );
    }
    finally
    {
            workflowModule.unInitialize();
    }
}
```

Récupérer le prochain numéro de chrono de référence formatée

L'exemple suivant montre comment récupérer le prochain numéro de chrono de la référence formatée.

```
public void document_getNextChrono( IWorkflowModule workflowModule, IResourceDefinition definition
) throws WorkflowModuleException
{
    // récupération du prochain chrono de référence calculée
    int nextChrono = workflowModule.generateChrono( definition );
}
```

Ajouter une ligne dans l'historique

L'exemple suivant montre comment ajouter un évènement dans l'historique d'un document.

```
public void document addEventHistory( IContext context, IWorkflowModule workflowModule,
IWorkflowInstance workflowInstance )
  throws WorkflowModuleException
{
  IUser qui = workflowModule.getUser( getNavigator().getLoggedOnUser() );
  IUser pourQui = qui;

  workflowInstance.getHistory().addEvent("Action","Etat",qui,pourQui,"Description","En tant que");
}
```

Ajouter le droit de lecture sur un document

L'exemple suivant montre comment ajouter le droit de lecture à un document.

```
public void document_addRights(IWorkflowModule workflowModule, IWorkflowInstance workflowInstance)
  throws WorkflowModuleException
{
  IUser user = workflowModule.getUser( getNavigator().getLoggedOnUser() );

  ISecurityController securityController = workflowModule.getSecurityController(workflowInstance);

  securityController.addPermission( user, Rights.Treatment.TreatmentLevel.READ CONTENT );
}
```

Positionner la valeur d'un sélecteur d'utilisateur simple

L'exemple suivant montre comment renseigner un sélecteur d'utilisateur simple. Cet exemple s'applique à tous les sélecteurs d'annuaire (IUser, IGroup, ILocalization, IOrganization), ainsi que le sélecteur de dossier ou fichier du module de plublication (IFile, IFolder).

```
public void
  document_setSelectorValue( IWorkflowModule workflowModule, IWorkflowInstance workflowInstance )
    throws WorkflowModuleException
{
    // positionner la valeur d'un sélecteur d'utilisateur simple
    IUser user = workflowModule.getUserByLogin( "user1" );
    workflowModule.setExternalUser( workflowInstance, "fldSingleUserSelector", user );
}
```

Positionner la valeur d'un sélecteur d'utilisateurs multiple

L'exemple suivant montre comment renseigner un sélecteur d'utilisateur multiple. Cet exemple s'applique à tous les sélecteurs d'annuaire (IUser, IGroup, ILocalization, IOrganization), ainsi que le sélecteur de dossier ou fichier du module de plublication (IFile, IFolder).

```
public void
  document setSelectorValues( IWorkflowModule workflowModule, IWorkflowInstance workflowInstance )
    throws WorkflowModuleException
{
    // constituer une liste d'utilisateurs
    ArrayList arrUsers = new ArrayList();
    arrUsers.add( workflowModule.getUserByLogin( "user1" ) );
    arrUsers.add( workflowModule.getUserByLogin( "user2" ) );

    // positionner la valeur d'un sélecteur d'utilisateurs multiple
    workflowModule.setExternalElements( workflowInstance, "fldMultipleUserSelector", arrUsers );
}
```

Utiliser une requête SQL pour remonter des documents

Pour une utilisation avancée, VDoc offre un système de requête SQL qui permet de remonter directement des éléments d'API SDK.

Tous les objets implémentant l'interface de marquage ISearchSupport peuvent être remontés directement via une requête SQL.

```
[com.axemble.vdoc.sdk.supports.ISearchSupport]
    [IAction]
    [ICatalog]
    [IResource]
            [ILinkedResource]
            [IWorkflowInstance]
    [IResourceDefinition]
    [IRole]
    [ITask]
    [ITaskInstance]
    [IUser]
            [IOperator]
    [IWorkflow]
    [IWorkflowContainer]
```

Exemple 1 : retrouver tous les documents d'une application

```
public void document_useSearch( IContext context, IWorkflowModule workflowModule )
{
  try
  {
    IJdbcReference jdbcReference = workflowModule.getJdbcReference( context, "vdocDatabase" );

    ISearchController searchController = workflowModule.getSearchController( jdbcReference );

    Collection docs = searchController.findElements( IWorkflowInstance.class,
      "select id from vdp treatments", null );

    for ( Iterator iter = worflowInstances.iterator() ; iter.hasNext() ; )
    {
      IWorkflowInstance workflowInstance = (IWorkflowInstance)iter.next();
    }
  }
  catch( Exception e )
  {
    getNavigator().processErrors( e );
  }
}
```

Exemple 2 : retrouver l'utilisateur dont le login est « froggy »

```
Object[] arrParams = new Object[1];
arrParams[0] = "froggy";
Collection users = searchController.findElements( IUser.class, "select id from vdp_users where
login = ?", arrParams );

for ( Iterator iterUser = users.iterator() ; iterUser.hasNext() ; )
{
    IUser user = (IUser)iterUser.next();
}
```

Parcourir un tableau dynamique

Pour récupérer les éléments d'un tableau dynamique, il suffit d'utiliser la méthode getLinkedResources(). Celle-ci renvoie une liste d'objets ILinkedResource.

```
public void document browseTableLines( IWorkflowInstance workflowInstance )
{
    Collection linkedResources = workflowInstance.getLinkedResources( "propertyName" );
    for ( Iterator iter = linkedResources.iterator() ; iter.hasNext() ; )
    {
            ILinkedResource linkedResource = (ILinkedResource)iter.next();
            String textValue = (String)linkedResource.getValue( "textColumn 1" );
            Float numberValue = (Float)linkedResource.getValue( "numberColumn_2" );
    }
}
```

Ajouter une ligne à un tableau dynamique

La méthode createLinkedResource() simplifie la façon de créer des lignes de tableau. Il suffit d'indiquer sur quel champ (nom système de la propriété) vous souhaitez créer une ligne.

```
public void document createTableLines( IWorkflowInstance workflowInstance )
{
    // création d'une ligne
    ILinkedResource linkedResource = workflowInstance.createLinkedResource( "propertyName" );

    String textValue = "some text";
    Float numberValue = new Float( 7 );

    // positionnement de quelques valeurs
    linkedResource.setValue( "textColumn_1", textValue );
    linkedResource.setValue( "numberColumn_2", numberValue );

    // ajout de la ligne au tableau
    workflowInstance.addLinkedResource( linkedResource );
}
```

Ajouter des pièces jointes

Dans cet exemple, trois méthodes sont présentées pour ajouter des fichiers au champ pièces jointes.

```
public void document_createAttachment( IWorkflowModule module, IWorkflowInstance instance )
  throws WorkflowModuleException
{
  // création d'un objet pièce jointe à partir d'un fichier sur le serveur
  IAttachment idisk = module.addAttachment(instance,"propertyName","xxx","c:/xxx.xls" );

  // création d'un objet pièce jointe à partir d'un java.io.InputStream
  InputStream inputStream = null;
  IAttachment istream=module.addAttachment(instance,"propertyName","document_name",inputStream );

  // création d'un objet pièce jointe à partir d'un objet java.io.File
  File file = new File( "c:/tmp/document_name.xls" );
  IAttachment ifile = module.addAttachment( instance, "propertyName", file );
}
```

Récupérer les fichiers d'un champ pièces jointes

La méthode getAttachments() du module de workflow permet de récupérer tous les fichiers d'un champ pièces jointes. Il est alors possible, via l'interface IAttachment, de retrouver, pour chaque fichier, toutes les informations et le contenu.

```
public void document_getAttachments( IWorkflowModule module, IWorkflowInstance instance )
  throws WorkflowModuleException, FileNotFoundException
{
    Collection attachments = module.getAttachments( instance, "propertyName" );
    for ( Iterator iter = attachments.iterator() ; iter.hasNext() ; )
    {
            IAttachment attachment = (IAttachment)iter.next();

            // retrouver le nom
            attachment.getName();

            // retrouver la taille
            attachment.getSize();

            // récupérer le contenu dans un OutputStream
            attachment.getContent( new FileOutputStream(new File("c:/tmp/xxx.yyy")) );

            // récupérer le contenu dans un fichier
            attachment.getContent( new File("c:/tmp/xxx.yyy") );
    }
}
```

Effectuer un changement d'étape

Dans cet exemple, plusieurs éléments sont présentés :

- récupération du contexte de l'utilisateur « froggy » pour évaluer s'il existe une tâche active sur le document ;
- récupération de l'action à partir du nom système de l'action ;
- utilisation du module de workflow pour effectuer le changement d'étape.

```
public void document changeStage( IWorkflowModule module, IWorkflowInstance instance )
  throws WorkflowModuleException
{
    IContext localContext = module.getContextByLogin( "froggy" );

    ITaskInstance taskInstance = instance.getCurrentTaskInstance( localContext );
    if ( taskInstance != null )
    {
            IAction action = taskInstance.getTask().getAction( "actionName" );
            if ( action != null )
            {
                    module.end( localContext, taskInstance, action, "decription" );
            }
    }
}
```

Transformer un document au format XML

Dans cet exemple sont montrées les deux méthodes de conversion des documents depuis et vers l'XML.

```
public void document_transform( IWorkflowModule module, IWorkflowInstance instance )
  throws WorkflowModuleException, IOException
{
  // récupération d'un objet de transformation
  ITransformer transformer = module.getTransformer();

  // conversion d'une instance de workflow en fichier XML
  FileOutputStream xmlOutput = new FileOutputStream( new File( "D:/tmp/sdk/XXX.xml" ) );
  transformer.resourceToXML( instance, xmlOutput );

  // conversion d'un fichier XML en instance de workflow
  InputStream inputStream = new FileInputStream( new File( "D:/tmp/sdk/XXX.xml" ) );
  transformer.xmlToResource( inputStream );
}
```

Tâches autour de la personnalisation

Récupérer le Controller de ressource

Le controller de ressources est un objet qui permet de manipuler le rendu graphique d'un document processus. Pour récupérer un objet de ce type, il suffit de lui passer un élément objet IResource. Dans cet exemple, nous passons directement l'objet WorkflowInstance courant (c'est une ressource).

```
getWorkflowModule().getController( getWorkflowInstance() );
```

Rendre des champs du document processus obligatoires ou non

```
getResourceController().setMandatory( "fldDocumentDescription", true );
getResourceController().setMandatory( "fldDocumentBody", false );
```

Rendre des champs du document processus éditables ou non

```
getResourceController().setEditable( "fldDocumentDescription", false );
getResourceController().setEditable ( "fldDocumentBody", true );
```

Masquer ou non des champs du document processus

```
getResourceController().setHidden( "fldDocumentDescription", false );
getResourceController().setHidden( "fldPassword", true );
```

Masquer des éléments d'interface identifiés du document processus

Ce code masque ou affiche des blocs identifiés. Ces blocs peuvent être des fragments ou des éléments identifiés dans les formulaires de personnalisation (attribut « id » défini).

```
getResourceController().showBodyBlock( "marker.DocumentSpace", true );
getResourceController().showBodyBlock( "marker.LifeCycle", false );
```

Masquer une action de workflow

Le code suivant indique comment retrouver une action de workflow et la masquer.

```
// récupération de l'utilisateur connecté, puis construction d'un contexte
IContext context = getWorkflowModule().getContext(
  Navigator.getNavigator().getLoggedOnUser().getExternId() );

// récupération de la définition de la tâche active pour la comparer avec l'étape souhaitée
ITask task = getWorkflowInstance().getCurrentTaskInstance( context ).getTask();
if ( "TASK_NAME".equals( task.getName() ) )
{
  // récupération de l'action
  IAction action = getWorkflowModule().getAction( context, task, "ACTION_NAME" );

  // retrouver le bouton correspondent à l'action (chercher dans le container du bas)
  CtlButton button = getResourceController().getButton( action.getLabel(),
    IResourceController.BOTTOM_CONTAINER );

  // si le bouton est retrouvé, le masquer
  if ( button != null )
    button.setHidden( true );
}
```

Ajouter un bouton sur un document processus

Code extrait de la classe PlayWithButtonsExtension

```
CtlButton demoAlertBtn = new CtlButton( "alert", new CtlLocalizedText( "LG ALERT" ) );
demoAlertBtn.addActionListener( new ActionListener()
{
```

```
  public void onClick( ActionEvent event )
  {
    Navigator.getNavigator().showAlertBox( "Ceci est un message Alert !" );
  }
} );
getResourceController().getButtonContainer( IResourceController.TOP_CONTAINER ).addLast(
  demoAlertBtn );
```

Tâches autour des scripts

Evaluer l'appartenance de l'utilisateur connecté à un rôle

```
var role = iWorkflowModule.getRole( iContext, iWorkflowInstance.getCatalog(), "Vérificateur" );
if ( iUser.isMemberOf(role, true) )
  log.info( "The user " + iUser.getFullName() + " is member of " + role.getLabel() + " role." );
else
  log.info( "The user " + iUser.getFullName() + " is member of " + role.getLabel() + " role." );
```

Chapitre 18 : Référence de la gestion documentaire

L'API SDK intègre un module de gestion documentaire qui permet de manipuler, directement depuis les extensions Process ou tout autre système au sein du Portail, les éléments de Document Management et les relations définies entre eux.

La communication entre Process et Document Management est quasi-transparente via le SDK.

Cette couche simplifie la réalisation de tâches administratives telles que :

- la création de versions, de liasses documentaires ;
- l'association entre les dossiers et les versions ;
- l'association entre les fiches, les données et les versions.

Quelques définitions

Eléments de définition

De nombreux éléments sont implantés dans la base VDoc. Parmi ces éléments de définition, nous retiendrons les suivants :

- IResourceDefinition : rassemble toutes les propriétés (champs VDoc ; champs libres et champs listes) définies dans l'administration.
- IProperty : description d'un champ du document ;
- IWorkflow : constitue le cycle de vie du document. Rassemble les tâches et les actions ;
- ITask : description d'une tâche;
- IAction : description d'une action ;
- IBundleDefinition : constitue la définition d'une liasse ;
- IDataForm : constitue l'élément de base des fiches ;
- ISimpleDataForm : fiche standard ;
- IMenuDataForm : fiche menu ;
- IViewDataForm : fiche vue ;
- IFolder : interface correspondant à un élément d'une structure arborescente ;
- IAttachmentTemplate : modèle de document word.

Diagramme de classes des éléments de définition

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes descriptives du système de gestion documentaire.

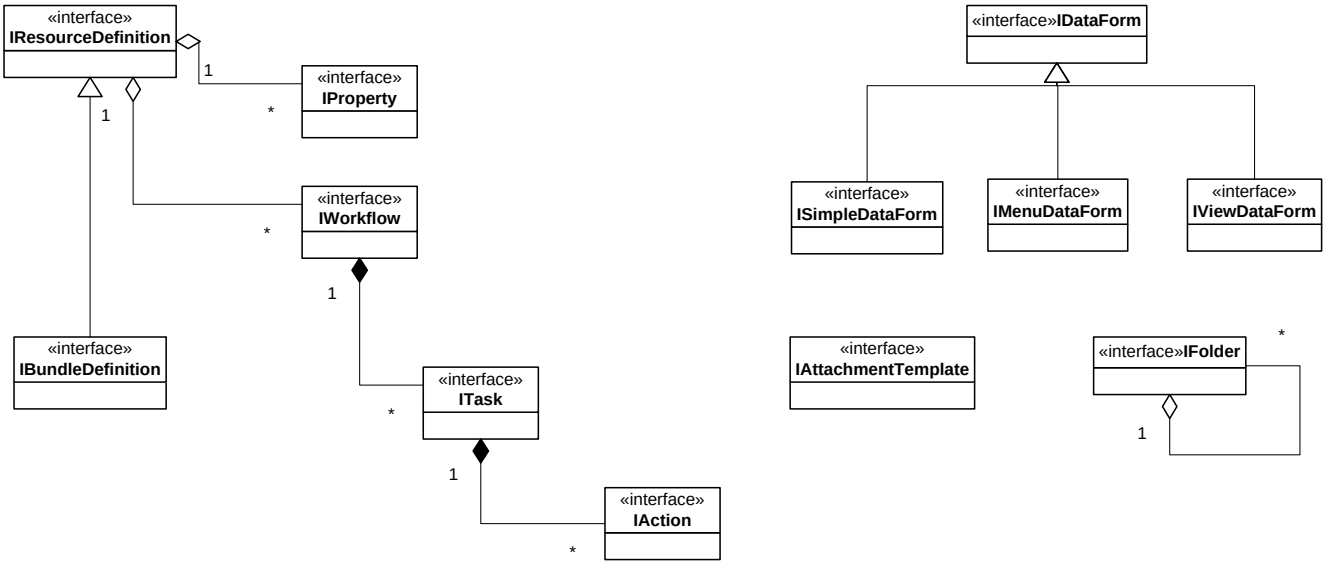


Tableau de correspondance des éléments de définition

Terme fonctionnel	Terme technique	Terme SDK	Terme Open SDK	Description
Type de document	Définition de resources	IResourceDefinition	ClsDocumentTypeItem	Elément constituant la la description d'un document en terme de valeurs supportées.
Champ Propriété Attributs documentaires	Propriété	IProperty	Property	Elément constituant la description d'une valeur supportée par le document.
Cycle de vie	Cycle de vie	IWorkflow	VersionGroup	Un cycle de vie est la modélisation du cheminement d'un document à travers plusieurs étapes.
Tâche Etape	Etape	ITask	ClsGeneralInformationsReader ClsStepWriter ClsStepReader	Une tâche est l'objet modélisé identifiant une étape du cycle de vie.
Action de changement d'étape Action de routage	Action de routage	IAction		Action utilisateur permettant de sortir d'une étape.
Modèle de liasse	Définition de liasse	IBundleDefinition	ClsBundleModelItem	Elément constituant la description d'une liasse documentaire.
Fiche	Fiche	IDataForm	ClsDataFormItem	Elément de base des fiches type standard, menu, et vue.
Fiche standard	Fiche standard	ISimpleDataForm	ClsDataFormItem	Fiche standard permettant de stocker des données, des menus et des sous-fiches.
Fiche menu	Fiche menu	IMenuDataForm	ClsDataFormItem	Fiche permetant d'amener des termes fonctionnels dans la navigation dans les données.
Fiche vue	Fiche vue	IViewDataForm	ClsDataFormItem	Fiche permettant de pointer sur une vue ou une table externe.
Dossier	Dossier	IFolder	ClsFolderReader ClsFolderItem	Elément d'une structure arborescente permetant de mettre en place la sécurité.
Modèle de document	Modèle de document	IAttachmentTemplate	ClsDocumentModelItem ClsWebFavouriteItem	Modèle de document Word.

Eléments dynamiques

A la création d'un document, plusieurs éléments sont inscrits en base. Parmi ces éléments dynamiques, nous retiendrons les suivants :

- IResource : interface représentant les données du document et à partir de laquelle il est possible de modifier les valeurs de champs ;
- IVersion : interface dérivant de l'interface IResource permettant de gérer le cycle de vie du document (changement d'état) ;
- ITaskInstance : interface représentant une tâche active du document sur laquelle des opérateurs peuvent intervenir ;
- IAttachment : interface permettant de manipuler des champs de type pièces jointes ;
- IOperator : interface représentant un utilisateur intervenant sur une étape ;
- IBundle : interface dérivant de l'interface IVersion permettant de gérer le cycle de vie des liasses ;
- IDataUnit : interface dérivant de l'interface IResource permettant de gérer les valeurs d'une donnée.

Diagramme de classes des éléments dynamiques

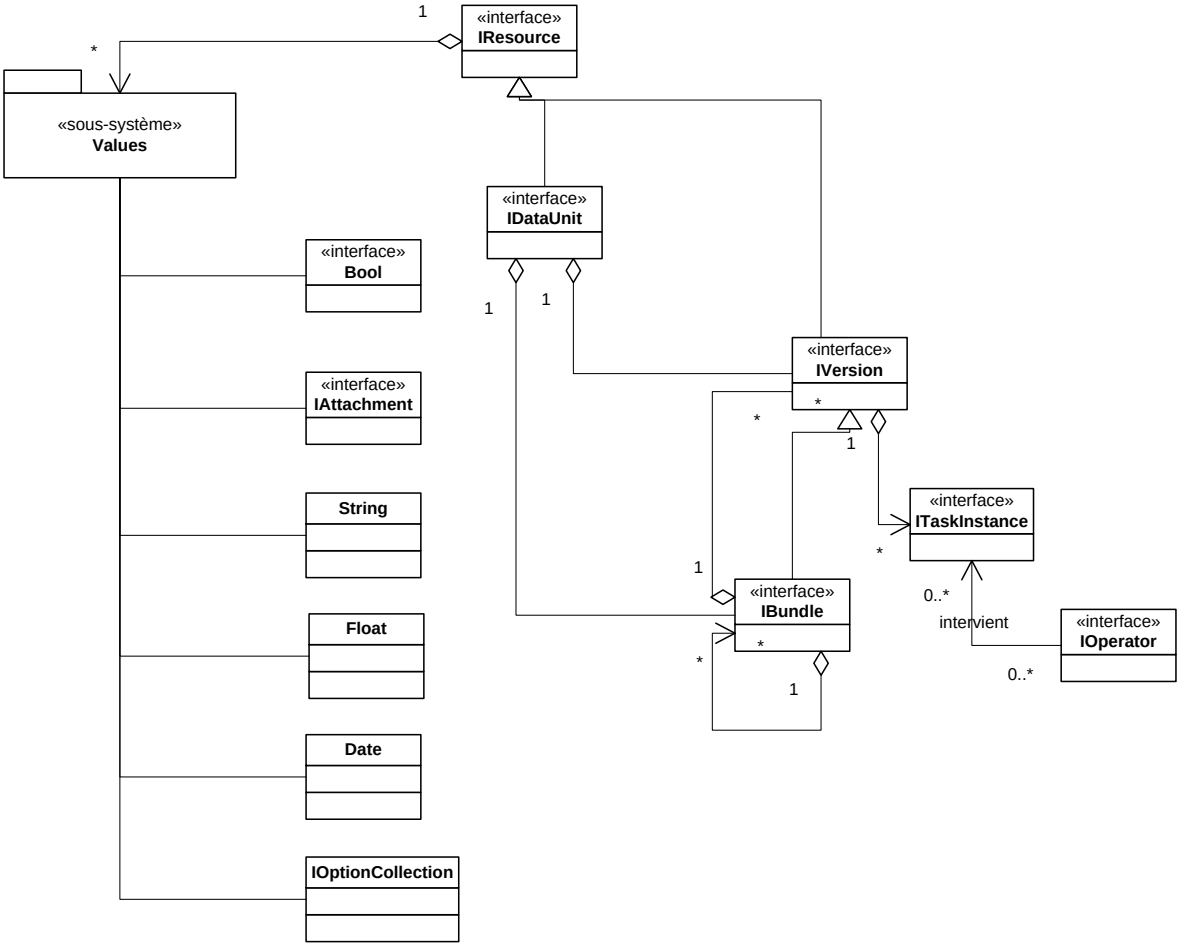


Tableau de correspondance des éléments dynamiques

Terme fonctionnel	Terme technique	Terme SDK	Terme Open SDK	Description
Ressource	Ressource	IResource	Resource	Le terme « Ressource » représente le conteneur de valeurs de champs. Elle permet de stocker l'ensemble des données d'un document.
Version	Version	IVersion	ClsVersionItem ClsBundleElementItem ClsVersionModifier ClsVersionReader ClsVersionBuilder ClsVersionDuplicator	Elément unitaire permettant de véhiculer des données utilisateur au travers d'un cycle de vie.
Tâche à traiter	Tâche à traiter	ITaskInstance	ClsVersionReader	Une tâche à traiter représente une intervention à réaliser par une personne physique ou un programme.
Pièce jointe, fichier	Enrobeur de fichier	IAttachment	ClsFileReader	Description d'un fichier attaché dans le document via un champ pièces jointes.
Opérateur Intervenant	Opérateur	IOperator	ClsOperatorReader ClsUserItem ClsGroupItem IUser IGroup	Il s'agit de l'association entre une personne et une tâche active.
Liasse documentaire	Liasse	IBundle	ClsBundleReader ClsBundleBuilder ClsVersionModifier ClsBundleItem ClsBundleElementItem	Liasse documentaire permettant de regrouper plusieurs documents, et sous-liasses.
Données	Données	IDataUnit	ClsDataBuilder ClsDataReader ClsDataItem	Données permettant de contenir des versions, et des fiches.

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc.

Pour des raisons de clarté, mais aussi pour marquer les différents niveaux d'API, nous avons décidé de classer ces développements en plusieurs sous-sections.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module document management des API SDK Récupérer un contexte pour utiliser les API SDK
Serveur	Récupérer les types de document Récupérer un type de document par son nom système Récupérer les propriétés d'un type de document Récupérer les modèles de document Récupérer un modèle de document par son nom système Récupérer les modèles de liasses

	Récupérer un modèle de liasse par son nom système
Dossiers	Créer un dossier sous la racine Créer un sous dossier Récupérer le dossier racine Récupération des dossiers fils Récupération d'un dossier fils
Versions	Créer une version à partir d'un type de document Créer une version à partir d'un modèle de document Créer une version à partir d'un fichier Effectuer un changement d'étape Créer une nouvelle version
Liasses	Créer une liasse à partir d'un modèle de liasse Effectuer un changement d'étape Créer une nouvelle liasse Parcourir le contenu d'une liasse Naviguer dans les liasses
Fiches	Récupérer la fiche racine Récupérer une fiche type menu Récupérer une fiche type standard Récupérer une fiche type vue Récupérer les sous-fiches d'une fiche
Données	Récupérer les données d'une fiche Créer une donnée Récupérer les versions d'une donnée Récupérer les attributs d'une donnée
Annuaire	Récupérer un utilisateur de la gestion documentaire Récupérer un utilisateur en tant qu'opérateur Récupérer l'ensemble des utilisateurs Récupérer un groupe de la gestion documentaire Récupérer un groupe en tant qu'opérateur

Tâches générales

Utiliser le module document management des API SDK

Le code suivant indique comment utiliser l'API document management. La méthode initialize() est utilisée pour positionner un certain nombre de paramètres.

Le module document management a la particularité d'utiliser une couche COM qui nécessite l'appel des méthodes Ole32.CoInitialize() et Ole32.CoUninitialize().

```
public void general useModule()
{
     try
     {
            Ole32.CoInitialize();

            IDocumentManagementModule module = null;

            try
            {
                  // création d'un object de module gestion documentaire
                  module = new
                    IDocumentManagementModule("VDoc.Sdk.Classes.DocumentManagementModule");

                  // positionnement des paramètres de connexion
                  module.AddParameter( module.getPROTOCOL(), "http://" );
                  module.AddParameter( module.getHOST(), "host-name" );
```

```
                    module.AddParameter( module.getPORT(), "80" );
                    module.AddParameter( module.getAPPLICATION(), "vdocopenweb" );
                    module.AddParameter( module.getDATABASE(), "database-name" );
                    module.AddParameter( module.getCHECK BUNDLE TIMEOUT(), "250" );

                    // initialisation du module
                    module.Initialize();

                    // do something here...
            }
            catch( Exception e )
            {}
            finally
            {
                    module.UnInitialize();
                    Ole32.CoUninitialize();
            }
    }
    catch( Exception e )
    {}
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general_getContext( IDocumentManagementModule module ) throws Exception
{
    // 1er cas : récupération du contexte à partir d'un objet utilisateur
    IUser user = module.GetUserByLogin( "sysadmin" );
    IContext userContext = module.GetContext( user );

    // 2ème cas : récupération du contexte par login
    IContext loginContext = module.GetContextByLogin( "sysadmin" );

    // 3ème cas : récupération du contexte par authentification
    IContext authenticationContext = module.GetContextByAuthentication( "adm", "admin" );
}
```

Tâches serveur

Récupérer les types de document

Exemple de parcours des types de document. L'exemple présente la classe _ResourceDefinitionCollection.

```
public void library getResourceDefinitions( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver tous les types de document
     ResourceDefinitionCollection definitions = module.GetResourceDefinitions( context );
    for ( int index = 0 ; index < definitions.getCount() ; index++ )
    {
            IResourceDefinition resourceDefinition = definitions.getItem( index );
            LOGGER.info( "resource definition: "
              + resourceDefinition.getId().getAsString()
              + ", name: " + resourceDefinition.getname()
              + ", label: " + resourceDefinition.getlabel() );
    }
}
```

Récupérer un type de document par son nom système

Exemple de récupération d'un type document par son nom système.

```
public void library getResourceDefinition( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver un type de document particulier
    IResourceDefinition resourceDefinition = module.GetResourceDefinition( context, "Type SDK" );
```

```
    LOGGER.info( "resource definition: "
      + resourceDefinition.getId().getAsString()
      + ", name: " + resourceDefinition.getname()
      + ", label: " + resourceDefinition.getlabel() );
}
```

Récupérer les propriétés d'un type de document

Exemple de parcours des propriétés d'un type de document. L'exemple présente la classe _PropertyCollection.

```
public void library getResourceDefinitionProperties( IDocumentManagementModule module, IContext
context, IResourceDefinition resourceDefinition ) throws Exception
{
    // retrouver toutes les propriétés d'un type de document
     PropertyCollection properties = module.GetPropertiesByResourceDefinition( resourceDefinition
);
    for ( int index = 0 ; index < properties.getCount() ; index++ )
    {
           IProperty property = properties.getItem( index );
           LOGGER.info( "property: " + property.getId().getAsString() + ", name: " +
property.getname() + ", label: " + property.getlabel() );
    }
}
```

Récupérer les modèles de document

Exemple de parcours des modèles de document. L'exemple présente la classe _AttachmentTemplateCollection.

```
public void library getAttachementTemplates( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver tous les types de document
    _AttachmentTemplateCollection attachmentTemplates = module.GetAttachmentTemplates( context );
    for ( int index = 0 ; index < attachmentTemplates.getCount() ; index++ )
    {
           IAttachmentTemplate attachmentTemplate = attachmentTemplates.getItem( index );
           LOGGER.info( "attachment template: "
             + attachmentTemplate.getId().getAsString()
             + ", name: " + attachmentTemplate.getname() );
    }
}
```

Récupérer un modèle de document par son nom système

Récupération d'un modèle de document à partir de son nom système.

```
public void library getAttachementTemplate( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver un type de document particulier
    IAttachmentTemplate attachmentTemplate = module.GetAttachmentTemplate( context, "Modèle SDK"
);
    LOGGER.info( "attachment template: "
      + attachmentTemplate.getId().getAsString() + ", name: " + attachmentTemplate.getname() );
}
```

Récupérer les modèles de liasses

Exemple de parcours des modèles de liasse. L'exemple présente la classe _BundleDefinitionCollection.

```
public void library getBundleDefinitions( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver tous les modèles de liasse
     BundleDefinitionCollection definitions = module.GetBundleDefinitions( context );
    for ( int index = 0 ; index < definitions.getCount() ; index++ )
    {
           IBundleDefinition bundleDefinition = definitions.getItem( index );
           IResourceDefinition resourceDefinition = bundleDefinition.getResourceDefinition();
           LOGGER.info( "bundle definition: "
             + resourceDefinition.getId().getAsString()
             + ", name: "
```

```
                + resourceDefinition.getname() + ", label: " + resourceDefinition.getlabel() );
    }
}
```

Récupérer un modèle de liasse par son nom système

Récupération d'un modèle de liasse à partir de son nom système.

```
public void library getBundleDefinition( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // retrouver un modèle de liasse particulier
    IBundleDefinition bundleDefinition = module.GetBundleDefinition( context, "Bundle SDK" );
    IResourceDefinition resourceDefinition = bundleDefinition.getResourceDefinition();
    LOGGER.info( "bundle definition: " + resourceDefinition.getId().getAsString() + ", name: "
      + resourceDefinition.getname() + ", label: " + resourceDefinition.getlabel() );
}
```

Tâches sur les dossiers

Créer un dossier à la racine

Depuis le module, il est possible de créer un dossier à la racine.

```
public void library_createFolder( IDocumentManagementModule module, IContext context )
  throws Exception
{
 // création d'un dossier à la sous le dossier racine
 IFolder folder = module.CreateFolder( context, "Folder SDK" );

 // positionnement des droits hérités
 folder.setInheritRights( true );

 // propagation des droits aux sous dossiers
 folder.setPropagateRights( true );

 // validation de la création du dossier
 folder.Save();
}
```

Créer un sous dossier

Depuis le module, il est possible de créer un sous dossier en passant le dossier parent.

```
public void library createChildFolder( IDocumentManagementModule module, IContext context ) throws
Exception
{
 IFolder rootFolder = module.getRootFolder();
 IFolder parentFolder = module.GetFolderByName( context, rootFolder, "Folder SDK" );

 // création d'un dossier fils
 IFolder folder = module.CreateFolderIntoFolder( context, parentFolder, "Sub folder SDK" );

 // positionnement des droits hérités
 folder.setInheritRights( true );

 // propagation des droits aux sous dossiers
 folder.setPropagateRights( true );

 // validation de la création du dossier fils
 folder.Save();
}
```

Récupérer le dossier racine

Depuis le module, il est possible de récupérer le dossier racine.

```
public void library_getRootFolder( IDocumentManagementModule module ) throws Exception
```

```
{
    // récupération du dossier racine
    IFolder rootFolder = module.getRootFolder();
}
```

Récupération des dossiers fils

Exemple de parcours des dossiers fils d'un dossier. L'exemple présente la classe _FolderCollection.

```
public void library getChildrenFolder( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IFolder rootFolder = module.getRootFolder();

    // récupération de tous les dossiers contenus sous le dossier racine
     FolderCollection folders = module.GetFolders( context, rootFolder );
    for ( int index = 0 ; index < folders.getCount() ; index++ )
    {
           IFolder folder = folders.getItem( index );
           LOGGER.info( "resource definition: "
            + folder.getId().getAsString() + ", name: " + folder.getname() );
    }
}
```

Récupération d'un dossier fils

Exemple de récupération d'un dossier fils par son nom.

```
public void library getChildFolder( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IFolder rootFolder = module.getRootFolder();

    // récupération d'un dossier particulier sous le dossier racine
    IFolder childFolder = module.GetFolderByName( context, rootFolder, "Folder SDK" );
}
```

Tâches sur les versions

Créer une version à partir d'un type de document

Exemple présentant les différentes façons de créer des versions à partir d'un type de document.

```
public void document_createVersion( IDocumentManagementModule module,
  IContext context, IResourceDefinition resourceDefinition ) throws Exception
{
    // créer une version en utilisant le paramétrage par défaut
    IVersion version = module.CreateVersionByResourceDefinition( context, resourceDefinition,
      "reference", "title" );

    // créer une version en utilisant le paramétrage par défaut et spécifiant l'emplacement dans
    // les dossiers
    IFolder folder = module.getRootFolder().GetFolder( "Folder SDK" );
    IVersion folderVersion = module.CreateVersionIntoFolder( context, resourceDefinition,
      "reference", "title", folder );

    // créer une version en utilisant le paramétrage par défaut et spécifiant l'emplacement dans
    // les données
    ISimpleDataForm form = module.GetSimpleDataFormByName( module.getRootDataForm(),
      "Form SDK" );
    IDataUnit dataUnit = module.CreateDataUnit( form );
    IVersion dataVersion = module.CreateVersionIntoDataUnit( context, resourceDefinition,
      "reference", "title", dataUnit );

    // créer une version en spécifiant les emplacements dans les dossiers et les données
    folder = module.getRootFolder().GetFolder( "Folder SDK" );
    form = module.GetSimpleDataFormByName( module.getRootDataForm(), "Form SDK" );
    dataUnit = module.CreateDataUnit( (ISimpleDataForm)form );
    IVersion folderAndDataVersion = module.CreateVersion( context, resourceDefinition,
      "reference", "title", folder, dataUnit );
```

```
    // positionnement du numéro de version
    version.setVersion( "1.0" );

    // commentaire concernant la création de la version
    version.setComment( "Création d'une version via la SDK VDoc" );

    IResource resource = version.getResource();

    // valeur de type boolean
    resource.SetValueAsBool( "Boolean", true );

    // valeur de type date
    resource.SetValueAsDate( "DateTime", new Date() );
    resource.SetValueAsDate( "SmallDateTime", new Date() );

    // valeur de type liste simple
    resource.SetValueAsString( "Single selector", "A1" ); // A1, faisant partie de la liste
    resource.SetValueAsString( "Simple list", "AZERTY" ); // AZERTY, faisant partie de la liste

    // valeur de type liste multiple
    _OptionCollection arr = module.CreateCollection();
    arr.Add( "BB" );
    arr.Add( "CC" );
    // BB et CC, faisant toutes deux parties de la liste
    resource.SetValueAsCollection( "Multiple list", arr );

    arr = module.CreateCollection();
    arr.Add( "B1" );
    arr.Add( "B2" );
    arr.Add( "B3" );
    // B1, B2 et B3, faisant toutes trois parties de la liste
    resource.SetValueAsCollection( "Multiple selector", arr );

    resource.Save();
}
```

Créer une version à partir d'un modèle de document

Exemple de création d'une version à partir d'un modèle de document.

```
public void document_createVersion( IDocumentManagementModule module, IContext context,
  IAttachmentTemplate template ) throws Exception
{
    // récupération d'un dossier
    IFolder folder = module.getRootFolder().GetFolder( "Folder SDK" );

    // récupération d'une donnée
    ISimpleDataForm form = module.GetSimpleDataFormByName( module.getRootDataForm(),
      "Form SDK" );
    IDataUnit dataUnit = module.GetDataUnitByDataForm( form, new Object[]{"123"} );

    // créer une version en utilisant un modèle de document
    IVersion version = module.CreateVersionByAttachmentTemplate( context, template, "reference",
      "title", folder, dataUnit );

    // récupérer la ressource associée à la version
    IResource resource = version.getResource();

    // enregistrer la ressource
    resource.Save();
}
```

Créer une version à partir d'un fichier

Exemple de création de version avec ajout de fichier. L'exemple présente la classe IAttachment.

```
public void document createVersion( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // créer une version
    IVersion version = null; // module.CreateVersion...;

    // récupérer la ressource associée à la version
    IResource resource = version.getResource();
```

```
    // enregistrer la ressource
    resource.Save();

    // ajouter la pièce jointe
    IAttachment attachement = module.AddAttachment(version, "C:\\sdk\\any.doc");
}
```

Effectuer un changement d'étape

Exemple de changement d'étape. L'exemple présente les classes ITaskInstance et IAction.

```
static void document_changeState( IDocumentManagementModule module, IContext context,
  IVersion version ) throws Exception
{
    // récupération d'une tâche active sur la version
    ITaskInstance taskInstance = module.GetOneTaskInstance( context, version );

    // récupération de l'action "validate" associée
    IAction action = taskInstance.gettask().GetAction( module.getVALIDATE_ACTION() );

    // effectuer le changement d'étape
    taskInstance.End( action, "Commentaire de changement d'étape" );
}
```

Créer une nouvelle version

Exemple de création d'une nouvelle version.

```
public void document createNewVersion( IDocumentManagementModule module, IContext context,
  IVersion version ) throws Exception
{
    IVersion newVersion = module.CreateNewVersion( context, version, version.getReference(),
      version.getTitle(), "comment pour la nouvelle version." );

    IResource resource = newVersion.getResource();
    resource.Save();

    LOGGER.info( "version:" + resource.getId().getAsString()
      + ", reference:" + newVersion.getReference() + ", title:"
      + newVersion.getTitle() + ", version:"
      + newVersion.getVersion() + ", comment:" + newVersion.getComment() );
}
```

Tâches sur les liasses

Créer une liasse à partir d'un modèle de liasse

Exemple de création d'une liasse à partir d'un modèle de liasse. Plusieurs cas sont présentés dans cet exemple :

- création par défaut ;
- création en spécifiant le dossier de destination ;
- création en spécifiant la donnée liée.

```
public void document createBundle( IDocumentManagementModule module, IContext context,
  IBundleDefinition bundleDefinition ) throws Exception
{
    // créer une liasse en utilisant le paramétrage par défaut
    IBundle bundle = module.CreateBundleByBundleDefinition( context, bundleDefinition,
      "reference", "title" );

    // créer une liasse en utilisant le paramétrage par défaut et spécifiant l'emplacement dans
      les dossiers
    IFolder folder = module.GetFolderByName( context, module.getRootFolder(), "Folder SDK" );
    IBundle folderBundle = module.CreateBundleIntoFolder( context, bundleDefinition, "reference",
      "title", folder );

    // créer une liasse en utilisant le paramétrage par défaut et spécifiant l'emplacement dans
```

```
      les données
    ISimpleDataForm form = module.GetSimpleDataFormByName( module.getRootDataForm(),
      "Form SDK" );
    IDataUnit dataUnit = module.CreateDataUnit( form );
    IBundle dataBundle = module.CreateBundleIntoDataUnit( context, bundleDefinition, "reference",
      "title", dataUnit );

    // créer une liasse en spécifiant les emplacements dans les dossiers et les données
    folder = module.getRootFolder().GetFolder( "Folder SDK" );
    form = module.GetSimpleDataFormByName( module.getRootDataForm(), "Form SDK" );
    dataUnit = module.CreateDataUnit( (ISimpleDataForm)form );
    IBundle folderAndDataBundle = module.CreateBundle( context, bundleDefinition, "reference",
      "title", folder, dataUnit );

    // positionnement du numéro de version de la liasse
    bundle.getVersion().setVersion( "1.0" );

    // commentaire concernant la création de la liasse
    bundle.getVersion().setComment( "Création d'une liasse via la SDK VDoc" );

    IResource resource = bundle.getVersion().getResource();
    // valeur de type boolean
    resource.SetValueAsBool( "Boolean", true );

    // valeur de type date
    resource.SetValueAsDate( "DateTime", new Date() );
    resource.SetValueAsDate( "SmallDateTime", new Date() );

    // valeur de type liste simple
    resource.SetValueAsString( "Liste selector", "A1" ); // A1, faisant partie de la liste
    resource.SetValueAsString( "Liste simple", "AZERTY" ); // AZERTY, faisant partie de la liste

    // valeur de type liste multiple
    _OptionCollection arr = module.CreateCollection();
    arr.Add( "BB" );
    arr.Add( "CC" );
    // BB et CC, faisant toutes deux parties de la liste
    resource.SetValueAsCollection( "Liste multiple", arr );

    arr = module.CreateCollection();
    arr.Add( "B1" );
    arr.Add( "B2" );
    arr.Add( "B3" );
    // B1, B2 et B3, faisant toutes trois parties de la liste
    resource.SetValueAsCollection( "Liste selector multiple", arr );

    resource.Save();
}
```

Créer une nouvelle liasse

Exemple de création d'une nouvelle liasse.

```
public void document createNewBundle( IDocumentManagementModule module, IContext context,
  IBundle bundle ) throws Exception
{
    IBundle newBundle = module.CreateNewBundle( context, bundle,
      bundle.getVersion().getReference(), bundle.getVersion().getTitle(),
      "comment for the new version." );

    IVersion version = newBundle.getVersion();
    IResource resource = version.getResource();
    resource.Save();

    LOGGER.info( "version:" + resource.getId().getAsString() + ", reference:"
      + version.getReference() + ", title:" + version.getTitle() + ", version:"
      + version.getVersion() + ", comment:" + version.getComment() );
}
```

Parcourir le contenu d'une liasse

.Exemple de parcours du contenu de la liasse

```
public void document getBundleContent( IDocumentManagementModule module, IContext context,
  IBundle bundle ) throws Exception
```

```
{
    // récupérer les éléments de la liasses sous forme de versions
     VersionCollection elements = bundle.getElements();
    for ( int index = 0 ; index < elements.getCount() ; index++ )
    {
            IVersion element = elements.getItem( index );
            // vérifier le type de l'élément
            if ( element.getIsBundle() )
            {
                    // récupérer la liasse
                    IBundle childBundle = element.getBundle();

                    // récupérer la version de la liasse
                    IVersion childVersion = childBundle.getVersion();

                    // récupérer la ressource associée à la version
                    IResource childResource = childVersion.getResource();

                    LOGGER.info( "bundle:" + childResource.getId().getAsString()
                      + ", reference:" + element.getReference() + ", title:"
                      + element.getTitle() + ", version:" + element.getVersion()
                      + ", version:" + element.getComment() );
            }
            else
            {
                    // récupérer la ressource associée à l'élément
                    IResource resource = element.getResource();
                    LOGGER.info( "version:" + resource.getId().getAsString()
                      + ", reference:" + element.getReference() + ", title:"
                      + element.getTitle() + ", version:" + element.getVersion()
                      + ", version:" + element.getComment() );
            }
    }
}
```

Pacourir les liasses à la racine du serveur

Récupérer l'ensemble des liasses à la racine du serveur. L'exemple présente la classe _BundleCollection.

```
public void document_getBundles( IDocumentManagementModule module, IContext context )
  throws Exception
{
    // récupération de toutes les liasses
     BundleCollection bundles = module.GetBundles( context );
    for ( int index = 0 ; index < bundles.getCount() ; index++ )
    {
            IBundle bundle = bundles.getItem( index );

            // récupération de la version associée
            IVersion version = bundle.getVersion();

            // récupération de la resource associée à la version
            IResource resource = version.getResource();

            LOGGER.info( "bundle:" + resource.getId().getAsString() + ", reference:"
              + version.getReference() + ", title:" + version.getTitle() + ", version:"
              + version.getVersion() + ", comment:" + version.getComment() );
    }
}
```

Naviguer dans les liasses

Récupérer l'ensemble des liasses à la racine du serveur. L'exemple présente la classe _BundleCollection.

```
public void document_getChildBundles( IDocumentManagementModule module, IContext context,
  IBundle bundle ) throws Exception
{
    // récupération des sous-liasses
    _BundleCollection bundles = module.GetChildBundles( context, bundle );
    for ( int index = 0 ; index < bundles.getCount() ; index++ )
    {
            IBundle child = bundles.getItem( index );

            // récupération de la version associée à la sous-liasse
            IVersion version = child.getVersion();
```

```
            // récupération de la resource associée à la version
            IResource resource = version.getResource();

            LOGGER.info( "bundle:" + resource.getId().getAsString() + ", reference:"
              + version.getReference() + ", title:" + version.getTitle() + ", version:"
              + version.getVersion() + ", comment:" + version.getComment() );
    }
}
```

Tâches sur les fiches

Récupérer la fiche racine

Depuis le module, il est possible de récupérer la fiche racine.

```
public void data_getRootForm( IDocumentManagementModule module, IContext context ) throws
Exception
{
    // récupérer l'élément racine des données
    IDataForm form = module.getRootDataForm();
}
```

Récupérer une fiche type menu

Exemple de navigation dans les fiches et récupération d'une fiche menu.

```
public void data getMenuForm( IDocumentManagementModule module, IContext context ) throws
Exception
{
    IDataForm form = module.getRootDataForm();
    IMenuDataForm menuForm = module.GetMenuDataFormByName( form, "Menu SDK" );
}
```

Récupérer une fiche type standard

Exemple de navigation dans les fiches et récupération d'une fiche standard.

```
public void data getSimpleForm( IDocumentManagementModule module, IContext context ) throws
Exception
{
    IDataForm form = module.getRootDataForm();
    ISimpleDataForm simpleForm = module.GetSimpleDataFormByName( form, "Form SDK" );
}
```

Récupérer une fiche type vue

Exemple de navigation dans les fiches et récupération d'une fiche view.

```
public void data_getViewForm( IDocumentManagementModule module, IContext context ) throws
Exception
{
    IDataForm form = module.getRootDataForm();
    IViewDataForm viewForm = module.GetViewDataFormByName( form, "View SDK" );
}
```

Récupérer des sous-fiches d'une fiche

Exemple de navigation dans les sous-fiches.

```
public void data getForms( IDocumentManagementModule module, IContext context ) throws Exception
{
    _DataFormCollection forms = module.getRootDataForm().getDataForms();
    for ( int index = 0 ; index < forms.getCount() ; index++ )
    {
            IDataForm form = forms.getItem( index );
            if ( form.getIsMenu() ) // est une fiche menu
```

```
            {
                    IMenuDataForm menuDataForm = form.getMenuDataForm();

                    // récupérer les fiches filles (standard, menu ou vue)
                    _DataFormCollection dataForms = menuDataForm.getDataForm().getDataForms();
            }
            else if ( form.getIsSimple() ) // est une fiche standard
            {
                    ISimpleDataForm simpleDataForm = form.getSimpleDataForm();

                    // récupérer les fiches filles (standard, menu ou vue)
                     DataFormCollection dataForms = simpleDataForm.getDataForm().getDataForms();

                    // récupérer les données d'une fiche standard
                     DataUnitCollection dataUnits = simpleDataForm.getDataUnits();
            }
            else if ( form.getIsView() ) // est une fiche vue
            {
                    IViewDataForm viewDataForm = form.getViewDataForm();

                    // récupérer les fiches filles (standard, menu ou vue)
                    _DataFormCollection dataForms = viewDataForm.getDataForm().getDataForms();

                    // récupérer les données d'une fiche vue
                     DataUnitCollection dataUnits = viewDataForm.getDataUnits();
            }
    }
}
```

Tâches sur les données

Récupérer les données d'une fiche

Exemple de récupération des données d'une fiche. Une donnée peut être manipulée comme une ressource.

```
public void data getData( IDocumentManagementModule module, IContext context,
  ISimpleDataForm form ) throws Exception
{
    _DataUnitCollection dataUnits = module.GetDataUnits( form );
    for ( int index = 0 ; index < dataUnits.getCount() ; index++ )
    {
            IDataUnit dataUnit = dataUnits.getItem( index );

            IResource resource = dataUnit.getResource();

            LOGGER.info( "data:" + resource.getId().getAsString() );
    }
}
```

Créer une donnée

Exemple de création d'une donnée sous une fiche standard.

```
public void data createData( IDocumentManagementModule module, IContext context, ISimpleDataForm
form ) throws Exception
{
    // création d'une donnée à partir d'une fiche standard
    IDataUnit dataUnit = module.CreateDataUnit( form );

    IResource resource = dataUnit.getResource();

    resource.SetValueAsFloat( "bigint", 1 );
    resource.SetValueAsBool( "bit", true );
    resource.SetValueAsString( "char", "a" );
    resource.SetValueAsDate( "datetime", new Date() );
    resource.SetValueAsFloat( "decimal", (float)12.500 );
    resource.SetValueAsFloat( "float", (float)7.5 );
    resource.SetValueAsString( "nchar", "AZERTY" );

    resource.Save();
}
```

Récupérer les versions d'une donnée

Exemple de récupération des versions d'une donnée.

```
public void data_getDataVersions( IDocumentManagementModule module, IContext context,
  IDataUnit dataUnit ) throws Exception
{
    _VersionCollection versions = dataUnit.getVersions();
    for ( int index = 0 ; index < versions.getCount() ; index++ )
    {
            IVersion version = versions.getItem( index );
            LOGGER.info( "version:" + version.getResource().getId().getAsString()
              + ", reference:" + version.getReference() + ", title:" + version.getTitle()
              + ", version:" + version.getVersion() + ", comment:" + version.getComment() );
    }
}
```

Tâches sur l'annuaire

Récupérer les attributs d'une donnée

Exemple de récupération des attributs d'une donnée.

```
public void data_getDataAttributes( IDocumentManagementModule module, IContext context,
  IDataUnit dataUnit ) throws Exception
{
     PropertyCollection properties = module.GetPropertiesBySimpleDataForm(
      dataUnit.getParentDataForm() );
    for ( int index = 0 ; index < properties.getCount() ; index++ )
    {
            IProperty property = properties.getItem( index );

            if ( property.getType() == property.getPropertyType().getSTRING() )
                    LOGGER.info( "attribute '" + property.getname() + "' = " +
                      dataUnit.getResource().GetValueAsString( property.getname() ) );
    }
}
```

Récupérer un utilisateur de la gestion documentaire

Exemple de récupération d'un utilisateur.

```
public void directory_getUser( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IUser user = module.GetUserByLogin( "zorgly" );
}
```

Récupérer un utilisateur en tant qu'opérateur

Exemple de récupération d'un utilisateur en tant qu'opérateur sur une étape.

```
public void directory getUserAsOperator( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IUser user = module.GetUserByLogin( "sboirol" );

    // transformation d'un utilisateur en opérateur
    IOperator operator = user.getAsOperator();
}
```

Récupérer l'ensemble des utilisateurs

Exemple de récupération des utilisateurs de la gestion documentaire.

```
public void directory_getUsers( IDocumentManagementModule module, IContext context )
  throws Exception
```

```
{
    // thanks to François Edom!
     UserCollection users = module.getUsers();
    for ( int index = 0 ; index < users.getCount() ; index++ )
    {
            IUser user = users.getItem( index );
            LOGGER.info( "user:" + user.getId().getAsString() + ", login:" + user.getlogin() +
              ", first name:" + user.getFirstName() + ", last name:" + user.getLastName()
              + ", email:" + user.getEMail() );
    }
}
```

Récupérer un groupe de la gestion documentaire

Exemple de récupération d'un groupe.

```
public void directory getGroup( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IGroup group = module.GetGroupById( 123 );
}
```

Récupérer un groupe en tant qu'opérateur

Exemple de récupération d'un groupe en tant qu'opérateur sur une étape.

```
public void directory_getGroupAsOperator( IDocumentManagementModule module, IContext context )
  throws Exception
{
    IGroup group = module.GetGroupById( 123 );

    // transformation d'un groupe en opérateur
    IOperator operator = group.getAsOperator();
}
```

Chapitre 19 : Référence de la gestion des forums

L'API SDK intègre un module de gestion de forums qui permet de manipuler, directement depuis des extensions ou tout autre système au sein du serveur VDoc, les éléments des forums

Cette couche simplifie la réalisation de tâches telles que :

- la création de forums ;
- la création de discussions et posts ;
- la gestion de la sécurité.

Quelques définitions

Eléments de définition

De nombreux éléments sont implantés dans la base VDoc. Parmi ces éléments de définition, nous retiendrons les suivants :

- IForumSpace : espace dans lequel peuvent être créés plusieurs forums ;
- IForum : forum dédié à une thématique

Diagramme de classes des éléments de définition

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes descriptives du système de gestion des forums.

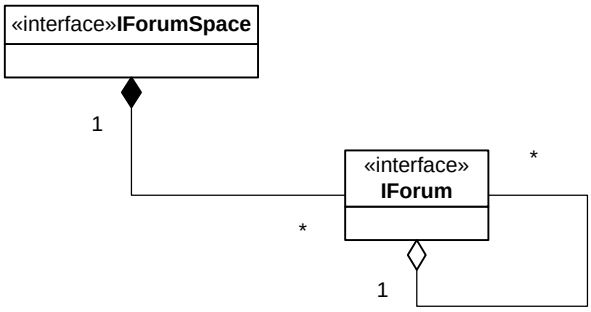


Tableau de correspondance des éléments de définition

Terme fonctionnel	Terme technique	Terme SDK	Terme natif	Description
Espace forums	Espace forums	IForumSpace	ForumSpace	Elément regroupant plusieurs forums
Forum	Forum	IForum	ForumNode	Elément regroupant un ensemble de discussions autour d'une thématique.

Eléments dynamiques

A la création d'une discussion, plusieurs éléments sont inscrits en base. Parmi ces éléments dynamiques, nous retiendrons les suivants :

- IDiscussion : sujet de discussion d'un forum ;
- IPost : réponse d'une discussion ;
- IAttachment : pièce jointe d'une discussion ou d'un post.

Diagramme de classes des éléments dynamique

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes descriptives du système de gestion des forums.

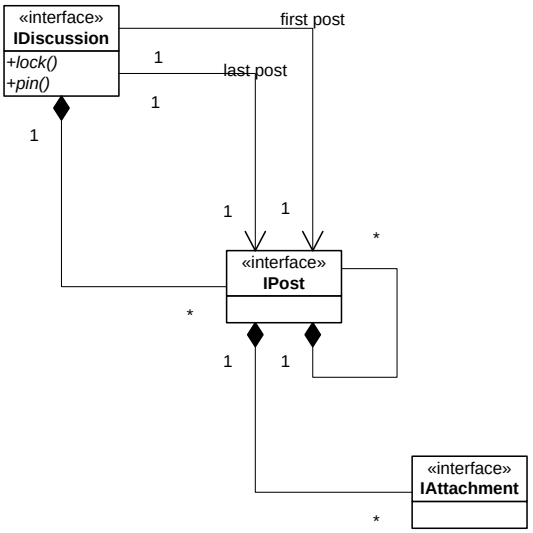


Tableau de correspondance des éléments dynamiques

Terme fonctionnel	Terme technique	Terme SDK	Terme natif	Description
Sujet, Discussion	Discussion	IDiscussion	ThreadNode	Elément regroupant un sujet associé à un ensemble de réponses.
Réponse, Post	Post	IPost	PostNode	Elément de réponse à une discussion ou un post.

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc.

Pour des raisons de clarté, mais aussi pour marquer les différents niveaux d'API, nous avons décidé de classer ces développements en plusieurs sous-sections.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module de forums des API SDK Récupérer un contexte pour utiliser les API SDK
Serveur	Récupérer les espaces forums Récupérer un espace forums par son nom système
Espace forums	Récupérer les forums d'un espace Récupérer un forum d'un espace Créer un forum dans un espace
Discussions	Récupérer les discussions d'un forum Récupérer une discussion d'un forum Récupérer les posts d'une discussion Créer une discussion Répondre à une discussion Punaiser une discussion Verrouiller une discussion

Tâches générales

Utiliser le module de forums des API SDK

Le code suivant indique comment utiliser l'API forum. La méthode initialize() est utilisée pour positionner un certain nombre de paramètres.

```
public void general_useModule()
{
     // création d'un object de module de Forum
     IForumModule forumModule = new PortalForumModule();
     try
     {
            // initialisation du module
            forumModule.initialize( null );
     }
     catch( Exception e )
     {
            // si une erreur s'est produite
            getNavigator().processErrors( e );
     }
     finally
     {
            forumModule.unInitialize();
     }
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general_getContext( IForumModule forumModule ) throws Exception
{
     // 1er cas : récupération du contexte à partir d'un objet utilisateur
     IUser user = forumModule.getUserByLogin( "sysadmin" );
     IContext userContext = forumModule.getContext( user );

     // 2ème cas : récupération du contexte à partir d'un ID externe
     IContext externalIDContext = forumModule.getContext(
       getNavigator().getLoggedOnUser().getExternId() );

     // 3ème cas : récupération du contexte à partir d'un login
     IContext loginContext = forumModule.getContextByLogin( "sysadmin" );
}
```

Tâches serveur

Récupérer les espaces forums

Exemple de parcours des espaces forums.

```
public void server getForumSpaces( IForumModule module, IContext context ) throws Exception
{
    // retrouver l'organization de l'utilisateur connecté
    IOrganization organization = context.getUser().getOrganization();

    // retrouver l'ensemble des espaces forum de l'organisation
    Collection forumSpaces = module.getForumSpaces( context, organization );
    for ( Iterator iterator = forumSpaces.iterator() ; iterator.hasNext() ; )
    {
            IForumSpace forumSpace = (IForumSpace)iterator.next();
            System.out.println( "Espace forum : " + forumSpace.getName() );
    }
}
```

Récupérer un espace forums par son nom système

Exemple de récupération d'un espace forum.

```
public void server_getForumSpace( IForumModule module, IContext context, String forumName )
  throws Exception
{
    // retrouver l'organization de l'utilisateur connecté
    IOrganization organization = context.getUser().getOrganization();

    // retrouver un espace forum par son nom
    IForumSpace forumSpace = module.getForumSpace( context, organization, forumName );
    System.out.println( "Espace forum : " + forumSpace.getName() );
}
```

Tâches sur les espaces forums

Récupérer les forums d'un espace

Exemple de parcours des forums d'un espace.

```
public void forum getForums( IForumModule module, IContext context, IForumSpace forumSpace )
  throws Exception
{
    // retrouver l'ensemble des forums d'un espace de forums
    Collection forums = module.getForums( context, forumSpace );
    for ( Iterator iterator = forums.iterator() ; iterator.hasNext() ; )
    {
            IForum forum = (IForum)iterator.next();
            System.out.println( "Forum : " + forum.getLabel() );
    }
}
```

Récupérer un forum d'un espace

Exemple de récupération d'un forum.

```
public void forum_getForum( IForumModule module, IContext context, IForumSpace forumSpace )
  throws Exception
{
    // retrouver un forum particulier d'un espace de forums
    IForum forum = module.getForum( context, forumSpace, "Les forums SDK" );
    System.out.println( "Forum : " + forum.getLabel() );
}
```

Créer un forum dans un espace

Exemple de création d'un forum.

```
public void forum_createForum( IForumModule module, IContext context, IForumSpace forumSpace )
  throws Exception
{
    // créer un forum dans un espace de forums
    IForum forum = module.createForum( context, forumSpace, "Les forums SDK",
      "Discussion sur les forums SDK" );
    System.out.println( "Forum : " + forum.getLabel() );
}
```

Tâches sur les discussions

Récupérer les discussions d'un forum

Exemple de parcours des discussions d'un forum.

```
public void forum getDiscussions( IForumModule module, IContext context, IForum forum )
  throws Exception
{
    // retrouver l'ensemble des discussions d'un forum
    Collection discussions = module.getDiscussions( context, forum );
    for ( Iterator iterator = discussions.iterator() ; iterator.hasNext() ; )
    {
            IDiscussion discussion = (IDiscussion)iterator.next();
    }
}
```

Récupérer une discussion d'un forum

Exemple de récupération d'une discussion d'un forum.

```
public void forum_getDiscussion( IForumModule module, IContext context, IForum forum )
  throws Exception
{
    IDiscussion discussion = module.getDiscussion( context, forum,
      "Cas d'intégration avec les forums" );
}
```

Créer une discussion

Exemple de création d'une discussion.

```
public void forum createDiscussion( IForumModule module, IContext context, IForum forum )
  throws Exception
{
    IDiscussion discussion = module.createDiscussion( context, forum,
      "Cas d'intégration avec les forums",
      "Comment peut-on intégrer les forums dans les documents processus ?" );
}
```

Récupérer les posts d'une discussion

Exemple de récupération des réponses d'une discussion.

```
public void forum_getPosts( IForumModule module, IContext context, IDiscussion discussion )
  throws Exception
{
    Collection posts = module.getPosts( context, discussion );
    for ( Iterator iterator = posts.iterator() ; iterator.hasNext() ; )
    {
            IPost post = (IPost)iterator.next();
            System.out.println( "Post : " + post.getLabel() );
    }
}
```

Répondre à une discussion

Exemple de réponse à une discussion ou un post. Cet exemple présente la méthode **getFirstPost**() et la possibilité de spécifier une réponse en privé.

```
public void forum reply( IForumModule module, IContext context, IDiscussion discussion )
  throws Exception
{
    // répondre à un début de discussion
    IPost post = module.replyTo( context, discussion, discussion.getFirstPost().getLabel(),
      "Pourquoi pas, très bonne idée." );

    // répondre à une réponse en privé
    IPost repliedPost = module.replyTo( context, post, post.getLabel(),
      "Oui, mais c'est peut-être un peu chaud, non ?" );
    repliedPost.setPrivate( true );
```

Punaiser une discussion

Exemple permettant de gérer les punaises sur les discussions.

```
public void forum_pinDiscussion( IForumModule module, IContext context, IDiscussion discussion )
  throws Exception
{
    module.pin( context, discussion );
    if ( discussion.isPinned() )
           module.unPin( context, discussion );
}
```

Verrouiller une discussion

Exemple permettant de gérer les verrous sur les discussions.

```
public void forum lockDiscussion( IForumModule module, IContext context, IDiscussion discussion )
  throws Exception
{
    module.lock( context, discussion );
    if ( discussion.isLocked() )
           module.unLock( context, discussion );
}
```

Chapitre 20 : Référence de gestion de sites

L'API SDK intègre un module de gestion de sites qui permet de manipuler, directement depuis les extensions ou tout autre système, les éléments de site.

Cette couche simplifie la réalisation de tâches telles que :

- la création de sites, de rubriques, de pages, d'alias ;
- la construction de page ;
- la gestion de la sécurité.

Quelques définitions

Eléments

A l'utilisation du module de sites, de nombreux éléments sont implantés dans la base VDoc. Parmi ces éléments, nous retiendrons les suivants :

- ISite : site ;
- ITopic : rubrique conenant des pages, des alias ou des sous-rubriques ;
- IPageContainer : page intégrant plusieurs versions (versions brouillon et approuvée) ;
- ISharedBlockContainer : bloc partagé ;
- IBlockTemplateContainer : modèle de blocs ;
- IContent : version d'une page. Celle-ci contient un bloc racine qui, lui-même, contient des composants et des sous-blocs ;
- IBlock : partie d'une page contenant des composants ou des sous-blocs.
- IComponent : élément unitaire pouvant être ajouté à un bloc.

Diagramme de classes des éléments

Ce diagramme de classes représente les relations principales qui existent entre toutes les classes du système de gestion de sites.

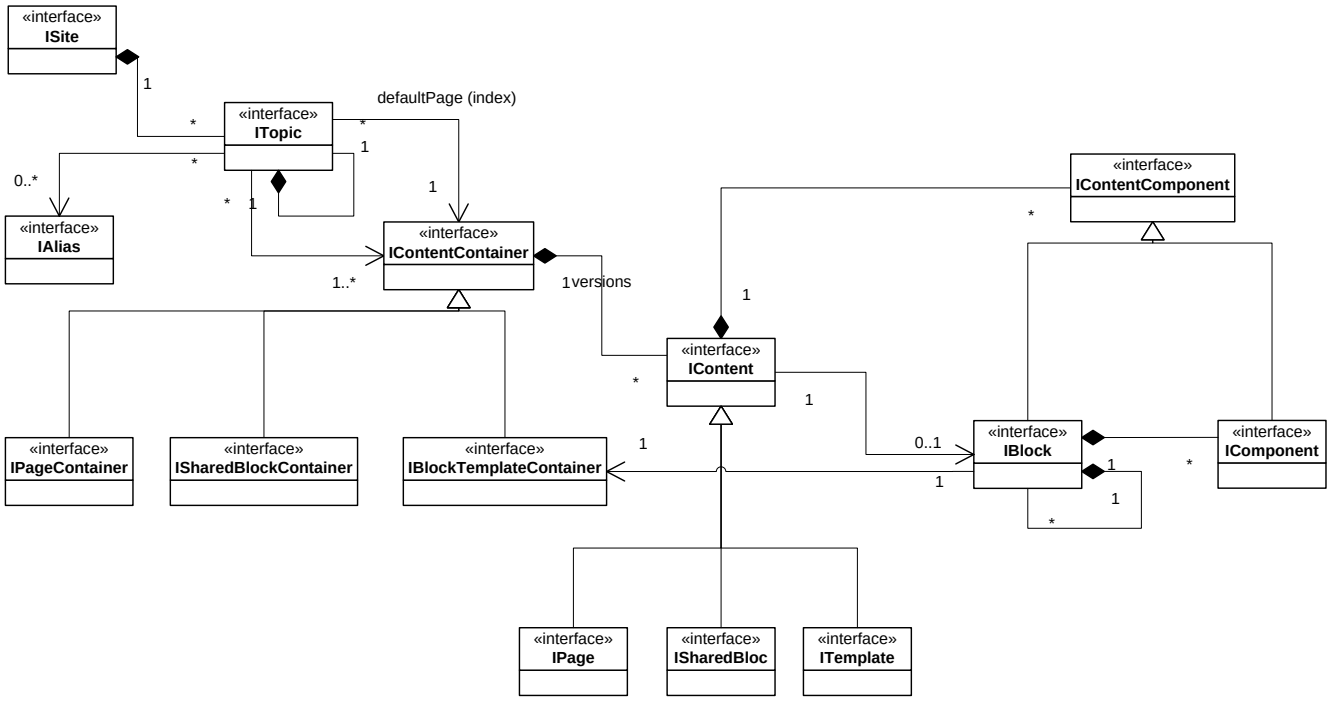


Tableau de correspondance des éléments

Terme fonctionnel	Terme technique	Terme SDK	Terme natif	Description
Site	Site	ISite	SiteImpl	Elément constituant un site.
Rubrique, dossier	Topic	ITopic	FolderImpl	Elément regroupant un ensemble de pages, de sous-rubriques, d'alias..
Page	Page	IPageContainer	PageImpl	Elément constituant une page.
Bloc partagé	Bloc partagé	ISharedBlockContainer	CommonBlockImp	Elément constituant un bloc partagé.
Modèle de bloc	Modèle de bloc	IBlockTemplateContainer	TemplateImpl	Elément pouvant être réutilisé et configuré au sein de plusieurs blocs ou pages.
Contenu d'une page	Content	IContent	ContentVersionImpl	Elément constituant une version particulière d'une page (ex. brouillon).
Bloc	Block	IBlock	ContentVersionImpl	Elément composé de composants et sous-blocs.
Composant, élément	Composant	IComponent	ElementImpl	Elément unitaire pouvant être ajouté au sein d'un bloc.

Tâches de programmation

Cette section décrit certains développements qui peuvent être réalisés à l'aide des API SDK de VDoc.

Pour des raisons de clarté, mais aussi pour marquer les différents niveaux d'API, nous avons décidé de classer ces développements en plusieurs sous-sections.

Tableaux récapitulatif des tâches de programmation

Catégorie	Tâches de programmation
Générale	Utiliser le module de site via l'API SDK Récupérer un contexte pour utiliser les API SDK Utiliser les transactions
Serveur	Créer un site Récupérer un site Supprimer un site
Site	Créer une rubrique Récupérer une rubrique Récupérer des rubriques d'un site ou d'une rubrique Créer d'une page Récupérer une page Récupérer des pages d'un site ou d'une rubrique
Page	Récupérer le contenu d'une page Récupérer le bloc principal d'un contenu de page Ajouter des composants au bloc d'une page Ajouer des blocs au bloc d'une page Approuver une page
Récipient universel	Déclarer un récipient Supprimer un récipient nommé Envoyer un message à un récipient nommé Réceptionner un message envoyé Positionner des informations pour le référencement depuis une extension de page Positionner des informations pour le référencement depuis un plugin

Tâches générales

Utiliser le module de site via l'API SDK

Le code suivant indique comment utiliser l'API de gestion de site.

```
public void general useModule()
{
     // création d'un object de module de site
     ISiteModule siteModule = (ISiteModule)Modules.getSiteModule();
     try
     {}
     catch( Exception e )
     {
            // si une erreur s'est produite
            getNavigator().processErrors( e );
     }
     finally
     {
            Modules.releaseModule( siteModule );
     }
}
```

Récupérer un contexte pour utiliser les API SDK

La notion de contexte permet d'exécuter des traitements avec le compte d'un utilisateur. Plusieurs cas sont présentés. Leur utilisation dépend du contexte d'exécution et des objets disponibles.

```
public void general getContext( ISiteModule siteModule ) throws Exception
{
     // 1er cas : récupération du contexte à partir d'un objet utilisateur
     IUser user = null;
     IContext userContext = siteModule.getContext( user );

     // 2ème cas : récupération du contexte à partir d'un ID externe
     IContext externalIDContext = siteModule.getContext( siteModule.getLoggedOnUser() );

     // 3ème cas : récupération du contexte à partir d'un login
     IContext loginContext = siteModule.getContextByLogin( "sysadmin" );
}
```

Utiliser les transactions

Dans de nombreux cas, il est nécessaire de rassembler plusieurs traitements dans une même transaction. Chaque module possède son propre mécanisme transactionnel.

L'exemple suivant montre comment utiliser ce mécanisme transactionnel depuis le module de site.

```
public void general useTransaction( IContext context, ISiteModule siteModule )
{
     try
     {
            // démarrage d'une transaction
            siteModule.beginTransaction( this );

            // réaliser quelque chose
            // ...

            // validation de la transaction
            siteModule.commitTransaction( this );
     }
     catch( Exception e )
     {
            // si une erreur s'est produite, annuler le traitement précédent
            siteModule.rollbackTransaction( this );
     }
}
```

Tâches serveur

Créer un site

Un site ne peut être créé qu'au sein d'une organisation. Par ailleurs, la méthode « createSite » construit et associe au site un espace documentaire qui regroupera toutes les images, fichiers.

```
public void server createSite( ISiteModule siteModule, IContext context ) throws
SiteModuleException
{
    // création d'un site
    IOrganization organization = null;
    ISite site = siteModule.createSite( context, organization, "test-site" );

    // création d'un site en spécifiant la bibliothèque à utiliser
    ILibrary library = null;
    site = siteModule.createSite( context, library, "test-site" );
}
```

Récupérer un site

La récupération d'un site s'effectue à l'aide du nom système.

```
public void server getSite( ISiteModule siteModule, IContext context ) throws SiteModuleException
{
    // récupération d'un site
    ISite site = siteModule.getSiteByName( context, "test-site" );
    System.out.println( site.getName() );
}
```

Supprimer un site

La suppression d'un site peut s'effectuer de deux façons :

- suppression du site complet avec l'espace documentaire associé ;
- suppression unique du site.

```
public void server_deleteSite( ISiteModule siteModule, IContext context ) throws
SiteModuleException
{
    // récupération d'un site
    ISite site = siteModule.getSiteByName( context, "test-site" );

    // 1er cas : suppression définitive du site complet
    site.delete( context );

    // 2ème cas : suppression du site et maintient de l'espace documentaire associé
    site.delete( context, false );
}
```

Tâches sur le site

Créer une rubrique

Les rubriques peuvent être créées soit à la racine du site soit dans une rubrique parent.

```
public void site createTopic( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // 1er cas : création d'une rubrique à la racine d'un site
    ITopic topic = siteModule.createTopic( context, site, "test-topic", "test topic label" );

    // 2ème cas : création d'une rubrique dans une rubrique parent
    ITopic parentTopic = null;
    topic = siteModule.createTopic( context, parentTopic, "test-topic", "test topic label" );
```

```
}
```

Récupérer une rubrique

La récupération d'une rubrique racine ou sous rubrique s'effectue à l'aide du nom système de la rubrique.

```
public void site getTopic( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // récupération d'une rubrique depuis un site
    ITopic topic = siteModule.getTopic( context, site, "test-topic" );

    // récupération d'une rubrique depuis une rubrique parent
    ITopic parentTopic = null;
    topic = siteModule.getTopic( context, parentTopic, "test-topic" );

    System.out.println( topic.getName() );
}
```

Récupérer des rubriques d'un site ou d'une rubrique

Le module de site prévoie de pouvoir récupérer les rubriques racines ou les sous-rubriques d'une rubrique particulière.

```
public void site getTopics( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // récupération des rubriques racines d'un site
    Collection topics = siteModule.getTopics( context, site );
    for ( Iterator iterator = topics.iterator() ; iterator.hasNext() ; )
    {
           ITopic topic = (ITopic)iterator.next();
           System.out.println( topic.getName() );
    }

    // récupération des rubriques racines d'une rubrique parent
    ITopic parentTopic = null;
    topics = siteModule.getTopics( context, parentTopic );
    for ( Iterator iterator = topics.iterator() ; iterator.hasNext() ; )
    {
           ITopic topic = (ITopic)iterator.next();
           System.out.println( topic.getName() );
    }
}
```

Créer d'une page

Les pages peuvent être créées directement à la racine d'un site ou sous une rubrique particulière.

```
public void site createPage( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // 1er cas : création d'un page à la racine du site
    IPageContainer sitePage = siteModule.createPageContainer(context,site,"site-page","My page");

    ITopic topic = null;
    // 2ème cas : création d'une page sous une rubrique
    IPageContainer topicPage = siteModule.createPageContainer(context,topic,"topic-page","page");
}
```

Récupérer une page

Le module de gestion de site prévoie de pouvoir récupérer une page à partir de la racine du site ainsi que depuis une rubrique.

```
public void site getPage( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // 1er cas : récupérer une page depuis la racine d'un site
```

```
    IPageContainer pageContainer = siteModule.getPageContainer( context, site, "site-page" );

    ITopic topic = null;
    // 2ème cas : récupérer une page depuis une rubrique
    IPageContainer childPageContainer = siteModule.getPageContainer(context,topic,"topic-page" );
}
```

Récupérer des pages d'un site ou d'une rubrique

Le module de gestion de site permet de retrouver toutes les pages présentes à la racine d'un site ainsi que celle présentes dans une sous-rubrique.

```
public void site getPages( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // récupérer les pages présentes à la racine d'un site
    Collection pageContainers = siteModule.getPageContainers( context, site );
    for ( Iterator iterator = pageContainers.iterator() ; iterator.hasNext() ; )
    {
            IPageContainer pageContainer = (IPageContainer)iterator.next();
            System.out.println( pageContainer.getLabel() );
    }

    // récupérer une rubrique
    ITopic topic = siteModule.getTopic( context, site, "my-lovely-topic" );

    // récupérer les pages présentes à la racine d'une rubrique
    pageContainers = siteModule.getPageContainers( context, topic );
    for ( Iterator iterator = pageContainers.iterator() ; iterator.hasNext() ; )
    {
            IPageContainer pageContainer = (IPageContainer)iterator.next();
            System.out.println( pageContainer.getLabel() );
    }
}
```

Tâches sur la page

Récupérer le contenu d'une page

A tout moment, il est possible de récupérer le contenu d'une page. A ce jour, une page peut contenir au maximum deux versions :

- version approuvée disponible pour les utilisateurs du site ;
- version brouillon, diponible pour le concepteur de la page.

A terme, d'autres versions compléteront cette liste (ex : version archivée).

L'exemple suivant montre comment récupérer une version particulière d'une page.

```
public void page_getPageContent( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    IPageContainer pageContainer = siteModule.getPageContainer( context, site, "my-lovely-page"
);

    // 1er cas : récupérer la version brouillon d'une page
    IContent draftContent = pageContainer.getContent( IContent.IStatus.DRAFT );

    // 2ème cas : récupérer la version approuvée d'une page
    IContent approuvedContent = pageContainer.getContent( IContent.IStatus.APPROVED );
}
```

Récupérer le bloc principal d'un contenu de page

L'exemple suivant montre comment récupérer le bloc racine d'une page. A l'aide de ce bloc, ou tout autre élément de l'arborescence il sera possible de réaliser des traitements tels que :

- l'ajout ou la construction de composants ou sous-blocs ;
- le masquage d'éléments de l'arborescence.

```
public void page_getPageContentBlock( ISiteModule siteModule, IContext context, ISite site )
throws SiteModuleException
{
    IContent content = null;

    // récupérer le bloc principal d'une version
    IBlock block = content.getBlock();
    System.out.println( content.getOwner().getFullName() );
}
```

Ajouter des composants au bloc d'une page

Depuis une page d'extension, il est possible d'ajouter dynamiquement des composants. Cet exemple montre comment utiliser « l'usine de composants » pour ajouter un titre de niveau 1 au bloc racine de la page.

```
public void page completeBlockWithComponents( ISiteModule siteModule, IContext context, ISite site
) throws SiteModuleException
{
    // récupérer la version d'une page
    IContent content = null;

    // utiliser l'usine à composants pour obtenir un composant de titre
    HeadingComponent headingComponent = siteModule.getComponentsFactory().newHeadingComponent();
    headingComponent.setContent( "Hello everyone!!!" );
    headingComponent.setLevel( "1" );

    // ajouter le composant au bloc principal de la page
    content.getBlock().addComponent( headingComponent );
}
```

Ajouer des blocs au bloc d'une page

Depuis une page d'extension, il est possible d'ajouter dynamiquement des blocs, sous-blocs. Cet exemple montre comment utiliser « l'usine de blocs » pour ajouter un formulaire de site au bloc racine de la page.

```
public void page completeBlockWithBlocks( ISiteModule siteModule, IContext context, ISite site )
throws SiteModuleException
{
    // récupérer la version d'une page
    IContent content = null;

    // utiliser l'usine à blocs pour obtenir un bloc type formulaire
    FormBlock formBlock = siteModule.getBlocksFactory().newFormBlock();
    formBlock.setName( "frmContacts" );
    formBlock.setLabel( "Les contacts" );

    // créer un bloc de champs
    FieldsetBlock fieldsetBlock = siteModule.getBlocksFactory().newFieldsetBlock();

    // créer deux champs de saisie type texte
    TextInputComponent firstName = siteModule.getComponentsFactory().newTextInputComponent();
    firstName.setLabel( "First name" );
    firstName.setName( "fldFirstName" );

    TextInputComponent lastName = siteModule.getComponentsFactory().newTextInputComponent();
    lastName.setLabel( "Last name" );
    lastName.setName( "fldLastName" );

    // ajouter les champs au bloc
    fieldsetBlock.addComponent( firstName );
    fieldsetBlock.addComponent( lastName );

    // créer deux boutons de validation du formulaire
    ButtonInputComponent createButton =
           siteModule.getComponentsFactory().newButtonInputComponent();
    createButton.setLabel( "Create" );
    createButton.setValue( "type", ButtonInputComponent.Action.SAVEANDCLOSE );

    ButtonInputComponent resetButton =
           siteModule.getComponentsFactory().newButtonInputComponent();
```

```
    resetButton.setLabel( "Reset" );
    resetButton.setValue( "type", ButtonInputComponent.Action.RESET );

    // ajouter le bloc de champs au formulaire
    formBlock.addBlock( fieldsetBlock );

    // ajouter les boutons sur le bloc de formulaire
    formBlock.addComponent( createButton );
    formBlock.addComponent( resetButton );

    // ajouter le bloc de formulaire au bloc principal de la page
    content.getBlock().addBlock( formBlock );

    // enregistrer les modifications (contexte transactionnel nécessaire)
    content.save( context );
}
```

Approuver une page

Chaque page est liée à un cycle de validation. Après modification de la version « brouillon » d'une page il est nécessaire d'approuver page pour que cette dernière puisse être consultée par les utilisateurs du site. Si toutefois, l'utilisateur approuvant la page n'a pas suffisamment de droits il doit passer par une demande d'approbation. Le responsable pourra alors approuver ou rejettér la demande.

L'exemple suivant illustre le cycle de validation d'une page.

```
public void page_approuvePage( ISiteModule siteModule, IContext context, ISite site ) throws
SiteModuleException
{
    // récupérer une page à approuver
    IPageContainer pageContainer = siteModule.getPageContainer( context, site, "my-lovely-page"
);

    // demander l'approbation d'une page
    siteModule.requestApproval( context, pageContainer, "Please..." );

    // approuver une page
    siteModule.approve( context, pageContainer );

    // rejeter l'approbation
    siteModule.reject( context, pageContainer, "No way." );
}
```

Tâches dynamiques

Déclarer un récipient

Deux natures de récipient peuvent être déclarées :

- Récipient simple : toute extension de page peut recevoir le message envoyé ;
- Récipient avec classe associée : seul la classe associée recevra le message envoyé.

Classe récipient

Voici un exemple d'une classe représentant un récipient universel. Celle-ci dérive d'une classe de base nommée BaseRecipient. Elle doit impérativement implémenter la méthode onMessage.

```
public class AnyRecipient extends BaseRecipient
{
    interface SupportedEventTypes
    {
            int ADD = 0;
            int REMOVE = 1;
            int CLEAR = 2;
    }

    public boolean onMessage( IMessage message )
    {
            switch ( message.getEventType() )
            {
                    case SupportedEventTypes.ADD :
                            // TODO : do something while adding
                            break;
                    case SupportedEventTypes.REMOVE :
                            // TODO : do something while removing
                            break;
                    case SupportedEventTypes.CLEAR :
                            // TODO : do something while clearing
                            break;
                    default :
                            // TODO : do something for default case
                            break;
            }
            return true;
    }
}
```

Dans cet exemple nous montrons comment déclarer les deux natures de récipient.

```
public void dynamic_declareRecipient( ISiteModule siteModule, IContext context ) throws
SiteModuleException
{
    // déclarer un récipient simple
    siteModule.getMessageController().registerRecipient( "simple-recipient" );

    // déclarer un récipient associé à une classe
    AnyRecipient anyRecipient = (AnyRecipient)
      siteModule.getMessageController().registerRecipient("any-recipient",AnyRecipient.class );
}
```

Supprimer un récipient nommé

Il est possible de supprimer un récipient à tout moment.

```
public void dynamic_deleteRecipient( ISiteModule siteModule, IContext context ) throws
SiteModuleException
{
    // annuler la déclaration d'un récipient
    siteModule.getMessageController().unregisterRecipient( "simple-recipient" );
}
```

Envoyer un message à un récipient nommé

Les messages peuvent être envoyés depuis une extension de page, un provider ou tout autre système tel que les entensions dynamique du module de workflow.

L'exemple suivant montre comment envoyer un message depuis un « Listener » associé à un bouton.

```
public void dynamic sendRecipientMessage( ISiteModule siteModule, IContext context ) throws
SiteModuleException
{
    CtlButton addButton = new CtlButton( "add", new CtlText( "Add to basket" ) );
    addButton.addActionListener( new ActionListener()
    {
      public void onClick( ActionEvent event )
      {
            try
            {
              // envoyer un message à un récipient depuis un provider
              getSiteModule().getMessageController().sendMessage( "simple-recipient", 2, null );
            }
            catch( SiteModuleException e )
            {
                  Navigator.getNavigator().processErrors( e, true );
            }
      };
    } );
}
```

Réceptionner un message envoyé à un récipient

Pour réceptionner un message envoyé à un récipient, il suffit de d'implémenter la méthode onMessage(). Celle-ci est obligatoire dans le cas d'une classe dérivant de BaseRecipient, mais peut être implémentée dans le cas d'une extension de page.

L'exemple suivant montre comment extraire depuis le message le type de l'événement ainsi que son contenu (body).

```
public boolean onMessage( IMessage message )
{
    switch ( message.getEventType() )
    {
            case BasketHandler.IEventTypes.ADD :
                  this.add( message.getBody() );
                  break;
            case BasketHandler.IEventTypes.REMOVE :
                  this.remove( message.getBody() );
                  break;
            case BasketHandler.IEventTypes.CLEAR :
                  this.clear();
                  break;
            case BasketHandler.IEventTypes.VALIDATE :
                  this.validate();
                  break;
    }
    return true;
}
```

Positionner des informations pour le référencement depuis une extension de page

L'exemple suivant montre comment, depuis une classe d'extension de page, positionner des informations relatives au référencement.

```
public boolean onBeforeLoad()
{
 SEO seo = this.getExecutionContext().getPageContainer().getSEO();
 if ( seo != null )
 {
  // modification du titre de la page
  seo.setTitle( "Kit de développement VDoc" );

  // positionnement de quelques "meta"
  seo.addMeta( newMeta( HTMLConstants.Meta.DESCRIPTION, "Ensemble de composants logiciel" ) );
  seo.addMeta( newMeta( HTMLConstants.Meta.KEYWORDS, "api, framework, navigation" ) );
 }
 return super.onBeforeLoad();
}
```

Positionner des informations pour le référencement depuis un plugin

L'exemple suivant montre comment, depuis un contrôleur de plugin, positionner des informations relatives au référencement.

```
protected String preparePage( IPluginRequest pluginRequest, IRenderModel renderModel )
  throws Exception
{
 // création d'une structure SEO pour le référencement
 SEO seo = newSEO();

 // positionnement du titre de la page
 seo.setTitle( (String)renderModel.getValue( FIELD_NAME ) );

 // positionnement d'un certain nombre de 'meta'
 seo.addMeta( newMeta( HTMLConstants.Meta.DESCRIPTION, "Exemple d'un contrôleur de plugin" ) );
 seo.addMeta( newMeta( HTMLConstants.Meta.KEYWORDS, "file, plugin" ) );

 // stockage dde la structure SEO dans le modèle
 renderModel.setValue( IRenderModel.VirtualPageMarkers.SEO, seo );

 // positionnement du masque de rendu.
 return ( "content" );
}
```