Guide des développeurs pour les services de données Sun Cluster pour SE Solaris

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Référence : 817–6386
Mai 2004, Révision A

Copyright 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

040525@9061

Table des matières

Préface

Le *Guide des développeurs pour les services de données Sun Cluster pour SE Solaris* contient des informations sur l'utilisation de l'interface de programmation d'application de gestion des ressources pour développer des services de données Sun™ Cluster sur des systèmes basés sur SPARC® et x86.

Remarque – dans ce document, le terme "x86" fait référence à la gamme de puces microprocesseurs de la gamme 32 bits d'Intel et aux puces microprocesseurs conçues par AMD.

Remarque – le logiciel Sun Cluster fonctionne sur deux plates-formes, SPARC et x86. Les informations contenues dans ce document s'appliquent aux deux, sauf indication contraire dans un chapitre, une rubrique, une remarque, une liste à puces, une figure, un tableau ou un exemple spécifique.

Utilisateurs de ce manuel

Le présent document s'adresse à des développeurs expérimentés possédant une connaissance approfondie des logiciels et matériels Sun. Les informations qu'il contient supposent une bonne connaissance du système d'exploitation Solaris™.

Organisation de ce manuel

Le *Guide des développeurs pour les services de données Sun Cluster pour SE Solaris* contient les chapitres et les annexes suivantes :

- Le Chapitre 1 présente les concepts requis pour développer un service de données.
- Le Chapitre 2 fournit des informations détaillées sur le développement d'un service de données.
- Le Chapitre 3 offre une vision globale des connaissances requises pour mettre à niveau un type de ressources et migrer une ressource.
- Le Chapitre 4 sert de référence aux fonctions d'accès et méthodes de rappel constituant l'interface de programmation d'application de gestion des ressources (API GR).
- Le Chapitre 5 propose un exemple de service de données Sun Cluster pour l'application `in.named()`.
- Le Chapitre 6 présente brièvement les interfaces de programmation d'application constituant la bibliothèque de développement de services de données (BDSD).
- Le Chapitre 7 explique l'usage habituel de la BDSD dans la conception et la mise en oeuvre des types de ressource.
- Le Chapitre 8 présente un exemple de type de ressources mis en oeuvre avec la bibliothèque BDSD.
- Le Chapitre 9 présente SunPlex ™Agent Builder.
- Le Chapitre 10 décrit la procédure de création d'un service de données générique.
- Le Chapitre 11 présente les fonctions de l'interface de programmation d'application de la bibliothèque BDSD.
- Le Chapitre 12 présente le protocole CRNP (Cluster Reconfiguration Notification Protocol). Le protocole CRNP permet aux applications de basculement et évolutives d'être "prises en charge sur le cluster".
- L'Annexe A présente les propriétés des types de ressources standard, des groupes de ressources et des ressources.
- L'Annexe B fournit le code complet de chaque méthode du service de données échantillon.
- L'Annexe C répertorie le code complet de chaque méthode du type de ressources `SUNW.xfnts()`.
- L'Annexe D répertorie les exigences en matière de caractères légaux pour les noms et valeurs RGM (Resource Group Manager).
- L'Annexe E répertorie les exigences requises pour pouvoir conférer aux applications non cluster ordinaires un haut niveau de disponibilité.
- L'Annexe F répertorie les définitions de type de document pour le protocole CRNP.

- L'Annexe G présente l'application `CrnpClient.java` complète, décrite au Chapitre 12.

Documentation connexe

Le tableau suivant présente les manuels contenant des informations sur des sujets connexes associés à Sun Cluster. L'ensemble de la documentation Sun Cluster est disponible à l'adresse suivante : `http://docs.sun.com`.

Sujet	Documentation
Concepts	*Sun Cluster Concepts Guide for Solaris OS*
Présentation générale	*Sun Cluster Overview for Solaris OS*
Administration du matériel	*Sun Cluster 3.x Hardware Administration Manual for Solaris OS*
	Guides d'administration matérielle individuelle
Installation du logiciel	*Sun Cluster Software Installation Guide for Solaris OS*
Administration des services de données	*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*
	Guides des services de données individuels
Développement de services de données	*Sun Cluster Data Services Developer's Guide for Solaris OS*
Administration du système	*Sun Cluster System Administration Guide for Solaris OS*
Messages d'erreur	*Sun Cluster Error Messages Guide for Solaris OS*
Références sur les commandes et les fonctions	*Sun Cluster Reference Manual for Solaris OS*

Une liste complète de la documentation Sun Cluster est disponible dans les notes de version de Sun Cluster sur le site `http://docs.sun.com`.

Accès à l'aide

Si vous rencontrez des problèmes lors de l'installation ou de l'utilisation de Sun Cluster, adressez-vous à votre prestataire de services et communiquez-lui les renseignements suivants :

- votre nom et votre adresse de courrier électronique (le cas échéant) ;
- le nom, l'adresse et le numéro de téléphone de votre société ;
- les numéros de modèle et de série de vos systèmes ;
- le numéro de version du système d'exploitation (par exemple Solaris 10) ;
- le numéro de version de Sun Cluster (par exemple Sun Cluster 3.1).

Pour obtenir ces informations, exécutez les commandes suivantes :

Commande	Fonction
`prtconf -v`	Indique la taille de la mémoire système et affiche des informations sur les périphériques.
`psrinfo -v`	Affiche des informations sur les processeurs.
`showrev -p`	Indique les patchs installés.
SPARC : `prtdiag -v`	Affiche des informations diagnostiques sur le système.
`/usr/cluster/bin/scinstall -pv`	Affiche les informations de version de Sun Cluster et les informations de version du package

Gardez également à disposition le contenu du fichier `/var/adm/messages`.

Accès à la documentation Sun en ligne

Le site Web docs.sun.com[SM] vous permet d'accéder à la documentation technique Sun en ligne. Vous pouvez le parcourir ou y rechercher un titre de manuel ou un sujet particulier. L'URL de ce site est `http://docs.sun.com`.

Commande de documents Sun

Sun Microsystems offre une sélection de documentation produit imprimée. Pour obtenir une liste de ces documents et savoir comment les commander, consultez la rubrique "Acheter la documentation imprimée" sur le site http://docs.sun.com .

Conventions typographiques

Le tableau suivant présente les modifications typographiques utilisées dans ce manuel.

TABLEAU P–1 Conventions typographiques

Type de caractère ou symbole	Signification	Exemple
`AaBbCc123`	Noms de commandes, de fichiers et de répertoires, messages système s'affichant à l'écran	Modifiez votre fichier `.login`. Utilisez `ls -a` pour afficher la liste de tous les fichiers. `nom_machine% Vous avez reçu du courrier.`
`AaBbCc123`	Ce que vous entrez, par opposition à ce qui s'affiche à l'écran.	`nom_machine%` **`su`** `Password:`
AaBbCc123	Paramètre substituable de ligne de commande, à remplacer par un nom ou une valeur	Pour supprimer un fichier, entrez **`rm`** *nom_fichier* .
AaBbCc123	Titres de manuels, termes nouveaux ou mis en évidence.	Reportez-vous au chapitre 6 du *Manuel d'utilisation*. Ces options sont appelées options de *classe*. Vous devez être *superutilisateur* pour effectuer cette action.

Invites du shell dans les exemples de commandes

Le tableau suivant présente les invites système et les invites de superutilisateur par défaut des C shell, Bourne shell et Korn shell.

TABLEAU P–2 Invites de shell

Shell	Invite
Invite en C shell	`nom_machine%`
Invite du superutilisateur en C shell	`nom_machine#`
Invite en Bourne et Korn shells	`$`
Invite de superutilisateur en Bourne et Korn shells	`#`

CHAPITRE 1

Présentation de la gestion des ressources

Ce manuel fournit des procédures de création d'un type de ressources pour des applications telles que Oracle®, Sun Java™ System Web Server (précédemment Sun™ ONE Web Server), DNS, etc. Par conséquent, il s'adresse aux développeurs de types de ressources.

Ce chapitre présente les concepts que vous devez maîtriser pour développer un service de données et contient les informations suivantes :

Remarque – les expressions *type de ressources* et *service de données* sont utilisées de façon interchangeables dans ce manuel. Le terme *agent*, rarement usité dans ce manuel, signifie également *type de ressources* et *service de données*.

Environnement d'application Sun Cluster

Le système Sun Cluster permet d'exécuter et d'administrer les applications en tant que ressources évolutives et hautement disponibles. La fonction de cluster appelée RGM (gestionnaire de groupes de ressources) est à l'origine du haut niveau de disponibilité et d'évolutivité. Les éléments constituant l'interface de programmation vers cette fonction sont :

- Un ensemble de méthodes de rappel, que vous avez rédigées, permettant au RGM de contrôler une application sur le cluster.
- L'interface de programme d'application de gestion des ressources (API GR), un ensemble de commandes et de fonctions API de bas niveau que vous pouvez utiliser pour rédiger des méthodes de rappel. Ces interfaces API sont mises en oeuvre dans la bibliothèque `libscha.so`.
- Les fonctions de gestion de processus pour contrôler et redémarrer les processus sur le cluster.
- La bibliothèque de développement de services de données (BDSD), un ensemble de fonctions de bibliothèque qui encapsule les fonctionnalités de gestion des processus et de l'interface API de bas niveau à un niveau supérieur et confère d'autres fonctionnalités pour faciliter l'écriture des méthodes de rappel. Ces fonctions sont mises en oeuvre dans le fichier `libdsdev.so`.

L'illustration indiquée ci-après présente l'interrelation entre ces éléments.

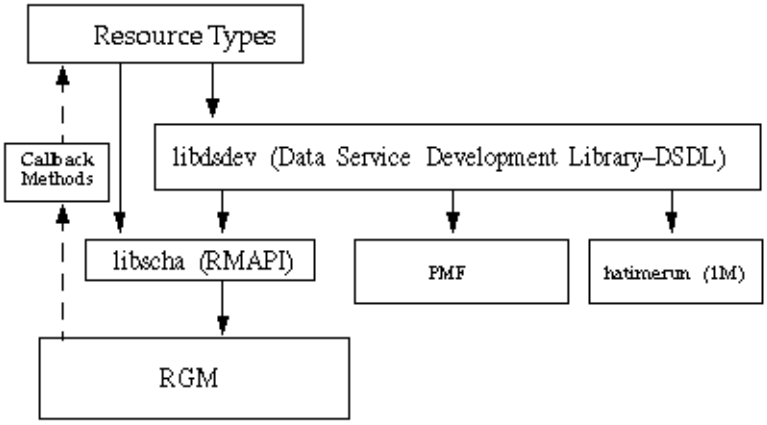


FIGURE 1–1 Architecture de programmation

SunPlex™ Agent Builder est inclus dans le package Sun Cluster. Cet outil automatise le processus de création d'un service de données (reportez-vous au Chapitre 9). Agent Builder génère un code de service de données en C (à l'aide des fonctions BDSD pour rédiger des méthodes de rappel) ou en korn shell (`ksh`) (à l'aide des commandes API de bas niveau pour rédiger des méthodes de rappel).

Le gestionnaire RGM s'exécute comme un démon sur chaque noeud du cluster. Il démarre automatiquement et arrête les ressources sur les noeuds sélectionnés conformément à des règles prédéfinies. Le gestionnaire RGM assure le haut niveau de disponibilité d'une ressource en cas de dysfonctionnement ou de redémarrage d'un noeud en arrêtant la ressource sur le noeud concerné et en la démarrant sur un autre

noeud. En outre, le gestionnaire RGM démarre et arrête automatiquement les détecteurs dédiés aux ressources pouvant détecter les dysfonctionnements au niveau des ressources, réallouer ces ressources sur un autre noeud ou contrôler d'autres aspects en termes de performances des ressources.

Le gestionnaire RGM prend en charge les ressources de basculement pouvant être connectées à un seul noeud à la fois et les ressources évolutives pouvant être connectées à plusieurs noeuds simultanément.

Modèle RGM

Cette rubrique traite de la terminologie fondamentale et présente de façon détaillée le gestionnaire RGM et les interfaces connexes.

Le gestionnaire RGM gère trois types fondamentaux d'objets interreliés : types de ressources, ressources et groupes de ressources. Un exemple peut être utilisé pour présenter ces objets, comme indiqué ci-après.

Un développeur met en oeuvre un type de ressources, `ha-oracle`, garantissant le haut niveau de disponibilité d'une application Oracle DBMS. Un utilisateur final définit des bases de données distinctes pour le marketing, l'ingénierie et la finance, chacune d'entre elles représentant une ressource de type `ha-oracle`. L'administrateur du cluster intègre ces ressources dans des groupes de ressources distincts, afin de pouvoir les exécuter sur différents noeuds et les basculer de façon indépendante. Un développeur crée un second type de ressources, ha-calendar, pour mettre en oeuvre un serveur d'agendas à haut niveau de disponibilité requérant une base de données Oracle. L'administrateur de cluster place la ressource dédiée à l'agenda financier dans le groupe de ressources dans lequel figure déjà la base de données financière. Ainsi, ces deux ressources sont exécutées sur le même noeud et basculent en même temps.

Types de ressources

Un type de ressources est constitué d'une application à exécuter sur un cluster, de programmes de gestion utilisés, comme les méthodes de rappel, par le gestionnaire RGM pour gérer l'application en tant que ressource de cluster et d'un ensemble de propriétés faisant partie intégrante de la configuration statique d'un cluster. Le gestionnaire RGM utilise des propriétés de type de ressources pour gérer les ressources d'un type spécifique.

Remarque – en plus d'une application, un type de ressources peut représenter d'autres ressources système comme les adresses réseau.

Le développeur indique les propriétés du type de ressources qu'il conçoit et définit leur valeur dans un fichier RTR (Resource Type Registration). Le fichier RTR est conforme à un format bien défini dont vous trouverez une description dans la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 34 et à la page de manuel `rt_reg`(4). Reportez-vous également à la rubrique "Définition du fichier d'enregistrement du type de ressource" à la page 92 pour obtenir la description du fichier d'enregistrement d'un exemple de type de ressources.

Le Tableau A–1 répertorie les propriétés de type de ressources.

L'administrateur du cluster installe et enregistre sur un cluster l'application sous-jacente et de mise en oeuvre des types de ressources. La procédure d'enregistrement ajoute à la configuration du cluster les informations issues du fichier d'enregistrement du type de ressources. Le document *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* présente la procédure d'enregistrement d'un service de données.

Ressources

Une ressource hérite des propriétés et des valeurs de son type de ressources. En outre, un développeur peut déclarer des propriétés de ressource dans le fichier d'enregistrement du type de ressources. Reportez-vous au Tableau A–2 pour obtenir la liste des propriétés de ressource.

L'administrateur de cluster peut modifier les valeurs de certaines propriétés suivant la façon dont elles sont spécifiées dans le fichier RTR. Par exemple, les définitions de propriété peuvent indiquer une plage de valeurs permises et spécifier quand la propriété est configurable (par exemple à la création, à tout moment ou jamais). L'administrateur du cluster peut modifier les propriétés dans le respect de ces spécifications à l'aide des commandes d'administration.

L'administrateur du cluster peut créer de nombreuses ressources du même type, chaque ressource possédant un nom et un ensemble de valeurs de propriété qui lui sont propres, afin qu'il soit possible d'exécuter plusieurs instances de l'application sous-jacente sur le cluster. Chaque instanciation requiert un nom unique au sein du cluster.

Groupes de ressources

Chaque ressource doit être configurée dans un groupe de ressources. Le gestionnaire RGM connecte et déconnecte simultanément toutes les ressources d'un groupe sur un même noeud. Lorsque le gestionnaire RGM connecte ou déconnecte un groupe de ressources, il exécute des méthodes de rappel sur les ressources individuelles du groupe.

Les noeuds auxquels un groupe de ressources est connecté sont appelés les *noeuds principaux* de ce groupe. Un groupe de ressources est *géré* par chacun de ses noeuds principaux. Chaque groupe de ressources possède une propriété `Liste_noeud` associée. Cette propriété est définie par l'administrateur de cluster et identifie tous les *noeuds principaux potentiels* ou maîtres du groupe de ressources.

Un groupe de ressources possède également un jeu de propriétés. Ces propriétés comprennent les propriétés de configuration définissables par l'administrateur du cluster et les propriétés dynamiques reflétant l'état actif du groupe de ressources et définies par le gestionnaire RGM.

Le gestionnaire RGM définit deux types de groupes de ressources : les groupes de ressources de basculement et les groupes de ressources évolutives. Un groupe de ressources de basculement ne peut être connecté qu'à un seul noeud à la fois tandis qu'un groupe de ressources évolutives peut être connecté simultanément à plusieurs noeuds. Le gestionnaire RGM fournit un ensemble de propriétés pour prendre en charge la création de chaque type de groupe de ressources. Reportez-vous aux rubriques "Transfert d'un service de données sur un cluster" à la page 34 et "Mise en oeuvre des méthodes de rappel" à la page 43 pour de plus amples informations sur ces propriétés.

Reportez-vous au Tableau A–3 pour obtenir la liste des propriétés de groupe de ressources.

Gestionnaire de groupes de ressources

Le gestionnaire RGM (Resource Group Manager) est mis en oeuvre comme un démon, `rgmd` qui s'exécute sur chaque noeud appartenant au cluster. Tous les processus `rgmd` communiquent entre eux et agissent ensemble comme une fonction unique à l'échelle du cluster.

Le gestionnaire RGM prend en charge les fonctions suivantes :

- Chaque fois qu'un noeud est initialisé ou tombe en panne, le gestionnaire RGM tente de préserver la disponibilité de tous les groupes de ressources gérés en les connectant automatiquement aux noeuds maîtres appropriés.

- Si une ressource spécifique tombe en panne, son programme de contrôle peut demander le redémarrage du groupe de ressources sur le même ou un nouveau noeud maître.
- L'administrateur du cluster peut lancer une commande administrative demandant l'exécution d'une des actions suivantes :
  - modification des noeuds maîtres d'un groupe de ressources ;
  - activation ou désactivation d'une ressource spécifique au sein d'un groupe de ressources ;
  - création, suppression ou modification d'une ressource, d'un groupe de ressources ou d'un type de ressource.

Chaque fois que le gestionnaire RGM active des modifications de configuration, il coordonne ses actions sur tous les noeuds appartenant au cluster. Cette activité porte le nom de reconfiguration. Pour modifier l'état d'une ressource individuelle, le gestionnaire RGM lui applique une méthode de rappel propre au type de ressources.

Méthodes de rappel

La structure de Sun Cluster utilise un mécanisme de rappel pour assurer la communication entre un service de données et le gestionnaire RGM. Cette structure définit un ensemble de méthodes de rappel, arguments et valeurs de retour compris, ainsi que les circonstances dans lesquelles le gestionnaire RGM appelle chaque méthode.

Vous pouvez créer un service de données en codant un ensemble de méthodes de rappel individuelles et en mettant en oeuvre chaque méthode sous la forme d'un programme que le logiciel RGM peut appeler. Aussi, le service de données n'est pas constitué d'un seul fichier exécutable mais plutôt de plusieurs scripts exécutables (`ksh`) ou programmes en binaire (C) que le gestionnaire RGM peut appeler directement.

Les méthodes de rappel sont enregistrées avec le gestionnaire RGM dans le fichier RTR (Resource Type Registration). Vous identifiez dans le fichier RTR chaque méthode mise en oeuvre pour le service de données. Lorsqu'un administrateur système enregistre le service de données sur un cluster, le gestionnaire RGM lit le fichier RTR, qui fournit, entre autres informations, l'identité des programmes de rappel.

Un type de ressources ne requiert que deux méthodes de rappel : une méthode de démarrage (`Démarrage` ou `Démarrage_avant_réseau`) et une méthode d'arrêt (`Arrêt` ou `Arrêt_après_réseau`).

Les méthodes de rappel peuvent être regroupées dans les catégories suivantes :

- Méthodes de contrôle et d'initialisation

- `Démarrage` lance et `Arrêt` arrête les ressources dans un groupe connecté ou déconnecté.
- `Init`, `Fini`, `Initialisation` exécutent le code d'initialisation et de fin sur les ressources.

- Méthodes de prise en charge administrative
  - `Validation` vérifie les propriétés définies par l'action administrative.
  - `Mise_à_jour` met à jour les paramètres de propriété d'une ressource en ligne.
- Méthodes relatives au réseau
  - `Démarrage_avant_réseau` et `Arrêt_après_réseau` exécutent des actions de démarrage ou d'arrêt spéciales avant que les adresses réseau du même groupe de ressources ne soient connectées ou après leur déconnexion.
- Méthodes de contrôle des détecteurs
  - `Contrôle_détecteur` et `Arrêt_détecteur` démarrent ou arrêtent le contrôleur d'une ressource.
  - `Contrôle_détecteur` évalue la fiabilité d'un noeud avant qu'un groupe de ressources ne soit basculé sur ce noeud.

Reportez-vous au Chapitre 4 et à la page de manuel `rt_callbacks`(1HA) pour de plus amples informations sur les méthodes de rappel. Reportez-vous également au Chapitre 5 et au Chapitre 8 pour de plus amples informations sur les méthodes de rappel dans les services de données échantillon.

Interfaces de programmation

Afin d'écrire le code du service de données, l'architecture de gestion des ressources offre une interface API de base et de bas niveau, une bibliothèque d'un niveau plus élevé et un outil, SunPlex Agent Builder, générant automatiquement un service de données à partir de l'entrée de base fournie.

API GR

L'interface API GR (interface de programme d'application des gestion des ressources) propose un ensemble de routines de bas niveau permettant au service de données d'accéder aux informations sur les ressources, les types de ressources et les groupes de ressources du système, demandant un redémarrage local ou un basculement et définissant l'état des ressources. La bibliothèque `libscha.so` vous permet d'accéder à ces fonctions. L'interface API GR fournit ces méthodes de rappel sous la forme de commandes shell et de fonctions C. Reportez-vous à `scha_calls`(3HA) et au

Chapitre 4 pour de plus amples informations sur les routines API GR. Reportez-vous également au Chapitre 5 pour découvrir des exemples d'utilisation de ces routines dans les méthodes de rappel de service de données échantillon.

Bibliothèque de développement de services de données (BDSD)

La bibliothèque BDSD est générée au sommet de l'interface API GR. Elle fournit une structure intégrée de plus haut niveau tout en conservant le modèle de rappel/méthode sous-jacente de l'interface RGM. La bibliothèque BDSD regroupe diverses fonctions de développement de services de données, y compris :

- `libscha.so` : les interfaces API de gestion des ressources de bas niveau.
- PMF : la fonction de gestion de processus offrant un moyen de contrôler les processus et leurs descendants et de les redémarrer en cas d'arrêt (reportez-vous à `pmfadm`(1M) et `rpc.pmfd`(1M)).
- `hatimerun` : une fonction d'exécution de programmes suivant un délai d'expiration (reportez-vous à `hatimerun`(1M)).

Avec la plupart des applications, la bibliothèque BDSD fournit toutes ou presque toutes les fonctionnalités dont vous avez besoin pour concevoir un service de données. Notez, toutefois, que la bibliothèque BDSD ne remplace pas l'interface API de bas niveau mais l'encapsule et l'étend. En fait, de nombreuses fonctions BDSD appellent les fonctions `libscha.so`. Vous pouvez également appeler directement les fonctions `libscha.so` tout en utilisant la librairie BDSD pour coder la majeure partie du service de données. La librairie `libdsdev.so` contient les fonctions BDSD.

Reportez-vous au Chapitre 6 et à la page de manuel `scha_calls`(3HA) pour de plus amples informations sur la bibliothèque BDSD.

SunPlex Agent Builder

Agent Builder est un outil automatisant la création d'un service de données. Une fois entrées les informations élémentaires sur l'application cible et le service de données à créer, Agent Builder génère un service de données, complété par un code source ou exécutable (C ou Korn shell), un fichier RTR personnalisé et un package Solaris™.

Vous pouvez utiliser Agent Builder avec la plupart des applications pour générer un service de données complet en n'y apportant que des modifications manuelles mineures. Les applications dont les exigences sont plus complexes (ajout de contrôles de validation aux propriétés additionnelles par exemple) peuvent requérir des actions qu'Agent Builder ne peut réaliser. Cependant, même dans ces circonstances vous pouvez utiliser Agent Builder pour générer la majeure partie du code avant de coder manuellement la partie restante. Vous pouvez, au minimum, utiliser Agent Builder pour générer le package Solaris pour vous.

Interface administrative du gestionnaire de groupes de ressources

Sun Cluster offre une interface utilisateur graphique et un ensemble de commandes d'administration d'un cluster.

SunPlex Manager

SunPlex Manager est un outil basé sur le Web vous permettant d'exécuter les tâches suivantes :

- installer un cluster ;
- administrer un cluster ;
- créer et configurer des ressources et des groupes de ressources ;
- configurer des services de données avec le logiciel Sun Cluster.

Reportez-vous au *Sun Cluster Software Installation Guide for Solaris OS* pour de plus amples informations sur la procédure d'installation de SunPlex Manager et son utilisation pour installer un logiciel de cluster. SunPlex Manager dispose d'une aide en ligne sur la plupart des tâches administratives uniques.

Commandes administratives

Les commandes d'administration Sun Cluster des objets RGM sont `scrgadm`(1M), `scswitch`(1M) et `scstat`(1M) `-g`.

La commande `scrgadm` permet de visualiser, créer, configurer et supprimer le type de ressources, le groupe de ressources et les ressources utilisés par le gestionnaire RGM. La commande fait partie intégrante de l'interface administrative du cluster. Vous ne devez pas l'utiliser dans le même contexte que l'interface d'application présentée dans le reste de ce chapitre. La configuration du cluster, au sein de laquelle l'interface API fonctionne, est réalisée à l'aide de l'outil `scrgadm`. La compréhension de l'interface d'application passe par la compréhension de l'interface administrative. Reportez-vous à la page de manuel `scrgadm`(1M) pour de plus amples informations sur les tâches administratives que cette commande vous permet de réaliser.

La commande `scswitch` connecte et déconnecte le groupe de ressources sur les noeuds indiqués et active ou désactive une ressource ou son détecteur. Reportez-vous à la page de manuel `scswitch`(1M) pour de plus amples informations sur les tâches administratives que cette commande peut réaliser.

La commande `scstat -g` indique l'état dynamique courant de tous les groupes de ressources et de toutes les ressources.

CHAPITRE 2

Développement d'un service de données

Ce chapitre fournit des informations détaillées sur le développement d'un service de données.

Ce chapitre contient les rubriques suivantes :

Analyse du caractère approprié de l'application

La première étape de création d'un service de données consiste à s'assurer que l'application répond effectivement aux conditions requises en matière de haute disponibilité et d'évolutivité. Si l'application ne satisfait pas toutes les conditions requises, vous devez être en mesure de modifier son code source pour pallier le problème.

La liste indiquée ci-après présente brièvement les exigences requises pour qu'une application soit hautement disponible ou évolutive. Pour obtenir de plus amples informations ou si vous devez modifier le code source de l'application, reportez-vous à l'Annexe B.

Remarque – pour être hautement disponible, un service évolutif doit remplir toutes les conditions suivantes, ainsi que certains critères supplémentaires.

- Dans l'environnement Sun Cluster, vous pouvez rendre hautement disponibles ou évolutives des applications réseau (modèle client/serveur) et non prévues pour être utilisées en réseau (sans client). Cependant, Sun Cluster ne peut pas optimiser la disponibilité au sein d'environnements en temps partagé sous lesquels des applications sont exécutées sur un serveur accessible via `telnet` ou `rlogin`.
- L'application doit être tolérante aux pannes. Cela signifie qu'elle doit restaurer (si nécessaire) les données de disque au démarrage après qu'un noeud soit tombé en panne de façon inattendue. En outre, le temps de reprise après une panne doit être limité. La tolérance aux pannes est indispensable pour rendre une application hautement disponible car la possibilité de restaurer le disque et de redémarrer l'application est une question d'intégrité des données. Il n'est pas nécessaire que le service de données puisse restaurer les connexions.
- L'application ne doit pas dépendre du nom d'hôte physique du noeud sur lequel elle est exécutée. Reportez-vous à la rubrique "Noms d'hôtes" à la page 333 pour obtenir de plus amples informations.
- L'application doit fonctionner correctement dans les environnements sous lesquels plusieurs adresses IP sont configurées en amont, comme par exemple les environnements comportant des hôtes multihome (caractérisés par le fait que le noeud figure sur plusieurs réseaux publics) et les environnements comportant des noeuds sur lesquels plusieurs interfaces logiques sont configurées en amont sur une seule interface matérielle.
- Pour être hautement disponibles, les données de l'application doivent résider dans les systèmes de fichiers du cluster. Reportez-vous à la rubrique "Données multi-hôtes" à la page 331.

  Si l'application utilise un nom de chemin d'accès câblé pour localiser les données, vous pouvez le modifier par un lien symbolique renvoyant à un emplacement dans le système de fichiers du cluster, sans modifier le code source de l'application. Reportez-vous à la rubrique "Utilisation de liens symboliques pour déterminer l'emplacement des données multi-hôtes" à la page 332 pour obtenir de plus amples informations.
- Les codes binaires applicatifs et les bibliothèques peuvent résider localement sur chaque noeud ou dans le système de fichiers du cluster. Avantage lorsqu'ils résident dans le système de fichiers du cluster : une seule installation suffit. Inconvénient : la mise à niveau devient problématique car les codes binaires sont en cours d'utilisation tandis que l'application est exécutée sous le contrôle du gestionnaire RGM.

- Le client doit pouvoir essayer de relancer automatiquement une requête si le délai de la première tentative est dépassé. Si l'application et le protocole prennent déjà en charge le redémarrage d'un serveur tombé en panne, ils peuvent également gérer le basculement ou la commutation du groupe de ressources. Reportez-vous à la rubrique "Relance d'un client" à la page 335 pour obtenir de plus amples informations.
- L'application ne doit comporter ni prise de domaine Unix ni pipe nommé dans le système de fichiers du cluster.

De plus, les services évolutifs doivent satisfaire les conditions suivantes :

- L'application doit pouvoir exécuter plusieurs instances fonctionnant toutes sur les mêmes données d'application dans le système de fichiers du cluster.
- L'application doit assurer la cohérence des données dans le cadre d'accès simultanés à partir de plusieurs noeuds.
- L'application doit mettre en oeuvre un verrouillage suffisant au moyen d'un mécanisme visible à l'échelle du système, comme le système de fichiers du cluster.

Dans le cadre d'un service évolutif, les caractéristiques de l'application déterminent également la règle d'équilibrage de la charge. Par exemple, la règle d'équilibrage de la charge `équilibrage_charge_pondéré`, qui permet à n'importe quelle instance de répondre aux requêtes des clients, ne fonctionne pas avec une application utilisant une mémoire cache d'entrée sur le serveur pour les connexions client. Dans ce cas, vous devez spécifier une règle d'équilibrage de la charge qui restreint le trafic d'un client donné à une instance de l'application. Les règles d'équilibrage de la charge `équilibrage_charge_sticky` et `équilibrage_charge_sticky_jocker` transmettent à plusieurs reprises toutes les requêtes d'un client à la même instance de l'application où elles peuvent utiliser la mémoire cache d'entrée. Notez que si plusieurs requêtes client proviennent de différents clients, le gestionnaire RGM les distribue entre les instances du service. Reportez-vous à la rubrique "Mise en oeuvre d'une ressource de basculement" à la page 52 pour de plus amples informations sur le paramétrage de la règle d'équilibrage de la charge des services de données évolutifs.

Détermination de l'interface à utiliser

Le package de support développeur de Sun Cluster (`SUNWscdev`) intègre deux ensembles d'interfaces de codage des méthodes de services de données :

- L'interface de programme d'application de gestion des ressources (API GR), un ensemble de routines de bas niveau (dans la bibliothèque `libscha.so`).
- La bibliothèque de développement de services de données (BDSD). Cet ensemble de fonctions d'un niveau plus élevé (dans la bibliothèque `libdsdev.so`) encapsule les fonctionnalités de l'interface API GR et fournit d'autres fonctionnalités.

Ce package comprend également SunPlex Agent Builder, un outil d'automatisation de la création d'un service de données.

Approche recommandée pour développer un service de données :

1. Choisir le type de code, C ou korn shell, à utiliser. Si vous choisissez d'utiliser le code korn shell, vous ne pouvez pas utiliser la bibliothèque BDSD qui fournit uniquement une interface C.
2. Exécuter Agent Builder, indiquer les entrées demandées et générer un service de données qui comprend un code source et exécutable, un fichier RTR et un package.
3. Si le service de données généré doit être personnalisé, vous pouvez ajouter un code BDSD aux fichiers sources générés. Agent Builder indique, à l'aide de commentaires, les emplacements spécifiques où vous pouvez ajouter votre propre code dans les fichiers sources.
4. S'il est nécessaire de personnaliser davantage le code pour prendre en charge l'application cible, vous pouvez ajouter des fonctions API GR au code source existant.

Dans la pratique, de nombreuses approches vous permettent de créer un service de données. Par exemple, plutôt que d'ajouter votre propre code aux emplacements spécifiques générés par Agent Builder dans le code source, vous pouvez remplacer l'intégralité d'une des méthodes générées ou le programme détecteur généré par un programme que vous avez écrit à l'aide des fonctions BDSD ou API GR. Quelle que soit votre façon de procéder, commencer par utiliser Agent Builder n'est pas dénué de sens pour les raisons suivantes :

- Le code généré par Agent Builder, bien que générique à cause de sa nature, a été testé dans de nombreux services de données.
- Agent Builder génère un fichier RTR, un fichier makefile, un package pour la ressource et d'autres fichiers support pour le service de données. Même si vous n'utilisez pas le code de service de données, l'utilisation de ces autres fichiers peut vous permettre de réduire considérablement la quantité de travail.
- Vous pouvez modifier le code généré.

Remarque – contrairement à l'interface API GR fournissant un ensemble de fonctions C et un ensemble de commandes à utiliser dans les scripts, la bibliothèque BDSD n'offre qu'une interface de fonction C. Par conséquent, si vous spécifiez une sortie korn shell (`ksh`) dans Agent Builder, le code source généré appelle l'interface API GR car il n'y a pas de commande BDSD `ksh`.

Paramétrage d'un environnement de développement dédié à l'écriture d'un service de données

Avant de développer un service de données, vous devez installer le package de développement de Sun Cluster (`SUNWscdev`) pour accéder aux fichiers d'en-tête et de bibliothèque de Sun Cluster. Bien que ce package soit déjà installé sur tous les noeuds du cluster, vous ne procédez pas au développement sur un noeud du cluster mais sur une machine de développement distincte n'appartenant pas au cluster. Dans ce cas, vous devez utiliser `pkgadd` pour installer le package `SUNWscdev` sur la machine de développement.

Lors de la compilation et de la liaison du code, vous devez définir un ensemble d'options spécifiques pour identifier les fichiers d'en-tête et de bibliothèque. À la fin du développement (sur un noeud non-cluster), vous pouvez transférer le service de données terminé sur un cluster pour l'exécuter et le tester.

Remarque – veillez à utiliser une version de développement de Solaris 5.8 ou supérieure.

Les procédures décrites dans cette rubrique permettent de réaliser les actions suivantes :

- installation du package de développement de Sun Cluster (`SUNWscdev`) et définition des options du compilateur et de l'éditeur de liens ;
- transfert du service de données sur un cluster.

▼ Paramétrage de l'environnement de développement

Cette procédure décrit comment installer le package `SUNWscdev` et configurer les options du compilateur et de l'éditeur de liens pour développer le service de données.

1. **Connectez-vous en tant que superutilisateur ou équivalent et remplacez le répertoire par le répertoire du CD de votre choix.**

   # **cd** *Répertoire_CD*

2. **Installez le package SUNWscdev dans le répertoire courant.**

   # **pkgadd -d . SUNWscdev**

3. **Dans le fichier `Makefile`, indiquez les options du compilateur et de l'éditeur de liens identifiant les fichiers inclus et de bibliothèque de votre code de service de données.**

   Utilisez l'option `-I` pour identifier les fichiers d'en-tête de Sun Cluster, l'option `-L` pour spécifier le chemin de recherche de la bibliothèque de compilation sur le système de développement et l'option `-R` pour indiquer le chemin de recherche de la bibliothèque à l'éditeur de liens d'exécution sur le cluster.

```
# Fichier Makefile d'un service de données modèle...

-I /usr/cluster/include

-L /usr/cluster/lib

-R /usr/cluster/lib
...
```

Transfert d'un service de données sur un cluster

Lorsque vous avez terminé de développer un service de données sur une machine de développement, vous devez le transférer sur un cluster pour le tester. Pour réduire les risques d'erreur, le meilleur moyen de réaliser ce transfert est encore de constituer un package avec le code du service de données et le fichier RTR, puis de l'installer sur tous les noeuds du cluster.

Remarque – que vous installiez le service de données avec `pkgadd` ou de quelque autre façon que ce soit, vous devez placer le service de données sur tous les noeuds du cluster. Agent Builder crée automatiquement un package avec le fichier RTR et le code du service de données.

Paramétrage des propriétés de ressources et de types de ressources

Sun Cluster propose un ensemble de propriétés de ressources et de types de ressources servant à définir la configuration statique d'un service de données. Les propriétés de types de ressources spécifient le type de ressource, la version, la version de l'interface API, etc., ainsi que les chemins d'accès aux méthodes de rappel. Le Tableau A–1 répertorie toutes les propriétés de types de ressources.

Les propriétés de ressources, telles que `Mode_basculement` et `Intervalle_sonde_complet` et les délais d'attente de la méthode définissent également la configuration statique de la ressource. Les propriétés de ressource dynamique, comme `État_ressource` et `Statut`, reflètent l'état actif d'une ressource gérée. Le Tableau A–2 présente les propriétés de ressources.

Vous déclarez les propriétés de ressources et de types de ressources dans le fichier RTR. Ce fichier est un élément indispensable d'un service de données. Le fichier RTR définit la configuration d'origine du service de données au moment où l'administrateur du cluster l'enregistre sous Sun Cluster.

Nous vous recommandons d'utiliser Agent Builder pour générer le fichier RTR de votre service de données car cette application déclare l'ensemble des propriétés qui sont à la fois utiles et requises pour n'importe quel service de données. Par exemple, certaines propriétés (comme `Type_ressource`) doivent être déclarées dans le fichier RTR car, dans le cas contraire, l'enregistrement du service de données échoue. Bien que cela ne soit pas indispensable, d'autres propriétés ne seront pas accessibles par l'administrateur système si vous ne les déclarez pas dans le fichier RTR tandis que d'autres propriétés sont disponibles que vous les déclariez ou non car le gestionnaire RGM les définit et leur octroie une valeur par défaut. Pour éviter de gérer un tel niveau de complexité, il vous suffit d'utiliser Agent Builder pour garantir la génération d'un fichier RTR correct que vous pouvez éditer ultérieurement pour modifier des valeurs spécifiques si nécessaire.

À la fin de cette rubrique, vous trouverez un modèle de fichier RTR créé par Agent Builder.

Déclaration des propriétés de type de ressources

L'administrateur du cluster ne peut pas configurer les propriétés de type de ressources que vous déclarez dans le fichier RTR. Ces propriétés font partie intégrante de la configuration permanente du type de ressources.

Remarque – seule une propriété de type de ressources, `Noeuds_installés` est configurable par l'administrateur système. En fait, elle ne peut être configurée que par un administrateur système. Vous ne pouvez pas la déclarer dans le fichier RTR.

Syntaxe de déclaration des types de ressources :

nom_propriété = *valeur*;

Remarque – le gestionnaire RGM est insensible à la casse dans les noms de propriétés. Les fichiers RTR fournis par Sun sont basés sur la convention suivante : la première lettre du nom est en majuscules tandis que toutes les autres sont en minuscules. Cette convention s'applique aux noms de propriété mais pas aux noms de méthode. Les noms de méthodes et les attributs de propriété sont en majuscules.

Voici les déclarations de type de ressources contenues dans le fichier RTR d'un service de données (`smpl`) échantillon :

```
# Sun Cluster Data Services Builder template version 1.0
# Enregistrement de données et de ressources pour smpl
#
#REMARQUE : les mots clés sont insensibles à la casse. Cela signifie
que vous pouvez utiliser les minuscules
#ou les majuscules au choix.
#
Type_ressource = "smpl";
id_fournisseur = SUNW;
Description_TR = "Sample Service on Sun Cluster";

Version_RT ="1.0";
Version_API = 2;
Basculement = VRAI;

noeuds_init = Éléments_principaux_GR;

rép_base_TR=/opt/SUNWsmpl/bin;

Démarrage = smpl_svc_start;
Arrêt = smpl_svc_stop;

Validation = smpl_validate;
Mise_à_jour = smpl_update;

Démarrage_détecteur = smpl_monitor_start;
Arrêt_détecteur = smpl_monitor_stop;
Contrôle_détecteur = smpl_monitor_check;
```

Astuce – la propriété `Type_ressource` doit être déclarée en tant que première entrée dans le fichier RTR. Dans le cas contraire, l'enregistrement du type de ressources échoue.

Le premier ensemble de déclarations de type de ressources fournit des informations élémentaires sur le type de ressources :

`Type_ressource` et `id_fournisseur` — Indique le nom du type de ressources. Vous pouvez spécifier le nom du type de ressources de deux façons : au moyen de la

	propriété `Type_ressources` (`smpl`) uniquement ou en utilisant le préfixe `id_fournisseur` suivi d'un point "." comme séparateur et du nom du type de ressources (`SUNW.smpl`), comme dans notre exemple. Si vous utilisez `id_fournisseur`, définissez le type de ressources à l'aide du symbole boursier de l'entreprise. Le nom du type de ressources doit être unique sur le cluster. **Remarque –** les noms de types de ressources (*Type_ressourcesId_fournisseur*) sont traditionnellement utilisés comme nom de package. Les noms de package ne peuvent pas contenir plus de 9 caractères. C'est pourquoi il est préférable de limiter le nombre total de caractères à 9 maximum lorsque vous attribuez un nom à ces deux propriétés bien que le gestionnaire RGM n'impose pas cette restriction. Par contre, comme Agent Builder génère explicitement le nom du package à partir du nom du type de ressources, il impose cette restriction.
`Version_RT`	Identifie la version du service de données échantillon.
`Version_API`	Identifie la version de l'interface API. Par exemple, `Version_API = 2` signifie que le service de données fonctionne sous Sun Cluster, version 3.0.
`Basculement = VRAI`	Indique qu'il est impossible d'exécuter le service de données dans un groupe de ressources pouvant être en ligne simultanément sur plusieurs noeuds car il s'agit d'un service de données de basculement. Reportez-vous à la rubrique "Transfert d'un service de données sur un cluster" à la page 34 pour de plus amples informations.
`Démarrage`, `Arrêt`, `Validation`, etc.	Indiquent les chemins d'accès aux programmes de méthode de rappel

correspondants que le gestionnaire RGM appelle. Ces chemins d'accès dépendent du répertoire spécifié par `rép_base_TR`.

Les autres déclarations de type de ressources fournissent des informations sur la configuration :

`noeuds_init = Éléments_principaux_GR`
: Indique que le gestionnaire RGM n'appelle les méthodes `Init`, `Initialisation`, `Fini` et `Validation` que sur les noeuds pouvant gérer le service de données. Les noeuds spécifiés par `Éléments_principaux_GR` constituent un sous-ensemble de tous les noeuds sur lesquels le service de données est installé. Définissez la valeur sur `noeuds_installés_TR` pour indiquer que le gestionnaire RGM appelle ces méthodes sur tous les noeuds sur lesquels le service de données est installé.

`Rép_base_TR`
: Sélectionnez le chemin d'accès au répertoire `/opt/SUNWsample/bin` pour compléter les chemins relatifs, comme les chemins d'accès aux méthodes de rappel.

`Démarrage`, `Arrêt`, `Validation`, etc.
: Indiquent les chemins d'accès aux programmes de méthode de rappel correspondants que le gestionnaire RGM appelle. Ces chemins d'accès dépendent du répertoire spécifié par `rép_base_TR`.

Déclaration des propriétés de ressources

Comme pour les propriétés de type de ressources, vous déclarez des propriétés de ressources dans le fichier RTR. Les déclarations de propriétés de ressources suivent traditionnellement les déclarations de type de ressources dans le fichier RTR. Les déclarations de ressource sont rédigées sous la forme d'un ensemble de paires de valeur d'attribut entre crochets :

```
{
    Attribut = Valeur;
    Attribut = Valeur;
               .
               .
               .
    Attribut = Valeur;
}
```

Vous pouvez modifier, dans le fichier RTR, les attributs spécifiques aux propriétés de ressources fournies par Sun Cluster et appelées propriétés *définies par le système*. Par exemple, Sun Cluster fournit des propriétés de délai d'attente pour chaque méthode de rappel en précisant des valeurs par défaut. Vous pouvez modifier ces valeurs dans le fichier RTR.

Vous pouvez également y définir de nouvelles propriétés de ressource, appelées propriétés d'*extension*, à l'aide d'un ensemble d'attributs de propriété fourni par Sun Cluster. Le Tableau A–4 répertorie les attributs permettant de modifier et de définir des propriétés de ressources. Les déclarations de propriété d'extension suivent les déclarations de propriété définies par le système dans le fichier RTR.

Le premier ensemble de propriétés de ressource définies par le système indique le délai d'attente des méthodes de rappel :

```
...

# Les déclarations de propriétés de ressource apparaissent sous
# la forme d'une liste d'entrées entre crochets après les déclarations
# de type de ressources. Le premier attribut suivant le crochet
# d'ouverture d'une entrée de propriété de ressource doit être la
# déclaration du nom de la propriété.
#
# Définissez les délais d'attente minimum et par défaut de la méthode.
{
        PROPRIÉTÉ = Délai_démarrage;
MIN=60;
PAR DÉFAUT=300;
}
{
PROPRIÉTÉ = Délai_arrêt;
MIN=60;
PAR DÉFAUT=300;
}
{
PROPRIÉTÉ = Délai_validation;
MIN=60;
PAR DÉFAUT=300;
}
{
PROPRIÉTÉ = Délai_mise_à_jour;
MIN=60;
PAR DÉFAUT=300;
}
{
PROPRIÉTÉ = Délai_démarrage_détecteur;
MIN=60;
PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Délai_arrêt_détecteur;
MIN=60;
PAR DÉFAUT=300;
{
PROPRIÉTÉ = Délai_contrôle_détecteur;
MIN=60;
PAR DÉFAUT=300;
}
```

Le premier attribut de chaque déclaration de propriétés de ressources doit être le nom de la propriété (PROPRIÉTÉ = *valeur*). Vous pouvez configurer des propriétés de ressources conformément aux limites définies par les attributs de propriété figurant dans le fichier RTR. Par exemple, le délai d'attente par défaut de chaque méthode est de 300 secondes dans notre exemple. Un administrateur peut modifier cette valeur,sachant cependant que la valeur minimum autorisée, conformément à l'attribut MIN, est de 60 secondes. Reportez-vous au Tableau A–4 pour consulter la liste complète des attributs de propriétés de ressources.

L'ensemble de propriétés de ressources suivant définit les propriétés ayant une utilisation spécifique dans le service de données.

```
{
        PROPRIÉTÉ = Mode_basculement;
PAR DÉFAUT=LOGICIEL;
RÉGLABLE = À_TOUT_MOMENT;
}
{
PROPRIÉTÉ = Intervalle_sonde_complet;
MIN=1;
MAX=3600;
PAR DÉFAUT=60;
RÉGLABLE = À_TOUT_MOMENT;
}

#Nombre de tentatives à exécuter dans une période donnée avant de conclure
# qu'il est impossible de démarrer l'application avec succès sur ce noeud.
{
PROPRIÉTÉ = Nombre_nouvelles_tentatives;
MAX=10;
PAR DÉFAUT=2;
RÉGLABLE = À_TOUT_MOMENT;
}

# Configurez Intervalle_nouvelles_tentatives sur un multiple de 60 car ce paramètre est
# arrondi lors du convertissement des secondes en minutes. Par exemple, 50 secondes
# sera arrondi à 1 minute. Cette propriété vous permet de minuter le nombre
# de tentatives (Nombre_nouvelles_tentatives).
{
PROPRIÉTÉ = Intervalle_nouvelles_tentatives;
MAX=3600;
PAR DÉFAUT=300;
RÉGLABLE = À_TOUT_MOMENT;
}

{
PROPRIÉTÉ = Ressources_réseau_utilisées;
RÉGLABLE = LORSQU'ELLE EST DÉSACTIVÉE;
PAR DÉFAUT = "";
}
{
PROPRIÉTÉ = Évolutivité;
PAR DÉFAUT = FAUX;
RÉGLABLE = À_LA_CRÉATION;
```

```
}
{
PROPRIÉTÉ = Règle_équilibrage_charge;
PAR DÉFAUT = Équilibrage_charge_pondéré;
RÉGLABLE = À_LA_CRÉATION;
}
{
PROPRIÉTÉ = Poids_équilibrage_charge;
PAR DÉFAUT = "";
RÉGLABLE = À_TOUT_MOMENT;
}
{
PROPRIÉTÉ = Liste_ports;
RÉGLABLE = À_LA_CRÉATION;
PAR DÉFAUT = ;
}
```

Ces déclarations de propriétés de ressources créent l'attribut `RÉGLABLE`, qui restreint les occasions permettant à l'administrateur système de modifier leurs valeurs. `À_LA_CRÉATION` signifie que l'administrateur ne peut spécifier la valeur qu'à la création de la ressource et qu'il ne pourra pas la modifier ultérieurement.

Vous pouvez accepter les valeurs par défaut qu'Agent Builder génère pour la plupart de ces propriétés à moins d'avoir une raison de les modifier. Vous trouverez ci-après de plus amples informations sur ces propriétés (vous pouvez également vous reportez à la rubrique "Propriétés des ressources" à la page 257 ou à la page de manuel `r_properties`(5)) :

`Mode_basculement`
: Indique si le gestionnaire RGM doit relocaliser le groupe de ressources ou arrêter le noeud en cas d'échec d'une méthode `Démarrage` ou `Arrêt`.

`Intervalle_sonde_complet`, `Nombre_nouvelles_tentatives`, `Intervalle_nouvelles_tentatives`
: Propriétés utilisées dans le système de détection des pannes. `Réglable` et `À_tout_moment` sont identiques de sorte que l'administrateur système peut les ajuster si le système de détection des pannes ne fonctionne pas de façon optimale.

`Ressources_réseau_utilisées`
: Liste des ressources (noms d'hôte logique et adresses partagées) utilisées par le service de données. Agent Builder déclare cette propriété, afin qu'un administrateur système puisse spécifier une liste de ressources, le cas échéant, lors de la configuration du service de données.

`Évolutivité`
: Définissez cette propriété sur `FAUX` pour indiquer que la ressource correspondante n'utilise pas la fonction de gestion de réseaux du cluster (adresse partagée). Ce paramétrage est compatible avec la propriété de type de ressources `Basculement` définie sur `VRAI` pour indiquer qu'il s'agit d'un service de basculement. Reportez-vous aux rubriques "Transfert d'un service de données sur un cluster" à la page 34 et "Mise en oeuvre des méthodes de rappel" à la page 43 pour de plus amples informations sur l'utilisation de cette ressource.

`Règle_équilibrage_charge`, `Poids_équilibrage_charge`
Déclarent automatiquement ces propriétés, sinon elles ne sont pas utilisées dans un type de ressources de basculement.

`Liste_ports`
Identifie la liste des ports de réception du serveur. Agent Builder déclare cette propriété, afin qu'un administrateur système puisse identifier une liste de ports lors de la configuration du service de données.

Déclaration des propriétés d'extension

Les propriétés d'extension se trouvent à la fin du fichier RTR échantillon, comme indiqué dans la liste ci-après.

```
# Propriétés d'extension #

# L'administrateur du cluster doit définir la valeur de cette propriété pour désigner
# le répertoire contenant les fichiers de configuration utilisés par l'application.
# Pour cette application, smpl, indiquez le chemin d'accès du fichier de configuration sur
# PXFS (généralement named.conf).
{
PROPRIETE = Liste_répconf;
EXTENSION;
TABLEAU DE CHAÎNES DE CARACTÈRES;
RÉGLABLE = À_LA_CRÉATION;
DESCRIPTION = "Chemin(s) Répertoire de Configuration";
}
 # Les deux propriétés suivantes contrôlent le redémarrage du système de détection des pannes.
{
PROPRIÉTÉ = Nombre_nouvelles_tentatives_détecteur;
EXTENSION;
INT;
PAR DÉFAUT = 4;
RÉGLABLE = À_TOUT_MOMENT;
DESCRIPTION = "Nombre de redémarrages de PMF autorisé par le détecteur de panne.";
}
{
PROPRIÉTÉ = Intervalle_nouvelles_tentatives_détecteur;
EXTENSION;
INT;
PAR DÉFAUT = 2;
RÉGLABLE = À_TOUT_MOMENT;
DESCRIPTION = "Fenêtre de temps (en minutes) relative au redémarrage du détecteur de pannes.";
}
# Délai d'attente en seconde pour la détection.
{
PROPRIÉTÉ = Délai_test;
EXTENSION;
INT;
PAR DÉFAUT = 120;
RÉGLABLE = À_TOUT_MOMENT;
DESCRIPTION = "Valeur du délai imparti au test (en secondes)";
```

```
}
# Niveau de contrôle du processus enfant de PMF (option -C de pmfadm).
# La valeur par défaut -1 signifie qu'il ne faut pas utiliser l'option -C de pmfadm.
# Une valeur égale ou supérieure à 0 indique le niveau souhaité de contrôle
# du processus enfant.
{
PROPRIÉTÉ = Niveau_de_contrôle_enfant;
EXTENSION;
INT;
PAR DÉFAUT = -1;
RÉGLABLE = À_TOUT_MOMENT;
DESCRIPTION = "Niveau de contrôle enfant pour PMF";
}
# Code utilisateur ajouté -- DÉBUT VVVVVVVVVVV
# Code utilisateur ajouté -- FIN  ^^^^^^^^^^^^
```

Agent Builder crée quelques propriétés d'extension qui sont utiles à la plupart des services de données. Les voici :

`Liste_rép_conf`
: Indique le chemin d'accès au répertoire de configuration de l'application. Cette information est utile pour la plupart des applications. L'administrateur peut indiquer l'emplacement de ce répertoire lors de la configuration du service de données.

`Nombre_nouvelles_tentatives_détecteur`, `Intervalle_nouvelles_tentatives_détecteur`, `Délai_test`
: Contrôlent les redémarrages du système de détection des pannes et non du démon du serveur.

`Niveau_de_contrôle_enfant`
: Définit le niveau de contrôle que la fonction PMF doit effectuer. Reportez-vous à `pmfadm`(1M) pour de plus amples informations.

Vous pouvez créer d'autres propriétés d'extension dans les emplacements délimités par les commentaires *Code ajouté par l'utilisateur*.

Mise en oeuvre des méthodes de rappel

Cette rubrique fournit des informations sur la mise en oeuvre des méthodes de rappel, en règle générale.

Accès aux informations sur les propriétés de ressources et de groupe de ressources

En règle générale, les méthodes de rappel doivent accéder aux propriétés de la ressource. L'interface API GR fournit des commandes shell et des fonctions C que vous pouvez utiliser dans les méthodes de rappel pour accéder aux propriétés d'extension définies par l'utilisateur des ressources. Reportez-vous aux pages de manuel `scha_resource_get`(1HA) et `scha_resource_get`(3HA).

La bibliothèque BDSD fournit un ensemble de fonctions C (une par propriété) pour accéder aux propriétés définies par le système et une fonction d'accès aux propriétés d'extension. Reportez-vous aux pages de manuel `scds_property_functions`(3HA) et `scds_get_ext_property`(3HA).

Vous ne pouvez pas utiliser le mécanisme de propriété pour enregistrer des données d'état dynamiques pour un service de données car aucune fonction API n'est disponible pour paramétrer les propriétés de ressources (à l'exception du paramétrage de `Statut` et `msg_statut`). Le cas échéant, il est préférable d'enregistrer les données d'état dynamiques dans des fichiers globaux.

Remarque – l'administrateur du cluster peut définir certaines propriétés de ressources à l'aide de la commande `scrgadm`, d'une commande administrative graphique disponible ou d'une interface administrative graphique. Par contre, n'appelez pas `scrgadm` à partir d'une méthode de rappel car cette commande échoue pendant la reconfiguration du cluster (c'est-à-dire lorsque le gestionnaire RGM appelle la méthode.)

Idempotence des méthodes

En règle générale, le gestionnaire RGM n'appelle pas successivement plus d'une fois une méthode sur la même ressource avec les mêmes arguments. Par contre, si une méthode `Démarrage` échoue, le gestionnaire RGM peut appeler une méthode `Arrêt` sur une ressource même si cette dernière n'a jamais été exécutée. De même, le gestionnaire RGM peut exécuter la méthode `Arrêt` sur le démon d'une ressource qui s'est pourtant interrompu de lui-même. Les mêmes scénarios s'appliquent aux méthodes `Démarrage_détecteur` et `Arrêt_détecteur`.

C'est pourquoi vous devez créer une relation d'idempotence entre les méthodes `Arrêt` et `Arrêt_détecteur`. Les appels répétés d'`Arrêt` ou `Arrêt_détecteur` sur la même ressource en utilisant les mêmes paramètres fournissent les mêmes résultats qu'un appel unique.

L'idempotence se caractérise notamment par le fait qu'`Arrêt` et `Arrêt_détecteur` doivent revenir à 0 (succès) même si la ressource ou le détecteur sont déjà arrêtés ou qu'aucun travail n'est effectué.

Remarque – les méthodes `Init`, `Fini`, `Initialisation` et `Mise_à_jour` doivent également être idempotentes. Il n'est pas nécessaire qu'une méthode `Démarrage` soit idempotente.

Service de données générique

Un service de données générique (GDS) est un mécanisme permettant de rendre hautement disponibles et évolutives des applications uniques en les connectant à la structure du gestionnaire RGM de Sun Cluster. Ce mécanisme évite l'utilisation d'un agent de programmation, procédure habituelle pour rendre une application hautement disponible ou évolutive.

Le modèle GDS s'appuie sur un type de ressources précompilées, SUNW.gds, pour assurer l'interaction avec la structure RGM.

Reportez-vous au Chapitre 10 pour de plus amples informations.

Contrôle d'une application

Les méthodes de rappel permettent au gestionnaire RGM de contrôler les ressources sous-jacentes (application) chaque fois que des noeuds sont liés au processus d'entrée/sortie du cluster.

Démarrage et arrêt d'une ressource

La mise en oeuvre d'un type de ressources requiert au minimum une méthode `Démarrage` et une méthode `Arrêt`. Le gestionnaire RGM appelle les programmes de méthode du type de ressources aux moments opportuns et sur les noeuds appropriés pour connecter ou déconnecter les groupes de ressources. Par exemple, après la défaillance d'un noeud de cluster, le gestionnaire RGM transfère les groupes de ressources gérés par ce noeud sur un autre noeud. Vous devez mettre en oeuvre une méthode `Démarrage` pour permettre au gestionnaire RGM de redémarrer chaque ressource sur le noeud hôte restant.

Une méthode `Démarrage` ne doit pas être retournée tant que la ressource n'a pas été exécutée et n'est pas disponible sur le noeud local. Veillez à ce que les types de ressources nécessitant une longue période d'initialisation disposent d'un délai d'attente suffisant dans la méthode `Démarrage` (configurez la propriété `Délai_démarrage` sur une valeur minimum dans le fichier d'enregistrement du type de ressources).

Vous devez mettre en oeuvre une méthode `Arrêt` lorsque le gestionnaire RGM déconnecte un groupe de ressources. Par exemple, supposons qu'un groupe de ressources est déconnecté du Noeud1 puis reconnecté au Noeud2. Tout en déconnectant le groupe de ressources, le gestionnaire RGM appelle la méthode `Arrêt` sur les ressources du groupe dont vous souhaitez arrêter toute activité sur le Noeud1. Une fois que les méthodes `Arrêt` ont été exécutées sur toutes les ressources du Noeud1, le gestionnaire RGM connecte le groupe de ressources au Noeud2.

Une méthode `Arrêt` ne doit pas être retournée tant que la ressource n'a pas cessé totalement toute activité au niveau du noeud local et qu'elle n'est pas complètement arrêtée. La mise en oeuvre la plus fiable d'une méthode `Arrêt` se traduit par l'achèvement de tous les processus sur le noeud local lié à la ressource. Pour les types de ressources mettant longtemps à s'arrêter, il est nécessaire de définir un délai d'attente suffisamment long dans leur méthode `Arrêt`. Définissez la propriété `Délai_arrêt` dans le fichier d'enregistrement du type de ressources.

Lorsqu'une méthode `Arrêt` échoue ou dépasse le délai imparti, le groupe de ressources bascule dans un mode d'erreur requérant l'intervention de l'opérateur. Pour éviter cela, les mises en oeuvre des méthodes `Arrêt` et `Arrêt_détecteur` doivent tenter une récupération à partir de toutes les conditions d'erreur possibles. Dans l'idéal, la fermeture de ces méthodes doit s'accompagner d'un état d'erreur nul (succès) signifiant que les méthodes ont réussi à interrompre correctement toutes les activités de la ressource et de son détecteur sur le noeud local.

Choix des méthodes `Démarrage` et `Arrêt` à utiliser

Cette rubrique présente des astuces permettant de savoir quand il est préférable d'utiliser les méthodes `Démarrage` et `Arrêt` par opposition aux méthodes `Démarrage_avant_réseau` et `Arrêt_après_réseau`. Pour déterminer les méthodes à utiliser, vous devez posséder une connaissance approfondie du client et du protocole de gestion des réseaux client-serveur du service de données.

Par ailleurs, il est possible qu'avec les services utilisant des ressources d'adresse réseau, les étapes de démarrage et d'arrêt doivent être exécutées dans un ordre précis dépendant de la configuration de l'adresse du nom d'hôte logique. Les méthodes de rappel en option `Démarrage_avant_réseau` et `Arrêt_après_réseau` permettent à la mise en oeuvre d'un type de ressources d'effectuer des actions de démarrage et d'arrêt spéciales avant et après la configuration en amont ou en aval des adresses réseau dans le même groupe de ressources.

Le gestionnaire RGM appelle les méthodes plombant les adresses réseau (sans les configurer en amont) avant d'appeler la méthode `Démarrage_avant_réseau` du service de données. Le gestionnaire RGM appelle les méthodes déplombant les adresses réseau après avoir appelé les méthodes `Arrêt_après_réseau` du service de données. Séquence applicable lorsque le gestionnaire RGM connecte un groupe de ressources :

1. Plombage des adresses réseau.
2. Appel de la méthode `Démarrage_avant_réseau` du service de données (le cas échéant).
3. Configuration en amont des adresses réseau.
4. Appel de la méthode `Démarrage` du service de données (le cas échéant).

Séquence applicable lorsque le gestionnaire RGM déconnecte un groupe de ressources (séquence inverse de la précédente) :

1. Appel de la méthode `Arrêt` du service de données (le cas échéant).
2. Configuration en aval des adresses réseau.
3. Appel de la méthode `Arrêt_après_réseau` du service de données (le cas échéant).
4. Déplombage des adresses réseau.

Pour choisir les méthodes `Démarrage`,`Arrêt`, `Démarrage_avant_réseau` ou `Arrêt_après_réseau` à utiliser, considérez tout d'abord le côté serveur. Lors de la connexion d'un groupe de ressources contenant des ressources d'adresse réseau et d'application de service de données, le gestionnaire RGM appelle des méthodes de configuration en amont des adresses réseau avant d'appeler les méthodes de ressources de `Démarrage` du service de données. Par conséquent, si un service de données requiert des adresses configurées en amont à son démarrage, utilisez la méthode `Démarrage` pour le démarrer.

De même, lors de la déconnexion d'un groupe de ressources contenant des ressources d'adresse réseau et de service de données, le gestionnaire RGM appelle des méthodes de configuration en aval des adresses réseau avant d'appeler les méthodes de ressources d'`Arrêt` du service de données. Par conséquent, si un service de données requiert des adresses configurées en amont à son arrêt, utilisez la méthode `Arrêt` pour l'arrêter.

Par exemple, pour démarrer ou arrêter un service de données, vous devrez peut-être exécuter ses bibliothèques ou ses utilitaires d'administration. Le service de données contient parfois des bibliothèques ou des utilitaires d'administration utilisant une interface de gestion de réseaux client-serveur pour accomplir les tâches administratives. Le cas échéant, un utilitaire d'administration appelle le démon du serveur. L'adresse réseau doit donc être en amont pour utiliser la bibliothèque ou l'utilitaire d'administration. Dans ce cas, utilisez les méthodes `Démarrage` et `Arrêt`.

Par contre, vous devez démarrer ou arrêter le service de données à l'aide des méthodes `Démarrage_avant_réseau` et `Arrêt_après_réseau` si son démarrage ou son arrêt nécessite que les adresses réseau soient configurées en aval. Vous devez tenir compte du fait que les logiciels clients peuvent réagir différemment suivant que l'adresse réseau ou le service de données se connecte en premier après la reconfiguration d'un cluster (soit `scha_control()` avec l'argument `SCHA_GIVEOVER` ou une commutation avec `scswitch`). Par exemple, la mise en oeuvre de clients peut nécessiter un minimum de tentatives, l'abandon survenant rapidement une fois que l'indisponibilité du port du service de données a été détectée.

Si le service de données ne requiert pas qu'une adresse réseau soit configurée en amont, démarrez-le avant de configurer l'interface réseau en amont. Vous avez ainsi l'assurance que le service de données peut répondre immédiatement aux requêtes des clients dès que l'adresse réseau a été configurée en amont. Par conséquent, les clients sont moins susceptibles d'interrompre leurs tentatives. Dans ce cas, démarrez le service de données à l'aide de la méthode `Démarrage_avant_réseau` plutôt qu'avec la méthode `Démarrage`.

Si vous utilisez la méthode `Arrêt_après_réseau`, la ressource de service de données se trouve encore en amont lors de la configuration en aval de l'adresse réseau. La méthode `Arrêt_après_réseau` n'est donc appelée qu'après la configuration en aval de l'adresse réseau. En conclusion, le port du service TCP ou UDP du service de données (ou son numéro de programme RPC) semble toujours disponible aux clients du réseau, hormis lorsque l'adresse réseau elle-même ne répond pas.

La décision d'utiliser les méthodes `Démarrage` et `Arrêt` plutôt que les méthodes `Démarrage_avant_réseau` et `Arrêt_après_réseau` ou de combiner les deux, doit tenir compte des exigences et du comportement du client et du serveur.

Méthodes `Init`, `Fini` et `Initialisation`

Les trois méthodes facultatives d'`Init`, de `Fini` et d'`Initialisation` permettent au gestionnaire RGM d'exécuter un programme d'initialisation ou d'arrêt sur une ressource. Le gestionnaire RGM appelle la méthode `Init` pour exécuter l'initialisation unique d'une ressource qui devient gérée soit parce que le groupe de ressources auquel elle appartient bascule d'un état non géré à un état géré ou parce qu'elle est créée dans un groupe de ressources déjà géré.

Le gestionnaire RGM appelle la méthode `Fini` pour supprimer la ressource qui devient non gérée soit parce que le groupe de ressources auquel elle appartient bascule d'un état géré à un état non géré ou parce qu'elle est supprimée d'un groupe de ressources géré. La suppression doit être idempotente. Par conséquent, si la suppression a déjà été effectuée, le résultat de `Fini` est 0 (succès).

Le gestionnaire RGM appelle la méthode `Initialisation` sur les noeuds qui viennent de se connecter au cluster, c'est-à-dire, qui viennent d'être initialisés ou réinitialisés.

L'initialisation induite par les méthodes `Initialisation` et `Init` est généralement identique. Elle doit être idempotente. Par conséquent, si la ressource a déjà été initialisée sur le noeud local, le résultat d'`Initialisation` et `Init` est 0 (succès).

Contrôle d'une ressource

En règle générale, vous mettez en oeuvre des moyens de contrôle pour procéder périodiquement à une recherche de pannes sur les ressources, afin de déterminer si elles fonctionnent correctement. En cas d'échec, le détecteur peut tenter un redémarrage local ou transmettre une requête de basculement du groupe de ressources concerné en exécutant les fonctions API GR `scha_control ()` ou BDSD `scds_fm_action()`.

Vous pouvez également surveiller les performances d'une ressource et régler ou signaler les performances. Il n'est absolument pas nécessaire d'écrire un système de détection des pannes propre au type de ressources puisque même si vous le faites, le type de ressources bénéficie de la surveillance de base du cluster par Sun Cluster lui-même. Sun Cluster détecte les pannes du matériel hôte, les défaillances majeures du système d'exploitation de l'hôte et les échecs de connexion de l'hôte aux réseaux publics.

Bien que le gestionnaire RGM n'appelle pas un détecteur de ressource directement, il doit pouvoir le faire pour démarrer automatiquement des détecteurs pour les ressources. Lors de la déconnexion d'une ressource, le gestionnaire RGM appelle la méthode `Arrêt_détecteur` pour arrêter le détecteur de la ressource sur les noeuds locaux avant d'arrêter la ressource elle-même. Lors de la connexion d'une ressource en ligne, le gestionnaire RGM appelle la méthode `Démarrage_détecteur` après le démarrage de la ressource.

Les fonctions API GR `scha_control()` et BDSD `scds_fm_action()` (qui appelle `scha_control ()`) permettent aux détecteurs de ressource de demander le basculement d'un groupe de ressources sur un autre noeud. La fonction `scha_control ()` s'assure de la faisabilité de l'opération en exécutant notamment `Contrôle_détecteur` (si cette fonction est définie) pour déterminer si le noeud requis est suffisamment fiable pour gérer le groupe de ressources contenant la ressource. Si `Contrôle_détecteur` signale que le noeud n'est pas fiable ou que le délai d'attente de la méthode est dépassé, le gestionnaire RGM recherche un autre noeud pour honorer la requête de basculement. Si `Contrôle_détecteur` échoue sur tous les noeuds, le basculement est annulé.

Le détecteur de ressource peut définir les propriétés `Statut` et `msg_Statut` pour refléter l'affichage de l'état de la ressource sur le détecteur. Définissez ces propriétés à l'aide de la fonction API GR `scha_resource_setstatus()`, de la commande `scha_resource_setstatus` ou de la fonction BDSD `scds_fm_action()`.

Remarque – bien que l'utilisation de `Statut` et `msg_statut` soient propres à un détecteur de ressource, n'importe quel programme peut définir ces propriétés.

La rubrique "Définition d'un détecteur de pannes" à la page 109 présente un exemple de système de détection des pannes mis en oeuvre avec l'interface API GR. La rubrique "Détecteur de pannes `SUNW.xfnts`" à la page 158 présente un exemple de système de détection des pannes mis en oeuvre avec la bibliothèque BDSD. Reportez-vous au document *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour de plus amples informations sur les systèmes de détection des pannes créés à l'intérieur des systèmes de données fournis par Sun.

Ajout d'un journal de messages à une ressource

Si vous souhaitez enregistrer des messages d'état dans le même fichier journal que les autres messages du cluster, utilisez la fonction `scha_cluster_getlogfacility()`. Très pratique, elle permet d'extraire le numéro de fonction utilisé pour consigner les messages du cluster.

Utilisez ensuite ce numéro avec la fonction `syslog()` standard de Solaris pour consigner des messages dans le journal du cluster. Vous pouvez également accéder au contenu du journal du cluster à l'aide de l'interface `scha_cluster_get()` générique.

Gestion des processus

L'interface API GR et la BDSD offrent des fonctions de gestion des processus pour mettre en oeuvre des détecteurs et des rappels de contrôle des ressources. L'interface API GR définit les fonctions suivantes (reportez-vous aux pages de manuel pour de plus amples informations sur chaque commande et programme) :

Fonction de contrôle des processus :

`pmfadm` et `rpc.pmfd` — La fonction PMF (Process Monitor Facility) offre un moyen de contrôler les processus et leurs descendants et de redémarrer les processus s'ils sont arrêtés. Elle comprend la

	commande `pmfadm` pour démarrer et contrôler les processus gérés et le démon `rpc.pmfd`.
`halockrun`	Programme d'exécution d'un programme enfant sans déverrouiller le fichier. Cette commande est pratique à utiliser dans les scripts shell.
`hatimerun`	Programme d'exécution d'un programme enfant suivant un délai d'attente. Cette commande est pratique à utiliser dans les scripts shell.

La BDSD intègre la fonction `scds_hatimerun` permettant de mettre en oeuvre la fonctionnalité `hatimerun`.

Elle fournit également un ensemble de fonctions (`scds_pmf_*`) de mise en oeuvre de la fonctionnalité PMF. Reportez-vous à la rubrique "Fonctions PMF" à la page 214. Elle présente brièvement la fonctionnalité PMF de la bibliothèque PMF et contient une liste des fonctions individuelles.

Support administratif d'une ressource

Entre autres actions administratives possibles sur les ressources figurent le paramétrage et la modification des propriétés de ressources. L'interface API définit les méthodes de rappel `Validation` et `Mise_à_jour`, afin que vous puissiez mettre en oeuvre ces actions.

Le gestionnaire RGM appelle la méthode `Validation` facultative lorsqu'une ressource est créée et qu'une action administrative met à jour les propriétés de la ressource ou de son groupe. Il transfère la valeur de propriété de cette ressource ou de son groupe à la méthode `Validation`. Il appelle `Validation` sur l'ensemble des noeuds du cluster spécifiés par la propriété `noeuds_init` du type de ressources (reportez-vous à la rubrique "Propriétés des types de ressources" à la page 249 ou à la page de manuel `rt_properties`(5) pour de plus amples informations sur `noeuds_init`). Le gestionnaire RGM appelle `Validation` avant que la création ou la mise à jour ne soit appliquée. Par conséquent, l'échec d'un code de sortie d'une méthode sur n'importe quel noeud entraîne l'échec de la création ou de la mise à jour.

Il n'appelle `Validation` que lorsque les propriétés de la ressource ou du groupe sont modifiées par une opération de l'administrateur, et non lorsque le RGM définit des propriétés, ou lorsqu'un détecteur définit les propriétés `Statut` et `msg_statut` de la ressource.

Le gestionnaire RGM appelle la méthode `Mise_à_jour` facultative pour notifier à une ressource en cours d'exécution que des propriétés ont été modifiées. Il appelle `Mise_à_jour` après l'exécution réussie d'une action administrative de paramétrage des propriétés d'une ressource ou de son groupe. Le gestionnaire RGM appelle cette méthode sur les noeuds sur lesquels cette méthode est en ligne. Cette méthode peut utiliser les fonctions d'accès de l'interface API pour lire les valeurs de propriété pouvant affecter une ressource active et régler les ressources en cours d'exécution en conséquence.

Mise en oeuvre d'une ressource de basculement

Un groupe de ressources de basculement contient des adresses réseau (par exemple, le nom d'hôte logique et l'adresse partagée des types de ressources intégrés) et des ressources de basculement (ressources d'application de service de données dédiées à un service de données de basculement par exemple). Les ressources d'adresse réseau, et leurs ressources de service de données dépendantes passent d'un noeud de cluster à l'autre lors du basculement ou de la commutation des services de données. Le gestionnaire RGM propose de nombreuses propriétés prenant en charge la mise en oeuvre d'une ressource de basculement.

Définissez la propriété booléenne du type de ressources `Basculement` sur `VRAI` pour empêcher la configuration d'une ressource dans un groupe pouvant être en ligne sur plusieurs noeuds simultanément. Cette propriété est définie sur `FAUX` par défaut. Vous devez donc la déclarer comme `VRAI` dans le fichier RTR lorsqu'il s'agit d'une ressource de basculement.

La propriété de ressource `Évolutivité` détermine si la ressource utilise la fonction d'adresse partagée du cluster. Pour une ressource de basculement, définissez `Évolutivité` sur `FAUX` car elle ne doit pas utiliser d'adresses partagées.

La propriété de groupe de ressources `Mode_GR` permet à l'administrateur du cluster d'identifier un groupe de ressources comme étant évolutif ou de basculement. Si `Mode_GR` est défini sur `BASCULEMENT`, le gestionnaire RGM configure la propriété `Éléments_principaux_max.` du groupe sur 1 et limite la gestion du groupe de ressources à un seul noeud. Le gestionnaire RGM ne permet pas de créer une ressource dont la propriété `Basculement` est définie sur `VRAI` dans un groupe de ressources dont la propriété `Mode_GR` est définie sur `ÉVOLUTIVITÉ`.

La propriété de groupe de ressources `Dépendances_réseau_implicites` spécifie que le gestionnaire RGM doit appliquer les dépendances implicites fortes des ressources d'adresse non prévues pour être utilisées en réseau à toutes les ressources

d'adresse réseau (nom d'hôte logique et adresse partagée) au sein du groupe. Cela signifie que les méthodes `Démarrage` des ressources d'adresse non prévues pour être utilisées en réseau ne sont pas appelées tant que les adresses réseau du groupe ne sont pas configurées en amont. La propriété `Dépendances_réseau_implicites` est définie par défaut sur `VRAI`.

Mise en oeuvre d'une ressource évolutive

Une ressource évolutive peut être en ligne sur plusieurs noeuds simultanément. Les ressources évolutives comprennent les services de données comme Sun Cluster HA for Sun One Web Server (précédemment Sun Cluster HA for Sun ONE Web Server) et Sun Cluster HA for Apache.

Le gestionnaire RGM propose de nombreuses propriétés prenant en charge la mise en oeuvre d'une ressource évolutive.

Définissez la propriété booléenne du type de ressources `Basculement` sur `FAUX` pour autoriser la configuration d'une ressource dans un groupe pouvant être ligne sur plusieurs noeuds simultanément.

La propriété de ressource `Évolutivité` détermine si la ressource utilise la fonction d'adresse partagée du cluster. Définissez cette propriété sur `VRAI` car une ressource évolutive utilise une ressource d'adresse partagée pour que les multiples instances du service évolutif soient présentées comme un service unique au client.

La propriété `Mode_GR` permet à l'administrateur du cluster d'identifier un groupe de ressources comme étant évolutif ou de basculement. Si la propriété `Mode_GR` est définie sur `ÉVOLUTIVITÉ`, le gestionnaire RGM permet de configurer `Éléments_principaux_max` sur une valeur supérieure à 1, ce qui signifie que le groupe peut être géré par plusieurs noeuds simultanément. Le gestionnaire RGM permet d'instancier une ressource dont la propriété `Basculement` est définie sur `FAUX` dans un groupe de ressources dont la propriété `Mode_GR` est définie sur `ÉVOLUTIVITÉ`.

L'administrateur du cluster crée un groupe de ressources évolutives pour contenir les ressources de service évolutives et un groupe de ressources de basculement pour contenir les ressources d'adresse partagée dont la ressource évolutive dépend.

L'administrateur du cluster utilise la propriété de groupe de ressources `Dépendances_groupe_ressources` pour spécifier dans quel ordre les groupes de ressources sont connectés à un noeud et en sont déconnectés. Cet ordre est important dans le cadre d'un service évolutif car les ressources évolutives et les ressources

d'adresse partagée dont elles dépendent se trouvent dans des groupes différents. Un service de données évolutif nécessite que son adresse réseau (adresse partagée) soit configurée en amont avant d'être démarré. Par conséquent, l'administrateur doit définir la propriété `Dépendances_groupe_ressources` (du groupe de ressources contenant le service évolutif) pour inclure le groupe de ressources contenant les ressources d'adresse partagée.

Lorsque vous déclarez la propriété Évolutivité d'une ressource dans le fichier RTR, le gestionnaire RGM crée automatiquement l'ensemble des propriétés évolutives suivantes pour cette ressource :

`Ressources_réseau_utilisées`
: Identifie les ressources d'adresse partagée utilisées par cette ressource. Cette propriété est définie par défaut sur une chaîne de caractères vide. Par conséquent, l'administrateur du cluster doit fournir la liste réelle des adresses partagées que le service évolutif utilise lors de la création de la ressource. La commande `scsetup` et SunPlex Manager offrent des fonctions permettant de paramétrer automatiquement les ressources et les groupes requis pour les services évolutifs.

`Règle_équilibrage_charge`
: Spécifie la règle d'équilibrage de la charge de la ressource. Vous pouvez explicitement définir cette règle dans le fichier RTR (ou activer la valeur par défaut Équilibrage_charge_pondéré). Dans tous les cas, l'administrateur du cluster peut modifier la valeur lors de la création de la ressource (à moins que vous ne définissiez la valeur `Réglable` de la propriété `Règle_équilibrage_charge` sur `AUCUN` ou `FAUX` dans le fichier RTR). Valeurs légales :

  `Équilibrage_charge_pondéré`
  : La charge est répartie entre plusieurs noeuds en fonction des poids définis dans la propriété `Poids_équilibrage_charge`.

  `Équilibrage_charge_sticky`
  : Un client donné (identifié par son adresse IP) du service évolutif est toujours envoyé au même noeud du cluster.

`Équilibrage_charge_sticky_étendu`
: Un client donné (identifié par son adresse IP), connecté à l'adresse IP d'un service sticky à caractère joker est toujours envoyé sur le même noeud du cluster, indépendamment du port vers lequel il est dirigé.

  Dans le cadre d'un service évolutif défini sur `Règle_équilibrage_charge` `Équilibrage_charge_sticky` ou `Équilibrage_charge_sticky_étendu`, la modification de la propriété `Poids_équilibrage_charge` alors que le service est en ligne peut réinitialiser les affinités des clients existants. Le cas échéant, un autre noeud peut prendre en charge une requête ultérieure du client, même si ce dernier était précédemment géré par un autre noeud du cluster.

  Le redémarrage d'une nouvelle instance du service sur un cluster peut également réinitialiser les affinités des clients existants.

`Poids_équilibrage_charge`
: Spécifie la charge à envoyer à chaque noeud. Format : *poids*`@`*noeud*, *poids*`@`*noeud*, *poids* correspondant à un nombre entier reflétant la part relative de la charge distribuée au noeud spécifié. La fraction de la charge distribuée à un noeud correspond au poids de ce noeud divisé par la somme de tous les poids des instances actives. Par exemple, `1@1,3@2` indique que le noeud 1 reçoit 1/4 de la charge et que le noeud 2 en reçoit les 3/4.

`Liste_ports`
: Identifie les ports d'écoute du serveur. Cette propriété est définie par défaut sur une chaîne de caractères vide. Vous pouvez fournir une liste des ports dans le fichier RTR. Sinon, l'administrateur du cluster doit fournir la liste réelle des ports lors de la création de la ressource.

Vous pouvez créer un service de données que l'administrateur pourra configurer en tant que service de basculement ou évolutif. Pour ce faire, déclarez la propriété de type de ressources `Basculement` et la propriété de ressource `Évolutivité` sur `FAUX` dans le fichier RTR du service de données. Spécifiez à la création que la propriété `Évolutivité` est réglable.

La valeur de la propriété `Basculement` (`FAUX`) permet de configurer la ressource dans un groupe de ressources évolutives. L'administrateur peut activer les adresses partagées en définissant la valeur de la propriété `Évolutivité` sur `VRAI` lors de la création de la ressource, créant ainsi un service évolutif.

D'un autre côté, même si la propriété `Basculement` est définie sur `FAUX`, l'administrateur peut configurer la ressource dans un groupe de ressources de basculement pour mettre en oeuvre un service de basculement. L'administrateur ne modifie pas la valeur de la propriété `Évolutivité` (déjà définie sur `FAUX`). Pour prendre en charge cette éventualité, vous devez fournir un contrôle de la propriété `Évolutivité` dans la méthode `Validation`. Si la propriété `Évolutivité` est définie sur `FAUX`, vérifiez que la ressource est configurée dans un groupe de ressources de basculement.

Vous trouverez de plus amples informations sur les ressources évolutives dans le *Sun Cluster Concepts Guide for Solaris OS*.

Contrôles de validation des services évolutifs

Chaque fois qu'une ressource est créée et mise à jour alors que la propriété d'Évolutivité est définie sur `VRAI`, le gestionnaire RGM valide diverses propriétés de ressources. Si les propriétés ne sont pas configurées correctement, le gestionnaire RGM rejette les tentatives de mise à jour ou de création. Le gestionnaire RGM effectue les contrôles suivants :

- La propriété `Ressources_réseau_utilisées` ne doit pas être vide. Elle doit contenir le nom des ressources d'adresse partagée existantes. Chaque noeud de la `Liste_noeuds` du groupe de ressources contenant la ressource évolutive doit apparaître dans la propriété `Liste_NetIf`ou `Liste_noeud_aux` de chaque ressource d'adresse partagée nommée.
- La propriété Dépendances_groupe_ressources du groupe de ressources contenant la ressource évolutive doit inclure les groupes de ressources de toutes les ressources d'adresse partagée répertoriée dans la propriété `Ressources_réseau_utilisées` de cette ressource évolutive.
- La propriété Liste_ports ne doit pas être vide. Elle doit contenir une liste des paires port/protocole de sorte que le protocole est soit TCP soit UDP. Par exemple,

  ```
  Liste_ports=80/tcp,40/udp
  ```

Écriture et test des services de données

Cette rubrique fournit des informations sur l'écriture et la vérification de services de données.

Utilisation du mécanisme Keep-Alives

Côté serveur, l'utilisation du mécanisme Keep-Alives au niveau des connexions TCP protège le serveur contre les gaspillages de ressources système d'un client en aval (ou partitionné en réseau). Si ces ressources ne sont pas nettoyées (dans un serveur restant allumé suffisamment longtemps), elles finissent par s'étendre de façon illimitée lorsque les clients tombent en panne et sont réinitialisés.

Si les communications client-serveur passent par un flux TCP, le serveur et le client doivent activer le mécanisme Keep-Alives au niveau des connexions TCP. Cette disposition s'applique également dans le cas d'un serveur unique non-HA.

D'autres protocoles orientés connexion peuvent intégrer un mécanisme Keep-Alives.

Côté client, l'utilisation d'un mécanisme Keep-Alives au niveau des connexions TCP permet au client d'être averti du basculement ou de la commutation d'une ressource d'adresse réseau d'un hôte physique vers un autre. Ce transfert de la ressource d'adresse réseau interrompt la connexion TCP. Pourtant, à moins que le client ait activé le mécanisme Keep-Alives, il n'est pas nécessairement informé de la déconnexion si la connexion est latente à ce moment-là.

Par exemple, supposons que le client attend une réponse du serveur à une requête longue durée, que le serveur a déjà reçu cette requête et qu'il en a déjà accusé réception au niveau de la couche TCP. Dans cette situation, le module TCP du client n'a pas besoin de continuer de transmettre la requête alors que l'application cliente est bloquée car elle attend la réponse.

Chaque fois que cela est possible, en plus d'utiliser le mécanisme Keep-Alives au niveau de la connexion TCP, l'application cliente doit également exécuter un mécanisme Keep-Alives périodique à son niveau. De fait, ce mécanisme n'est pas parfait dans tous les cas de limite possibles. L'utilisation d'un mécanisme Keep-Alives au niveau de l'application requiert généralement que le protocole client-serveur prenne en charge une opération Null ou au moins une opération de lecture seule efficace, telle une opération d'état.

Test d'un service de données à haut niveau de disponibilité

Cette rubrique apporte quelques suggestions quant aux procédures de test relatives à la mise en oeuvre d'un service de données dans un environnement à haut niveau de disponibilité. Les cas présentés sont suggestifs et non exhaustifs. Vous devez accéder à une configuration banc d'essai de Sun Cluster, afin que la procédure de test n'ait aucune répercussion sur les machines de production.

Vérifiez que votre service de données à haut niveau de disponibilité adopte un comportement approprié dans toutes les situations lorsqu'un groupe de ressources est déplacé d'un hôte physique vers un autre, y compris en cas de panne du système ou d'utilisation de la commande `scswitch`. Vérifiez que les machines clientes continuent de recevoir le service après ces événements.

Testez l'idempotence des méthodes. Par exemple, remplacez chaque méthode temporairement par un script shell court appelant la méthode d'origine au moins deux fois.

Coordination des dépendances entre les ressources

Il arrive parfois qu'un service de données client-serveur transmette des requêtes à un autre service de données client-serveur tout en exécutant une requête pour un client. Plus simplement, un service de données A dépend d'un service de données B ; par conséquent, pour que A puisse fournir son service, B doit fournir le sien. Pour satisfaire à cette exigence, Sun Cluster autorise la configuration de dépendances entre les ressources au sein d'un groupe de ressources. Les dépendances affectent l'ordre dans lequel Sun Cluster démarre et arrête les services de données. Reportez-vous à la page de manuel `scrgadm`(1M) pour de plus amples informations.

Si les ressources de votre type de ressources dépendent des ressources d'un autre type, vous devez notifier à l'utilisateur de configurer les ressources et les groupes de ressources de façon appropriée ou fournir des scripts ou des outils permettant de les configurer correctement. Si la ressource dépendante doit être exécutée sur le même noeud que la ressource dont d'autres dépendent, vous devez configurer ces deux ressources dans le même groupe.

Déterminez si vous souhaitez utiliser des dépendances explicites entre les ressources ou les omettre, puis testez la disponibilité du ou des autres services de données dans le code de votre service de données à haut niveau de disponibilité. Si la ressource dépendante/dont d'autres dépendent est exécutable sur différents noeuds, configurez chaque ressource dans un groupe distinct. Le cas échéant, une interrogation est requise car il est impossible de configurer ce type de dépendance entre les groupes.

Certains services de données ne sauvegardent directement aucune donnée. Pour ce faire, ils dépendent d'un autre service de données back-end. Un tel service de données convertit toutes les requêtes de lecture et de mise à jour en appels au service de données back-end. Par exemple, considérons un service d'agenda client-serveur hypothétique conservant toutes ses données dans une base de données SQL comme Oracle. Ce service possède son propre protocole réseau client-serveur. Par exemple, son protocole a pu être défini au moyen d'un langage RPC, comme ONC RPC.

Dans l'environnement Sun Cluster, vous pouvez utiliser HA-ORACLE pour rendre hautement disponible la base de données Oracle back-end, puis écrire des méthodes simples de démarrage et d'arrêt du démon d'agenda. L'utilisateur final enregistre le type de ressource d'agenda avec Sun Cluster.

Si l'application d'agenda doit fonctionner sur le même noeud que la base de données Oracle, l'utilisateur final configure la ressource d'agenda dans le même groupe de ressources que la ressource HA-ORACLE. Ensuite, il fait dépendre la ressource d'agenda de la ressource HA-ORACLE. Cette dépendance est spécifiée à l'aide de la balise de propriété `Dépendances_ressource` dans `scrgadm`.

Si la ressource HA-ORACLE peut fonctionner sur un autre noeud que la ressource d'agenda, l'utilisateur final les configure dans deux groupes de ressources distincts. Il peut définir la dépendance du groupe de ressources d'agenda sur le groupe de ressources Oracle. Cependant, les dépendances entre les groupes de ressources ne sont effectives que si vous démarrez ou arrêtez simultanément ces deux groupes sur le même noeud. Par conséquent, après avoir été démarré, le démon du service de données d'agenda doit attendre que la base de données Oracle devienne disponible. La méthode `Démarrage` du type de ressources d'agenda doit juste retourner qu'elle a réussi dans ce cas. De fait, si elle est indéfiniment bloquée, son groupe de ressources bascule en mode occupé et il devient impossible d'en modifier l'état (édition, basculement ou commutation notamment). En outre, si la méthode `Démarrage` de la ressource d'agenda dépasse le délai d'attente ou se ferme avec une valeur différente de zéro, le groupe de ressources risque de basculer continuellement entre plusieurs noeuds tandis que la base de données Oracle reste indisponible.

CHAPITRE 3

Mise à niveau d'un type de ressources

Ce chapitre offre une vision globale des connaissances requises pour mettre à niveau un type de ressources et migrer une ressource.

Présentation générale

Les administrateurs système doivent pouvoir installer et enregistrer une nouvelle version d'un type de ressources existant, pour permettre l'enregistrement de plusieurs versions d'un type de ressources donné et la migration d'une ressource existante vers une nouvelle version du type de ressources sans devoir ni supprimer, ni recréer la ressource. Les développeurs de ressources doivent maîtriser les exigences requises pour permettre la mise à niveau d'un type de ressources et la migration des ressources.

Lorsqu'un type de ressources est conçu dans le but de pouvoir être mis à niveau, il est dit qu'il *prend en charge des mises à niveau*.

La nouvelle version d'un type de ressources peut s'avérer sensiblement différente de la version antérieure. Voici la liste des caractéristiques qui ont pu être modifiées :

- les attributs de propriétés du type de ressources ;
- l'ensemble des propriétés de ressources déclarées, y compris les propriétés standard et d'extension ;
- les attributs de propriétés de ressource, comme `par défaut`, `min`, `max`, `min_tableau`, `max_tableau` ou la capacité de réglage ;
- l'ensemble des méthodes déclarées ;
- la mise en oeuvre des méthodes ou des détecteurs.

Le développeur du type de ressources décide quand il est possible de migrer une ressource existante vers une nouvelle version en choisissant l'une des options de capacité de réglage suivantes. Ces options sont classées de la moins à la plus restrictive :

- à tout moment (`À_tout_moment`) ;
- lorsque la ressource n'est pas contrôlée (`Lorsque_non_contrôlée`) ;
- lorsque la ressource est hors ligne (`Lorsque_hors_ligne`) ;
- lorsque la ressource est désactivée (`Lorsque_désactivée`) ;
- lorsque le groupe de ressources n'est pas géré (`Lorsque_non_géré`) ;
- à la création (`À_création`).

Reportez-vous à la rubrique "Propriété de ressources `Version_type`" à la page 64 pour de plus amples informations sur chaque option.

Remarque – ce chapitre traite de la procédure de mise à niveau au moyen de la commande `scrgadm` mais l'administrateur n'est pas contraint d'utiliser cette commande. Il peut également effectuer la mise à niveau par le biais de l'interface utilisateur graphique ou au moyen de la commande `scsetup`.

Fichier RTR (Resource Type Registration)

Nom du type de ressources

Les trois composantes du nom du type de ressources sont des propriétés spécifiées dans le fichier RTR sous *id_fournisseur*, *type_ressources* et *version_type_res*. La commande `scrgadm` insère les délimiteurs (points et deux points) pour créer le nom du type de ressources :

```
vendor_id.resource_type:rt_version
```

Le préfixe *id_fournisseur* permet de distinguer deux fichiers de connexion portant le même nom mais provenant de deux fournisseurs différents. La *version_type_res* permet de distinguer plusieurs versions enregistrées (mises à niveau) du même type de ressources de base. Pour avoir l'assurance que l'*id_fournisseur* est unique, nous vous recommandons d'utiliser le symbole boursier de l'entreprise en tant que type de ressources.

La mise en ligne d'une ressource échoue si la chaîne *version_TR* res comprend un espace vide, une tabulation, une barre oblique (/) ou (\), un astérisque (*), un point d'interrogation (?), une virgule (,), un point virgule (;), un crochet gauche ([) ou un crochet droit (]).

La propriété `Version_TR`, qui était facultative dans Sun Cluster 3.0, est désormais obligatoire sous Sun Cluster 3.1.

Le nom qualifié est le nom que renvoie la commande suivante :

`scha_resource_get -O Type -R` *nom_ressource* `-G` *nom_groupe_ressources*

Les noms de type de ressources que vous avez enregistrés sous les versions antérieures à Sun Cluster 3.1 ont toujours la syntaxe suivante :

```
type_ressource.id_fournisseur
```

Instructions

Les fichiers RTR des types de ressources supportant les mises à niveau doivent inclure une instruction `#$upgrade`, suivie de zéro ou plusieurs instructions comme suit :

`#$upgrade_from` *version capacité_réglage*

L'instruction `upgrade_from` comprend la chaîne `#$upgrade_from`, suivie de `Version_RT`, de la contrainte de capacité de réglage de la ressource. Si le type de ressources en cours de mise à jour ne possède pas de version, la propriété `Version_RT` est définie en tant que chaîne de caractères vide, comme illustré dans l'exemple ci-après :

```
#$upgrade_from "1.1" Lorsque_hors_ligne
#$upgrade_from "1.2" Lorsque_hors_ligne
#$upgrade_from "1.3" Lorsque_hors_ligne
#$upgrade_from "2.0" Lorsque_non_contrôlée
#$upgrade_from "2.1" À_tout_moment
#$upgrade_from "" Lorsque_non_gérée
```

Le gestionnaire RGM exécute ces contraintes sur une ressource lorsque l'administrateur système tente de modifier la ressource `Version_type`. Si la version actuelle du type de ressources n'apparaît pas dans la liste, le gestionnaire RGM impose la capacité de réglage de l'instruction `Lorsque_non_gérée`.

Ces instructions doivent apparaître entre la rubrique relative aux déclarations de propriétés de type de ressources et la rubrique relative aux déclarations de ressources du fichier RTR. Reportez-vous à `rt_reg`(4).

Modification de la chaîne `Version_RT` dans un fichier RTR

Modifiez la chaîne `Version_RT` dans un fichier RTR chaque fois que le contenu de ce fichier change. La valeur de cette propriété doit permettre de reconnaître la nouvelle version du type de ressources de la précédente. Il n'est pas nécessaire de modifier la chaîne `Version_TR` si le fichier RTR n'a pas été modifié.

Noms des types de ressources dans les versions précédentes de Sun Cluster

Les noms de type de ressources dans Sun Cluster 3.0 ne contenaient pas le suffixe de version :

```
nom_ressource.id_fournisseur
```

Le format du nom d'un type de ressources initialement enregistré sous Sun Cluster 3.0 est conservé même après la mise à niveau du logiciel vers Sun Cluster 3.1 ou versions ultérieures. De la même façon, un nom de format Sun Cluster 3.0 sans version de suffixe est attribué à un type de ressources dont le fichier RTR ne contient pas d'instruction `#$upgrade`, si ce fichier est enregistré sur un cluster fonctionnant sous Sun Cluster 3.1 ou versions ultérieures.

Vous pouvez enregistrer des fichiers RTR avec l'instruction `#$upgrade` ou `#$upgrade_from` dans Sun Cluster 3.0. Par contre, la migration des ressources existantes vers les nouveaux types de ressources dans Sun Cluster 3.0 n'est pas prise en charge.

Propriété de ressources `Version_type`

La propriété de ressources standard propriété `Version_type` contient la propriété `RT_Version` d'un type de ressources. Cette propriété n'apparaît pas dans le fichier RTR. L'administrateur système édite la valeur de cette propriété à l'aide de la commande suivante :

`scrgadm -c -j` *ressource* `-y version_type=` *nouvelle_version*

La capacité de réglage de cette propriété provient de :

- la version actuelle du type de ressources ;
- les instructions `#$upgrade_from` figurant dans le fichier RTR.

Utilisez les valeurs de capacité de réglage dans les instructions `#$upgrade_from` :

`À_tout_moment`
: En l'absence de restrictions quand il est possible de mettre une ressource à niveau. La ressource peut être en ligne.

`Lorsque_non_contrôlée`
: lorsque les méthodes `Mise_à_jour`, `Arrêt`, `Contrôle_détecteur` et `Arrêt_après_réseau` de la nouvelle version du type de ressources sont connues pour être compatibles avec les méthodes de démarrage de la version antérieure (`Arrêt_avant_réseau` et `Démarrage`) et lorsque la méthode `Fini` de la nouvelle version du type de ressources est connue pour être compatible avec la méthode `Init` des versions antérieures. Dans ce cas, il faut simplement arrêter le programme du détecteur de ressources avant la mise à niveau.

`Lorsque_hors_ligne`
: Lorsque les méthodes de la nouvelle version du type de ressources `Mise_à_jour`, `Arrêt`, `Contrôle_détecteur` ou `Arrêt_après_réseau` sont connues pour ne pas être compatibles avec les méthodes de démarrage de la version antérieure du type de ressources (`Arrêt_avant_réseau` et `Démarrage`) mais pour être compatibles avec la méthode `Init` des versions précédentes, la ressource doit être hors ligne lors de la mise à niveau du type auquel elle appartient.

`Lorsque_désactivée`
: Identique à `Lorsque_hors_ligne`. Cette valeur de capacité de réglage implique que la ressource soit désactivée.

`Lorsque_non_gérée`
: Lorsque la méthode de la version `Fini` de la nouvelle version du type de ressources n'est pas compatible avec la méthode `Init` des versions antérieures. Lorsque cette valeur de capacité de réglage est définie, vous ne pouvez mettre à niveau la ressource qu'après avoir basculé l'état du groupe de ressources existant en mode non géré.

`À_la_création`
: Lorsqu'il est impossible de mettre à niveau les ressources vers la nouvelle version de type de ressources. Vous ne pouvez que créer de nouvelles ressources avec la nouvelle version.

  La capacité de réglage de `À_la_création` signifie que le développeur du type de ressources peut empêcher la migration d'une ressource existante vers un nouveau type. Le cas échéant, l'administrateur système doit supprimer, puis recréer la ressource. Ceci revient à déclarer que la version de la ressource ne peut être définie qu'au moment de la création.

Migration d'une ressource vers une version différente

La nouvelle version du type de ressources est appliquée à une ressource existante lorsque l'administrateur système édite la propriété `Version_type` de cette ressource. Les conventions appliquées sont identiques à celles servant à éditer les autres propriétés de ressources, si ce n'est que certaines informations peuvent provenir de la nouvelle version du type de ressources au lieu de la version actuelle :

- Les attributs de propriétés de ressources de toutes les propriétés (`min`, `max`, `min_tableau`, `max_tableau`), les valeurs par défaut et la capacité de réglage sont issus de la nouvelle version du type de ressources.
- La capacité de réglage applicable à la propriété `Version_type` est fournie par l'instruction `#$upgrade_from` figurant dans le fichier RTR et la propriété `Version_RT` du type de ressources de la ressource existante. Elle est différente de la capacité de réglage décrite dans `property_attributes`(5).
- La méthode `Validation` de la nouvelle version du type de ressources est appliquée, garantissant ainsi que les attributs de propriétés sont valides pour le nouveau type de ressources. Si les attributs de propriétés de ressources existants ne satisfont pas aux conditions de validation de la nouvelle version du type de ressources, l'administrateur système doit attribuer des valeurs valides aux propriétés concernées sur la ligne de commande `scrgadm`. Cette situation peut se présenter si la dernière version du type de ressources exécute une propriété qui n'a pas été déclarée dans la version précédente et pour laquelle aucune valeur par défaut n'a été définie. Vous pouvez également y être confronté si la ressource existante possède une propriété dont la valeur, qui a déjà été attribuée, est invalide pour la dernière version du type de ressources.
- Les propriétés de ressources qui ont été déclarées dans la version précédente du type de ressources peuvent ne pas être déclarées dans la nouvelle version. Lors de la migration de la ressource vers la nouvelle version, ces propriétés sont supprimées.

Remarque – la méthode `Validation` peut interroger la propriété `Version_type` actuelle de la ressource (avec `scha_resource_get`), ainsi que la nouvelle propriété `Version_type` (transmise à la ligne de commande `Validation`). Par conséquent, `Validation` peut exclure les mises à jour des versions non prises en charge.

Mise à niveau ou adaptation vers une version antérieure d'un type de ressources

La rubrique "Upgrading a Resource Type" du *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* fournit de plus amples informations sur la mise à niveau et la migration d'un type de ressources.

▼ Procédure de mise à niveau d'un type de ressources

1. **Lisez la documentation de mise à niveau du nouveau type de ressources pour découvrir les modifications apportées au type de ressources et les contraintes de capacités de réglage des ressources.**

2. **Installez le package de mise à niveau du type de ressources sur tous les noeuds du cluster.**

   La méthode d'installation recommandée de nouveaux packages de type de ressources est en cours de révision : `pkgadd` est exécuté lors de l'initialisation d'un noeud en mode non-cluster.

   Certaines situations permettent d'installer de nouveaux packages de type de ressources sur un noeud en mode cluster :

   - Si l'installation du package de type de ressources ne modifie pas le code de méthode tout en mettant uniquement à jour le détecteur, vous devez interrompre la surveillance de toutes les ressources de ce type pendant l'installation.
   - Si l'installation du package de type de ressources ne modifie ni le code de méthode ni le code de contrôle, il n'est pas nécessaire d'interrompre la surveillance des ressources lors de l'installation. De fait, cette opération installe juste un nouveau fichier RTR sur le disque.

3. **Enregistrez la nouvelle version du type de ressources à l'aide de la commande `scrgadm` (ou une commande équivalente), en indiquant le fichier RTR de la mise à niveau.**

   Le gestionnaire RGM crée un nouveau type de ressources dont le nom possède le format suivant :

   ```
   id_fournisseur.type_ressource:version
   ```

4. **Si la mise à niveau du type de ressources n'est installée que sur un sous-ensemble des noeuds, vous devez définir la propriété `Noeuds_installés`**

du nouveau type de ressources sur les noeuds sur lesquels elle est réellement installée.

Lorsque le nouveau type est attribué à une ressource (venant d'être créée ou mise à jour), le gestionnaire RGM requiert que le groupe de ressources `liste_noeud` soit un sous-ensemble de la liste `Noeuds_installés` du type de ressources.

```
scrgadm -c -t type_ressources -h liste_noeuds_installés
```

5. **Pour chaque ressource du type pré-mis à niveau à migrer vers le type mis à niveau, exécutez `scswitch` pour basculer l'état de la ressource ou de son groupe sur l'état approprié, comme indiqué dans la documentation de mise à niveau.**

6. **Pour chaque ressource du type pré-mis à niveau à migrer vers le type mis à niveau, éditez la ressource et remplacez la propriété `Version_type` par la nouvelle version.**

   ```
   scrgadm -c -j ressource -y version_type=nouvelle_version
   ```

   Remplacez, si nécessaire, les autres propriétés de la même ressource par les valeurs appropriées dans la même commande.

7. **Restaurez l'état précédent de la ressource ou du groupe de ressources en inversant la commande exécutée à l'Étape 5.**

▼ Procédure d'adaptation d'une ressource vers une version antérieure de son type de ressources

Vous pouvez adapter une ressource vers une version antérieure de son type de ressources. Les conditions requises pour adapter une ressource vers une version antérieure du type de ressources sont plus restrictives que celles requises pour mettre à niveau le type de ressources. Vous devez commencer par basculer le groupe de ressources en mode non géré. En outre, vous ne pouvez adapter une ressource vers une version antérieure que si cette version est compatible avec la mise à jour du type de ressources. Vous pouvez identifier ces versions à l'aide de la commande `scrgadm -p`. Les versions compatibles avec la mise à jour apparaissent avec le suffixe `:version`.

1. **Déconnectez le groupe de ressources qui contient la ressource que vous souhaitez adapter vers une version antérieure.**

   ```
   scswitch -F -g groupe_ressources
   ```

2. **Déconnectez la ressource que vous souhaitez adapter vers une version antérieure, ainsi que toutes les ressources du groupe de ressources.**

   ```
   scswitch -n -j ressource_vers_version_antérieure
   scswitch -n -j ressource1
   scswitch -n -j ressource2
   scswitch -n -j ressource3
   ...
   ```

Remarque – désactivez les ressources par ordre de dépendance, en commençant par les plus dépendantes (ressources d'application) et en terminant par les moins dépendantes (ressources d'adresse réseau).

3. **Basculez le groupe de ressources en mode non géré.**

   `scswitch -u -g` *groupe_ressources*

4. **La version antérieure du type de ressources vers laquelle vous souhaitez faire l'adaptation figure-t-elle toujours sur le cluster ?**

   - Si oui, passez à l'étape suivante.
   - Si non, réenregistrez la version antérieure qui vous intéresse.

     `scrgadm -a -t` *nom_type_ressource*

5. **Adaptez la ressource vers le bas en spécifiant la version antérieure sous `Version_type`.**

   `scrgadm -c -j` *ressource_vers_version_antérieure* **`-y version_type=`***version_antérieure*

   Remplacez, si nécessaire, les autres propriétés de la même ressource par les valeurs appropriées dans la même commande.

6. **Basculez le groupe contenant la ressource à adapter vers une version antérieure en mode géré, activez toutes les ressources et connectez le groupe de ressources.**

   `scswitch -Z -g` *groupe_ressources*

Valeur par défaut des propriétés

Le gestionnaire RGM enregistre toutes les ressources. Par conséquent, toutes les propriétés qui ne sont pas explicitement définies par l'administrateur (et configurées sur leur valeur par défaut) ne sont pas enregistrées dans l'entrée des ressources du CCR (cluster configuration repository). Le gestionnaire RGM trouve les valeurs par défaut d'une propriété de ressource manquante dans le type de ressources (ou, si elles n'y figurent pas, à l'aide des valeurs par défaut définies par le système) lorsqu'une ressource est lue à partir du CCR. Cette méthode d'enregistrement des propriétés permet à un type de ressources mis à niveau de définir de nouvelles propriétés ou de nouvelles valeurs par défaut pour les propriétés existantes.

Lors de l'édition de propriétés de ressources à l'aide de la commande appropriée, le gestionnaire RGM les enregistre dans le CCR.

Si une version mise à niveau du type de ressources déclare une nouvelle valeur par défaut pour une propriété par défaut, les ressources existantes héritent de cette valeur, même si la propriété est déclarée comme étant réglable uniquement `À_la_création` ou `Lorsque_désactivée`. Si l'application de la nouvelle valeur par défaut engendre l'échec d'une méthode `Arrêt`, `Arrêt_après_réseau` ou `Fini` par exemple, la personne chargée de la mise en application du type de ressources doit modifier en conséquence l'état de la ressource au moment de sa mise à niveau. Pour ce faire, il doit restreindre la capacité de réglage de la propriété `Version_type`.

La méthode `Validation` de la nouvelle version du type de ressources peut effectuer un contrôle pour vérifier que les attributs de propriétés existants sont appropriés. S'ils ne le sont pas, l'administrateur système peut définir de façon appropriée les valeurs des propriétés d'une ressource existante dans la commande servant à modifier la propriété `Version_type` pour mettre à niveau la ressource vers la nouvelle version du type de ressources.

Remarque – les ressources créées dans Sun Cluster 3.0 n'héritent pas des attributs de propriétés par défaut du type de ressources lors de leur migration vers une version ultérieure car leurs propriétés par défaut sont enregistrées dans le CCR.

Informations à fournir par le développeur de type de ressources

Le développeur de type de ressources doit fournir les informations suivantes avec la nouvelle ressource :

- décrire les ajouts, modifications ou suppressions de propriétés ;
- décrire la procédure à appliquer pour rendre les propriétés conformes aux nouvelles exigences ;
- définir les contraintes de capacité de réglage au niveau des ressources ;
- afficher les nouveaux attributs de propriétés par défaut ;
- informer l'administrateur système qu'il est possible de modifier les propriétés de ressources existantes sur les valeurs appropriées à l'aide de la commande servant à modifier la propriété `Version_type` pour mettre à niveau la ressource vers la nouvelle version du type de ressources.

Mise en oeuvre du nom et du détecteur du type de ressources

Vous pouvez enregistrer un type de ressources supportant les mises à niveau dans Sun Cluster 3.0. Par contre, son nom est enregistré dans le CCR sans le suffixe de version. Pour fonctionner correctement dans Sun Cluster 3.0 et Sun Cluster 3.1 (et versions ultérieures), le détecteur de ce type de ressources doit pouvoir gérer deux conventions d'attribution des noms :

```
id_fournisseur.nom_ressource:version
id_fournisseur.nom_ressource
```

Le code de contrôle peut déterminer le nom approprié à utiliser en exécutant l'équivalent de

```
scha_resourcetype_get -O RT_VERSION -T VEND.myrt
scha_resourcetype_get -O RT_VERSION -T VEND.myrt:vers
```

puis en comparant les valeurs de sortie avec `vers`. Seule une de ces commandes n'échouera pas pour une valeur spécifique de `vers` car il est impossible d'enregistrer la même version d'un type de ressources deux fois sous deux noms différents.

Mises à niveau de l'application

La mise à niveau du code d'application diffère de celle du code de l'agent, bien que certains points se recoupent. La mise à niveau de l'application peut être accompagnée ou non de la mise à niveau d'un type de ressources.

Exemples de mise à niveau d'un type de ressources

Ces exemples présentent plusieurs scénarios de mise à niveau et d'installation d'un type de ressources. Les informations relatives au package et à la capacité de réglage ont été choisies en fonction des types de modifications apportés à la mise en oeuvre du type de ressources. La capacité de réglage s'applique à la migration de la ressource vers le nouveau type de ressources.

Tous les exemples supposent que :

- Le type de ressources est fourni dans un package Solaris. Reportez-vous à `pkgadd`(1M) et `pkgrm`(1M).
- Il existe une seule version antérieure du type de ressources et, par conséquent, une seule instruction `#$upgrade_from` dans le nouveau fichier RTR.
- La procédure d'installation ne peut pas supprimer ou écraser les méthodes dans la mesure où le gestionnaire RGM peut les exécuter alors qu'elles ont été effacées du disque.
- Les nouvelles méthodes sont compatibles avec les méthodes antérieures à moins que l'inverse soit spécifié.
- Les ressources et les groupes de ressources sont basculés dans l'état requis avant l'installation ou la migration à l'aide de la commande `scswitch`(1M) appropriée ou d'une commande équivalente. Exemples de basculement du groupe de ressources en mode non géré :

  ```
  scswitch -M -n -j ressource
  scswitch -n -j ressource
  scswitch -F -g groupe_ressources
  scswitch -u -g groupe_ressources
  ```

- Vous pouvez enregistrer un type de ressources à l'aide de la commande suivante :

  ```
  scrgadm -a -t type_ressource -f transfert_vers_fichier_RTR
  ```

- Vous pouvez migrer un type de ressources à l'aide de la commande suivante :

  ```
  scrgadm -c -j ressource -y Type_version=version \
             -y propriété=valeur \
             -x propriété=valeur ...
  ```

- L'état précédent des ressources et des groupes de ressources est restauré après la migration à l'aide de la commande `scswitch`(1M) ou d'une commande équivalente :

  ```
  scswitch -M -e -j ressource
  scswitch -e -j ressource
  scswitch -o -g groupe_ressources
  scswitch -Z -g groupe_ressources
  ```

Le développeur du type de ressources peut devoir spécifier des valeurs de capacité de réglage plus restrictives que celles utilisées dans ces exemples. Les valeurs de réglage dépendent des modifications précises qui ont été apportées à la mise en oeuvre du type de ressources. En outre, le développeur peut choisir d'utiliser un package différent du package Solaris employé dans ces exemples.

TABLEAU 3–1 Exemples de mise à niveau d'un type de ressources

Type de modification	Capacité de réglage	Configurations	Procédure
Les propriétés sont uniquement modifiées dans le fichier RTR.	`À_tout_moment`	Fournit uniquement un nouveau fichier RTR.	Exécutez la commande `pkgadd` du nouveau fichier RTR sur tous les noeuds. Enregistrez le nouveau type de ressources. Migrez la ressource.
Les méthodes sont mises à jour.	`À_tout_moment`	Le chemin d'accès aux méthodes mises à jour doit différer du chemin d'accès aux méthodes antérieures.	Exécutez la commande `pkgadd` des méthodes mises à jour sur tous les noeuds. Enregistrez le nouveau type de ressources. Migrez la ressource.
Nouveau programme détecteur.	`Lorsque_non_contrôlée`	Écrasez uniquement la méthode précédente du détecteur.	Désactivez la surveillance. Exécutez la commande `pkgadd` du nouveau programme détecteur sur tous les noeuds. Enregistrez le nouveau type de ressources. Migrez la ressource. Activez la surveillance.
Les méthodes sont mises à jour. Les nouvelles méthodes `Mise_à_jour/Arrêt` sont incompatibles avec les méthodes de `Démarrage` antérieures.	`Lorsque_hors_ligne`	Le chemin d'accès aux méthodes mises à jour doit différer du chemin d'accès aux méthodes antérieures.	Exécutez la commande `pkgadd` des méthodes mises à jour sur tous les noeuds. Enregistrez le nouveau type de ressources. Déconnectez la ressource. Migrez la ressource. Connectez la ressource.

TABLEAU 3–1 Exemples de mise à niveau d'un type de ressources *(Suite)*

Type de modification	Capacité de réglage	Configurations	Procédure
Les méthodes sont mises à jour et les nouvelles propriétés sont ajoutées aux fichiers RTR. Les nouvelles méthodes requièrent de nouvelles propriétés. (L'objectif est de permettre au groupe de ressources correspondant de rester en ligne tout en évitant de connecter la ressource, même si le groupe de ressources déconnecté est mis en ligne sur un noeud).	`Lorsque_désactivée`	Écrasez les versions précédentes des méthodes.	Désactivez la ressource. Au niveau de chaque noeud : ■ Retirez le noeud du cluster. ■ Exécutez la commande `pkgrm/pkgadd` des méthodes en cours de mise jour ■ Restaurez le noeud sur le cluster. Enregistrez le nouveau type de ressources. Migrez la ressource. Activez la ressource.
Les méthodes sont mises à jour et les nouvelles propriétés sont ajoutées aux fichiers RTR. Les nouvelles méthodes ne requièrent pas de nouvelles propriétés.	`À_tout_moment`	Écrasez les versions précédentes des méthodes.	Au niveau de chaque noeud : ■ Retirez le noeud du cluster. ■ Exécutez la commande `pkgrm/pkgadd` des méthodes en cours de mise à jour ■ Restaurez le noeud sur le cluster. Lors de cette procédure, le gestionnaire RGM exécute les nouvelles méthodes même si la migration (qui doit configurer les nouvelles propriétés) n'a pas encore été réalisée. Il est important que les nouvelles méthodes soient fonctionnelles sans les nouvelles propriétés. Enregistrez le nouveau type de ressources. Migrez la ressource.

TABLEAU 3–1 Exemples de mise à niveau d'un type de ressources *(Suite)*

Type de modification	Capacité de réglage	Configurations	Procédure
Les méthodes sont mises à jour. La nouvelle méthode `Fini` est incompatible avec la méthode `Init` antérieure.	`Lorsque_non_gérée`	Le chemin d'accès aux méthodes mises à jour doit différer du chemin d'accès aux méthodes antérieures.	Basculez le groupe de ressources correspondant en mode non géré. Exécutez la commande `pkgadd` des méthodes mises à jour sur tous les noeuds. Enregistrez le type de ressources. Migrez la ressource. Basculez le groupe de ressources correspondant en mode géré.
Les méthodes sont mises à jour. Le fichier RTR n'est pas modifié.	Sans objet. Le fichier RTR n'est pas modifié.	Écrasez les versions précédentes des méthodes.	Au niveau de chaque noeud : ■ Retirez le noeud du cluster. ■ Exécutez la commande `pkgadd` des méthodes mises à jour ■ Restaurez le noeud sur le cluster. Comme le fichier RTR n'est pas modifié, il n'est pas nécessaire d'enregistrer ou de migrer la ressource.

Conditions requises pour l'installation des packages de type de ressources

L'installation de nouveaux packages de type de ressources comportent deux conditions requises :

- Lorsqu'un nouveau type de ressources est enregistré, son fichier RTR doit être accessible sur le disque.
- Lorsqu'une ressource du nouveau type est créée, tous les noms de chemin d'accès aux méthodes déclarés, ainsi que le programme détecteur du nouveau type doivent figurer sur le disque et être exécutables. Les programmes détecteurs et de méthode antérieurs doivent être conservés au même emplacement tant que la ressource est utilisée.

Pour choisir le package le plus approprié, la personne chargée de la mise en application du type de ressources doit tenir compte des points suivants :

- Le fichier RTR change-t-il ?
- La valeur par défaut ou la capacité de réglage d'une propriété change-t-elle ?
- La valeur `min` ou `max` d'une propriété change-t-elle ?
- La mise à niveau supprime-t-elle ou ajoute-t-elle des propriétés ?
- Le code de méthode change-t-il ?
- Le code de contrôle change-t-il ?
- Les nouvelles méthodes ou le nouveau code de contrôle est-il compatibles avec la version antérieure ?

Informations utiles avant la modification d'un fichier RTR

Certaines mises à niveau du type de ressources n'impliquent ni une nouvelle méthode ni un nouveau code de contrôle. Par exemple, la mise à niveau d'un type de ressources peut n'induire que la modification de la valeur par défaut ou de la capacité de réglage d'une propriété de ressources. Comme le code de méthode ne change pas, il est uniquement nécessaire, pour installer la mise à niveau, de disposer d'un nom de chemin d'accès valide vers un fichier RTR lisible.

Il n'est pas nécessaire de réenregistrer le type de ressources précédent car le nouveau fichier RTR peut écraser la version antérieure. Dans le cas contraire, vous devez enregistrer le nouveau fichier RTR sous un nouveau nom de chemin d'accès.

Si la mise à niveau modifie la valeur par défaut ou la capacité de réglage d'une propriété, la méthode `Validation` de la nouvelle version peut vérifier, au moment de la migration, que les attributs de propriétés existants sont valides pour le nouveau type de ressources. Si la mise à niveau modifie les attributs `min`, `max` ou `type` d'une propriété, la commande `scrgadm` valide automatiquement ces contraintes au moment de la migration.

Les informations relatives à la mise à niveau doivent présenter tous les nouveaux attributs de propriétés par défaut. Elles doivent indiquer à l'administrateur système qu'il est possible de modifier les propriétés de ressources existantes sur les valeurs appropriées à l'aide de la commande servant à modifier la propriété `Version_type` pour mettre à niveau la ressource vers la nouvelle version du type de ressources.

Si la mise à niveau ajoute ou supprime des propriétés, il est probable qu'il faille également modifier le code de contrôle ou certaines méthodes de rappel.

Modification du code de contrôle

Si seul le code de contrôle a été modifié dans le type de ressources mis à jour, l'installation du package peut écraser les codes de contrôle binaires. La documentation doit spécifier à l'administrateur système de suspendre la surveillance avant d'installer le nouveau package.

Modification du code de méthode

Si seul le code de méthode a été modifié dans le type de ressources mis à jour, il est crucial de déterminer si le nouveau code de méthode est compatible avec la version antérieure. Ceci vous permet de savoir si vous devez enregistrer ce nouveau code sous un nouveau nom de chemin d'accès ou si vous pouvez écraser les méthodes antérieures.

Si vous pouvez appliquer les nouvelles méthodes d'`Arrêt`, `Arrêt_après_réseau` et `Fini` (à condition d'être déclarées) aux ressources initialisées ou exécutées par les versions antérieures des méthodes de `Démarrage`, `Arrêt_avant_réseau` ou `Init`, vous pouvez écraser les anciennes méthodes par les nouvelles.

Si le nouveau code de méthode n'est pas compatible avec la version antérieure, vous devez arrêter une ressource ou la supprimer de la configuration à l'aide des versions antérieures avant de pouvoir la migrer vers le type de ressources mis à niveau. Si les nouvelles méthodes écrasent les anciennes, il peut s'avérer nécessaire d'arrêter (voire de basculer en mode non géré) toutes les ressources de ce type avant de mettre ce dernier à niveau. Si les nouvelles et les anciennes méthodes sont enregistrées séparément (tout en étant accessibles simultanément), vous pouvez, même en l'absence de rétrocompatibilité, installer la nouvelle version du type de ressources et mettre les ressources à niveau individuellement.

Même si les nouvelles méthodes sont rétrocompatibles, il peut s'avérer nécessaire de mettre à niveau une ressource pour utiliser les nouvelles méthodes tandis que les autres ressources continuent d'utiliser les méthodes antérieures. Vous devez encore enregistrer les nouvelles méthodes dans un répertoire distinct plutôt que d'écraser les méthodes antérieures.

L'enregistrement des méthodes de chaque version du type de ressources dans un répertoire distinct permet de rebasculer les ressources vers la version antérieure en cas de problème avec la nouvelle version.

Une autre approche consiste à inclure toutes les versions précédentes encore prises en charge dans le package. Le cas échéant, le nouveau package remplace la version antérieure sans écraser ni supprimer les chemins d'accès aux méthodes antérieures. C'est au développeur du type de ressources de décider du nombre de versions antérieures prises en charge.

Remarque – nous vous recommandons d'écraser les méthodes ou les méthodes `pkgrm`/`pkgadd` sur un noeud appartenant au cluster. Si le gestionnaire RGM doit exécuter une méthode qui n'est pas accessible sur le disque, ceci risque d'engendrer des résultats inattendus. La suppression ou le remplacement du code binaire d'une méthode en cours d'exécution peut également engendrer des résultats inattendus.

CHAPITRE 4

Référence concernant l'API de gestion des ressources

Ce chapitre fournit une référence concernant les fonctions d'accès et les méthodes de rappel constituant l'API de gestion des ressources (API GR). Il énumère et décrit brièvement chaque fonction et méthode. Toutefois, la référence faisant foi pour ces fonctions et méthodes se trouve dans les pages de manuel consacrées à l'API GR.

Ce chapitree fournit les informations suivantes :

- "Méthodes d'accès à l'API GR" à la page 80 : sous la forme de commandes de script Shell et de fonctions C.
  - `scha_resource_get`(1HA), `scha_resource_close`(3HA), `scha_resource_get`(3HA), `scha_resource_open`(3HA) ;
  - `scha_resource_setstatus`(1HA), `scha_resource_setstatus`(3HA) ;
  - `scha_resourcetype_get`(1HA), `scha_resourcetype_close`(3HA), `scha_resourcetype_get`(3HA), `scha_resourcetype_open`(3HA) ;
  - `scha_resourcegroup_get`(1HA), `scha_resourcegroup_get`(3HA), `scha_resourcegroup_close`(3HA), `scha_resourcegroup_open`(3HA) ;
  - `scha_control`(1HA), `scha_control`(3HA)
  - `scha_cluster_get`(1HA), `scha_cluster_close`(3HA), `scha_cluster_get`(3HA), `scha_cluster_open`(3HA) ;
  - `scha_cluster_getlogfacility`(3HA) ;
  - `scha_cluster_getnodename`(3HA) ;
  - `scha_strerror`(3HA).
- "Méthodes de rappel API GR" à la page 85 : décrites à la page de manuel `rt_callbacks`(1HA).
  - `Démarrage` ;
  - `Arrêt` ;
  - `Init` ;
  - `Fini` ;
  - `Initialisation` ;

- `Démarrage_avant_réseau;`
- `Arrêt_après_réseau;`
- `Démarrage_détecteur;`
- `Arrêt_détecteur;`
- `Contrôle_détecteur;`
- `Mise_à_jour;`
- `Validation`.

Méthodes d'accès à l'API GR

L'API propose des fonctions permettant d'accéder aux propriétés de la ressource, du type de ressource et du groupe de ressources, ainsi qu'à d'autres informations du cluster. Ces fonctions sont fournies sous la forme de commandes shell et de fonctions C, ce qui permet aux fournisseurs de types de ressource de mettre en oeuvre des programmes de contrôle sous la forme de scripts Shell ou de programmes C.

Commandes shell API GR

Les commandes Shell doivent être utilisées dans les mises en oeuvre du script Shell des méthodes de rappel pour les types de ressource représentant des services contrôlés par le RGM du cluster. Vous pouvez utiliser ces commandes pour :

- accéder à des informations sur les ressources, les types de ressource, les groupes de ressources et les clusters ;
- définir les propriétés `Statut` et `msg_statut` d'une ressource, avec un détecteur ;
- demander le redémarrage ou le déplacement d'un groupe de ressources.

Remarque – bien que cette rubrique fournisse de brèves descriptions des commandes shell, les différentes pages de manuel, dans la rubrique 1HA, constituent la référence faisant foi pour les commandes shell. Sauf mention contraire, chaque commande correspond à une page de manuel portant le même nom.

Commandes de la ressource API GR

Ces commandes vous permettent d'accéder aux informations relatives à une ressource ou de définir les propriétés `Statut` et `msg_statut` de celle-ci.

`scha_resource_get`
: Accède aux informations relatives à une ressource ou à un type de ressource sous le contrôle du RGM. Elle fournit les mêmes informations que la fonction `scha_resource_get ()`.

`scha_resource_setstatus`
Définit les propriétés `Statut` et `msg_statut` d'une ressource sous le contrôle du RGM. Elle est utilisée par le détecteur de la ressource afin d'indiquer l'état de celle-ci tel qu'il le perçoit. Elle propose la même fonctionnalité que la fonction C `scha_resource_setstatus()`.

Remarque – bien que `scha_resource_setstatus()` soit d'un intérêt particulier pour les détecteurs de ressources, tout programme peut l'appeler.

Commande du type de ressource

Cette commande accède à des informations sur un type de ressource enregistré auprès du RGM.

`scha_resourcetype_get`
Cette commande fournit la même fonctionnalité que la fonction C `scha_resourcetype_get ()`.

Commandes du groupe de ressources

Ces commandes vous permettent d'accéder à des informations sur un groupe de ressources ou de redémarrer celui-ci.

`scha_resourcegroup_get`
Accède aux informations relatives au groupe de ressources sous le contrôle du RGM. Cette commande fournit la même fonctionnalité que la fonction C `scha_resourcetype_get ()`.

`scha_control`
Demande le redémarrage d'un groupe de ressources sous le contrôle du RGM ou son déplacement vers un autre noeud. Elle propose la même fonctionnalité que la fonction C `scha_control()`.

Commande du cluster

Cette commande accède aux informations relatives à un cluster, telles que les noms des noeuds, les ID, les états, le nom du cluster, les groupes de ressources, etc.

`scha_cluster_get`
Cette commande fournit les mêmes informations que la fonction C `scha_cluster_get ()`.

Fonctions C

Les fonctions C doivent être utilisées dans les mises en oeuvre du programme C des méthodes de rappel pour les types de ressource représentant des services contrôlés par le RGM du cluster. Ces fonctions vous permettent d'effectuer les opérations suivantes :

- accéder à des informations sur les ressources, les types de ressource, les groupes de ressources et les clusters ;
- définir les propriétés `Statut` et `msg_statut` d'une ressource, avec un détecteur ;
- demander le redémarrage ou le déplacement d'un groupe de ressources ;
- convertir un code d'erreur en message d'erreur approprié.

Remarque – bien que cette rubrique fournisse de brèves descriptions des fonctions C, les différentes pages de manuel `(3HA)` constituent la référence faisant foi pour les fonctions C. Sauf mention contraire, chaque fonction correspond à une page de manuel portant le même nom. Reportez-vous à la page de manuel `scha_calls`(3HA) pour obtenir des informations sur les arguments de sortie et les codes de retour des fonctions C.

Fonctions des ressources

Ces fonctions accèdent aux informations relatives à une ressource gérée par le RGM ou indiquent l'état de celle-ci tel qu'il est perçu par le détecteur.

`scha_resource_open()`, `scha_resource_get ()` et `scha_resource_close()`
Ensemble, ces fonctions accèdent à des informations sur une ressource gérée par le RGM. La fonction `scha_resource_open()` initialise l'accès à une ressource et renvoie un identificateur pour `scha_resource_get ()`, lequel accède aux informations de la ressource. La fonction `scha_resource_close ()` invalide l'identificateur et libère la mémoire allouée aux valeurs de retour `scha_resource_get()`.

Une ressource peut changer, à la suite d'une reconfiguration du cluster ou d'une action de l'administrateur, après que la fonction `scha_resource_open()` a renvoyé l'identificateur de la ressource, auquel cas les informations obtenues par `scha_resource_get()` par le biais de celui-ci risquent d'être erronées. Dans le cas d'une reconfiguration du cluster ou d'une action de l'administrateur sur une ressource, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resource_get()` pour indiquer qu'il est possible que les informations concernant la ressource aient changé. Il s'agit d'un message d'erreur non fatale. La fonction renvoie une réussite. Vous pouvez choisir d'ignorer le message et d'accepter l'information renvoyée ou de fermer l'identificateur actuel et d'en ouvrir un nouveau pour accéder aux informations relatives à la ressource.

Ces trois fonctions sont décrites sur une même page de manuel. Vous pouvez accéder à cette page par le biais des fonctions `scha_resource_open`(3HA), `scha_resource_get`(3HA) ou `scha_resource_close`(3HA).

`scha_resource_setstatus()`
Définit les propriétés `Statut` et `msg_statut` d'une ressource sous le contrôle du RGM. Le détecteur de la ressource utilise cette fonction pour indiquer l'état de celle-ci.

Remarque – bien que `scha_resource_setstatus()` soit d'un intérêt particulier pour les détecteurs de ressources, tout programme peut l'appeler.

Fonctions du type de ressource

Ensemble, ces fonctions accèdent aux informations sur le type de ressource enregistré avec le RGM.

`scha_resourcetype_open()`, `scha_resourcetype_get()`, `scha_resourcetype_close()`
La fonction `scha_resourcetype_open()` initialise l'accès à une ressource et renvoie un identificateur pour `scha_resourcetype_get ()`, lequel accède aux informations du type de ressource. La fonction `scha_resourcetype_close ()` invalide l'identificateur et libère la mémoire allouée aux valeurs de retour `scha_resourcetype_get()`.

Une ressource peut changer, à la suite d?une reconfiguration du cluster ou d?une action de l?administrateur, après que la fonction `scha_resourcetype_open()` a renvoyé l?identificateur de la ressource, auquel cas les informations obtenues par `scha_resourcetype_get ()` par le biais de celui-ci risquent d?être erronées. Dans le cas d'une reconfiguration du cluster ou d'une action de l'administrateur sur un type de ressource, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resourcetype_get()` pour indiquer qu'il est possible que les informations concernant le type de ressource aient changé. Il s'agit d'un message d'erreur non fatale. La fonction renvoie une réussite. Vous pouvez choisir d'ignorer le message et d'accepter l'information renvoyée ou de fermer l'identificateur actuel et d'en ouvrir un nouveau pour accéder aux informations relatives au type de ressource.

Ces trois fonctions sont décrites sur une même page de manuel. Vous pouvez accéder à cette page de manuel par le biais de chacune des fonctions `scha_resourcetype_open`(3HA), `scha_resourcetype_get`(3HA) ou `scha_resourcetype_close`(3HA).

Fonctions du groupe de ressources

Ces fonctions vous permettent d'accéder aux informations relatives au groupe de ressources ou de redémarrer celui-ci.

`scha_resourcegroup_open`(3HA), `scha_resourcegroup_get`(3HA) et `scha_resourcegroup_close`(3HA)
: Ensemble, ces fonctions accèdent à des informations sur un groupe de ressources géré par le RGM. La fonction `scha_resourcegroup_open()` initialise l'accès à un groupe de ressources et renvoie un identificateur pour `scha_resourcegroup_get ()`, lequel accède aux informations du groupe de ressources. La fonction `scha_resourcegroup_close ()` invalide l'identificateur et libère la mémoire allouée aux valeurs de retour `scha_resourcegroup_get()`.

  Un groupe de ressources peut changer, à la suite d'une reconfiguration du cluster ou d'une action de l'administrateur, après que la fonction `scha_resourcegroup_open()` a renvoyé l'identificateur du groupe de ressources, auquel cas les informations obtenues par `scha_resourcegroup_get ()` par le biais de celui-ci risquent d'être erronées. Dans le cas d'une reconfiguration du cluster ou d'une action de l'administrateur sur un groupe de ressources, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resourcegroup_get()` pour indiquer qu'il est possible que les informations concernant le groupe de ressources aient changé. Il s'agit d'un message d'erreur non fatale. La fonction renvoie une réussite. Vous pouvez choisir d'ignorer le message et d'accepter l'information renvoyée ou de fermer l'identificateur actuel et d'en ouvrir un nouveau pour accéder aux informations relative au groupe de ressources.

`scha_control`(3HA)
: Demande le redémarrage d'un groupe de ressources sous le contrôle du RGM ou son déplacement vers un autre noeud.

Fonctions du cluster

Ces fonctions accèdent à des informations sur un cluster ou en renvoient.

`scha_cluster_open`(3HA), `scha_cluster_get`(3HA), `scha_cluster_close`(3HA)
: Ensemble, ces fonctions accèdent à des informations relatives à un cluster, telles que les noms des noeuds, les ID, les états, le nom du cluster, les groupes de ressources, etc.

  Un cluster peut changer, à la suite d'une reconfiguration du cluster ou d'une action de l'administrateur, après que la fonction `scha_cluster_open()` a renvoyé l'identificateur du cluster, auquel cas les informations obtenues par `scha_cluster_get()` par le biais de celui-ci risquent d'être erronées. Dans le cas d'une reconfiguration ou d'une action de l'administrateur sur un cluster, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_cluster_get()` pour indiquer qu'il est possible que les informations concernant le cluster aient changé. Il s'agit d'un message d'erreur non fatale. La fonction renvoie une réussite. Vous pouvez choisir d'ignorer le message et d'accepter l'information renvoyée ou de fermer l'identificateur actuel et d'en ouvrir un nouveau pour accéder aux informations relatives au cluster.

`scha_cluster_getlogfacility`(3HA)
Renvoie le numéro de la fonction syslog utilisée comme journal du cluster. Utilise la valeur renvoyée à l'aide de la fonction `syslog()` de Solaris pour enregistrer les événements et les messages d'état dans le journal du cluster.

`scha_cluster_getnodename`(3HA)
Renvoie le nom du noeud du cluster sur lequel est appelée la fonction.

Fonction de l'utilitaire

Cette fonction convertit un code d'erreur en message d'erreur.

`scha_strerror`(3HA)
Traduit un code d'erreur, renvoyé par l'une des fonctions `scha_`, en message d'erreur. Utilisez cette fonction avec la commande `logger` pour consigner les messages dans le journal système (`syslog`).

Méthodes de rappel API GR

Les méthodes de rappel sont les éléments clés fournis par l'API pour la mise en oeuvre d'un type de ressource. Les méthodes de rappel permettent au RGM de contrôler les ressources du cluster en cas de modification des membres de celui-ci, par exemple en cas d'initialisation ou de blocage du noeud.

Remarque – les méthodes de rappel sont exécutées par le RGM avec les autorisations racine étant donné que les programmes client contrôlent les services HA sur le système du cluster. Installez et gérez ces méthodes avec des membres et des autorisations de fichiers restrictifs. Plus spécifiquement, donnez-leur un propriétaire privilégié, par exemple `bin` ou `root` et rendez-les inaccessibles en écriture.

Cette rubrique décrit les arguments et codes de sortie des méthodes de rappel. Par ailleurs, elle énumère et décrit les méthodes de rappel des différentes catégories :

- méthodes de contrôle et d'initialisation ;
- méthodes d'assistance à l'administration ;
- méthodes relatives au Net ;
- méthodes de contrôle des détecteurs.

Remarque – bien que cette rubrique fournisse de brèves descriptions des méthodes de rappel, comprenant le moment auquel la méthode est appelée et l'effet escompté sur la ressource, la page de manuel `rt_callbacks`(1HA) constitue la référence faisant foi pour les méthodes de rappel.

Arguments de méthode

Le RGM exécute les méthodes de rappel de la manière suivante :

méthode `-R` *nom_ressource* `-T` *nom_type* `-G` *nom_groupe*

La méthode est le nom du chemin d'accès enregistré comme `Démarrage`, `Arrêt` ou autre rappel. Les méthodes de rappel d'un type de ressource sont déclarées dans son fichier d'enregistrement.

Tous les arguments des méthodes de rappel sont transmis comme des valeurs marquées, avec `-R` indiquant le nom de l'instance de la ressource, `-T` indiquant le type de la ressource et `-G` indiquant le groupe dans lequel est configurée la ressource. Ayez recours aux arguments disposant de fonctions d'accès pour récupérer des informations sur la ressource.

La méthode `Validation` est appelée avec des arguments supplémentaires (les valeurs des propriétés de la ressource et du groupe de ressources ciblées par l'appel).

Reportez-vous à `scha_calls`(3HA) pour obtenir de plus amples informations.

Codes de sortie

Toutes les méthodes de rappel possèdent les mêmes codes de sortie définis pour spécifier l'effet de l'exécution de la méthode sur l'état de la ressource. La page de manuel `scha_calls`(3HA) décrit tous les codes de sortie. Les codes de sortie sont les suivants :

- 0 : méthode couronnée de succès ;
- valeur différente de zéro : méthode se soldant par un échec.

Le RGM gère également les échecs anormaux rencontrés lors de l'exécution de la méthode de rappel, tels que les dépassements de délai imparti et les core dumps.

Les mises en oeuvre de la méthode doivent sortir des informations d'échec à l'aide de `syslog` sur chaque noeud. L'envoi des sorties écrites sur `stdout` ou sur `stderr` à l'utilisateur n'est pas garanti, bien que celui-ci apparaisse sur la console du noeud local.

Méthodes de rappel de contrôle et d'initialisation

Les principales méthodes de rappel de contrôle et d'initialisation démarrent et arrêtent les ressources. Les autres méthodes exécutent un code d'initialisation et d'arrêt sur les ressources.

`Démarrage`
: Cette méthode requise est exécutée sur un noeud du cluster lorsque le groupe contenant la ressource est mis en ligne sur ce noeud. Cette méthode active la ressource sur ce noeud.

  Une méthode `Démarrage` ne doit pas se fermer avant que la ressource qu'elle active ait été démarrée et soit disponible sur le noeud local. Par conséquent, avant sa fermeture, la méthode `Démarrage` doit interroger la ressource afin de déterminer si elle a démarré. Par ailleurs, vous devez définir un délai imparti suffisamment long pour cette méthode. Par exemple, certaines ressources, telles que les démons de base de données, prennent plus de temps au démarrage et ont donc besoin d'une méthode prévoyant un délai plus long.

  La manière dont le RGM réagit à l'échec de la méthode `Démarrage` dépend du paramètre de la propriété `Mode_basculement`.

  La propriété `DÉLAI_DÉMARRAGE` du fichier d'enregistrement du type de ressource définit la valeur du délai imparti pour une méthode de `Démarrage` de ressource.

`Arrêt`
: Cette méthode requise est appelée sur un noeud du cluster lorsque le groupe contenant la ressource est mis hors ligne sur ce noeud. Cette méthode désactive la ressource si elle est active.

  Une méthode `Arrêt` ne doit pas se fermer avant que la ressource qu'elle contrôle ait complètement arrêté toute activité sur le noeud local et ait fermé tous les descripteurs de fichier. Dans le cas contraire, comme le RGM suppose que la ressource est arrêtée alors qu'elle est toujours active, une corruption de données peut avoir lieu. La manière la plus sûre d'éviter une corruption de données consiste à mettre un terme à tous les processus sur le noeud local associé à la ressource.

  Avant sa fermeture, la méthode `Arrêt` doit interroger la ressource afin de déterminer si elle est arrêtée. Par ailleurs, vous devez définir un délai imparti suffisamment long pour cette méthode. Par exemple, certaines ressources, telles que les démons de base de données, prennent plus de temps à l'arrêt et ont donc besoin d'une méthode prévoyant un délai plus long.

  La manière dont le RGM réagit à l'échec de la méthode `Arrêt` dépend du paramètre de la propriété `Mode_basculement` (reportez-vous au Tableau A–2).

  La propriété `DÉLAI_ARRÊT` du fichier d'enregistrement du type de ressource définit la valeur du délai imparti pour une méthode `Arrêt` de ressource.

`Init`
Cette méthode optionnelle est appelée pour effectuer une initialisation unique de la ressource lorsque celle-ci est gérée, soit quand le groupe de ressources auquel elle appartient passe d'un état non géré à un état géré soit lorsque la ressource est créée dans un groupe de ressources déjà géré. La méthode est appelée sur les noeuds déterminés par la propriété de ressource `Noeuds_init`.

`Fini`
Cette méthode facultative est appelée pour faire le ménage lorsque la ressource n'est plus gérée, soit quand le groupe auquel elle appartient n'est plus géré soit quand elle est supprimée d'un groupe de ressources géré. La méthode est appelée sur des noeuds déterminés par la propriété de ressource `Noeuds_init`.

`Initialisation`
Cette méthode facultative, analogue à Init, est appelée pour initialiser la ressource sur des noeuds rejoignant le cluster après que le groupe contenant la ressource a déjà été placé sous le contrôle du RGM. La méthode est appelée sur des noeuds déterminés par la propriété de ressource `Noeuds_init`. La méthode `Initialisation` est appelée lorsque le noeud adhère à un cluster à la suite d'une initialisation ou d'une réinitialisation.

Remarque – un échec des méthodes `Init`, `Fini` ou `Initialisation` entraîne la génération, par la fonction `syslog()`, d'un message d'erreur, mais n'affecte en rien la gestion de la ressource par le RGM.

Méthodes d'assistance à l'administration

Les tâches d'administration concernant les ressources comprennent entre autres le paramétrage et la modification des propriétés de la ressource. Les méthodes de rappel `Validation` et `Mise_à_jour` permettent à la mise en oeuvre d'un type de ressource de passer par ces opérations d'administration.

`Validation`
Cette méthode facultative est appelée lorsqu'une ressource est créée et lorsque l'administrateur met à jour les propriétés de la ressource ou du groupe la contenant. Cette méthode est appelée sur le jeu de noeuds du cluster indiqué par la propriété Noeuds_init du type de ressource. `Validation` est appelée avant la création ou la mise à jour, et un code de sortie avec échec renvoyé par la méthode sur un noeud entraîne l'annulation de l'opération envisagée.

`Validation` n'est appelée que lorsque les propriétés de la ressource ou du groupe sont modifiées par une opération de l'administrateur, et non lorsque le RGM définit des propriétés, ou lorsqu'un détecteur définit les propriétés `Statut` et `msg_statut` de la ressource.

`Mise_à_jour`
Cette méthode facultative est appelée pour signaler à une ressource en cours d'exécution que les propriétés ont été modifiées. `Mise_à_jour` est appelée après la réussite d'une action de l'administrateur visant à définir les propriétés d'une ressource ou de son groupe. Cette méthode est appelée sur les noeuds lorsque la ressource est en ligne. La méthode utilise les fonctions d'accès de l'API pour lire les valeurs des propriétés susceptibles d'affecter une ressource active et régler la ressource en cours d'exécution en conséquence.

Un échec de la méthode `Mise_à_jour` pousse la fonction `syslog()` à générer un message d'erreur mais n'affecte en rien la gestion de la ressource par le RGM.

Méthodes de rappel relatives au Net

Il est possible que les services utilisant des ressources d'adresses réseau exigent que les étapes menant à un démarrage ou à un arrêt soient effectuées dans un certain ordre, en fonction de la configuration de l'adresse réseau. Les méthodes de rappel facultatives `Démarrage_avant_réseau` et `Arrêt_après_réseau` permettent à la mise en oeuvre d'un type de ressource d'effectuer des actions spéciales de démarrage et de fermeture avant et après la configuration ou la déconfiguration d'une adresse réseau liée.

`Démarrage_avant_réseau`
Cette méthode facultative est appelée pour entreprendre des actions de démarrage spéciales avant la configuration des adresses réseau du même groupe de ressources.

`Arrêt_après_réseau`
Cette méthode facultative est appelée pour effectuer des actions de fermeture spéciales une fois les adresses réseau du même groupe de ressources déconfigurées.

Méthodes de rappel de contrôle des détecteurs

La mise en oeuvre d'un type de ressource peut inclure, de manière facultative, un programme permettant de surveiller la performance d'une ressource, de faire rapport sur son état ou de réagir en cas d'échec. Les méthodes `Démarrage_détecteur`, `Arrêt_détecteur` et `Contrôle_détecteur` prennent en charge la mise en oeuvre d'un détecteur de ressources dans le type de ressource.

`Démarrage_détecteur`
Cette méthode facultative est appelée pour démarrer un détecteur destiné à la ressource lorsque celle-ci a démarré.

`Arrêt_détecteur`
Cette méthode facultative est appelée pour arrêter un détecteur destiné à la ressource avant l'arrêt de celle-ci.

`Contrôle_détecteur`
: Cette méthode facultative est appelée pour évaluer la fiabilité d'un noeud avant le déplacement du groupe de ressources sur celui-ci. La méthode `Contrôle_détecteur` doit être mise en oeuvre de manière à ne pas entrer en conflit avec d'autres méthodes exécutées simultanément.

CHAPITRE 5

Service de données modèle

Ce chapitre décrit un service de données Sun Cluster modèle, HA-DNS, pour l'application `in.named`. Le démon `in.named` constitue la mise en oeuvre, sous Solaris, du service DNS (Domain Name Service). Ce service de données modèle montre comment rendre une application hautement disponible à l'aide de l'API de gestion des ressources.

Celle-ci prend en charge une interface de script Shell ainsi qu'une interface de programme C. L'application modèle présentée dans ce chapitre est rédigée à l'aide de l'interface de script Shell.

Les informations fournies par ce chapitre sont les suivantes :

Présentation du service de données modèle

Le service de données modèle lance, arrête, redémarre et bascule l'application DNS entre les noeuds du cluster en réponse à des événements de cluster, tels qu'une action de l'administrateur, un échec d'application ou une erreur de noeud.

Le redémarrage de l'application est géré par le gestionnaire de processus. Si les morts d'application dépassent le nombre d'échecs autorisés pour le délai de comptabilisation des erreurs, le détecteur de pannes bascule le groupe de ressources contenant la ressource d'application vers un autre noeud.

Le service de données modèle fournit une surveillance des pannes sous la forme d'une méthode `SONDE`. Celle-ci utilise la commande `nslookup` pour s'assurer que l'application est saine. Si la sonde détecte un service DNS bloqué, elle tente de rectifier la situation en redémarrant l'application DNS localement. Si cette opération ne résout pas le problème et si la sonde détecte à plusieurs reprises des blocages de ce service, elle tente de le basculer vers un autre noeud du cluster.

Plus particulièrement, le service de données modèle comprend :

- Un fichier d'enregistrement du type de ressource identifiant les propriétés statiques du service de données.
- Une méthode de rappel de `Démarrage` appelée par le RGM de manière à démarrer le démon `in.named` lorsque le groupe de ressources contenant le service de données HA-DNS est mis en ligne.
- Une méthode de rappel d'`Arrêt` appelée par le RGM pour arrêter le démon `in.named` lorsque le groupe de ressources contenant HA-DNS passe hors ligne.
- Un détecteur de pannes permettant de vérifier la disponibilité du service en s'assurant que le serveur DNS tourne. Le détecteur de pannes est mis en oeuvre par une méthode `SONDE` définie par l'utilisateur. Il est lancé et arrêté par les méthodes de rappel de `Démarrage_détecteur` et d'`Arrêt_détecteur`.
- Une méthode de rappel de `Validation` appelée par le RGM afin de valider l'accessibilité du répertoire de configuration du service.
- Une méthode de rappel de `Mise_à_jour` appelée par le RGM afin de redémarrer le détecteur de pannes quand l'administrateur système modifie la valeur d'une propriété de ressource.

Définition du fichier d'enregistrement du type de ressource

Le fichier d'enregistrement du type de ressource (RTR) de cet exemple définit la configuration statique du type de ressource DNS. Les ressources de ce type héritent des propriétés définies dans le fichier RTR.

Les informations contenues dans le fichier RTR sont lues par le RGM lorsque l'administrateur du cluster enregistre le service de données HA-DNS.

Présentation du fichier RTR

Le fichier RTR suit un format bien défini. Les propriétés du type de ressources sont définies en premier lieu dans le fichier, suivent les propriétés de ressource définies par le système, puis les propriétés d'extension. Reportez-vous à la page de manuel `rt_reg`(4) et à la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 34 pour de plus amples informations.

Cette rubrique décrit les propriétés spécifiques du fichier RTR modèle. Elle fournit également une liste des différentes parties du fichier. Pour obtenir une liste complète du contenu de ce fichier, reportez-vous à la rubrique "Liste de code du fichier RTR" à la page 277.

Propriétés du type de ressource dans le fichier RTR modèle

Le fichier RTR modèle commence par des commentaires, suivis des propriétés du type de ressource définissant la configuration de HA-DNS, comme vous pouvez le voir dans la liste suivante :

```
#
# Copyright (c) 1998-2003 by Sun Microsystems, Inc.
# Tous droits réservés
#
# Informations relatives à l'enregistrement du service DNS (Domain Name Service)
#

#pragma ident   "@(#)SUNW.sample   1.1   00/05/24 SMI"

TYPE_RESSOURCE = "modèle";
ID_FOURNISSEUR = SUNW;
DESCRIPTION_TR = "service DNS sur Sun Cluster"


VERSION_TR ="1.0";
VERSION_API = 2;
BASCULEMENT = VRAI;

RÉP_BASE_TR=/opt/SUNWsample/bin;
LISTE_PACKAGES = SUNWsample;

DÉMARRAGE = dns_svc_start;
ARRÊT = dns_svc_stop;

VALIDATION = dns_validate;
MISE_À_JOUR = dns_update;

DÉMARRAGE_DÉTECTEUR = dns_monitor_start;
ARRÊT_DÉTECTEUR = dns_monitor_stop;
CONTRÔLE_DÉTECTEUR = dns_monitor_check;
```

Astuce – la propriété `Type_ressource` doit être déclarée en tant que première entrée dans le fichier RTR. Dans le cas contraire, l'enregistrement du type de ressources échoue.

Remarque – le gestionnaire RGM est insensible à la casse dans les noms de propriété. Les fichiers RTR fournis par Sun sont basés sur la convention suivante : la première lettre du nom est en majuscules tandis que toutes les autres sont en minuscules. Cette convention s'applique aux noms de propriété mais pas aux noms de méthode. Les noms de méthodes et les attributs de propriété sont en majuscules.

Vous trouverez présentées ci-dessous des informations sur ces propriétés :

- Le nom du type de ressources peut être spécifié par la propriété `Type_ressource` seule (`sample`) ou en utilisant l'`Id_fournisseur` comme préfixe avec un "." le séparant du type de ressources (`SUNW.sample`).

  Si vous utilisez `id_fournisseur`, définissez le type de ressources à l'aide du symbole boursier de l'entreprise. Le nom du type de ressources doit être unique sur le cluster.
- La propriété `Version_TR` identifie la version du service de données modèle telle que spécifiée par le fournisseur.
- La propriété `Version_API` identifie la version de Sun Cluster. Par exemple, `Version_API = 2` indique que le service de données tourne sous Sun Cluster version 3.0.
- `Basculement = VRAI` indique que le service de données ne peut pas tourner dans un groupe de ressources susceptible d'être en ligne sur plusieurs noeuds à la fois.
- `rép_base_TR` pointe vers `/opt/SUNWsample/bin` comme le chemin de répertoire permettant de compléter les chemins relatifs, tels que les chemins de méthode de rappel.
- `Démarrage`, `Arrêt`, `Validation`, proposent les chemins pointant vers les programmes de méthode de rappel respectifs appelés par le RGM. Ces chemins d'accès dépendent du répertoire spécifié par `rép_base_TR`.
- `Liste_packages` identifie `SUNWsample` comme le package contenant l'installation du service de données modèle.

Les propriétés du type de ressource non spécifiées dans ce fichier RTR, par exemple `Instance_unique`, `noeuds_init` et `Noeuds_installés`, reçoivent leur valeur par défaut. Reportez-vous au Tableau A–1 pour obtenir une liste complète des propriétés du type de ressource, avec leurs valeurs par défaut.

L'administrateur du cluster ne peut pas modifier les valeurs spécifiées pour les propriétés du type de ressource dans le fichier RTR.

Propriétés de ressource dans le fichier RTR modèle

Par convention, déclarez les propriétés de ressource selon les propriétés du type de ressource dans le fichier RTR. Les propriétés de ressource comprennent les propriétés définies par le système et fournies par Sun Cluster, ainsi que les propriétés d'extension définies par l'utilisateur. Pour chaque type, vous pouvez spécifier un nombre d'attributs de propriété fournis par Sun Cluster, telles que des valeurs minimum et maximum ainsi que des valeurs par défaut.

Propriétés définies par le système dans le fichier RTR

La liste énumérée ci-dessous présente les propriétés définies par le système dans le fichier RTR.

```
# Une liste des déclarations de propriétés de ressources entre crochets suit les
# déclarations relatives au type de ressource. La déclaration du nom de la propriété doit
# être le premier attribut après le crochet ouvrant de chaque entrée.

# Les propriétés <Délai_exécution_[méthode] définissent le délai (en secondes) après lequel
# le RGM conclut que l'appel de la méthode a échoué

# La valeur MIN de tous les délais impartis de la méthode est définie à 60 secondes. Ceci
# évite que les administrateurs définissent des délais plus courts qui non seulement
# n'améliorent pas les performances de basculement, mais peuvent générer des actions
# non souhaitées de la part du RGM (basculements erronés, réinitialisation de noeuds
# ou passage du groupe de ressources à l'état ERREUR_ÉCHEC_ARRÊT, requérant une intervention
# de l'opérateur). La définition de délais trop courts pour la méthode entraîne une
# *diminution* de la disponibilité générale du service de données.
{
   PROPRIÉTÉ = Délai_démarrage;
MIN=60;
PAR DÉFAUT=300;
}

{
   PROPRIÉTÉ = Délai_arrêt;
MIN=60;
PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Délai_validation;
MIN=60;
PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Délai_mise_à_jour;
```

```
MIN=60;
PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Délai_démarrage_détecteur;
MIN=60;
        PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Délai_arrêt_détecteur;
MIN=60;
PAR DÉFAUT=300;
}
{
        PROPRIÉTÉ = Intervalle_sonde_complet;
MIN=1;
MAX=3600;
PAR DÉFAUT=60;
RÉGLABLE = À_TOUT_MOMENT;
}

# Le nombre de nouvelles tentatives à effectuer au cours d'une période donnée avant de conclure
# que l'application ne peut pas démarrer correctement sur ce noeud.
{
       PROPRIÉTÉ = Nombre_nouvelles_tentatives;
MIN=0;
MAX=10;
PAR DÉFAUT=2;
RÉGLABLE = À_TOUT_MOMENT;
}

# Définissez une valeur multiple de 60 pour Intervalle_nouvelles_tentatives, étant donné que
# cette valeur est convertie de secondes en minutes, avec arrondissement au chiffre supérieur.
# Par exemple, une valeur de 50 (secondes) est convertie en 1 minute. Utilisez cette propriété
# pour établir le délai séparant les nouvelles tentatives (Nombre_nouvelles_tentatives).
{
        PROPRIÉTÉ = Intervalle_nouvelles_tentatives;
MIN=60;
MAX=3600;
PAR DÉFAUT=300;
RÉGLABLE = À_TOUT_MOMENT;
}

{
       PROPRIÉTÉ = Ressources_réseau_utilisées;
RÉGLABLE = À_LA_CRÉATION;
PAR DÉFAUT = "";
}
```

Bien que Sun Cluster fournisse les propriétés définies par le système, vous pouvez définir d'autres valeurs par défaut à l'aide des attributs de propriété de ressource. Reportez-vous à la rubrique "Attributs des propriétés de ressources" à la page 274 pour une liste complète des attributs disponibles pour une application aux propriétés de ressource.

Remarquez les points suivants sur les propriétés de ressource définies par le système dans le fichier RTE modèle :

- Sun Cluster fournit une valeur minimale (1 seconde) et une valeur par défaut (3600 secondes) pour tous les dépassements de délais impartis. Le fichier RTR modèle modifie le minimum à 60 et fait passer la valeur par défaut à 300 secondes. Un administrateur de clusters peut accepter cette valeur par défaut ou modifier la valeur du délai imparti (à 60 ou plus). Sun Cluster n'impose pas de valeur maximale autorisée.
- Les propriétés `Intervalle_sonde_complet`, `Nombre_nouvelles_tentatives` et `Intervalle_nouvelles_tentatives` ont la valeur `À_TOUT_MOMENT` pour l'attribut `RÉGLABLE`. Ce paramètre signifie que l'administrateur du cluster peut modifier la valeur de ces propriétés, même lorsque le service de données tourne. Celles-ci sont utilisées par le détecteur de pannes mis en oeuvre par le service de données modèle. Le service de données modèle met en oeuvre une méthode `Mise_à_jour` pour arrêter et redémarrer le détecteur de pannes lors de la modification de ces propriétés de ressource ou d'autres par l'administrateur. Reportez-vous à la rubrique "Méthode `Mise_à_jour`" à la page 123.
- Les propriétés de ressource sont classifiées comme :
  - *requises* : l'administrateur du cluster doit spécifier une valeur à la création d'une ressource.
  - *optionnelles* : si l'administrateur ne spécifie pas de valeur, le système en renseigne une par défaut.
  - *conditionnelles* : le RGM ne crée la propriété que si elle est déclarée dans le fichier RTR.

  Le détecteur de pannes du service de données modèle utilise les propriétés conditionnelles `Intervalle_sonde_complet`, `Nombre_nouvelles_tentatives`, `Intervalle_nouvelles_tentatives` et `Ressources_réseau_utilisées`, de telle manière que le développeur a dû les déclarer dans le fichier RTR. Reportez-vous à la page de manuel `r_properties`(5) ou à la rubrique "Propriétés des ressources" à la page 257 pour obtenir des informations sur la manière dont les propriétés sont classifiées.

Propriétés d'extension dans le fichier RTR

Les propriétés d'extension se trouvent à la fin du fichier RTR modèle, comme le montre la liste suivante :

```
# Propriétés d'extension

# L'administrateur du cluster doit définir la valeur de cette propriété afin qu'elle pointe
# vers le répertoire contenant les fichiers de configuration utilisés par l'application.
# Pour cette application, DNS, spécifiez le chemin du fichier de configuration DNS sur
# PXFS (généralement named.conf).
{
```

```
PROPRIÉTÉ = Rép_conf;
EXTENSION;
CHAÎNE;
RÉGLABLE = À_LA_CRÉATION;
DESCRIPTION = "Chemin du répertoire de configuration";
}

# Délai en secondes avant la déclaration d'un échec de la sonde.
{
PROPRIÉTÉ = Délai_sonde;
EXTENSION;
INT;
PAR DÉFAUT = 120;
RÉGLABLE = À_TOUT_MOMENT;
DESCRIPTION = "Délai imparti pour la sonde (secondes)";
}
```

Le fichier RTR modèle définit deux propriétés d'extension, `Rép_conf` et `Délai_sonde`. `Rép_conf` spécifie le chemin vers le répertoire de configuration DNS. Ce répertoire contient le fichier `in.named` requis par le DNS pour fonctionner correctement. Les méthodes `Démarrage` et de `Validation` du service de données modèle utilisent cette propriété pour vérifier que le répertoire de configuration et le fichier `in.named` sont accessibles avant de démarrer le DNS.

Lorsque le service de données est configuré, la méthode `Démarrage` vérifie que le nouveau répertoire est accessible.

La méthode `SONDE` du service de données modèle n'est pas une méthode de rappel de Sun Cluster, mais une méthode définie par l'utilisateur. C'est la raison pour laquelle Sun Cluster ne propose aucune propriété `Délai_sonde` pour celle-ci. Le développeur a défini une propriété d'extension dans le fichier RTR afin de permettre à un administrateur de cluster de configurer une valeur Délai_sonde.

Fonctionnalité commune à toutes les méthodes

Cette rubrique décrit la fonctionnalité suivante, utilisée dans toutes les méthodes de rappel du service de données modèle :

Identification de l'interpréteur de commandes et exportation du chemin

La première ligne d'un script Shell doit identifier l'interpréteur de commandes. Chacun des scripts de méthode du service de données modèle identifie l'interpréteur de commandes de la manière suivante :

```
#!/bin/ksh
```

Tous les scripts de méthode de l'application modèle exportent le chemin vers les binaires et les bibliothèques de Sun Cluster au lieu de se fier aux paramètres `PATH` de l'utilisateur.

```
#############################################################################
# MAIN
#############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH
```

Déclaration des variables `PMF_TAG` et `SYSLOG_TAG`

Tous les scripts de méthode (à l'exception de `Validation`) utilisent `pmfadm` pour lancer (ou arrêter) le service de données ou le détecteur, en transmettant le nom de la ressource. Chaque script définit une variable `PMF_TAG` pouvant être transmise à `pmfadm` pour identifier le service de données ou le détecteur.

De la même manière, chaque script de méthode a recours à la commande `logger` pour consigner des messages dans le journal système. Chaque script définit une variable `SYSLOG_TAG` pouvant être transmise à `logger` avec l'option `-t` pour identifier le type de ressource, le groupe de ressources ainsi que le nom de la ressource pour laquelle est consigné le message.

Toutes les méthodes définissent `SYSLOG_TAG` de la même manière, comme l'indique l'exemple suivant. Les méthodes `Sonde_dns`,, de `démarrage_svc_dns`, d'`arrêt_svc_dns` et de `contrôle_détecteur_dns` définissent `PMF_TAG` comme suit (l'usage de `pmfadm` et `logger` provient de la méthode `dns_svc_stop`) :

```
#############################################################################
# MAIN
#############################################################################

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # Envoi d'un signal SIGTERM au service de données et attente pendant 80 % du
   # délai total imparti.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
```

```
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.info \
           -t [$SYSLOG_TAG] \
           "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
            SIGKILL"
```

Les méthodes `Démarrage_détecteur_dns`, `Arrêt_détecteur_dns` et `Mise_à_jour_dns` définissent `PMF_TAG` comme suit (l'usage de `pmfadm` provient de la méthode `Arrêt_détecteur_dns`) :

```
##############################################################################
# MAIN
##############################################################################

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
...

# Voir si le détecteur tourne et, le cas échéant, l'arrêter.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

Analyse des arguments de la fonction

Le RGM appelle toutes les méthodes de rappel, à l'exception de `Validation`, de la manière suivante :

nom_méthode `-R` *nom_ressource* `-T` *nom_type_ressource* `-G` *nom_groupe_ressources*

Le nom de la méthode correspond au nom de chemin d'accès du programme mettant en oeuvre la méthode de rappel. Un service de données spécifie le nom de chemin d'accès de chaque méthode du fichier RTR. Ces noms dépendent du répertoire spécifié par la propriété `rép_base_TR`, également dans le fichier RTR. Par exemple, dans le fichier RTR du service de données modèle, le répertoire de base et les noms des méthodes sont spécifiés de la manière suivante :

```
RÉP_BASE_TR=/opt/SUNWsample/bin;
DÉMARRAGE = démarrage_svc_dns;
ARRÊT = arrêt_svc_dns;
...
```

Tous les arguments des méthodes de rappel sont transmis comme des valeurs marquées, avec `-R` indiquant le nom de l'instance de la ressource, `-T` indiquant le type de la ressource et `-G` indiquant le groupe dans lequel est configurée la ressource. Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour de plus amples informations sur les méthodes de rappel.

Remarque – la méthode `Validation` est appelée avec des arguments supplémentaires (les valeurs des propriétés de la ressource et du groupe de ressources sources de l'appel). Reportez-vous à la rubrique"Gestion des mises à jour des propriétés" à la page 119 pour de plus amples informations.

Chaque méthode de rappel a besoin d'une fonction pour analyser les arguments transmis. Étant donné que tous les rappels sont transmis avec les mêmes arguments, le service de données fournit une seule fonction d'analyse employée dans tous les rappels de l'application.

L'exemple suivant montre l'utilisation de la fonction `parse_args()` pour les méthodes de rappel de l'application modèle.

```
#############################################################################
# Analyse des arguments du programme.
#
function parse_args # [args ...]
{
      typeset opt

      while getopts 'R:G:T:' opt
      do
             case "$opt" in
             R)
                  # Nom de la ressource DNS.
                  RESOURCE_NAME=$OPTARG
                  ;;
             G)
                  # Nom du groupe de ressources dans laquelle est
                  # configurée la ressource.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
             T)
                  # Nom du type de ressources.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
             *)
                  logger -p ${SYSLOG_FACILITY}.err \
                  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                  "ERROR: Option $OPTARG unknown"
                  exit 1
                      ;;
             esac
    done
}
```

Remarque – bien que la méthode SONDE de l'application modèle soit définie par l'utilisateur (et qu'il ne s'agisse pas d'une méthode de rappel de Sun Cluster), elle est appelée avec les mêmes arguments que les méthodes de rappel. Par conséquent, cette méthode contient une fonction d'analyse identique à celle employée par les autres méthodes de rappel.

La fonction d'analyse est appelée dans MAIN sous la forme :

```
parse_args "$@"
```

Génération de messages d'erreur

Il est recommandé que les méthodes de rappel utilisent la fonction syslog pour émettre des messages d'erreur à l'adresse des utilisateurs finaux. Toutes les méthodes de rappel du service de données modèle utilisent la fonction scha_cluster_get() pour récupérer le numéro de la fonction syslog employée pour le journal du cluster, de la manière suivante :

```
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
```

La valeur est enregistrée dans une variable de shell, SYSLOG_FACILITY, et peut être utilisée comme fonction de la commande logger pour consigner des messages dans le journal du cluster. Par exemple, la méthode Démarrage du service de données modèle récupère la fonction syslog et consigne un message indiquant que le service de données a été lancé, de la manière suivante :

```
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
...

if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS successfully started"
fi
```

Reportez-vous à la page de manuel scha_cluster_get(1HA) pour de plus amples informations.

Obtention des informations des propriétés

La plupart des méthodes de rappel doivent obtenir des informations sur les propriétés des ressources et des types de ressource du service de données. L'API fournit la fonction scha_resource_get() à cette fin.

Deux types de propriétés de ressources, les propriétés définies par le système et les propriétés d'extension, sont disponibles. Les propriétés définies par le système sont prédéfinies alors que les propriétés d'extension du fichier RTR doivent être configurées par l'utilisateur.

Lorsque vous utilisez `scha_resource_get()` pour obtenir la valeur d'une propriété définie par le système, vous spécifiez le nom de celle-ci à l'aide du paramètre `-O`. La commande ne renvoie que la *valeur* de la propriété. Par exemple, dans le service de données modèle, la méthode `Démarrage_détecteur` doit localiser le programme de sonde de manière à pouvoir le lancer. Le programme de sonde réside dans le répertoire de base du service de données, vers lequel pointe la propriété `REP_BASE_TR`. Ainsi, la méthode `Démarrage_détecteur` récupère la valeur de `REP_BASE_TR` et la place dans la variable `REP_BASE_TR`, de la manière suivante :

```
RÉP_BASE_TR=‘scha_resource_get -O RÉP_BASE_TR -R $RESOURCE_NAME -G \ $RESOURCEGROUP_NAME‘
```

Pour les propriétés d'extension, vous devez spécifier, à l'aide du paramètre `-O`, qu'il s'agit d'une propriété d'extension et fournir le nom de la propriété comme dernier paramètre. La commande renvoie à la fois le *type* et la *valeur* de ce type de propriété. Par exemple, dans le service de données modèle, le programme de sonde récupère le type et la valeur dans la propriété d'extension `délai_sonde`, puis utilise `awk` pour ne placer la variable que dans la variable de shell `DÉLAI_SONDE`, de la manière suivante.

```
info_délai_sonde=‘scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME
Probe_timeout‘ DÉLAI_SONDE=‘echo $probe_timeout_info | awk ’{print $2}’‘
```

Contrôle du service de données

Un service de données doit fournir une méthode `Démarrage` ou de `Démarrage_avant_réseau` pour activer le démon d'application du cluster, ainsi qu'une méthode `Arrêt` ou d'`Arrêt_après_réseau` pour arrêter le démon d'application du cluster. Le service de données modèle met en oeuvre des méthodes `Démarrage` et d'`Arrêt`. Reportez-vous à la rubrique “Choix des méthodes `Démarrage` et `Arrêt` à utiliser” à la page 46 pour de plus amples informations sur les usages alternatifs de `Démarrage_avant_réseau` et de `Arrêt_après_réseau`.

Méthode `Démarrage`

Le RGM appelle la méthode `Démarrage` sur un noeud de cluster lorsque le groupe de ressources contenant la ressource du service de données est mis en ligne sur ce noeud ou lorsque le groupe de ressources est déjà en ligne et la ressource activée. Dans l'application modèle, la méthode `Démarrage` active le démon `in.named` sur ce noeud.

Cette rubrique décrit les principaux éléments de la méthode `Démarrage` pour l'application modèle. Elle ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction `parse_args()` et l'obtention de la fonction syslog décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour obtenir une liste complète de la méthode `Démarrage`, reportez-vous à la rubrique "Méthode `Démarrage`" à la page 280.

Présentation de la méthode `Démarrage`

Avant de tenter de lancer le DNS, la méthode `Démarrage` du service de données modèle vérifie que le répertoire et le fichier de configuration (`named.conf`) sont accessibles et disponibles. Les informations de `named.conf` sont essentielles à un bon fonctionnement du DNS.

Cette méthode de rappel utilise le gestionnaire des processus (`pmfadm`) pour lancer le démon DNS (`in.named`). Si DNS se bloque ou n'arrive pas à démarrer, le gestionnaire de processus tente de le lancer un nombre prédéfini de fois pendant un intervalle donné. Le nombre de tentatives et l'intervalle sont spécifiés par les propriétés du fichier RTR du service de données.

Vérification de la configuration

Pour fonctionner, le DNS a besoin des informations du fichier `named.conf` situé dans le répertoire de configuration. C'est la raison pour laquelle la méthode `Démarrage` effectue des contrôles d'intégrité afin de s'assurer que le répertoire et le fichier sont accessibles avant de tenter de lancer le DNS.

La propriété d'extension `Rép_conf` fournit le chemin pointant vers le répertoire de configuration. La propriété elle-même est définie dans le fichier RTR. Toutefois, l'administrateur du cluster spécifie l'emplacement exact lors de la configuration du service de données.

Dans le service de données modèle, la méthode `Démarrage` récupère l'emplacement du répertoire de configuration à l'aide de la fonction `scha_resource_get ()`.

Remarque – `rép_conf` étant une propriété d'extension, `scha_resource_get()` renvoie à la fois le type et la valeur. La commande `awk` récupère simplement la valeur et la place dans une variable de shell, `RÉP_CONFIG`.

```
# trouver la valeur de Rép_dir définie par l'administrateur du cluster au moment de
# l'ajout de la ressource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get renvoie le "type" et la "valeur" des
```

```
# propriétés d'extension. Obtention uniquement de la valeur de la propriété d'extension.
RÉP_CONFIG=`echo $config_info | awk '{print $2}'`
```

La méthode `Démarrage` utilise alors la valeur de `RÉP_CONFIG` pour vérifier si le répertoire est accessible. Si ce n'est pas le cas, `Démarrage` consigne un message d'erreur et se ferme en affichant un état d'échec. Reportez-vous à la rubrique "État de `Démarrage` à la fermeture" à la page 106.

```
# Vérifier si $CONFIG_DIR est accessible.
if [ ! -d $CONFIG_DIR ]; then
logger -p ${SYSLOG_FACILITY}.err \
-t [$SYSLOG_TAG] \
"${ARGV0} Directory $CONFIG_DIR is missing or not mounted"
exit 1
fi
```

Avant de démarrer le démon de l'application, cette méthode effectue un dernier contrôle afin de vérifier la présence du fichier `named.conf`. Si ce n'est pas le cas, `Démarrage` consigne un message d'erreur et se ferme en affichant un état d'échec.

```
# Passer au répertoire $CONFIG_DIR s'il y a des
# noms de chemin d'accès relatifs dans les fichiers de données.
cd $CONFIG_DIR
# Vérifier que le fichier named.conf est présent dans le répertoire $CONFIG_DIR
if [ ! -s named.conf ]; then
logger -p ${SYSLOG_FACILITY}.err \
-t [$SYSLOG_TAG] \
"${ARGV0} File $CONFIG_DIR/named.conf is missing or empty"
exit 1
fi
```

Démarrage de l'application

Cette méthode utilise le gestionnaire de processus (`pmfadm`) pour lancer l'application. La commande `pmfadm` vous permet de définir le nombre de fois où l'application est redémarrée pendant un délai donné. Le fichier RTR contient deux propriétés, `Nombre_nouvelles_tentatives`, indiquant le nombre de tentatives de redémarrage d'une application, et `Intervalle_nouvelles_tentatives`, indiquant le délai de ces tentatives.

La méthode `Démarrage` récupère les valeurs de `Nombre_nouvelles_tentatives` et de `Intervalle_nouvelles_tentatives` à l'aide de la fonction `scha_resource_get()` et enregistre leurs valeurs dans des variables de shell. Ensuite, elle transmet ces valeurs à `pmfadm` à l'aide des options `-n` et `-t`.

```
# Obtenir la valeur du nombre de nouvelles tentatives du fichier RTR.
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`
# Obtenir la valeur de l'intervalle entre les tentatives du fichier RTR. Cette valeur est
# exprimée en secondes et doit être convertie en minutes avant d'être transmise à pmfadm.
# Remarquez que la conversion arrondit au chiffre supérieur ; par exemple, 50 secondes sont
```

```
# arrondies à 1 minute.
((INTRVAL_NOUVELLES_TENTATIVES=‘scha_resource_get -O Intervalle_nouvelles_tentatives
-R $RESOURCE_NAME \ -G $RESOURCEGROUP_NAMÈ / 60))

# Démarrer le démon in.named sous le contrôle du gestionnaire des processus. Le laisser se
# bloquer et redémarrer autant de fois qu’indiqué dans $RETRY_COUNT pendant le délai de
# $RETRY_INTERVAL; s’il se bloque plus souvent que cela, le gestionnaire de processus cesse
# d’essayer de le redémarrer.
# Si un processus est déjà enregistré sous la balise
# <$PMF_TAG>, le gestionnaire de processus émet un message d’alerte
# indiquant que le processus est déjà en cours d’exécution.
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Consigner un message indiquant que HA-DNS a été démarré.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

État de Démarrage à la fermeture

Une méthode Démarrage ne doit pas se fermer correctement avant que l’application sous-jacente soit en cours d’exécution et disponible, surtout si d’autres services de données en dépendent. L’une des manières permettant de vérifier la réussite consiste à sonder l’application afin de contrôler qu’elle tourne avant de quitter la méthode Démarrage. Pour une application complexe telle qu’une base de données, dans le fichier RTR, veillez à définir pour la propriété Délai_démarrage une valeur suffisamment élevée pour que l’application s’initialise et exécute une reprise sur incident.

Remarque – étant donné que la ressource de l’application DNS du service de données modèle se lance rapidement, le service de données modèle n’effectue pas d’interrogation pour vérifier son bon fonctionnement avec de se fermer.

Si cette méthode n’arrive pas à démarrer le DNS et se ferme en affichant un état d’échec, le RGM contrôle la propriété Mode_basculement, qui détermine la réaction à adopter. Le service de données modèle ne définit pas explicitement la propriété Mode_basculement. Par conséquent, cette propriété a la valeur par défaut AUCUN (sauf si l’administrateur du cluster a contourné cette valeur et en a spécifié une autre). Dans ce cas, le RGM ne prend pas de mesure autre que la définition de l’état du service de données. Une intervention de l’utilisateur est nécessaire pour le redémarrage sur le même noeud ou le basculement vers un autre noeud.

Méthode Arrêt

La méthode Arrêt est appelée sur un noeud du cluster lorsque le groupe de ressources contenant la ressource HA-DNS passe hors ligne sur ce noeud ou si le groupe de ressources est en ligne et la ressource désactivée. Cette méthode arrête le démon in.named (DNS) sur ce noeud.

Cette rubrique décrit les principaux éléments de la méthode Arrêt pour l'application modèle. Elle ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction parse_args() et l'obtention de la fonction syslog décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour obtenir une liste complète de la méthode Arrêt, reportez-vous à la rubrique "Méthode Arrêt" à la page 283.

Présentation de la méthode Arrêt

Il existe deux points principaux à prendre en compte lors d'une tentative d'arrêt du service de données. D'une part, il convient de le fermer correctement. La meilleure manière pour y arriver consiste à envoyer un signal SIGTERM par l'intermédiaire de pmfadm.

D'autre part, il faut veiller à ce que le service de données soit effectivement arrêté afin d'éviter de le faire passer à l'état Échec_arrêt. La meilleure manière pour y arriver consiste à envoyer un signal SIGKILL par l'intermédiaire de pmfadm.

La méthode Arrêt du service de données modèle tient compte de ces deux considérations. Elle envoie d'abord un signal SIGTERM. Si celui-ci ne peut pas arrêter le service de données, la méthode envoie un signal SIGKILL.

Avant de tenter d'arrêter le DNS, cette méthode Arrêt vérifie si le processus tourne effectivement. Si c'est le cas, Arrêt utilise le détecteur de processus (pmfadm) pour l'arrêter.

L'idempotence de cette méthode Arrêt est garantie. Bien que le RGM ne doive pas appeler une méthode Arrêt deux fois sans d'abord avoir démarré le service de données avec un appel à sa méthode Démarrage, il peut appeler une méthode Arrêt sur une ressource même si celle-ci n'a jamais été démarrée ou si celle-ci est morte. C'est pourquoi cette méthode Arrêt se ferme correctement même si le DNS ne tourne pas.

Arrêt de l'application

La méthode Arrêt fournit une approche à deux niveaux pour arrêter le service de données : une approche sans heurt utilisant un signal SIGTERM par le biais de pmfadm et une approche abrupte utilisant un signal SIGKILL. La méthode Arrêt obtient la

valeur Délai_arrêt (le délai au cours duquel la méthode Arrêt doit avoir un retour). Arrêt alloue alors 80 % de ce temps à un arrêt sans heurt et 15 % à un arrêt abrupt (5 % sont réservés), de la manière indiquée dans l'exemple suivant.

```
DÉLAI_ARRÊT`‘scha_resource_get -O DÉLAI_ARRÊT -R $RESOURCE_NAME

-G $RESOURCEGROUP_NAMÈ
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))
```

La méthode Arrêt utilise pmfadm -q pour vérifier si le démon DNS fonctionne. Si c'est le cas, Arrêt utilise d'abord pmfadm -s pour envoyer un signal TERM afin de mettre un terme au processus DNS. Si ce signal ne peut pas arrêter le processus après l'expiration de 80 % du délai imparti, Arrêt envoie un signal SIGKILL. Si ce signal ne parvient pas non plus à ses fins en 15 % du délai imparti, la méthode consigne un message d'erreur et se ferme en affichant un état d'échec.

Si la commande pmfadm met un terme au processus, la méthode consigne un message indiquant l'arrêt du processus et se ferme correctement.

Si le processus DNS ne tourne pas, la méthode consigne un message l'indiquant et se ferme correctement. L'exemple de code suivant montre comment Arrêt utilise pmfadm pour arrêter le processus DNS.

```
# Regardez si in.named fonctionne et si c'est le cas, arrêtez-le.
if pmfadm -q $PMF_TAG; then
   # Envoyez un signal SIGTERM au service de données et attendez 80 % de la
# valeur totale du délai imparti.
   pmfadm -s $RESOURCE_NAME.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err \
          -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"

      # Le service de données ne s'étant pas arrêté avec un signal SIGTERM, utilisez à présent
      # SIGKILL et attendez 15 % de la valeur totale du délai imparti.
      pmfadm -s $PMF_TAG -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG]
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL"

          exit 1
      fi
fi
else
   # Le service de données ne fonctionne pas actuellement. Consignez un message et
   # un code de sortie avec succès.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS is not started"
```

```
   # Même si HA-DNS ne tourne pas, quittez avec succès pour éviter de faire passer
   # la ressource du service de données à un état ÉCHEC_ARRÊT.

   exit 0

fi

# Arrêt du DNS fructueux. Consignez un message et un code de sortie avec succès.
logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
    "HA-DNS successfully stopped"
exit 0
```

État d'`Arrêt` à la fermeture

Une méthode `Arrêt` ne doit pas se fermer correctement avant que l'application sous-jacente soit réellement arrêtée, surtout si d'autres services de données en dépendent. Dans le cas contraire, vous pourriez rencontrer une corruption de données.

Pour une application complexe telle qu'une base de données, dans le fichier RTR, veillez à définir pour la propriété `Délai_arrêt` une valeur suffisamment élevée pour que l'application se nettoie à l'arrêt.

Si cette méthode n'arrive pas à arrêter le DNS et se ferme en affichant un état d'échec, le RGM contrôle la propriété `Mode_basculement`, qui détermine la réaction à adopter. Le service de données modèle ne définit pas explicitement la propriété `Mode_basculement`. Par conséquent, elle possède la valeur par défaut `AUCUN` (sauf si l'administrateur du cluster a contourné cette valeur et en a spécifié une autre). Dans ce cas, le RGM ne prend pas de mesure autre que la définition de l'état du service de données à `Échec_arrêt`. Une intervention de l'utilisateur est nécessaire pour forcer l'arrêt de l'application et effacer l'état `Échec_arrêt`.

Définition d'un détecteur de pannes

L'application modèle met en oeuvre un détecteur de pannes de base afin de surveiller la fiabilité de la ressource DNS (`in.named`). Le détecteur de pannes se compose de :

- `Sonde_dns`, un programme défini par l'utilisateur employant la commande `nslookup` pour vérifier si la ressource DNS contrôlée par le service de données modèle fonctionne. Si le DNS ne fonctionne pas, cette méthode tente de le redémarrer localement ou, en fonction du nombre de tentatives de redémarrage, demande que le RGM déplace le service de données sur un autre noeud.

- `Démarrage_détecteur_dns`, une méthode de rappel lançant `sonde_dns`. Le RGM appelle automatiquement `Démarrage_détecteur_dns` une fois le service de données modèle mis en ligne si la surveillance est activée.
- `Arrêt_détecteur_dns`, une méthode de rappel arrêtant `sonde_dns`. Le RGM appelle automatiquement `arrêt_détecteur_dns` avant de mettre le service de données modèle hors ligne.
- `Contrôle_détecteur_dns`, une méthode de rappel appelant la méthode `Validation` afin de vérifier si le répertoire de configuration est disponible lorsque le programme de `SONDE` bascule le service de données vers un nouveau noeud.

Programme de sonde

Le programme `sonde_dns` met en oeuvre un processus vérifiant en permanence si la ressource DNS contrôlée par le service de données modèle fonctionne. La commande `sonde_dns` est lancée par la méthode `Démarrage_détecteur_dns`, appelée automatiquement par le RGM une fois le service de données en ligne. Le service de données est arrêté par la méthode `Arrêt_détecteur_dns`, appelée par le RGM avant de mettre le service de données modèle hors ligne.

Cette rubrique décrit les principaux éléments de la méthode `SONDE` pour l'application modèle. Elle ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction `parse_args()` et l'obtention de la fonction syslog décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour obtenir une liste complète de la méthode `SONDE`, reportez-vous à la rubrique "Programme `SONDE`" à la page 286.

Présentation du programme de Sonde

La sonde tourne en boucle infinie. Elle utilise la commande `nslookup` afin de vérifier si la bonne ressource DNS tourne. Si le DNS tourne, la sonde passe en mode de sommeil pour un délai donné (établi par la propriété définie par le système `Intervalle_sonde_complet`), puis procède à un nouveau contrôle. Dans le cas contraire, ce programme tente de le redémarrer localement ou, en fonction du nombre de tentatives de démarrage, demande au RGM de déplacer le service de données sur un autre noeud.

Obtention des valeurs des propriétés

Ce programme a besoin des valeurs des propriétés suivantes :

- `Intervalle_sonde_complet` : pour définir le délai pendant lequel la sonde passe en mode de sommeil.
- `Délai_sonde` : pour appliquer le délai imparti relatif à la sonde à la commande `nslookup` effectuant le sondage.

- Ressources_réseau_utilisées : pour obtenir l'adresse IP sur laquelle tourne le DNS.
- Nombre_nouvelles_tentatives et Intervalle_nouvelles_tentatives : pour déterminer le nombre de tentatives de redémarrage ainsi que la période sur laquelle elles se répartissent.
- Rép_base_TR : pour obtenir le répertoire contenant le programme de SONDE ainsi que l'utilitaire gettime.

La fonction scha_resource_get() obtient les valeurs de ces propriétés et les enregistre dans des variables de shell, de la manière décrite ci-dessous :

```
INTERVALLE_SONDE=‘scha_resource_get -O INTERVALLE_SONDE_COMPLET \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
Info_délai_sonde=‘scha_resource_get -O Extension -R $RESOURCE_NAME
\
-G $RESOURCEGROUP_NAME Délai_sondè
DÉLAI_SONDE=‘echo $probe_timeout_info | awk ’{print $2}’‘
HÔTE_DNS=‘scha_resource_get -O RESSOURCES_RÉSEAU_UTILISÉES -R $RESOURCE_NAME
\
-G $RESOURCEGROUP_NAMÈ
NOMBRE_NOUVELLES_TENTATIVES=‘scha_resource_get -O NOMBRE_NOUVELLES_TENTATIVES
-R $RESOURCE_NAME
-G\
$RESOURCEGROUP_NAMÈ
INTERVALLE_NOUVELLES_TENTATIVES=‘scha_resource_get -O INTERVALLE_NOUVELLES_TENTATIVES
-R $RESOURCE_NAME
-G\
$RESOURCEGROUP_NAMÈ
RÉP_BASE_TR=‘scha_resource_get -O RÉP_BASE_TR -R $RESOURCE_NAME -G\
$RESOURCEGROUP_NAMÈ
```

Remarque – pour les propriétés définies par le système, telles que Intervalle_sonde_complet, scha_resource_get() ne retourne que la valeur. Pour les propriétés d'extension, telles que Délai_sonde, scha_resource_get() retourne le type et la valeur. Utilisez la commande awk pour n'obtenir que la valeur.

Contrôle de la fiabilité du service

La sonde elle-même est une boucle while infinie de commandes nslookup. Avant cette boucle, un fichier temporaire est créé. Son but consiste à collecter les réponses à nslookup. Les variables *probefail* et *retries* sont remises à 0.

```
# Configurer un fichier temporaire pour les réponses de nslookup.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

La boucle while elle-même :

- Définit l'intervalle de sommeil de la sonde.

- Utilise `hatimerun` pour lancer la commande `nslookup` afin que celle-ci transmette la valeur `Délai_sonde` et identifie l'hôte cible.
- Définit la variable *probefail* sur la base de la réussite ou de l'échec du code de retour de `nslookup`.
- Vérifie si la réponse `nslookup` provient du service de données modèle ou d'un autre serveur DNS si *probefail* a la valeur 1 (échec).

Voici le code de la boucle `while`.

```
while :
do
   # L'intervalle auquel la sonde doit s'exécuter est spécifié dans la
   # propriété INTERVALLE_SONDE_COMPLET. Par conséquent, définir le sommeil de la sonde
   # à une durée de INTERVALLE_SONDE_COMPLET.
   sleep $PROBE_INTERVAL

   # Exécuter une commande nslookup de l'adresse IP sur laquelle le DNS fonctionne.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
          > $DNSPROBEFILE 2>&1

      retcode=$?
      if [ $retcode -ne 0 ]; then
            probefail=1
      fi

   # Vérifier que la réponse à nslookup provient du serveur HA-DNS
   # et pas d'un autre nom de serveur mentionné dans le fichier
   # /etc/resolv.conf.
   if [ $probefail -eq 0 ]; then
   # Obtenir le nom du serveur ayant répondu à la requête de nslookup.
   SERVER=` awk ' $1=="Server:" { print $2 }' \
   $DNSPROBEFILE | awk -F. ' { print $1 } ' `
   if [ -z "$SERVER" ]; then
      probefail=1
      else
         if [ $SERVER != $DNS_HOST ]; then
            probefail=1
         fi
   fi
fi
```

Évaluation : redémarrage ou basculement

Si la variable *probefail* est différente de 0 (réussite), cela signifie que la commande `nslookup` a dépassé le délai imparti ou que la réponse provient d'un serveur autre que le DNS du service modèle. Dans un cas comme dans l'autre, le serveur DNS ne fonctionne pas de la manière attendue et le détecteur appelle la fonction `decide_restart_or_failover()` afin de déterminer s'il convient ou non de redémarrer le service de données localement ou de demander que le RGM déplace le service de données sur un autre noeud. Si la variable *probefail* a la valeur 0, alors un message indiquant que la sonde a réussi est généré.

```
if [ $probefail -ne 0 ]; then
      decide_restart_or_failover
else
      logger -p ${SYSLOG_FACILITY}.err\
      -t [$SYSLOG_TAG]\
      "${ARGV0} Probe for resource HA-DNS successful"
fi
```

La fonction `decide_restart_or_failover()` utilise un délai (`Intervalle_nouvelles_tentatives`) et un compteur d'échecs (`Nombre_nouvelles_tentatives`) afin de déterminer s'il convient de redémarrer le DNS localement ou de demander à ce que le RGM déplace le service de données sur un autre noeud. Elle met en oeuvre le code conditionnel suivant (voir l'affichage du code pour `decide_restart_or_failover ()` dans la rubrique "Programme `SONDE`" à la page 286).

- S'il s'agit du premier échec, redémarrez le service de données. Consignez un message d'erreur et augmentez le compteur dans la variable `retries`.
- Si ce n'est pas le premier échec, mais si le délai a été dépassé, redémarrez le service de données. Consignez un message d'erreur et réinitialisez le compteur ainsi que le délai.
- Si le délai n'est pas dépassé et si le compteur des nouvelles tentatives a été dépassé, basculez vers un autre noeud. Si le basculement échoue, consignez une erreur et quittez le programme de sonde avec un état 1 (échec).
- Si ni le délai ni le compteur n'ont été dépassés, redémarrez le service de données. Consignez un message d'erreur et augmentez le compteur dans la variable `retries`.

Si le nombre de redémarrages atteint la limite pendant le délai, la fonction demande au RGM de déplacer le service de données vers un autre noeud. Si le nombre de redémarrages se situe sous la limite ou si l'intervalle a été dépassé, entraînant une réinitialisation du compteur, la fonction tente de redémarrer le DNS sur le même noeud. Remarquez les points suivants concernant cette fonction :

- L'utilitaire `gettime` sert à mesurer le délai entre les redémarrages. Il s'agit d'un programme C résidant dans le répertoire (`Rép_base_TR`).
- Les propriétés de ressource définies par le système `Nombre_nouvelles_tentatives` et `Intervalle_nouvelles_tentatives` déterminent le nombre de tentatives de redémarrage et le délai pendant lequel compter. Par défaut, elles définissent 2 tentatives sur une période de 5 minutes (300 secondes) dans le fichier RTR. Toutefois, l'administrateur du cluster peut modifier ces valeurs.
- La fonction `restart_service()` est appelée afin de tenter de redémarrer le service de données sur le même noeud. Reportez-vous à la rubrique suivante, "Redémarrage du service de données" à la page 114, pour obtenir de plus amples informations sur cette fonction.

- La fonction API `scha_control()`, avec l'option `GIVEOVER`, met le groupe de ressources contenant le service de données modèle hors ligne, puis de nouveau en ligne, sur un autre noeud.

Redémarrage du service de données

La fonction `restart_service()` est appelée par `decide_restart_or_failover()` pour tenter de rédemarrer le service de données sur le même noeud. Cette fonction effectue les opérations suivantes :

- Elle détermine si le service de données est toujours enregistré dans le gestionnaire de processus. Si c'est le cas, la fonction :
  - Obtient le nom de la méthode `Arrêt` ainsi que la valeur `Délai_arrêt` du service de données.
  - Utilise `hatimerun` pour lancer la méthode `Arrêt` pour le service de données, avec transmission de la valeur `Délai_arrêt`.
  - Obtient le nom de la méthode `Démarrage` ainsi que la valeur `Délai_démarrage` pour le service de données (si celui-ci s'arrête correctement).
  - Utilise `hatimerun` pour lancer la méthode `Démarrage` pour le service de données, avec transmission de la valeur `Délai_démarrage`.
- Si le service de données n'est plus enregistré dans le gestionnaire de processus, cela signifie qu'il a dépassé le nombre maximum de nouvelles tentatives autorisées par le gestionnaire et que la fonction `scha_control()` est appelée avec l'option `GIVEOVER` afin de basculer le service de données vers un autre noeud.

```
function restart_service
{

        # Pour redémarrer le service de données, d'abord vérifier que le
        # service de données lui-même est toujours enregistré auprès du
        # gestionnaire de processus.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # La BALISE du service de données étant toujours enregistrée
                # auprès du gestionnaire de processus, arrêter
                # le service de données et le redémarrer.

                # Obtenir le nom de la méthode Arrêt et la valeur de DÉLAI_ARRÊT
                # pour cette ressource.
                DÉLAI_ARRÊT=`scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                MÉTHODE_ARRÊT=`scha_resource_get -O STOP \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
```

```
                logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                        "${ARGV0} Stop method failed."
                return 1
        fi

        # Obtenir le nom de la méthode DÉMARRAGE et la valeur
        # de DÉLAI_DÉMARRAGE pour cette ressource.
        DÉLAI_DÉMARRAGE=`scha_resource_get -O START_TIMEOUT \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
        MÉTHODE_DÉMARRAGE=`scha_resource_get -O START \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
        hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                -T $RESOURCETYPE_NAME

        if [[ $? -ne 0 ]]; then
                logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                        "${ARGV0} Start method failed."
                return 1
        fi


else
        # L'absence de la BALISE du service de données
        # signifie que celui-ci a déjà dépassé le nombre
        # maximum de nouvelles tentatives autorisé par le gestionnaire
        # des processus. Ne pas essayer de le redémarrer
        # mais tenter de le basculer sur un autre noeud du serveur.
        scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                -R $RESOURCE_NAME
fi

return 0
}
```

État de la sonde à la fermeture

Le programme de `SONDE` du service de données modèle se ferme en affichant un état d'échec si les tentatives de redémarrage local ont échoué et si la tentative de basculement vers un autre noeud a également échoué. Il consigne le message, "Failover attempt failed" ("Échec de la tentative de basculement").

Méthode `Démarrage_détecteur`

Le RGM appelle la méthode `Démarrage_détecteur` pour lancer la méthode `Sonde_dns` une fois le service de données modèle en ligne.

Cette rubrique décrit les principaux éléments de la méthode `Démarrage_détecteur` pour l'application modèle. Elle ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction `parse_args ()` et l'obtention de la fonction syslog, décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour un affichage complet de la méthode `Démarrage_détecteur`, reportez-vous à la rubrique "Méthode `Démarrage_détecteur`" à la page 292.

Présentation de la méthode `Démarrage_détecteur`

Cette méthode utilise le gestionnaire de processus (`pmfadm`) pour lancer la sonde.

Démarrage de la sonde

La méthode `Démarrage_détecteur` obtient la valeur de la propriété `Rép_base_TR` pour construire le nom entier du chemin d'accès du programme de `SONDE`. Cette méthode lance la sonde à l'aide de l'option de nouvelles tentatives infinies de `pmfadm` (-n -1, -t -1), ce qui signifie que si le démarrage de la sonde échoue, le gestionnaire de processus tente de la démarrer un nombre infini de fois sur une période infinie.

```
# Trouver où réside le programme de sonde en obtenant la valeur de la propriété
# RT_BASEDIR de la ressource.
RT_BASEDIR=‘scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

# Démarrer la sonde pour le service de données sous le gestionnaire de processus.
# Utiliser l’option permettant un nombre infini de nouvelles tentatives pour démarrer
# la sonde, Transmettre le nom, le type et le groupe de la ressource au programme
# de sonde.
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
   $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME
```

Méthode `Arrêt_détecteur`

Le RGM appelle la méthode `Arrêt_détecteur` afin d'arrêter l'exécution de `sonde_dns` lorsque le service de données modèle est mis hors ligne.

Cette rubrique décrit les principaux éléments de la méthode `Arrêt_détecteur` pour l'application modèle. Elle ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction `parse_args ()` et l'obtention de la fonction syslog, décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour l'affichage complet de la méthode `Arrêt_détecteur`, reportez-vous à la rubrique "Méthode `Arrêt_détecteur`" à la page 294.

Présentation de la méthode Arrêt_détecteur

Cette méthode utilise le gestionnaire de processus (pmfadm) pour vérifier si la sonde tourne et, le cas échéant, pour l'arrêter.

Arrêt du détecteur

La méthode Arrêt_détecteur utilise pmfadm -q pour déterminer si la sonde fonctionne et, le cas échéant, pmfadm -s pour l'arrêter. Si la sonde est déjà arrêtée, la méthode se ferme correctement, ce qui garantit l'idempotence de la méthode.

```
# Voir si le détecteur tourne et, le cas échéant, l'arrêter.
if pmfadm -q $PMF_TAG; then
   pmfadm -s $PMF_TAG KILL
   if [ $? -ne 0 ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Could not stop monitor for resource " \
            $RESOURCE_NAME
           exit 1
   else
         # Arrêt du détecteur fructueux. Consigner un message.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor for resource " $RESOURCE_NAME \
            " successfully stopped"
   fi
fi
exit 0
```

Attention – veillez à utiliser le signal KILL avec pmfadm pour arrêter la sonde et non un signal masquable tel que TERM. Dans le cas contraire, la méthode Arrêt_détecteur peut se bloquer indéfiniment et finit par dépasser le délai imparti. Le motif de ce problème réside dans l'appel par la méthode SONDE de la fonction scha_control() lorsqu'il est nécessaire de redémarrer ou de basculer le service de données. Lorsque scha_control() appelle Arrêt_détecteur dans le cadre du processus de mise du service de données hors ligne, si Arrêt_détecteur utilise un signal masquable, il se bloque en attendant que la fonction scha_control() s'arrête et scha_control() se bloque en attendant qu'Arrêt_détecteur s'arrête.

État d'Arrêt_détecteur à la fermeture

La méthode Arrêt_détecteur consigne un message d'erreur si elle ne peut pas arrêter la méthode SONDE. Le RGM fait passer le service de données modèle à l'état ÉCHEC_CONTRÔLE sur le noeud primaire, ce qui peut créer une panique au niveau du noeud.

Arrêt_détecteur ne doit pas se fermer avant l'arrêt de la sonde.

Méthode Contrôle_détecteur

Le RGM appelle la méthode Contrôle_détecteur lorsque la méthode SONDE tente de basculer le groupe de ressources contenant le service de données vers un autre noeud.

Cette rubrique décrit les principaux éléments de la méthode Contrôle_détecteur pour l'application modèle. Cette rubrique ne décrit pas la fonctionnalité commune à toutes les méthodes de rappel, telles que la fonction parse_args () et l'obtention de la fonction syslog, décrites dans la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 98.

Pour un affichage complet de la méthode Contrôle_détecteur, reportez-vous à la rubrique "Méthode Contrôle_détecteur" à la page 296.

La méthode Contrôle_détecteur doit être mise en oeuvre de manière à ne pas entrer en conflit avec d'autres méthodes exécutées simultanément.

La méthode Contrôle_détecteur appelle la méthode Validation afin qu'elle vérifie que le répertoire de configuration DNS est disponible sur le nouveau noeud. La propriété d'extension Rép_conf pointe vers le répertoire de configuration DNS. C'est pourquoi Contrôle_détecteur obtient le chemin et le nom de la méthode Validation ainsi que la valeur de Rép_conf. Elle transmet cette valeur à Validation, comme le montre l'affichage suivant :

```
# Obtenir le chemin complet de la méthode Validation à partir de la propriété
# RT_BASEDIR du type de ressource.
REP_BASE_TR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAMÈ

# Obtenir le nom de la méthode Validation pour cette ressource.
MÉTHODE_VALIDATION=`scha_resource_get -O VALIDATE \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtenir la valeur de la propriété Rép_conf pour démarrer le
# service de données. Utiliser le nom de la ressource et le groupe entré pour
# obtenir la valeur Rép_conf définie au moment de l'ajout de la ressource.
info_config =`scha_resource_get -O Extension -R $RESOURCE_NAME \
 -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get renvoie le type ainsi que la valeur des
# propriétés d'extension. Utiliser awk pour n'obtenir que
# la valeur de la propriété d'extension.
REP_CONFIG=`echo $config_info | awk '{print $2}'`

# Appeler la méthode Validation de manière à ce que le service de données
# puisse être basculé sur un nouveau noeud.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

Reportez-vous à la rubrique "Méthode Validation" à la page 119 pour voir comment l'application modèle vérifie l'aptitude du noeud à héberger le service de données.

Gestion des mises à jour des propriétés

Le service de données modèle met en oeuvre les méthodes `Validation` et de `Mise_à_jour` afin de gérer la mise à jour des propriétés par l'administrateur du cluster.

Méthode `Validation`

Le RGM appelle la méthode `Validation` lorsqu'un groupe est créé et lorsqu'une opération de l'administrateur met à jour les propriétés de la ressource ou du groupe qui la contient. Le RGM appelle `Validation` avant la création ou la mise à jour, et un code de sortie avec échec issu de la méthode sur un noeud entraîne l'annulation de la création ou de la mise à jour.

Il n'appelle `Validation` que lorsque les propriétés de la ressource ou du groupe sont modifiées par une opération de l'administrateur, et non lorsque le RGM définit des propriétés, ou lorsqu'un détecteur définit les propriétés `Statut` et `msg_statut` de la ressource.

Remarque – la méthode `Contrôle_détecteur` appelle aussi explicitement la méthode `Validation` lorsque la méthode `SONDE` tente de basculer le service de données sur un autre noeud.

Présentation de la méthode `Validation`

Le RGM appelle `Validation` avec des arguments différents de ceux transmis aux autres méthodes, y compris les propriétés et valeurs mises à jour. C'est la raison pour laquelle cette méthode du service de données modèle doit mettre en oeuvre une fonction `parse_args()` pour gérer les autres arguments.

La méthode `Validation` vérifie une seule propriété, la propriété d'extension `Rép_conf`. Cette propriété pointe vers le répertoire de configuration DNS, essentiel pour un bon fonctionnement du DNS.

Remarque – le répertoire de configuration ne pouvant pas être modifié lorsque le DNS tourne, la propriété `Rép_conf` est déclarée `RÉGLABLE = À_LA_CRÉATION` dans le fichier RTR. C'est pourquoi la méthode `Validation` n'est jamais appelée pour vérifier la propriété `Rép_conf` à la suite d'une mise à jour, mais uniquement à la création de la ressource du service de données.

Si Rép_conf est l'une des propriétés transmises par le RGM à Validation, la fonction parse_args() récupère et enregistre sa valeur. Validation vérifie alors si le répertoire vers lequel pointe la nouvelle valeur de Confdir est accessible et si le fichier named.conf existe dans ce répertoire et contient certaines données.

Si la fonction parse_args() ne peut pas récupérer la valeur de Rép_conf à partir des arguments de ligne de commande transmis par le RGM, Validation tente néanmoins de valider la propriété Rép_conf. Validation utilise scha_resource_get () pour obtenir la valeur de Confdir à partir de la configuration statique. Ensuite, elle effectue les mêmes contrôles afin de s'assurer que le répertoire de configuration est accessible et contient un fichier named.conf qui ne soit pas vierge.

Si Validation se ferme en affichant un état d'échec, la mise à jour ou la création de toutes les propriétés, et pas seulement de Rép_conf, échoue.

Fonction d'analyse de la méthode Validation

Le RGM transmet à la méthode Validation un ensemble de paramètres différent de celui des autres méthodes de rappel, de telle manière que Validation a besoin d'une autre fonction d'analyse des arguments que les autres méthodes. Reportez-vous à la page de manuel rt_callbacks(1HA) pour de plus amples informations sur les paramètres transmis à Validation et aux autres méthodes de rappel. L'exemple suivant montre la fonction Validation parse_args().

```
#############################################################################
#Analyser les arguments Validation.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Nom de la ressource DNS
NOM_RESSOURCE=$OPTARG
                  ;;
         G)
                  # Nom du groupe dans lequel est configurée
                  # la ressource.
NOM_GROUPE_RESSOURCE=$OPTARG
                  ;;
         T)
                  # Nom du type de ressource.
NOM_TYPE_RESSOURCE=$OPTARG
                  ;;
         r)
                  # La méthode n'accède pas à des propriétés définies
                  # par le système. Aucune action.
```

```
                ;;
        g)
                # La méthode n'accède pas à des propriétés du groupe de
                # ressources. Aucune action.
                ;;
        c)
                # Indique que la méthode Validation est appelée lors
                # de la création de la ressource. Indicateur sans action.
                ;;
        u)
                # Indique la mise à jour d'une propriété quand la
                # ressource existe déjà. Si la mise à jour est adressée
                # à la propriété Rép_conf, alors Rép_conf doit apparaître dans
                # les arguments de ligne de commande. Dans le cas contraire,
                # la méthode doit le rechercher à l'aide de scha_resource_get.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Liste des propriétés d'extension. Séparer les couples propriété et
                # valeur en utilisant "=" comme séparateur.
                PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                VAL=`echo $OPTARG | awk -F= '{print $2}'`
                # Si la propriété d'extension se trouve sur la ligne de
                # commande, noter sa valeur.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
    done
}
```

Comme la fonction `parse_args()` des autres méthodes, cette fonction propose un indicateur (`R`) pour capturer le nom de la ressource, (`G`) pour capturer le nom du groupe de ressources et (`T`) pour capturer le type de ressource transmis par le RGM.

Les indicateurs `r` (propriété définie par le système), `g` (propriété du groupe de ressources) et `c` (validation à la création de la ressource) sont ignorés parce que cette méthode est appelée pour valider une propriété d'extension lors de la mise à jour de la ressource.

L'indicateur `u` définit la variable de shell `UPDATE_PROPERTY` à `1` (`TRUE`). Le témoin `x` capture les noms et les valeurs des propriétés mises à jour. Si `Rép_conf` est l'une des propriétés mises à jour, sa valeur est placée dans la variable de shell *CONFDIR* et la variable *CONFDIR_FOUND* est réglée sur `1` (`TRUE`).

Validation de Rép_conf

Dans sa fonction MAIN, Validation attribue d'abord une chaîne vide à la variable *CONFDIR* et une valeur 0 à *UPDATE_PROPERTY* et *CONFDIR_FOUND*.

```
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0
```

Validation appelle ensuite parse_args() pour analyser les arguments transmis par le RGM.

```
parse_args "$@"
```

Validation vérifie alors si Validation est appelée à la suite d'une mise à jour des propriétés et si la propriété d'extension Rép_conf se trouvait sur la ligne de commande. Validation vérifie alors si la propriété Rép_conf a une valeur et, si ce n'est pas le cas, elle se ferme en affichant un état d'échec et émet un message d'erreur.

```
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
         config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
            -G $RESOURCEGROUP_NAME Confdir`
         CONFDIR=`echo $config_info | awk '{print $2}'`
fi

# Vérifier que la propriété Rép_conf possède une valeur. Dans le cas contraire, il y a échec
# et sortie avec l'état 1
if [[ -z $CONFDIR ]]; then
         logger -p ${SYSLOG_FACILITY}.err \
            "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
         exit 1
fi
```

Remarque – le code précédent vérifie tout spécialement si Validation est appelée à la suite d'une mise à jour ($UPDATE_PROPERTY == 1) et si la propriété n'a *pas* été trouvée dans la ligne de commande (CONFDIR_FOUND == 0), auquel cas il récupère la valeur existante de Rép_conf à l'aide de scha_resource_get (). Si Rép_conf a été trouvé dans la ligne de commande (CONFDIR_FOUND == 1), la valeur de *RÉP_CONF* provient de la fonction parse_args(), et non de scha_resource_get ().

La méthode Validation utilise alors la valeur de *RÉP_CONF* pour vérifier si le répertoire est accessible. Si ce n'est pas le cas, Validation consigne un message d'erreur et se ferme en affichant un état d'échec.

```
# Vérifier si $CONFDIR est accessible.
if [ ! -d $CONFDIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Directory $CONFDIR missing or not mounted"
```

```
   exit 1
fi
```

Avant de valider la mise à jour de la propriété `Rép_conf`, `Validation` effectue un contrôle final pour vérifier si le fichier `named.conf` est présent. Si ce n'est pas le cas, la méthode enregistre un message d'erreur et se ferme en affichant un état d'échec.

```
# Vérifier si le fichier named.conf est présent dans le répertoire Rép_conf
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
         exit 1
fi
```

Si le dernier contrôle réussit, `Validation` consigne un message indiquant une réussite et se ferme avec un état de réussite.

```
# Consigner un message indiquant que la méthode Validation a réussi.
logger -p ${SYSLOG_FACILITY}.err \
   -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

État de `Validation` à la fermeture

Si `Validation` se ferme avec succès (0) `Rép_conf` est créé avec la nouvelle valeur. Si `Validation` se ferme avec un échec (1), `Rép_conf` et les autres propriétés ne sont pas créées et un message indiquant la raison de cet échec est envoyé à l'administrateur du cluster.

Méthode `Mise_à_jour`

Le RGM appelle la méthode `Mise_à_jour` pour notifier à une ressource en cours d'exécution que ses propriétés ont changé. Il appelle `Mise_à_jour` après l'exécution réussie d'une action administrative de paramétrage des propriétés d'une ressource ou de son groupe. Cette méthode est appelée sur les noeuds lorsque la ressource est en ligne.

Présentation de la méthode `Mise_à_jour`

La méthode `Mise_à_jour` ne met pas les propriétés à jour, cette opération est effectuée par le RGM. Par contre, elle notifie la mise à jour aux processus. Le seul processus du service de données modèle affecté par une mise à jour des propriétés est le détecteur de pannes. C'est donc ce processus qu'arrête et que démarre la méthode `Mise_à_jour`.

La méthode `Mise_à_jour` doit vérifier que le détecteur de pannes tourne, puis le tuer à l'aide de la commande `pmfadm`. La méthode obtient l'emplacement du programme de sonde mettant en oeuvre le détecteur de pannes, puis le redémarre à l'aide de `pmfadm`.

Arrêt du détecteur à l'aide de `Mise_à_jour`

La méthode `Mise_à_jour` utilise `pmfadm -q` pour vérifier que le détecteur tourne et, si c'est le cas, le tue à l'aide de la commande `pmfadm -s TERM`. Si le détecteur se ferme correctement, un message le signalant est envoyé à l'administrateur. Dans le cas contraire, `Mise_à_jour` se ferme en affichant un état d'échec et envoie un message d'erreur à l'administrateur.

```
if pmfadm -q $RESOURCE_NAME.monitor; then

# Arrêter le détecteur en cours d'exécution
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
                 "${ARGV0} Could not stop the monitor"
       exit 1
    else
    # Arrêt du DNS fructueux. Consigner un message.
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                 "Monitor for HA-DNS successfully stopped"
    fi
```

Redémarrage du détecteur

Pour redémarrer le détecteur, la méthode `Mise_à_jour` doit localiser le script mettant en oeuvre le programme de sonde. Le programme de sonde réside dans le répertoire de base du service de données, vers lequel pointe la propriété `Rép_base_TR`. `Mise_à_jour` récupère la valeur de `Rép_base_TR` et l'enregistre dans la variable *RÉP_BASE_TR* , de la manière suivante :

```
RÉP_BASE_TR=`scha_resource_get -O RÉP_BASE_TR -R $RESOURCE_NAME -G \ $RESOURCEGROUP_NAME`
```

`Mise_à_jour` utilise alors la valeur de *RÉP_BASE_TR* avec la commande `pmfadm` pour redémarrer le programme de sonde_dns. S'il y réussit, `Mise_à_jour` se ferme avec succès et envoie un message le signalant à l'administrateur. Si `pmfadm` ne peut pas lancer le programme de sonde, `Mise_à_jour` se ferme en affichant un état d'échec et consigne un message d'erreur.

État de `Mise_à_jour` à la fermeture

L'échec de la méthode `Mise_à_jour` entraîne le passage de son état à "update failed" ("échec de la mise à jour"). Cet état n'affecte en rien la gestion RGM de la ressource, mais indique l'échec de l'opération de mise à jour aux outils d'administration par le biais de la fonction syslog.

CHAPITRE 6

Bibliothèque de développement de services de données (BDSD)

Ce chapitre présente brièvement les interfaces de programmation d'application constituant la bibliothèque de développement de services de données ou BDSD. La BDSD est mise en oeuvre dans la bibliothèque `libdsdev.so` et incluse dans le package Sun Cluster.

Ce chapitre contient les rubriques suivantes :

Présentation générale de la BDSD

L'interface API de la BDSD est mise en couche au-dessus de l'interface API GR. À ce titre, elle ne remplace pas l'interface API GR mais l'encapsule et étend ses fonctionnalités. La BDSD simplifie le développement de services de données en offrant des solutions prédéfinies en fonction de problèmes d'intégration spécifiques dans Sun Cluster. Par conséquent, vous pouvez consacrer aux problèmes de haute disponibilité et d'évolutivité intrinsèques de votre application la plus grande partie du temps dédié au développement, sans perdre de temps à intégrer dans Sun Cluster les procédures d'arrêt, de démarrage et de surveillance de l'application.

Gestion des propriétés de configuration

Toutes les méthodes de rappel nécessitent d'accéder aux propriétés de configuration. La BDSD prend en charge l'accès aux propriétés en :

- Analysant l'environnement.
- Offrant un ensemble de fonctions de convenance permettant de rechercher et d'extraire les valeurs de propriétés.

La fonction `scds_initialize` à exécuter au début de chaque méthode de rappel :

- Contrôle et traite les arguments de la ligne de commande (`argc` et `argv[]`) que le gestionnaire RGM transfère à la méthode de rappel. Par conséquent, vous n'avez pas à écrire une fonction d'analyse sur la ligne de commande.
- Configure les structures de données internes que les autres fonctions BDSD vont utiliser. Par exemple, les fonctions de convenance permettant de rechercher et d'extraire les valeurs de propriétés du gestionnaire RGM enregistrent ces valeurs dans ces structures. De même, les valeurs de la ligne de commande, qui sont prioritaires sur les valeurs issues du gestionnaire RGM, sont enregistrées dans ces structures de données.

Remarque – pour la méthode `Validation`, scds_initialize analyse les valeurs de propriétés qui sont transférées à la ligne de commande, supprimant ainsi la nécessité d'écrire une fonction d'analyse pour cette méthode.

La fonction `scds_initialize` initialise également l'environnement de connexion et valide les paramètres de contrôle du système de détection des pannes.

La BDSD offre un ensemble de fonctions permettant de rechercher et d'extraire les propriétés de ressources, de type de ressources et de groupe de ressources, ainsi que les propriétés d'extension fréquemment utilisées. Ces fonctions standardisent l'accès aux propriétés à l'aide des conventions suivantes :

- Chaque fonction exécute uniquement un argument de gestion (retourné par `scds_initialize`).
- Chaque fonction correspond à une propriété particulière. Le type de valeur retournée de la fonction correspond au type de la valeur de propriété récupéré.
- Les fonctions ne retournent pas d'erreurs, les valeurs ayant été précalculées par `scds_initialize`. Les fonctions récupèrent les valeurs dans le gestionnaire RGM à moins qu'une nouvelle valeur soit passée sur la ligne de commande.

Démarrage et arrêt d'un service de données

Une méthode `Démarrage` doit réaliser les actions requises pour démarrer un service de données sur un noeud du cluster. En règle générale, ces actions comprennent la récupération des propriétés de ressources, la localisation des fichiers exécutables et de configuration propres à l'application et le lancement de l'application à l'aide des arguments de ligne de commande appropriés.

La fonction `scds_initialize` recherche et extrait la configuration des ressources. La méthode `Démarrage` peut utiliser les fonctions de convenance des propriétés pour récupérer les valeurs de propriétés spécifiques, comme `Liste_rép_conf`, servant à identifier les répertoires et les fichiers de configuration de l'application à lancer.

Une méthode `Démarrage` peut exécuter `scds_pmf_start` pour lancer une application sous le contrôle de la fonction PMF (Process Monitor Facility). Cette fonction vous permet de spécifier le niveau de surveillance à appliquer au processus et de redémarrer le processus en cas d'échec. Reportez-vous à la rubrique "Méthode `Démarrage_xfnts`" à la page 148 ; vous y découvrirez un exemple de méthode `Démarrage` mise en oeuvre avec la BDSD.

Une méthode `Arrêt` doit être idempotente, ce qui permet de la fermer avec succès même si elle est appelée sur un noeud lorsque l'application ne fonctionne pas. Si la méthode `Arrêt` échoue, la ressource arrêtée bascule en mode `ÉCHEC_ARRÊT`, ce qui engendre une réinitialisation forcée du cluster.

Pour éviter le basculement de la ressource en mode `ÉCHEC_ARRÊT`, la méthode `Arrêt` doit tout mettre en oeuvre pour arrêter la ressource. La fonction `scds_pmf_stop` se caractérise par une tentative d'arrêt de la ressource par phase. Elle tente tout d'abord de l'arrêter à l'aide du signal `SIGTERM`, puis, en cas d'échec, elle utilise un signal `SIGKILL`. Reportez-vous à `scds_pmf_stop`(3HA) pour de plus amples informations.

Mise en oeuvre d'un système de détection des pannes

La BDSD simplifie considérablement la mise en oeuvre d'un système de détection des pannes en fournissant un modèle prédéfini. Une méthode `Démarrage_détecteur` lance le système de détection des pannes, sous le contrôle de la fonction PMF, lors du

démarrage de la ressource sur un noeud. Ce système tourne en boucle tant que la ressource est exécutée sur le noeud. La logique de haut niveau d'un système de détection des pannes de la BDSD se présente comme suit :

- La fonction `scds_fm_sleep` utilise la propriété `Intervalle_sonde_complet` pour déterminer l'intervalle de temps entre les détections. Tout échec du processus d'application déterminé par la fonction PMF lors de cet intervalle engendre le redémarrage de la ressource.
- La détection elle-même renvoie une valeur indiquant la gravité des échecs : de `0` signifiant l'absence d'échec à `100` pour un échec total.
- La valeur de retour de la détection est transmise à la fonction `scds_action` qui conserve un historique des échecs cumulatifs dans l'intervalle de la propriété `Intervalles_nouvelles_tentatives`.
- La fonction `scds_action` détermine ce que vous devez faire en cas d'échec.
  - Si la gravité de l'échec cumulatif est inférieur à `100`, ne faites rien.
  - Si la gravité de l'échec cumulatif atteint `100` (échec total), redémarrez le service de données. Si `Intervalle_nouvelles_tentatives` est dépassé, réinitialisez l'historique.
  - Si le nombre de redémarrage dépasse la valeur de la propriété `Nombre_nouvelles_tentatives`, dans le délai spécifié par `Intervalle_nouvelles_tentatives`, basculez le service de données.

Accès aux données d'adresse réseau

La BDSD offre des fonctions de convenance pour renvoyer les données d'adresse réseau des ressources ou des groupes de ressources. Par exemple, `scds_get_netaddr_list` recherche et extrait les ressources d'adresse réseau utilisées par une ressource en permettant à un système de détection des pannes d'analyser l'application.

La BDSD comprend également un ensemble de fonctions pour la surveillance TCP. En règle générale, ces fonctions établissent une connexion de prise simple à un service, lisent et écrivent des données sur un service puis assurent la déconnexion du service. Le résultat de la détection peut être transmis à la fonction BDSD `scds_fm_action` pour déterminer l'action à mettre en oeuvre.

Reportez-vous à la rubrique "Méthode `Validation_xfnts`" à la page 163 pour obtenir un exemple de détection TCP des pannes.

Débogage de la mise en oeuvre d'un type de ressources

La BDSD intègre des fonctions facilitant le débogage de votre service de données.

L'utilitaire BDSD `scds_syslog_debug()` propose une structure de base pour ajouter des instructions de débogage à la mise en oeuvre d'un type de ressources. Le *niveau* de débogage (compris entre 1 et 9) peut être défini de façon dynamique par la mise en oeuvre de types de ressources par noeud du cluster. Toutes les méthodes de rappel du type de ressources lisent le fichier `/var/cluster/rgm/rt/`*nomtr*`/loglevel` (qui contient uniquement un nombre entier compris entre 1 et 9). La routine BDSD `scds_initialize()` lit ce fichier et définit le niveau de débogage en interne sur le niveau spécifié. Le niveau de débogage par défaut 0 indique que le journal ne contient aucun message de débogage.

La fonction `scds_syslog_debug()` utilise l'option retournée par la fonction `scha_cluster_getlogfacility()` comme une priorité de `LOG_DEBUG`. Vous pouvez configurer ces messages de débogage dans `/etc/syslog.conf`.

Vous pouvez transformer certains messages de débogage en message d'informations d'une opération standard du type de ressources (notamment au niveau de la priorité `LOG_INFO`) à l'aide de l'utilitaire `scds_syslog`. Si vous consultez l'application BDSD modèle dans le Chapitre 8, vous constaterez que les fonctions `scds_syslog_debug` et `scds_syslog` sont utilisées librement.

Activation des systèmes de fichiers locaux à haut niveau de disponibilité

Vous pouvez utiliser le type de ressources `HAStoragePlus` pour rendre un système de fichiers local hautement disponible dans un environnement Sun Cluster. Les partitions du système de fichiers local doivent se trouver sur les groupes de disques globaux. Vous devez activer les commutations d'affinité et configurer l'environnement Sun Cluster pour prendre en charge le basculement. Ceci permet à l'utilisateur de rendre n'importe quel système de fichiers disponible à n'importe quel hôte connecté directement à ces disques multi-hôtes. Nous vous recommandons fortement d'utiliser un système de fichiers local hautement disponible pour certains services de données à fort pourcentage d'E/S. Vous trouverez de plus amples informations sur la configuration du type de ressources `HAStoragePlus` dans la rubrique "Enabling Highly Available Local File Systems" in *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

CHAPITRE 7

Conception des types de ressources

Ce chapitre explique l'usage habituel de la BDSD dans la conception et la mise en oeuvre des types de ressources. Il se concentre également sur la conception du type de ressource permettant de valider la configuration de la ressource ainsi que de démarrer, arrêter et surveiller la ressource. Ce chapitre décrit enfin la manière d'utiliser la BDSD pour mettre en oeuvre les méthodes de rappel associées au type de ressource.

Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour obtenir de plus amples informations.

Pour accomplir ces tâches, vous devez accéder aux paramètres des propriétés de ressources. L'utilitaire BDSD `scds_initialize()` vous propose une méthode uniforme pour accéder aux propriétés de ressources. Il est conçu de manière à être appelé au début de chaque méthode de rappel. Cette fonction utilitaire récupère toutes les propriétés d'une ressource depuis la structure du cluster et les met à la disposition de la famille de fonctions `scds_get`*nom*`()`.

Ce chapitre contient les rubriques suivantes :

Fichier RTR

Le fichier RTR (Resource Type Registration) est un élément important du type de ressource. Ce fichier spécifie les détails du type de ressource dans Sun Cluster. Ces détails comprennent des informations telles que les propriétés requises par la mise en oeuvre, les types de données de ces propriétés, leurs valeurs par défaut, le chemin du système de fichiers pour les méthodes de rappel associées à la mise en oeuvre du type de ressource et différents paramètres pour les propriétés définies par le système.

Le fichier RTR modèle fourni avec la BDSD doit être suffisant pour la plupart des mises en oeuvre du type de ressource. Il vous suffit d'éditer certains éléments de base tels que le nom du type de ressource et le nom du chemin d'accès des méthodes de rappel associées au type de ressource. Si une nouvelle propriété est nécessaire pour mettre le type de ressource en oeuvre, vous pouvez la déclarer comme propriété d'extension dans le fichier RTR de la mise en oeuvre du type de ressource, puis y accéder à l'aide de l'utilitaire `scds_get_ext_property()` de la BDSD.

Méthode `Validation`

Il existe deux scénarios pour l'appel par le RGM de la méthode `Validation` dans le cadre de la mise en oeuvre d'un type de ressource : 1) à la création d'une nouvelle ressource d'un type donné et 2) à la mise à jour d'une propriété de la ressource ou du groupe de ressources. Ces deux scénarios se distinguent par la présence de l'option de ligne de commande `-c` (création) ou `-u` (mise à jour) transmise à la méthode `Validation` de la ressource.

La méthode `Validation` est appelée sur chaque membre d'un jeu de noeuds défini par la valeur de la propriété `NOEUDS_INIT` du type de ressource. Si `NOEUDS_INIT` a la valeur `ÉLÉMENTS_PRINCIPAUX_GR`, `Validation` est appelée sur chaque noeud pouvant héberger (en tant que primaire) le groupe contenant la ressource. Si `NOEUDS_INIT` a la valeur `NOEUDS_INSTALLÉS_TR`, `Validation` est appelée sur chaque noeud où est installé le logiciel du type de ressource, généralement tous les noeuds du cluster. La valeur par défaut de `NOEUDS_INIT` est `ÉLÉMENTS_PRINCIPAUX_GR` (reportez-vous à `rt_reg`(4)). Au moment où la méthode `Validation` est appelée, le RGM n'a pas encore créé la ressource (dans le cas d'un rappel de création) ou n'a pas encore appliqué les valeurs mises à jour pour la propriété actualisée (dans le cas d'un rappel de mise à jour). L'objectif de la méthode de rappel `Validation` pour la mise en oeuvre d'un type de ressource consiste à vérifier que les paramètres de la ressource (tels que spécifiés par les valeurs proposées pour les propriétés de la ressource) sont acceptables pour le type de ressource

Remarque – si vous utilisez des systèmes de fichiers locaux gérés par `HAStoragePlus`, vous employez la commande `scds_hasp_check` pour vérifier l'état de la ressource `HAStoragePlus`. Ces informations sont obtenues à partir de l'état (en ligne ou autre) de toutes les ressources `SUNW.HAStoragePlus`(5) dont dépend la ressource à l'aide des propriétés système `Dépendances_ressource` ou `Dépendances_ressource_faibles` définies pour la ressource. Reportez-vous à `scds_hasp_check`(3HA) pour obtenir une liste complète des codes d'état renvoyés par l'appel `scds_hasp_check`.

La fonction BDSD `scds_initialize()` gère ces situations de la manière suivante :

- Dans le cas de la création d'une ressource, elle analyse les propriétés proposées pour la ressource telles qu'elles sont transmises sur la ligne de commande. Les valeurs proposées pour les propriétés de la ressource sont donc disponibles pour le développeur du type de ressource comme si la ressource avait déjà été créée dans le système.
- Dans le cas de la mise à jour d'une ressource ou d'un groupe de ressources, les valeurs proposées pour les propriétés mises à jour par l'administrateur sont lues depuis la ligne de commande, et les autres propriétés (dont les valeurs ne sont pas modifiées) sont lues depuis Sun Cluster à l'aide de l'API de gestion des ressources. Un développeur du type de ressource utilisant BDSD ne doit pas se préoccuper de ces tâches administratives. La validation d'une ressource peut s'effectuer comme si toutes les propriétés de la ressource étaient à la disposition du développeur.

Supposons que la fonction mettant en oeuvre la validation des propriétés d'une ressource s'appelle `svc_validate()` et utilise la famille de fonctions `scds_get_`*nom*`()` pour examiner la propriété qu'elle doit valider. En admettant qu'un paramètre de ressource acceptable soit représenté par un code de retour 0 émis par cette fonction, la méthode de `Validation` du type de ressource peut donc être représentée par le fragment de code suivant :

```
int
main(int argc, char *argv[])
{
   scds_handle_t handle;
   int rc;

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Erreur d'initialisation */
   }
   rc = svc_validate(handle);
   scds_close(&handle);
   return (rc);
}
```

La fonction de validation doit également consigner le motif de l'échec de la validation de la ressource. Pour aller à l'essentiel (vous trouverez au chapitre suivant des informations plus réalistes sur la fonction de validation) un exemple simple de la fonction `svc_validate()` peut être mis en oeuvre de la manière suivante :

```
int
svc_validate(scds_handle_t handle)
{
   scha_str_array_t *confdirs;
   struct stat    statbuf;
   confdirs = scds_get_confdir_list(handle);
   if (stat(confdirs->str_array[0], &statbuf) == -1) {
   return (1);   /* Paramètre de propriété de ressource invalide */
   }
   return (0);   /* Paramètre acceptable */
}
```

Le développeur du type de ressource ne doit donc se préoccuper que de la mise en oeuvre de la fonction `svc_validate()`. Un exemple typique de la mise en oeuvre d'un type de ressource viserait à s'assurer qu'un fichier de configuration d'application appelé app.conf existe sous la propriété `Liste_rép_conf`. Cette vérification peut être facilement mise en oeuvre par un appel système `stat()` effectué sur le nom de chemin d'accès approprié, dérivé de la propriété `Liste_rép_conf`.

Méthode `Démarrage`

La méthode de rappel `Démarrage` associée à la mise en oeuvre d'un type de ressource est appelée par le RGM sur un cluster donné afin de démarrer la ressource. Les noms du groupe de ressources, de la ressource et du type de ressource sont transmis à la ligne de commande. La méthode `Démarrage` doit exécuter les opérations nécessaires pour démarrer une ressource de service de données sur le noeud du cluster. Généralement, ceci implique la récupération des propriétés de la ressource, la localisation des exécutables et/ou des fichiers de configuration spécifiques à l'application ainsi que le lancement des arguments de ligne de commande appropriés.

Avec la BDSD, la configuration de la ressource est déjà récupérée par l'utilitaire `scds_initialize()`. L'opération de démarrage de l'application peut être contenue dans une fonction `svc_start()`. Une autre fonction, `svc_wait()`, peut être appelée pour vérifier que l'application démarre bien. Le code simplifié de la méthode `Démarrage` devient :

```
int
main(int argc, char *argv[])
{
   scds_handle_t handle;

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
```

```
    return (1);   /* Erreur d'initialisation */
    }
    if (svc_validate(handle) != 0) {
    return (1);   /* Paramètres invalides */
    }
    if (svc_start(handle) != 0) {
    return (1);   /* Échec du démarrage */
    }
    return (svc_wait(handle));
}
```

Cette mise en oeuvre de la méthode de démarrage appelle `svc_validate()` pour valider la configuration de la ressource. En cas d'échec, soit les configurations de la ressource et de l'application sont conflictuelles soit ce noeud du cluster rencontre actuellement un problème lié au système. Par exemple, un système de fichiers global requis par la ressource peut ne pas être disponible sur ce noeud du cluster à ce moment. Dans ce cas, il ne rime à rien de tenter de démarrer la ressource sur ce noeud. Il est préférable de laisser le RGM tenter de démarrer la ressource sur un autre noeud. Toutefois, remarquez que les informations indiquées ci-dessus supposent que `svc_validate ()` est suffisamment conservatrice (en d'autres termes, qu'elle ne vérifie que les ressources du noeud du cluster absolument nécessaires pour l'application) ou que le démarrage de la ressource peut échouer sur tous les autres noeuds du cluster et ainsi se retrouver avec l'état `ECHEC_DEMARRAGE`. Reportez-vous à `scswitch`(1M) et au document *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour de plus amples explications sur cet état.

La fonction `svc_start()` doit renvoyer la valeur 0 pour un démarrage réussi de la ressource sur le noeud. Si la fonction de démarrage rencontre un problème, elle doit retourner une valeur différente de 0. Si cette fonction échoue, le RGM tente de démarrer la ressource sur un autre noeud du cluster.

Pour tirer avantage de la BDSD autant que possible, la fonction `svc_start()` peut recourir à l'utilitaire `scds_pmf_start()` pour démarrer l'application sous le gestionnaire de processus. Cet utilitaire permet également de tirer profit de l'action de rappel d'échec du gestionnaire de processus (reportez-vous à l'indicateur d'action `-a` dans `pmfadm`(1M)) pour mettre en oeuvre la détection des échecs.

Méthode `Arrêt`

La méthode de rappel d'`Arrêt` de la mise en oeuvre d'un type de ressource est appelée par le RGM sur le noeud du cluster pour arrêter l'application. La sémantique de rappel de la méthode `Arrêt` exige que les conditions suivantes soient remplies :

- La méthode `Arrêt` doit être *idempotente* parce qu'elle peut être appelée par le RGM même si la méthode `Démarrage` n'a pas fonctionné avec succès sur le noeud. Par conséquent, la méthode `Arrêt` doit réussir (valeur 0) même si l'application ne

tourne pas sur le noeud du cluster et n'a aucune tâche à accomplir.

- Si la méthode Arrêt du type de ressource échoue (valeur différente de 0) sur un noeud du cluster, la ressource arrêtée se retrouve à l'état ÉCHEC_ARRÊT. En fonction du paramètre Mode_basculement de la ressource, il est possible que cela entraîne un redémarrage abrupt du noeud du cluster par le RGM. Il est donc important de concevoir la méthode Arrêt de manière à ce qu'elle essaie vraiment d'arrêter l'application, même en la tuant de manière abrupte (par exemple, à l'aide de SIGKILL) si toutes les autres tentatives se soldent par un échec. Il convient également de s'assurer qu'elle le fasse en temps utile, étant donné que la structure traite l'expiration du Délai_arrêt comme un échec de l'arrêt et fait donc passer la ressource à l'état ÉCHEC_ARRÊT.

L'utilitaire BDSD scds_pmf_stop() doit suffire pour la plupart des applications étant donné qu'il essaie d'abord d'arrêter l'application à chaud (via SIGTERM) (il suppose qu'elle a été démarrée par le gestionnaire de processus à l'aide de scds_pmf_start()) avant d'émettre un SIGKILL à l'adresse du processus. Reportez-vous à la rubrique "Fonctions PMF" à la page 214 pour obtenir de plus amples informations sur cet utilitaire.

Sur la base du modèle de code utilisé jusqu'à présent, et en supposant que la fonction spécifique visant à arrêter l'application s'appelle svc_stop() (le fait que la mise en oeuvre de svc_stop() utilise ou non scds_pmf_stop() n'a aucune importance ici et dépend de la méthode de démarrage de l'application : par le gestionnaire de processus à l'aide de la méthode Démarrage ou par un autre biais), la méthode Arrêt peut être mise en oeuvre de la façon suivante :

```
if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR)
{
   return (1);   /* Erreur d'initialisation */
}
return (svc_stop(handle));
```

La méthode svc_validate() n'est pas utilisée dans la mise en oeuvre de la méthode Arrêt, parce que, même si le système rencontre actuellement un problème, la méthode Arrêt doit tenter d'arrêter l'application sur ce noeud.

Méthode Démarrage_détecteur

Le RGM appelle la méthode Démarrage_détecteur pour démarrer un détecteur de pannes pour la ressource. Les détecteurs de pannes surveillent la santé de l'application gérée par la ressource. Les mises en oeuvre du type de ressource mettent généralement en oeuvre un détecteur de pannes sur un démon séparé tournant en arrière-plan. La méthode de rappel Démarrage_détecteur est utilisée pour lancer ce démon à l'aide des arguments appropriés.

Le démon du détecteur étant lui-même sensible aux échecs (par exemple, il peut mourir, laissant ainsi l'application sans surveillance), vous devez utiliser le gestionnaire de processus pour démarrer le démon du détecteur. L'utilitaire `scds_pmf_start()` de BDSD présente une prise en charge intégrée pour le démarrage des détecteurs de pannes. Cet utilitaire utilise le nom du chemin d'accès relatif (basé sur `Rép_base_TR` pour l'emplacement des mises en oeuvre de la méthode de rappel du type de ressource) du programme du démon du détecteur. Il utilise les propriétés d'extension `Intervalle_nouvelles_tentatives_détecteur` et `Nombre_nouvelles_tentatives_détecteur` gérées par la BDSD pour éviter des redémarrages illimités du démon. Il impose une syntaxe de ligne de commande identique à celle définie pour toutes les méthodes de rappel (à savoir, `-R` *ressource* `-G` *groupe_ressources* `-T` *type_ressource*) sur le démon du détecteur bien que celui-ci ne soit jamais appelé directement par le RGM. Il permet à la mise en oeuvre du démon du détecteur de profiter de l'utilitaire `scds_initialize()` pour configurer son propre environnement. L'effort principal consiste à concevoir le démon du détecteur lui-même.

Méthode `Arrêt_détecteur`

Le RGM appelle la méthode `Arrêt_détecteur` pour arrêter le démon du détecteur de pannes démarré par le biais de la méthode `Démarrage_détecteur`. L'échec de la méthode de rappel est traité exactement de la même manière que l'échec de la méthode `Arrêt` ; c'est pourquoi la méthode `Arrêt_détecteur` doit être aussi idempotente et robuste que la méthode `Arrêt`.

Si vous employez l'utilitaire `scds_pmf_start()` pour démarrer le démon du détecteur de pannes, recourez à `scds_pmf_stop()` pour l'arrêter.

Méthode `Contrôle_détecteur`

La méthode de rappel `Contrôle_détecteur` d'une ressource est appelée sur un noeud pour la ressource spécifiée afin de s'assurer que le noeud du cluster est capable de maîtriser la ressource (en d'autres termes : les applications gérées par la ressource peuvent-elles tourner sans problème sur ce noeud ?). Généralement, cette situation suppose que l'on s'assure que toutes les ressources système requises par l'application sont effectivement disponibles sur le noeud du cluster. Comme abordé dans la rubrique "Méthode `Validation`" à la page 134, la fonction `svc_validate()` mise en oeuvre par le développeur est destinée à obtenir au moins cette certitude.

En fonction de l'application spécifique gérée par la mise en oeuvre du type de ressource, la méthode `Contrôle_détecteur` peut être rédigée de manière à exécuter quelques tâches supplémentaires. La méthode `Contrôle_détecteur` doit être mise en oeuvre de manière à ne pas entrer en conflit avec d'autres méthodes exécutées simultanément. Pour les développeurs utilisant la BDSD, il est recommandé que la méthode `Contrôle_détecteur` tire profit de la fonction `svc_validate()` rédigée dans le but de mettre en oeuvre la validation spécifique à l'application des propriétés de la ressource.

Méthode `Mise_à_jour`

Le RGM appelle la méthode `Mise_à_jour` de la mise en oeuvre d'un type de ressource pour appliquer les modifications apportées par l'administrateur système à la configuration de la ressource active. La méthode `Mise_à_jour` n'est appelée que sur les noeuds (le cas échéant) sur lesquels la ressource est en ligne.

Il ne fait aucun doute que les modifications venant d'être apportées à la ressource sont acceptables pour la mise en oeuvre du type de ressource étant donné que le RGM exécute la méthode `Validation` propre au type de ressource avant de lancer la méthode `Mise_à_jour`. La méthode `Validation` est appelée avant la modification des propriétés de la ressource ou du groupe de ressources, et elle peut s'opposer aux changements proposés. La méthode `Mise_à_jour` est appelée après l'application des modifications de manière à permettre à la ressource active (en ligne) de prendre les nouveaux paramètres en compte.

En tant que développeur du type de ressource, vous devez faire preuve de prudence lorsque vous décidez des propriétés pouvant être mises à jour de manière dynamique et les marquez à l'aide du paramètre `RÉGLABLE = À_TOUT_MOMENT` du fichier RTR. Généralement, vous pouvez spécifier que vous souhaitez pouvoir mettre à jour de manière dynamique une propriété relative à la mise en oeuvre d'un type de ressource et utilisée par le démon du détecteur de pannes, pour autant que la méthode `Mise_à_jour` redémarre le démon du détecteur.

Les candidats possibles sont les suivants :

- `Intervalle_sonde_complet;`
- `Nombre_nouvelles_tentatives;`
- `Intervalle_nouvelles_tentatives;`
- `Nombre_nouvelles_tentatives_détecteur;`
- `Intervalle_nouvelles_tentatives_détecteur;`
- `Délai_sonde`.

Ces propriétés influent sur la manière dont le démon du détecteur de pannes contrôle la santé du service, à quelle fréquence il le fait, quel intervalle d'historique il utilise pour assurer le suivi des erreurs et quels seuils de redémarrage sont définis pour lui par le gestionnaire de processus. L'utilitaire `scds_pmf_restart ()` fourni dans la BDSD permet de mettre ces propriétés à jour.

Si vous pouvez mettre à jour de manière dynamique une propriété de ressource mais que la modification de celle-ci risque d'influer sur l'application en cours d'exécution, vous devez prendre les mesures appropriées de manière à ce que les mises à jour de cette propriété soient appliquées correctement dans toutes les instances de l'application en cours d'exécution. Actuellement, la BDSD ne propose aucune solution permettant de faciliter cette opération. `Mise_à_jour` ne transmet pas les propriétés modifiées à la ligne de commande (comme le fait `Validation`).

Méthodes `Init`, `Fini` et `Initialisation`

Il s'agit de méthodes *à action unique* telles que définies par les spécifications de l'API de gestion des ressources. La mise en oeuvre modèle fournie avec la BDSD n'illustre pas l'utilisation de ces méthodes. Toutefois, toutes les fonctions de la BDSD sont également disponibles dans celles-ci, si un développeur de type de ressource devait en avoir besoin. Généralement, les méthodes d'`Init` et d'`Initialisation` sont exactement les mêmes pour le type de ressource qui mettrait en oeuvre une *action unique*. La méthode `Fini` exécute généralement une opération *désactivant* l'action des méthodes `Init` ou `Initialisation`.

Conception du démon du détecteur de pannes

Les mises en oeuvre du type de ressource utilisant la BDSD possèdent généralement un démon du détecteur de pannes assumant les responsabilités suivantes :

- Surveillance périodique de la santé de l'application gérée. Cet aspect particulier du démon du détecteur dépend fortement de l'application et peut varier considérablement d'un type de ressource à l'autre. La BDSD présente certaines fonctions intégrées permettant de contrôler la santé de services simples basés sur le protocole TCP. Les applications mettant en oeuvre des protocoles basés sur le langage ASCII tels que HTTP, NNTP, IMAP et POP3 peuvent être mises en oeuvre

à l'aide de ces utilitaires.

- Conservation d'une trace des problèmes rencontrés par l'application utilisant les propriétés de ressources `Intervalle_nouvelles_tentatives` et `Nombre_nouvelles_tentatives`. Décision concernant l'attitude à adopter en cas d'échec : le script d'action du gestionnaire de processus doit-il redémarrer le service ou les échecs de l'application se sont-ils accumulés si rapidement qu'un basculement doit être envisagé ? Les utilitaires `scds_fm_action()` et `scds_fm_sleep ()` de la BDSD sont destinés à vous aider à mettre ce mécanisme en oeuvre.
- Prise des mesures appropriées (généralement, redémarrer l'application ou tenter de basculer le groupe contenant la ressource). L'utilitaire BDSD `scds_fm_action()` met en oeuvre un tel algorithme. Il calcule l'accumulation actuelle des échecs de sonde au cours du délai Intervalle_nouvelles_tentatives (secondes) défini dans ce but.
- Mise à jour de l'état des ressources de manière à ce que la commande `scstat` puisse accéder à l'état de santé de l'application, tout comme l'IUG de la gestion du cluster.

Les utilitaires BDSD sont conçus de manière à ce que la boucle principale du démon du détecteur de pannes soit représentée par le pseudo-code suivant :

Pour les détecteurs de pannes mis en oeuvre par le biais de BDSD :

- La détection de la mort du processus de l'application par le biais de la fonction `scds_fm_sleep ()` est assez rapide, étant donné que sa notification par le gestionnaire de processus est asynchrone. Comparez ce cas à celui où un détecteur de pannes s'active au bout d'un délai donné pour vérifier la santé du service et constate la mort de l'application. Le délai de détection des pannes se trouve considérablement réduit, ce qui permet d'accroître la disponibilité du service.
- Si le RGM rejette la tentative de basculement du service par le biais de l'API `scha_control(3HA)`, `scds_fm_action()` *réinitialise* (oublie) son historique des pannes actuel. La raison en est que l'historique des pannes se situe déjà au-delà de `Nombre_nouvelles_tentatives`, et si le démon du détecteur s'active au cours de l'itération suivante et est incapable de contrôler la santé du démon, il essaie à nouveau d'appeler la fonction `scha_control()`, nouvelle tentative vouée à l'échec étant donné que la situation ayant donné lieu au rejet au cours de la dernière itération perdure. La réinitialisation de l'historique garantit que le détecteur de pannes tente au moins de corriger la situation localement (par exemple, par le biais d'un redémarrage de l'application) au cours de l'itération suivante.
- `scds_fm_action()` ne réinitialise *pas* l'historique des pannes de l'application dans le cas d'un échec du redémarrage, étant donné que l'utilisateur tente généralement d'utiliser `scha_control()` rapidement si la situation ne se règle pas d'elle-même.

- L'utilitaire `scds_fm_action()` met à jour l'état des ressources en le faisant passer à `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_DEGRADED` ou `SCHA_RSSTATUS_FAULTED`, sur la base de l'historique des pannes. Cet état est donc disponible pour la gestion du système de cluster.

Dans la plupart des cas, le contrôle de santé spécifique à l'application peut être mis en oeuvre dans un utilitaire autonome distinct (`svc_probe()`, par exemple) et intégré à cette boucle principale générique :

```
for (;;) {

   / * Sommeil pendant la durée de l'intervalle Intervalle_sonde_complet
   *  entre des sondages successifs. */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));

   /* Sonde de toutes les adresses IP utilisées. Passage en boucle sur
   * 1. Toutes les ressources net utilisées.
   * 2. Toutes les adresses IP d'une ressource donnée.
   * Pour chacune des adresses sondées,
   * calcul de l'historique des pannes. */
   probe_result = 0;
   /* Itération dans toutes les ressources pour obtenir toutes
    * les adresses IP à utiliser pour l'appel de svc_probe() */
   for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
   /* Obtention du nom de l'hôte et du port dont
   * la santé doit être surveillée.
   */
   hostname = netaddr->netaddrs[ip].hostname;
   port = netaddr->netaddrs[ip].port_proto.port;
   /*
   * HA-XFS ne prend en charge qu'un seul port et
   * obtient donc la valeur du port à partir de la
   * première entrée du tableau des ports.
   */
   ht1 = gethrtime(); /* Blocage de l'heure de démarrage de sondage */
   probe_result = svc_probe(scds_handle,

   hostname, port, timeout);
   /*
   * Mise à jour de l'historique des sondages du service,
   * action si nécessaire.
   * Blocage de l'heure de fin de sondage.
   */
   ht2 = gethrtime();
   /* Conversion en millisecondes */
   dt = (ulong_t)((ht2 - ht1) / 1e6);

   /*
   * Calcul de l'historique des pannes et
   * action si nécessaire
   */
   (void) scds_fm_action(scds_handle,
   probe_result, (long)dt);
```

```
}       /* Chaque ressource net */
}       /* Toujours continuer le sondage */
```

CHAPITRE 8

Mise en oeuvre du type de ressource BDSD modèle

Ce chapitre décrit un type de ressource modèle, `SUNW.xfnts`, mis en oeuvre avec BDSD. Le service de données est rédigé en C. L'application sous-jacente est le serveur X Font, un service basé sur le protocole TCP/IP.

Les informations fournies dans ce chapitre sont les suivantes :

Serveur X Font

Le serveur X Font est un service simple, basé sur le protocole TCP/IP, servant des fichiers de police à ses clients. Les clients se connectent au serveur pour demander un jeu de polices et le serveur lit les fichiers de police à partir du disque avant de les servir aux clients. Le démon du serveur X Font se compose d'un binaire de serveur `/usr/openwin/bin/xfs`. Le démon est normalement démarré à partir de `inetd`. Toutefois, pour le modèle présenté, supposez que l'entrée appropriée du fichier `/etc/inetd.conf` a été désactivée (par exemple, par la commande `fsadmin -d`). Le démon est donc sous le seul contrôle de Sun Cluster.

Fichier de configuration du serveur X Font

Par défaut, le serveur X Font lit ses informations de configuration à partir du fichier `/usr/openwin/lib/X11/fontserver.cfg`. L'entrée de catalogue de ce fichier contient une liste des répertoires de police que peut servir le démon. L'administrateur du cluster peut localiser ces répertoires sur le système de fichiers global (afin d'optimiser l'utilisation du serveur X Font sur Sun Cluster en gérant une seule copie de la base de données des polices sur le système). Dans ce cas, l'administrateur doit éditer `fontserver.cfg` afin de refléter les nouveaux chemins des répertoires contenant les polices.

Pour faciliter la configuration, l'administrateur peut également placer le fichier de configuration lui-même dans le système de fichiers global. Le démon `xfs` propose des arguments de ligne de commande permettant d'ignorer l'emplacement intégré par défaut de ce fichier. Le type de ressource `SUNW.xfnts` utilise la commande suivante pour démarrer le démon sous le contrôle de Sun Cluster :

```
/usr/openwin/bin/xfs -config <emplacement_du_fichier_cfg>/fontserver.cfg \
-port <numéroport>
```

Dans la mise en oeuvre du type de ressource `SUNW.xfnts`, vous pouvez utiliser la propriété `Liste_rép_conf` pour gérer l'emplacement du fichier de configuration `fontserver.cfg`.

Numéro du port TCP

Le numéro du port TCP qu'écoute le démon du serveur `xfs` est normalement le port "fs" (généralement défini comme `7100` dans le fichier `/etc/services`). Toutefois, l'option `-port` de la ligne de commande `xfs` permet à l'administrateur système d'ignorer la valeur par défaut. Vous pouvez utiliser la propriété `Liste_ports` dans le type de ressources `SUNW.xfnts` pour définir la valeur par défaut et pour prendre en charge l'utilisation de l'option `-port` dans la ligne de commande `xfs`. Vous définissez la valeur par défaut de cette propriété comme `7100/tcp` dans le fichier RTR. Dans la méthode `SUNW.xfnts Démarrage`, vous transmettez `Liste_ports` à l'option `-port` de la ligne de commande `xfs`. Par conséquent, un utilisateur de ce type de ressource n'est pas obligé de spécifier un numéro de port (le port par défaut est `7100/tcp` mais, s'il le souhaite, il peut en indiquer un autre lors de la configuration du type de ressource. Pour ce faire, il lui suffit d'entrer une valeur différente pour la propriété `Liste_ports`.

Conventions de dénomination

Vous pouvez identifier les différentes parties du code modèle en gardant les conventions suivantes à l'esprit :

- Les fonctions API GR commencent par `scha_`.
- Les fonctions BDSD commencent par `scds_`.
- Les méthodes de rappel commencent par `xfnts_`.
- Les fonctions rédigées par l'utilisateur commencent par `svc_`.

Fichier RTR `SUNW.xfnts`

Cette rubrique décrit plusieurs propriétés clés du fichier RTR `SUNW.xfnts`. Elle ne décrit pas l'objectif de chaque propriété du fichier. Pour une telle description, reportez-vous à la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 34.

La propriété d'extension `Liste_rép_conf` identifie le répertoire de configuration (ou une liste de répertoires) de la manière suivante :

```
{
        PROPRIÉTÉ = Liste_rép_conf;
        EXTENSION;
        CHAÎNE;
        RÉGLABLE = À_LA_CRÉATION;
        DESCRIPTION = "Chemin(s) du répertoire de configuration";
}
```

La propriété `Liste_rép_conf` ne spécifie pas de valeur par défaut. L'administrateur du cluster doit indiquer un répertoire au moment de la création de la ressource. Cette valeur ne peut plus être modifiée ultérieurement parce que son paramétrage est limité à `À_LA_CRÉATION`.

La propriété `Liste_ports` identifie le port qu'écoute le démon du serveur de la manière suivante :

```
{
       PROPRIÉTÉ = Liste_ports;
       PAR DÉFAUT = 7100/tcp;
       RÉGLABLE = À_LA_CRÉATION;
}
```

La propriété déclarant une valeur par défaut, l'administrateur du cluster a la possibilité de spécifier une nouvelle valeur ou d'accepter la valeur par défaut au moment de la création de la ressource. Cette valeur ne peut plus être modifiée ultérieurement parce que son paramétrage est limité à `À_LA_CRÉATION`.

Fonction `scds_initialize()`

BDSD exige que chaque méthode de rappel appelle la fonction `scds_initialize`(3HA) au début de la méthode. Cette fonction exécute les opérations suivantes :

- Vérifie et traite les arguments de la ligne de commande (`argc` et `argv`) transmis par la structure du service de données. La méthode ne doit pas traiter plus avant les arguments de la ligne de commande.
- Définit les structures de données internes pour une utilisation par les autres fonctions dans BDSD.
- Initialise l'environnement de consignation.
- Valide les paramètres de sondage du détecteur de pannes.

La fonction `scds_close()` vous permet de récupérer les ressources allouées par `scds_initialize()`.

Méthode `Démarrage_xfnts`

Le RGM appelle la méthode `Démarrage` sur un noeud du cluster lorsque le groupe contenant la ressource du service de données est mis en ligne sur ce noeud ou lorsque la ressource est activée. Dans le type de ressource modèle `SUNW.xfnts`, la méthode `Démarrage_xfnts` active le démon `xfs` sur ce noeud.

La méthode `Démarrage_xfnts` appelle `scds_pmf_start ()` pour démarrer le démon sous le gestionnaire de processus. Celui-ci propose une notification automatique des pannes ainsi que des fonctions de redémarrage, de même qu'une intégration avec le détecteur de pannes.

Remarque – le premier appel de `Démarrage_xfnts` est adressé à `scds_initialize()` qui exécute des fonctions de *gestion interne* nécessaires (pour de plus amples informations, reportez-vous à la rubrique “Fonction `scds_initialize()`” à la page 148 et à la page de manuel `scds_initialize`(3HA)).

Validation du service avant démarrage

Avant de tenter de démarrer le serveur X Font, la méthode `Démarrage_xfnts` appelle `validation_svc()` pour vérifier si une configuration correcte est en place pour prendre en charge le démon `xfs` (reportez-vous à la rubrique “Méthode `Validation_xfnts`” à la page 163 pour de plus amples informations), de la manière suivante :

```
rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }
```

Démarrage du service

La méthode `Démarrage_xfnts` appelle la méthode `Démarrage_svc()`, définie dans `xfnts.c` pour démarrer le démon `xfs`. Cette rubrique décrit `Démarrage_svc()`.

La commande permettant de lancer le démon `xfs` est la suivante :

`xfs -config` *répertoire_config*`/fontserver.cfg -port` *numéro_port*

La propriété d’extension `Liste_rép_conf` identifie le *répertoire_config* alors que la propriété système `Liste_port` identifie le *numéro_port*. Lorsque l’administrateur du cluster configure le service de données, il fournit des valeurs spécifiques pour ces propriétés.

La méthode `Démarrage_xfnts` déclare ces propriétés comme tableaux de chaînes et obtient les valeurs définies par l’administrateur à l’aide des fonctions `scds_get_ext_confdir_list()` et `scds_get_port_list()` (décrites dans `scds_property_functions`(3HA)), de la manière suivante :

```
scha_str_array_t *confdirs;
scds_port_list_t    *portlist;
scha_err_t   err;

   /* obtention du répertoire de configuration depuis la propriété liste_rép_conf */
   confdirs = scds_get_ext_confdir_list(scds_handle);
```

```
   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /* obtention du port à utiliser par XFS depuis la propriété Liste_port */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }
```

Remarquez que la variable `Rép_conf` pointe vers le premier élément (0) du tableau.

La méthode `Démarrage_xfnts` utilise `sprintf` pour former la ligne de commande pour `xfs` de la manière suivante :

```
/* Construction de la commande pour démarrer le démon xfs. */
(void) sprintf(cmd,
"/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null", xfnts_conf, portlist->ports[0].port);
```

Remarquez que la sortie est redirigée vers `dev/null` pour supprimer les messages générés par le démon.

La méthode `Démarrage_xfnts` transmet la ligne de commande `xfs` à `scds_pmf_start()` pour démarrer le service de données sous le contrôle du gestionnaire de processus, de la manière suivante :

```
scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

   if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
   }
```

Tenez compte des points suivants concernant l'appel vers `scds_pmf_start ()` :

- Le paramètre `SCDS_PMF_TYPE_SVC` identifie le programme pour démarrer comme application du service de données (cette méthode peut également démarrer un détecteur de pannes ou certains autres types d'application).
- Le paramètre `SCDS_PMF_SINGLE_INSTANCE` l'identifie comme une ressource à instance unique.
- Le paramètre `cmd` est la ligne de commande générée précédemment.
- Le paramètre final, `-1`, spécifie le niveau de surveillance enfant. Le -1 spécifie que le gestionnaire de processus surveille tous les processus enfants de même que le processus d'origine.

Avant son retour, `svc_pmf_start()` libère la mémoire allouée à la structure `liste_ports`, de la manière suivante :

```
scds_free_port_list(portlist); return (err);
```

Retour de `démarrage_svc()`

Même lorsque `démarrage_svc()` renvoie une réussite, il est possible que l'application sous-jacente n'arrive pas à démarrer. `démarrage_svc()` doit donc sonder l'application afin de vérifier qu'elle fonctionne avant de renvoyer un message de réussite. La sonde doit tenir compte du fait que l'application peut ne pas être immédiatement disponible parce que son démarrage prend un certain temps. La méthode `Démarrage_svc ()` appelle `svc_wait()`, définie dans `xfnts.c`, pour vérifier si l'application tourne, de la manière suivante :

```
/* Attendre que le service ait complètement démarré */
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling svc_wait to verify that service has started.");

rc = svc_wait(scds_handle);

scds_syslog_debug(DBG_LEVEL_HIGH,
"Returned from svc_wait");

if (rc == 0)
{ scds_syslog(LOG_INFO, "Successfully started the service.");
} else {
scds_syslog(LOG_ERR, "Failed to start the service.");
}
```

La fonction `svc_wait()` appelle `scds_get_netaddr_list`(3HA) pour obtenir les ressources d'adresses réseau nécessaires pour sonder l'application, de la manière suivante :

```
/* Obtention de la ressource réseau à utiliser pour le sondage */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
      return (1);
   }

   /* Renvoi d'une erreur en l'absence de ressources réseau */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

Ensuite, `svc_wait()` obtient les valeurs de `délai_démarrage` et de `délai_arrêt` de la manière suivante :

```
svc_start_timeout = scds_get_rs_start_timeout(scds_handle)
   probe_timeout = scds_get_ext_probe_timeout(scds_handle)
```

Pour tenir compte du délai nécessaire au serveur pour démarrer, `svc_wait()` appelle `scds_svc_wait()` et transmet une valeur de délai imparti équivalant à trois pourcent de la valeur de `délai_démarrage`. Ensuite, `svc_wait()` appelle `svc_probe()` pour vérifier que l'application a démarré. La méthode `svc_probe()` établit une connexion de socket simple au serveur sur le port spécifié. En cas d'échec de la connexion au port, `svc_probe()` renvoie une valeur de 100, indiquant un échec total. Si la connexion s'établit mais que la déconnexion du port échoue, `svc_probe()` renvoie une valeur de 50.

En cas d'échec total ou partiel de `svc_probe()`, `svc_wait()` appelle `scds_svc_wait()` avec un délai imparti de 5. La méthode `scds_svc_wait()` limite la fréquence des sondages à un intervalle de 5 secondes. Cette méthode comptabilise également le nombre de tentatives de démarrage du service. Si le nombre de tentatives dépasse la valeur de la propriété `Nombre_nouvelles_tentatives` de la ressource dans la période spécifiée par la propriété `Intervalle_nouvelles_tentatives` de la ressources, la fonction `scds_svc_wait()` renvoie un échec. Dans ce cas, la fonction `démarrage_svc()` renvoie également un échec.

```
#définir    SVC_CONNECT_TIMEOUT_PCT    95
#définir    SVC_WAIT_PCT       3
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

   do {
      /*
       * sondage du service de données sur l'adresse IP
       * de la ressource réseau et du nom du port
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Réussite. Libération des ressources et retour */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

       /* Appel de scds_svc_wait() au cas où le service échoue également
         if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }
```

```
   /* Appui sur le RGM pour arrêter le programme en cas de dépassement du délai imparti */
   } while (1);
```

Remarque – avant sa fermeture, la méthode `Démarrage_xfnts` appelle `scds_close()` de manière à récupérer les ressources allouées par `scds_initialize ()`. Reportez-vous à la rubrique "Fonction `scds_initialize ()`" à la page 148 et à la page de manuel `scds_close`(3HA) pour de plus amples informations.

Méthode `Arrêt_xfnts`

La méthode `Démarrage_xfnts` utilisant `scds_pmf_start()` pour démarrer le service sous le gestionnaire de processus, `Arrêt_xfnts` emploie `scds_pmf_stop()` pour arrêter celui-ci.

Remarque – le premier appel d'`Arrêt_xfntsp` est destiné à la fonction `scds_initialize()` exécutant des fonctions de *gestion interne* nécessaires (pour de plus amples informations, reportez-vous à la rubrique "Fonction `scds_initialize ()`" à la page 148 et à la page de manuel `scds_initialize`(3HA)).

La méthode `Arrêt_xfnts` appelle la méthode `Arrêt_svc ()`, définie dans `xfnts.c` de la manière suivante :

```
scds_syslog(LOG_ERR, "Issuing a stop request.");
   err = scds_pmf_stop(scds_handle,
       SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
       scds_get_rs_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Successfully stopped HA-XFS.");
   return (SCHA_ERR_NOERR); /* Arrêt réussi */
```

Tenez compte des points suivants concernant l'appel d'`Arrêt_svc()` destiné à la fonction `scds_pmf_stop()` :

- Le paramètre SCDS_PMF_TYPE_SVC identifie le programme à arrêter comme une application de service de données (cette méthode peut également arrêter un détecteur de pannes ou un autre type d'application).
- Le paramètre SCDS_PMF_SINGLE_INSTANCE identifie le signal.
- Le paramètre SIGTERM identifie le signal à utiliser pour arrêter l'instance de la ressource. Si le signal ne peut pas arrêter l'instance, scds_pmf_stop() envoie SIGKILL pour le faire et, en cas d'échec, renvoie une erreur de dépassement du délai imparti. Reportez-vous à la page de manuel scds_pmf_stop(3HA) pour de plus amples informations.
- La valeur du délai imparti correspond à la propriété Délai_arrêt de la ressource.

Remarque – avant sa fermeture, la méthode Arrêt_xfnts appelle scds_close() afin de récupérer les ressources réparties par scds_initialize (). Reportez-vous à la rubrique "Fonction scds_initialize()" à la page 148 et à la page de manuel scds_close(3HA) pour de plus amples informations.

Méthode Démarrage_détecteur_xfnts

Le RGM appelle la méthode Démarrage_détecteur sur un noeud pour démarrer le détecteur de pannes une fois une ressource démarrée sur le noeud. La méthode Démarrage_détecteur_xfnts utilise scds_pmf_start() pour démarrer le démon du détecteur sous le gestionnaire de processus.

Remarque – le premier appel de Démarrage_détecteur_xfnts est adressé à scds_initialize() qui exécute des fonctions de *gestion interne* nécessaires (pour de plus amples informations, reportez-vous à la rubrique "Fonction scds_initialize()" à la page 148 et à la page de manuel scds_initialize(3HA)).

La méthode Démarrage_détecteur_xfnts appelle la méthode Démarrage_mon, définie dans xfnts.c de la manière suivante :

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling Monitor_start method for resource <%s>.",
      scds_get_resource_name(scds_handle));

    /* Appel de scds_pmf_start et transmission du nom de la sonde. */
```

```
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Started the fault monitor.");

   return (SCHA_ERR_NOERR); /* Détecteur démarré avec succès */
}
```

Tenez compte des points suivants concernant l'appel de `svc_mon_start()` adressé à la fonction `scds_pmf_start()` :

- Le paramètre `SCDS_PMF_TYPE_MON` identifie le programme à démarrer comme un détecteur de pannes (cette méthode peut également démarrer un service de données ou un autre type d'application).
- Le paramètre `SCDS_PMF_SINGLE_INSTANCE` l'identifie comme une ressource à instance unique.
- Le paramètre `xfnts_probe` identifie le démon du détecteur à démarrer. Le système suppose que le démon du détecteur se trouve dans le même répertoire que les autres programmes de rappel.
- Le paramètre final, `0`, spécifie le niveau de surveillance enfant. Dans ce cas, seul le démon du détecteur est surveillé.

Remarque – avant sa fermeture, la méthode `Démarrage_détecteur_xfnts` appelle `scds_close()` pour récupérer les ressources allouées par `scds_initialize()`. Reportez-vous à la rubrique "Fonction `scds_initialize()`" à la page 148 et à la page de manuel `scds_close`(3HA) pour de plus amples informations.

Méthode `Arrêt_détecteur_xfnts`

La méthode `Démarrage_détecteur_xfnts` utilisant `scds_pmf_start()` pour démarrer le démon du détecteur sous le gestionnaire de processus, `Arrêt_détecteur_xfnts` utilise `scds_pmf_stop()` pour l'arrêter.

Remarque – le premier appel d'`Arrêt_détecteur_xfnts` est destiné à `scds_initialize()` effectuant les fonctions *de gestion interne* nécessaires (pour de plus amples informations, consultez la rubrique "Fonction `scds_initialize()`" à la page 148 et la page de manuel `scds_initialize`(3HA).)

La méthode `Arrêt_détecteur_xfnts()` appelle la méthode `mon_stop`, définie dans `xfnts.c`, de la manière suivante :

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");

   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Détecteur arrêté avec succès */
}
```

Tenez compte des points suivants concernant l'appel de `svc_mon_stop()` destiné à la fonction `scds_pmf_stop()` :

- Le paramètre `SCDS_PMF_TYPE_MON` identifie le programme à arrêter comme un détecteur de pannes (cette méthode peut également arrêter un service de données ou un autre type d'application).
- Le paramètre `SCDS_PMF_SINGLE_INSTANCE` l'identifie comme une ressource à instance unique.
- Le paramètre `SIGKILL` identifie le signal à utiliser pour arrêter l'instance de la ressource. Si ce signal ne peut pas arrêter l'instance, `scds_pmf_stop()` renvoie une erreur de dépassement du délai imparti. Reportez-vous à la page de manuel `scds_pmf_stop`(3HA) pour de plus amples informations.
- La valeur du délai imparti correspond à la propriété `Délai_arrêt_détecteur` de la ressource.

Remarque – avant sa fermeture, la méthode `Arrêt_détecteur_xfnts` appelle `scds_close()` pour récupérer les ressources allouées par `scds_initialize()`. Reportez-vous à la rubrique “Fonction `scds_initialize()`” à la page 148 et à la page de manuel `scds_close`(3HA) pour de plus amples informations.

Méthode `Contrôle_détecteur_xfnts`

Le RGM appelle la méthode `Contrôle_détecteur` lorsque le détecteur de pannes tente de basculer le groupe contenant la ressource sur un autre noeud. La méthode `Contrôle_détecteur_xfnts` appelle la méthode `Validation_svc()` pour vérifier qu’une configuration est en place pour prendre en charge le démon `xfs` (reportez-vous à la rubrique “Méthode `Validation_xfnts`” à la page 163 pour de plus amples informations). Le code de `Contrôle_détecteur_xfnts` est le suivant :

```
   /* Traitement des arguments transmis par le RGM et initialisation de syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "monitor_check method "
       "was called and returned <%d>.", rc);

   /* Libération de la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Renvoi du résultat de la méthode de "Validation" dans le cadre du contrôle du détecteur */
   return (rc);
}
```

Détecteur de pannes SUNW.xfnts

Le RGM n'appelle pas directement la méthode SONDE, mais plutôt la méthode Démarrage_détecteur pour démarrer le détecteur une fois une ressource démarrée sur un noeud. La méthode Démarrage_détecteur_xfnts démarre le détecteur de pannes sous le contrôle du gestionnaire de processus. La méthode Arrêt_détecteur_xfnts arrête le détecteur de pannes.

Le détecteur de pannes SUNW.xfnts effectue les opérations suivantes :

- Il surveille périodiquement la santé du démon du serveur xfs à l'aide d'utilitaires spécifiquement conçus pour vérifier des services simples basés sur le protocole TCP, tels que xfs.
- Il assure le suivi des problèmes rencontrés par une application dans un délai donné (à l'aide des propriétés Nombre_nouvelles_tentatives et Intervalle_nouvelles_tentatives) et décide s'il convient de redémarrer le service de données ou de le basculer dans le cas d'un échec total de l'application. Les fonctions scds_fm_action() et scds_fm_sleep() fournissent une assistance intégrée à ce mécanisme de suivi et de décision.
- Il met en oeuvre la décision de basculement ou de redémarrage à l'aide de la fonction scds_fm_action().
- Il met à jour l'état des ressources et le met à la disposition des outils d'administration et interfaces utilisateur graphiques.

Boucle principale xfonts_probe

La méthode xfonts_probe met une boucle en oeuvre. Avant cela, xfonts_probe :

- Récupère les ressources d'adresses réseau de la ressource xfnts de la manière suivante :

```
/* Obtention des adresses IP disponibles pour cette ressource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Renvoi d'une erreur en l'absence de ressources réseau */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

- Appelle `scds_fm_sleep()` et transmet la valeur de `intervalle_sonde_complet` sous la forme du délai imparti. La sonde passe en mode de sommeil pendant le délai défini par l'`Intervalle_sonde_complet` entre les sondages.

```
timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {
      /*
       * Sommeil pendant une durée d'Intervalle_sonde_complet entre les
       *  sondages successifs.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));
```

La méthode `xfnts_probe` met la boucle en oeuvre de la manière suivante :

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Obtention du nom d'hôte et du port sur lesquels
          * la santé a été surveillée.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS ne prend en charge qu'un seul port
          * et obtient la valeur du port depuis la
          * première entrée du tableau des ports.
          */
         ht1 = gethrtime(); /* Blocage du délai de démarrage du sondage */
         scds_syslog(LOG_INFO, "Probing the service on port: %d.", port);

         probe_result =
         svc_probe(scds_handle, hostname, port, timeout);

         /*
          * Mise à jour de l'historique des sondages du service,
          * action si nécessaire.
          * Blocage du délai de fin du sondage.
          */
         ht2 = gethrtime();

         /* Conversion en millisecondes */
         dt = (ulong_t)((ht2 - ht1) / 1e6);

         /*
          * Calcul de l'historique des pannes
          * et action si nécessaire
          */
         (void) scds_fm_action(scds_handle,
             probe_result, (long)dt);
      }   /* Chaque ressource réseau */
   }    /* Toujours continuer le sondage */
```

La fonction `svc_probe()` met la logique de sondage en oeuvre. La valeur retournée par `svc_probe()` est transmise à `scds_fm_action()`, qui détermine s'il convient de redémarrer l'application, de basculer le groupe de ressources ou de ne rien faire.

Fonction `svc_probe()`

La fonction `svc_probe()` établit une connexion de socket simple au port spécifié en appelant `scds_fm_tcp_connect()`. Si la connexion échoue, `svc_probe()` renvoie une valeur de `100` indiquant un échec total. Si la connexion s'établit, mais que la déconnexion échoue, `svc_probe()` renvoie une valeur de `50` indiquant un échec partiel. Si la connexion et la déconnexion réussissent, `svc_probe()` renvoie une valeur de `0`, indiquant une réussite.

Le code de `svc_probe()` est le suivant :

```
int svc_probe(scds_handle_t scds_handle,
char *hostname, int port, int timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
   char   testcmd[2048];
   int    time_used, time_remaining;
   time_t      connect_timeout;


   /*
    * Sondage du service de données par le biais d'une connexion de socket au port */
    * spécifié dans la propriété liste_port de l'hôte alimentant le service
    * de données XFS. Si le service XFS configuré pour écouter le port
    * spécifié répond à la connexion, le sondage est réussi.
    * Dans le cas contraire, nous attendons pendant une période définie
    * dans la propriété délai_sonde avant de conclure à l'échec du sondage.
    */

   /*
    * Utilisez le pourcentage SVC_CONNECT_TIMEOUT_PCT du délai imparti
    * pour vous connecter au port
    */
   connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
   t1 = (hrtime_t)(gethrtime()/1E9);

   /*
    * la sonde établit une connexion au nom d'hôte et au port spécifiés.
    * La connexion reçoit 95% du délai_sonde effectif.
    */
   rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
       connect_timeout);
   if (rc) {
      scds_syslog(LOG_ERR,
          "Failed to connect to port <%d> of resource <%s>.",
```

```
        port, scds_get_resource_name(scds_handle));
    /* il s'agit d'un échec total */
    return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Calculez le temps réel requis pour la connexion. Il doit être inférieur
 * ou égal à délai_connexion, le temps alloué à la connexion.
 * Si la connexion utilise tout le délai qui lui est imparti,
 * la valeur restante de délai_sonde transmise à cette fonction
 * est utilisée comme délai imparti à la déconnexion. Sinon, le
 * temps restant de l'appel de connexion est ajouté au délai
 * de déconnexion.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Utilisez le temps restant (timeout - time_took_to_connect) pour la déconnexion
 */

time_remaining = timeout - (int)time_used;

/*
 * Si tout le délai est écoulé, utilisez un délai imparti court à code permanent
 * pour essayer une nouvelle fois de provoquer la déconnexion.
 * Ceci évite la fuite fd.
 */
if (time_remaining <= 0) {
    scds_syslog_debug(DBG_LEVEL_LOW,
        "svc_probe used entire timeout of "
        "%d seconds during connect operation and exceeded the "
        "timeout by %d seconds. Attempting disconnect with timeout"
        " %d ",
        connect_timeout,
        abs(time_used),
        SVC_DISCONNECT_TIMEOUT_SECONDS);

    time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Renvoi d'un échec partiel en cas d'échec à la déconnexion.
 * Motif : l'appel de connexion réussit, ce qui signifie que
 * l'application vit. Un échec à la déconnexion peut* être
 * dû à une application bloquée ou à une lourde charge.
 * Dans ce dernier cas, ne déclarez pas que l'application
 * est morte en renvoyant un échec total. Au lieu de cela, déclarez
 * qu'il s'agit d'un échec partiel. Si cette situation perdure, l 'appel
 * de déconnexion échoue une nouvelle fois et l'application est
 * redémarrée.
 */
```

```
   rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
   if (rc != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to disconnect to port %d of resource %s.",
          port, scds_get_resource_name(scds_handle));
      /* il s'agit d'un échec partiel */
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }

   t2 = (hrtime_t)(gethrtime()/1E9);
   time_used = (int)(t2 - t1);
   time_remaining = timeout - time_used;

   /*
    * S'il ne reste pas de temps, n'effectuez pas le test complet avec
    * fsinfo. Au lieu de cela, renvoyez SCDS_PROBE_COMPLETE_FAILURE/2
    * Cette opération garantit que si le dépassement du délai
    * imparti persiste, le serveur sera redémarré.
    */
   if (time_remaining <= 0) {
      scds_syslog(LOG_ERR, "Probe timed out.");
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }

   /*
    * La connexion et la déconnexion au port ont réussi.
    * Exécutez la commande fsinfo pour effectuer un contrôle total de
    * la santé du serveur.
    * Redirigez stdout, sans quoi la sortie de fsinfo
    * se retrouve sur la console.
    */
   (void) sprintf(testcmd,
       "/usr/openwin/bin/fsinfo -server %s:%d > /dev/null",
       hostname, port);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Checking the server status with %s.", testcmd);
   if (scds_timerun(scds_handle, testcmd, time_remaining,
      SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

      scds_syslog(LOG_ERR,
         "Failed to check server status with command <%s>",
         testcmd);
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }
   return (0);
}
```

Après avoir terminé, `svc_probe()` renvoie une valeur indiquant une réussite (`0`), un échec partiel (`50`) ou un échec total (`100`). La méthode `xfnts_probe` transmet cette valeur à `scds_fm_action()`.

Définition de l'action du détecteur de pannes

La méthode `xfnts_probe` appelle `scds_fm_action()` pour déterminer la mesure à prendre. La logique `scds_fm_action()` est la suivante :

- Gérez un historique cumulé des pannes dans la valeur de la propriété `Intervalle_nouvelles_tentatives`.
- Si les pannes cumulées atteignent 100 (échec total), redémarrez le service de données. Si `Intervalle_nouvelles_tentatives` est dépassé, réinitialisez l'historique.
- Si le nombre de redémarrage dépasse la valeur de la propriété `Nombre_nouvelles_tentatives`, dans le délai spécifié par `Intervalle_nouvelles_tentatives`, basculez le service de données.

Par exemple, supposons que la sonde établisse une connexion au serveur xfs, mais ne puisse pas se déconnecter. Ceci indique que le serveur tourne mais qu'il peut être bloqué ou être provisoirement soumis à une forte charge. Un échec de la déconnexion renvoie une erreur partielle (`50`) à `scds_fm_action()`. Cette valeur se situe sous le seuil de redémarrage du service de données, mais la valeur est gérée dans l'historique des pannes.

Si, pendant le sondage suivant, le serveur n'arrive à nouveau pas à se déconnecter, une valeur `50` est ajoutée à l'historique des pannes géré par `scds_fm_action()`. La valeur cumulée des pannes est à présent de `100`. Par conséquent, `scds_fm_action()` redémarre le service de données.

Méthode `Validation_xfnts`

Le RGM appelle la méthode `Validation` quand une ressource est créée et quand une opération de l'administrateur met à jour les propriétés de la ressource ou du groupe la contenant. Le RGM appelle `Validation` avant la création ou la mise à jour, et un code de sortie avec échec issu de la méthode sur un noeud entraîne l'annulation de la création ou de la mise à jour.

Il n'appelle `Validation` que lorsque les propriétés de la ressource ou du groupe sont modifiées par une opération de l'administrateur, et non lorsque le RGM définit des propriétés, ou lorsqu'un détecteur définit les propriétés `Statut` et `msg_statut` de la ressource.

Remarque – la méthode `Contrôle_détecteur` appelle aussi explicitement la méthode `Validation` lorsque la méthode `SONDAGE` tente de basculer le service de données sur un autre noeud.

Le RGM appelle `Validation` avec des arguments différents de ceux transmis aux autres méthodes, y compris les propriétés et valeurs mises à jour. L'appel destiné à `scds_initialize()` au début de `Validation_xfnts` analyse tous les arguments que le RGM transmet à `Validation_xfnts` et enregistre les informations dans le paramètre `scds_handle`. Les sous-routines appelées par `Validation_xfnts` utilisent ces informations.

La méthode `Validation_xfnts` appelle `validation_svc ()` qui vérifie les éléments suivants :

- La propriété `Liste_rép_conf` a été paramétrée pour la ressource et définit un répertoire unique.

```
scha_str_array_t *confdirs;
   confdirs = scds_get_ext_confdir_list(scds_handle);

/* Renvoi d'une erreur en l'absence d'une propriété
 * d'extension liste_rép_conf
 */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Erreur de validation */
   }
```

- Le répertoire spécifié par `Liste_rép_conf` contient le fichier `fontserver.cfg`.

```
(void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * Suppression de l'erreur parce que le prototype errno.h
       * ne possède pas d'argument vide
       */
      scds_syslog(LOG_ERR,
          "Failed to access file <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }
```

- Le binaire du démon du serveur est accessible sur le noeud du cluster.

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
      scds_syslog(LOG_ERR,
          "Cannot access XFS binary : <%s> ", strerror(errno));
      return (1);
   }
```

- La propriété Liste_ports spécifie un port unique.

```
scds_port_list_t   *portlist;
   err = scds_get_port_list(scds_handle, &portlist);
```

```
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list: %s.",
         scds_error_string(err));
      return (1); /* Echec de la validation */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
      scds_syslog(LOG_ERR,
          "Property Port_list must have only one value.");
      scds_free_port_list(portlist);
      return (1); /* Echec de la validation */
   }
#endif
```

- Le groupe de ressources contenant le service de données contient également au moins une ressource d'adresse réseau.

```
scds_net_resource_list_t *snrlp;
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
      != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group: %s.",
         scds_error_string(err));
      return (1); /* Echec de la validation */
   }

   /* Renvoi d'une erreur en l'absence de ressources d'adresse réseau */
   if (snrlp == NULL || snrlp->num_netresources == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      rc = 1;
      goto finished;
   }
```

Avant son renvoi, `validation_svc()` libère toutes les ressources allouées.

```
finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* renvoi du résultat de la validation */
```

Remarque – avant sa fermeture, la méthode `Validation_xfnts` appelle `scds_close()` pour récupérer les ressources allouées par `scds_initialize()`. Reportez-vous à la rubrique "Fonction `scds_initialize()`" à la page 148 et à la page de manuel `scds_close`(3HA) pour de plus amples informations.

Méthode Mise_à_jour_xfnts

Le RGM appelle la méthode Mise_à_jour pour notifier à la ressource en cours d'exécution que ses propriétés ont changé. Les seules propriétés pouvant être modifiées par le service de données xfnts concernent le détecteur de pannes. C'est la raison pour laquelle, lorsqu'une propriété est mise à jour, la méthode Mise_à_jour_xfnts appelle scds_pmf_restart_fm () pour redémarrer le détecteur de pannes.

```
* vérifier si le détecteur de pannes tourne déjà et, dans ce cas, l'arrêter
* et le redémarrer. Le second paramètre pour scds_pmf_restart_fm()
* identifie de manière unique l'instance du détecteur de pannes
* à redémarrer.
*/
scds_syslog(LOG_INFO, "Restarting the fault monitor.");
result = scds_pmf_restart_fm(scds_handle, 0);
if (result != SCHA_ERR_NOERR) {
scds_syslog(LOG_ERR,
"Failed to restart fault monitor.");
/* Libère toute la mémoire allouée par scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
   "Completed successfully.");
```

Remarque – le second paramètre destiné à scds_pmf_restart_fm() identifie de manière unique l'instance du détecteur de pannes à redémarrer s'il en existe plusieurs. La valeur 0 de l'exemple indique qu'il n'existe qu'une seule instance du détecteur de pannes.

CHAPITRE 9

SunPlex Agent Builder

Ce chapitre présente SunPlex Agent Builder et le module Cluster Agent pour Agent Builder, deux outils d'automatisation de la création des types de ressources ou des services de données à exécuter sous le gestionnaire RGM (Resource Group Manager). Un type de ressources est principalement un wrapper qui enveloppe une application pour lui permettre de s'exécuter dans un environnement clusterisé sous le contrôle du gestionnaire RGM.

Agent Builder intègre une interface constituée d'écrans permettant l'entréee de données uniques sur votre application et le type de ressources que vous souhaitez créer. En fonction des données entrées, Agent Builder génère les logiciels suivants :

- Un ensemble de fichiers sources (C, korn shell (`ksh`) ou un service de données générique (GDS)) dédié à un type de ressources de basculement ou évolutives, correspondant aux méthodes de rappel du type de ressources.
- Un fichier RTR (Resource Type Registration) personnalisé si vous générez une source en C ou korn shell.
- Des scripts d'utilitaire personnalisés pour le démarrage, l'arrêt et la suppression d'une instance (ressource) d'un type de ressources, ainsi que des pages de manuel personnalisées fournissant de plus amples informations sur l'utilisation de chacun de ces fichiers.
- Un package Solaris comprenant les codes binaires (si vous générez une source C), un fichier RTR (si vous générez une source en C ou korn shell) et les scripts d'utilitaire.

Agent Builder prend en charge les applications/applications compatibles réseau utilisant le réseau pour communiquer avec les clients, ainsi que les applications non prévues pour fonctionner en réseau (ou autonomes). Agent Builder vous permet également de générer un type de ressources pour une application possédant plusieurs arborescences de processus indépendantes que la fonction PMF (Process Monitor Facility) doit surveiller et redémarrer individuellement (reportez-vous à la rubrique "Création de types de ressources contenant plusieurs arborescences de processus indépendantes" à la page 177).

Voici les rubriques traitées dans ce chapitre :

Utilisation d'Agent Builder

Cette rubrique présente la procédure d'utilisation d'Agent Builder ainsi que les tâches que vous devez effectuer avant de pouvoir l'utiliser. Vous y découvrirez également comment déployer Agent Builder après avoir généré le code de votre type de ressources.

Analyse de l'application

Avant d'utiliser Agent Builder, vous devez déterminer si votre application répond aux critères requis pour devenir hautement disponible ou évolutive. Agent Builder ne peut pas réaliser cette analyse s'appuyant uniquement sur les caractéristiques du temps d'exécution de l'application. Vous trouverez de plus amples informations à ce sujet dans la rubrique "Analyse du caractère approprié de l'application" à la page 29.

Il est possible qu'Agent Builder ne puisse pas toujours créer un type de ressources complet pour votre application. Dans la plupart des cas, cependant, il propose au moins une solution partielle. Par exemple, les applications particulièrement complexes peuvent nécessiter un code supplémentaire qu'Agent Builder ne génère pas par défaut. Il peut notamment s'agir d'un code permettant d'ajouter des contrôles de validation pour des propriétés supplémentaires ou de régler des paramètres qu'Agent Builder n'affiche pas. Le cas échéant, vous devez apporter des modifications au code source généré ou au fichier RTR. La conception d'Agent Builder vise justement à offrir cette souplesse.

De fait, Agent Builder insère des commentaires aux endroits du code source généré où vous pouvez ajouter votre propre code de type de ressources. Une fois le code source modifié, vous pouvez utiliser le fichier makefile généré par Agent Builder pour recompiler le code source et générer de nouveau le package de type de ressources.

Même si vous utilisez exclusivement le code de type de ressources que vous avez créé sans insérer de code généré par Agent Builder, vous pouvez déployer le fichier makefile et la structure fournis par Agent Builder pour créer le package Solaris de votre type de ressources.

Installation et configuration d'Agent Builder

Agent Builder ne requiert aucune installation spéciale. Agent Builder est intégré au package `SUNWscdev` qui est installé par défaut lors de l'installation standard de Sun Cluster (reportez-vous au *Sun Cluster Software Installation Guide for Solaris OS* pour obtenir de plus amples informations). Avant d'utiliser Agent Builder, vérifiez les points suivants :

- Java doit figurer dans la variable `$PATH`. Agent Builder dépend de Java (Java Development Kit version 1.3.1 ou version supérieure) et si Java ne figure pas dans la variable `$PATH`, `scdsbuilder` renvoie un message d'erreur.
- Vous avez installé le groupe de logiciels « Developer System Support » de Solaris 8 ou version supérieure.
- Le compilateur `cc` est fourni avec la variable `$PATH`. Agent Builder utilise la première occurrence de `cc` dans la variable `$PATH` pour identifier le compilateur dont il doit se servir pour générer un code binaire C pour le type de ressources. Si `cc` n'est pas intégré à la variable `$PATH`, Agent Builder désactive l'option de génération de code C (reportez-vous à la rubrique "Utilisation de l'écran Créer" à la page 171.

Remarque – vous pouvez utiliser un autre compilateur que le compilateur `cc` standard avec Agent Builder. Pour ce faire, vous pouvez par exemple créer un lien symbolique du compilateur `cc` vers un autre compilateur dans la variable `$PATH`, tel `gcc`. Vous pouvez également remplacer la spécification du compilateur dans le fichier makefile (actuellement `CC=cc`) par le chemin d'accès complet d'un autre compilateur. Par exemple, dans le fichier makefile généré par Agent Builder, remplacez `CC=cc` par `CC=`*pathname*`/gcc`. Le cas échéant, vous ne pouvez pas exécuter Agent Builder directement. Vous devez utiliser les commandes `make` et `make pkg` pour générer un code de service de données et un package.

Lancement d'Agent Builder

Lancez Agent Builder à l'aide de la commande suivante :

```
% /usr/cluster/bin/scdsbuilder
```

L'écran initial de Sun Builder (comme indiqué dans l'illustration suivante) apparaît.

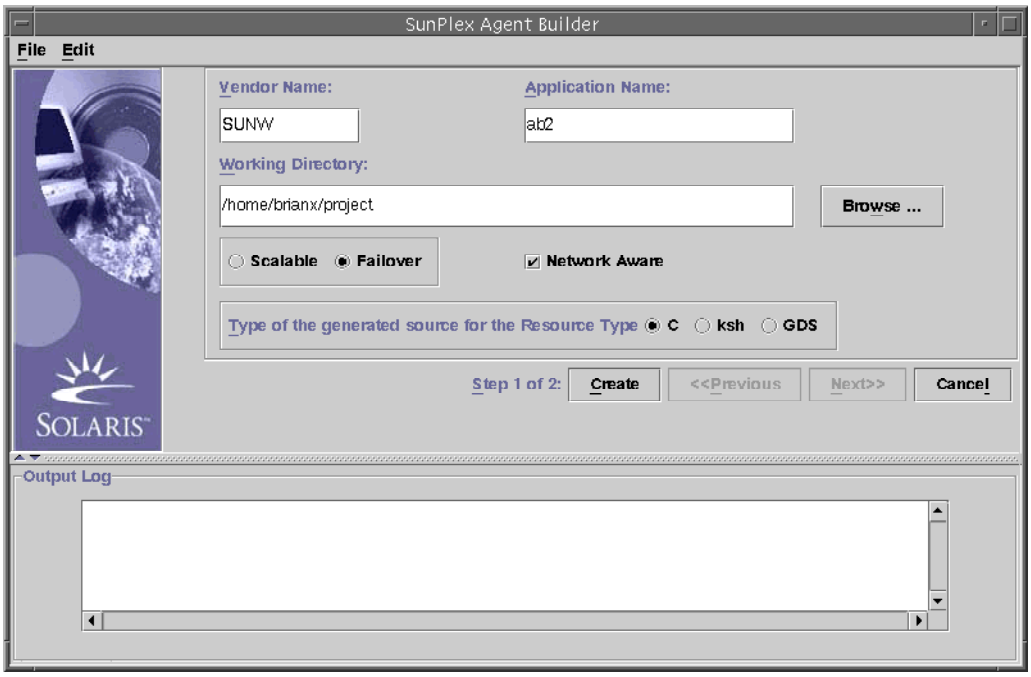


FIGURE 9–1 Écran initial

Remarque – vous pouvez accéder à Agent Builder par le biais de l'interface de ligne de commande (reportez-vous à la rubrique "Utilisation de la version ligne de commande d'Agent Builder" à la page 179) si l'interface utilisateur graphique n'est pas accessible.

Agent Builder comprend deux écrans vous guidant tout au long du processus de création d'un nouveau type de ressources :

1. **Créer :** cet écran contient des informations élémentaires sur le type de ressources à créer, comme son nom et le répertoire de travail (c'est-à-dire le répertoire dans lequel vous créez et configurez le modèle de type de ressources) des fichiers générés. Vous pouvez également identifier le type de ressources à créer (évolutives ou de basculement), si l'application de base est compatible réseau (c'est-à-dire si elle passe par le réseau pour communiquer avec ses clients) et le type de code (C, `ksh` ou GDS (service de données générique)) à générer. Pour de plus amples informations sur le code GDS (service de données générique), reportez-vous au Chapitre 10. Vous devez renseigner les champs de cet écran, puis sélectionner **Créer** pour générer la sortie correspondante, avant d'afficher l'écran Configurer.
2. **Configurer :** vous devez indiquer dans cet écran une commande permettant de démarrer l'application. Vous pouvez, de façon facultative, fournir des commandes pour interrompre et sonder l'application. Si vous n'entrez pas ces commandes, la

sortie générée utilise des signaux pour arrêter l'application et fournit un mécanisme de détection par défaut (reportez-vous à la description de la commande correspondante dans la rubrique "Utilisation de l'écran Configurer" à la page 174). Cet écran vous permet également de modifier les valeurs de délai pour chacune de ces trois commandes.

Remarque – si vous lancez Agent Builder à partir du répertoire de travail d'un type de ressources existant, Agent Builder initialise les écrans Créer et Configurer en reprenant les valeurs du type de ressources existant.

Reportez-vous à la rubrique "Navigation dans Agent Builder" à la page 184 si vous avez des questions sur l'utilisation des boutons ou des commandes de menu de l'un ou l'autre des écrans d'Agent Builder.

Utilisation de l'écran Créer

La première étape de création d'un type de ressources consiste à renseigner l'écran Créer qui s'affiche au lancement d'Agent Builder. L'illustration suivante présente l'écran Créer une fois que vous avez renseigné les champs.

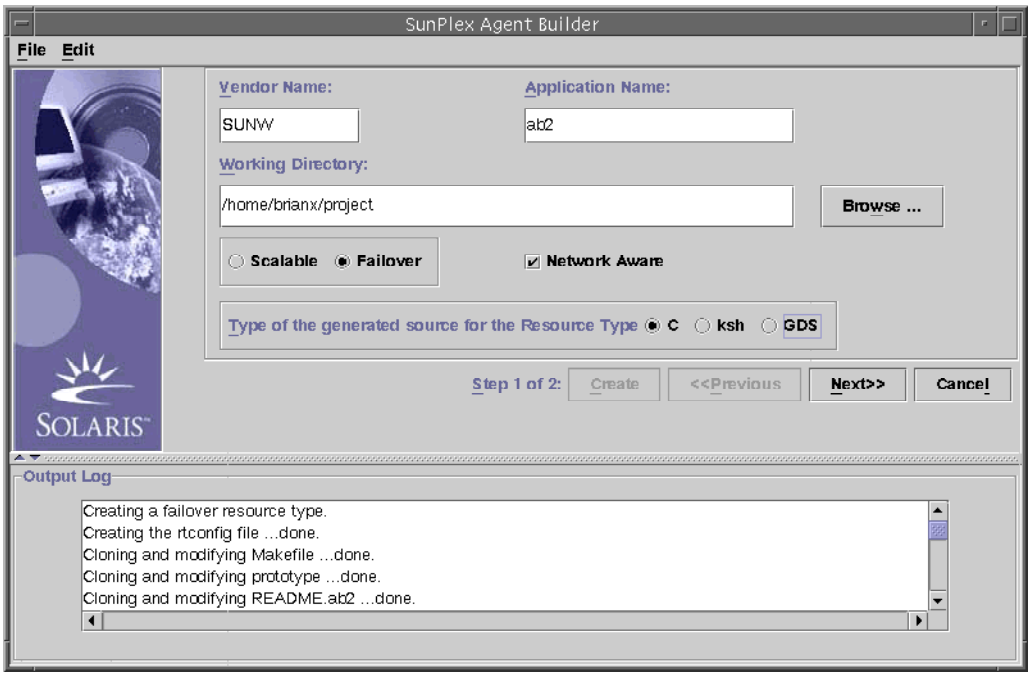


FIGURE 9–2 Écran Créer

L'écran Créer contient les champs, les cases d'option et les cases à cocher suivantes :

- **Nom du fournisseur :** nom permettant d'identifier le fournisseur du type de ressources. Si vous indiquez en règle générale le symbole de référencement du fournisseur, sachez toutefois que n'importe quel nom unique d'identification du fournisseur est valide. N'utilisez que des caractères alphanumériques.
- **Nom d'application :** nom du type de ressources. N'utilisez que des caractères alphanumériques.

Remarque – le nom du fournisseur et le nom d'application constituent le nom complet du type de ressources. Ce nom complet ne doit pas excéder neuf caractères.

- **Répertoire de travail :** répertoire dans lequel Agent Builder crée une structure de répertoire pour contenir tous les fichiers créés pour le type de ressources cible. Quel que soit le répertoire de travail, vous ne pouvez y créer qu'un seul type de ressources. Agent Builder initialise ce champ sur le chemin d'accès au répertoire de lancement d'Agent Builder. Vous pouvez cependant entrer un nom différent ou localiser un autre répertoire à l'aide du bouton **Parcourir**.

Agent Builder crée un sous-répertoire contenant le nom du type de ressources dans le répertoire de travail. Par exemple, si le nom du fournisseur et le nom d'application sont respectivement `SUNW` et `ftp`, Agent Builder appelle ce sous-répertoire `SUNWftp`.

Agent Builder insère tous les répertoires et fichiers du type de ressources cible dans ce répertoire (reportez-vous à la rubrique "Structure de répertoire" à la page 180).

- **Évolutif ou Basculement :** indique si le type de ressources cible est évolutif ou de basculement.
- **Compatible réseau :** indique si l'application de base est compatible ; c'est-à-dire si elle passe par le réseau pour communiquer avec ses clients. Cochez cette case pour indiquer que l'application est compatible réseau et ne la cochez pas pour indiquer qu'elle n'est pas prévue pour être utilisée en réseau. Le code korn shell nécessite que l'application soit compatible réseau. Par conséquent, Agent Builder sélectionne cette case et la grise si vous cochez le bouton **ksh** ou **GDS**.
- **C, ksh :** indique le langage du code source généré. Bien que ces options soient mutuellement exclusives, Agent Builder vous permet de créer un type de ressources avec un code généré `ksh`, puis de réutiliser les mêmes données pour créer un code généré C (reportez-vous à la rubrique "Clonage d'un type de ressources existant" à la page 178).
- **GDS :** indique que ce service est un service de données générique. Reportez-vous au Chapitre 10 pour de plus amples informations sur la création et la configuration d'un service de données générique.

Remarque – si le compilateur `cc` ne figure pas dans la variable `$PATH`, Agent Builder grise la case d'option C et sélectionne le bouton **ksh**. Pour spécifier un compilateur différent, reportez-vous à la remarque figurant à la fin de la rubrique "Installation et configuration d'Agent Builder" à la page 169.

Une fois les données requises entrées, cliquez sur le bouton **Créer**. Le Journal en bas de l'écran présente les actions qu'Agent Builder entreprend. La commande **Enregistrer journal** du menu Éditer vous permet d'enregistrer les données dans le journal.

Une fois cette opération terminée, Agent Builder affiche soit un message indiquant que cette étape a été réalisée avec succès soit un message d'avertissement précisant qu'il est impossible d'achever correctement cette tâche. Dans ce cas, vous devez consulter le journal pour obtenir plus d'informations.

Si Agent Builder termine cette tâche avec succès, vous pouvez cliquer sur le bouton **Suivant** pour afficher l'écran Configurer depuis lequel vous pouvez terminer de générer le type de ressources.

Remarque – bien que la génération d'un type de ressources complet s'exécute suivant une procédure en deux étapes, vous pouvez quitter Agent Builder une fois la première étape (création) terminée sans perdre les données que vous avez entrées ou le travail qu'Agent Builder a achevé (reportez-vous à la rubrique "Réutilisation d'un travail terminé" à la page 177).

Utilisation de l'écran Configurer

L'écran Configurer, illustré ci-après, apparaît une fois qu'Agent Builder a terminé de créer le type de ressources et que vous avez cliqué sur le bouton **Suivant** de l'écran Créer. Vous ne pouvez pas accéder à l'écran Configurer avant d'avoir créé le type de ressources.

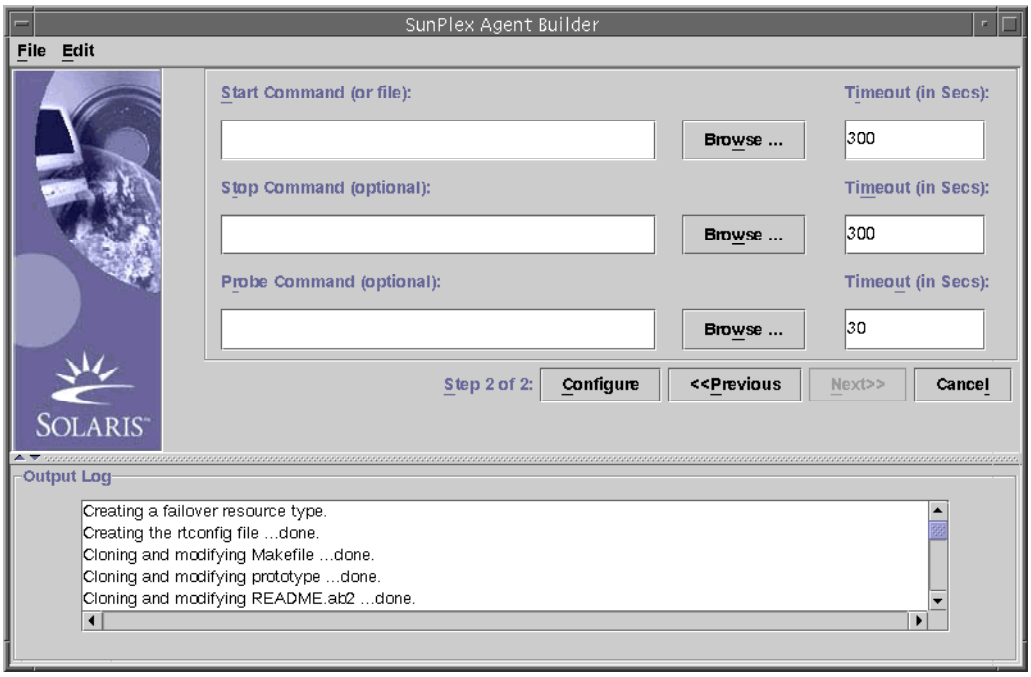


FIGURE 9–3 Écran Configurer

L'écran Configurer contient les champs suivants :

- **Commande de démarrage :** ligne de commande complète pouvant être transférée à n'importe quel shell UNIX pour lancer l'application de base. Nous vous recommandons de spécifier cette commande. Vous pouvez entrer cette commande dans le champ correspondant ou localiser le fichier contenant la commande de

démarrage de l'application à l'aide du bouton **Parcourir**.

La ligne de commande complète doit contenir toutes les informations requises pour démarrer l'application (noms d'hôte, numéros de port, chemin d'accès aux fichiers de configuration, etc.). Si votre application nécessite d'indiquer un nom d'hôte sur la ligne de commande, utilisez la variable `$hostnames` définie par Agent Builder (reportez-vous à la rubrique "Utilisation de la variable `$hostnames` d'Agent Builder" à la page 176).

N'utilisez pas de guillemets ("") avec la commande.

Remarque – si l'application de base comprend plusieurs arborescences de processus indépendantes et que chaque processus est exécuté à partir de sa propre balise sous la fonction PMF, vous ne pouvez pas indiquer une seule commande. Vous devez plus exactement créer un fichier texte contenant les commandes individuelles de démarrage de chaque arborescence de processus et indiquer le chemin d'accès à ce fichier dans le champ Commande de démarrage. Reportez-vous à la rubrique "Création de types de ressources contenant plusieurs arborescences de processus indépendantes" à la page 177. Vous y découvrirez quelques-unes des caractéristiques spécifiques que requiert ce fichier pour fonctionner correctement.

- **Commande d'arrêt :** ligne de commande complète pouvant être transférée à n'importe quel shell UNIX pour arrêter l'application de base. Vous pouvez entrer cette commande dans le champ correspondant ou localiser le fichier contenant la commande d'arrêt de l'application à l'aide du bouton Parcourir. Si votre application nécessite d'indiquer un nom d'hôte sur la ligne de commande, utilisez la variable `$hostnames` définie par Agent Builder (reportez-vous à la rubrique "Utilisation de la variable `$hostnames` d'Agent Builder" à la page 176).

  Cette commande est facultative. Si vous n'indiquez pas de commande d'arrêt, le code généré utilise les signaux (de la méthode `Arrêt`) pour arrêter l'application, comme suit :

  - La méthode `Arrêt` envoie le signal `SIGTERM` pour arrêter l'application et laisse s'écouler 80% du délai d'attente pour que l'application se ferme.
  - Si le signal `SIGTERM` échoue, la méthode `Arrêt` envoie le signal `SIGKILL` pour arrêter l'application et laisse s'écouler 15% du délai d'attente pour que l'application se ferme.
  - Si le signal `SIGKILL` échoue, la méthode `Arrêt` ne ferme pas l'application correctement (les 5% restants du délai d'attente sont considérés comme du temps supplémentaire).

Attention – veillez à ce que la commande d'arrêt ne soit pas renvoyée avant l'arrêt complet de l'application.

- **Commande de détection :** commande que vous pouvez exécuter périodiquement pour vérifier la maintenance de l'application et obtenir un état de sortie compris entre 0 (succès) et 100 (échec total). Cette commande est facultative. Vous pouvez entrer le chemin d'accès complet à cette commande ou localiser le fichier contenant la commande de détection de l'application à l'aide du bouton Parcourir.

  En règle générale, vous indiquez un client unique de l'application de base. Si vous n'indiquez pas de commande de détection, le code généré se contente de se connecter au port utilisé par la ressource puis de se déconnecter. Si cette opération réussit, l'application est déclarée comme étant parfaitement opérationnelle. L'utilisation d'une commande de détection n'est possible qu'avec les applications réseau. Agent Builder génère toujours une commande de détection mais désactive cette fonction avec les applications non prévues pour être utilisées en réseau.

  Si votre application nécessite d'indiquer un nom d'hôte sur la ligne de commande de détection, utilisez la variable `$hostnames` définie par Agent Builder (reportez-vous à la rubrique "Utilisation de la variable `$hostnames` d'Agent Builder" à la page 176).
- **Délai d'attente :** (par commande) : délai d'attente en secondes pour chaque commande. Vous pouvez indiquer une nouvelle valeur ou accepter la valeur par défaut fournie par Agent Builder (300 secondes pour le démarrage et l'arrêt et 30 secondes pour la détection).

Utilisation de la variable `$hostnames` d'Agent Builder

Pour de nombreuses applications, notamment les applications réseau, le nom d'hôte sur lequel l'application écoute et gère les requêtes des clients doit être transmis à l'application sur la ligne de commande. Par conséquent, le nom d'hôte est généralement un paramètre que vous devez indiquer pour les commandes de démarrage, d'arrêt et de détection du type de ressources cible (dans l'écran Configurer). Par contre, le nom d'hôte est propre au cluster. Il est déterminé à l'exécution de la ressource sur un cluster et ne peut pas l'être lorsqu'Agent Builder génère le code de votre type de ressources.

Pour pallier ce problème, Agent Builder propose la variable `$hostnames` que vous pouvez spécifier sur la ligne de commande pour les commandes de démarrage, d'arrêt et de détection. Vous spécifiez la variable `$hostnames` exactement comme vous indiqueriez un nom d'hôte réel. Exemple :

```
/opt/network_aware/echo_server -p n°_port -l $hostnames
```

Lorsqu'une ressource du type de ressources cible est exécutée sur un cluster, le nom d'hôte LogicalHostname (NomHôteLogique) ou SharedAddress (AdressePartagée) configuré pour cette ressource (dans la propriété de ressource `Ressources_réseau_utilisées` de cette ressource) est remplacé par la valeur de la variable `$hostnames`.

Si vous configurez la propriété `Ressources_réseau_utilisées` avec plusieurs noms d'hôte, ils apparaissent tous dans la variable `$hostnames`, séparés par des virgules.

Création de types de ressources contenant plusieurs arborescences de processus indépendantes

Agent Builder peut créer des types de ressources pour les applications intégrant plusieurs arborescences de processus indépendantes. Ces arborescences de processus sont indépendantes dans le sens où la fonction PMF les contrôle et les démarre individuellement. La fonction PMF démarre chaque arborescence de processus avec sa propre balise.

Remarque – Agent Builder ne vous permet de créer des types de ressources contenant plusieurs arborescences de processus indépendantes que si le code source généré que vous spécifiez est C. Vous ne pouvez pas utiliser Agent Builder pour créer ces types de ressources pour `ksh` ou GDS. Pour ce faire, vous devez écrire le code manuellement.

Dans le cadre d'une application de base contenant plusieurs arborescences de processus indépendantes, la spécification d'une ligne de commande unique ne suffit pas pour démarrer l'application. Dans ce cas, vous devez créer un fichier texte dont chaque ligne indique le chemin d'accès complet à une commande permettant de démarrer l'une des arborescences de processus de l'application. Ce fichier ne doit contenir aucune ligne vide et être spécifié dans le champ Commande de démarrage de l'écran Configurer.

Vous devez vous assurer qu'il ne contient aucune autorisation d'exécution, ce qui permet à Agent Builder de distinguer ce fichier visant à démarrer plusieurs arborescences de processus d'un simple script exécutable contenant plusieurs commandes. Si ce fichier texte contient des autorisations d'exécution, les ressources s'affichent sans problème ni erreur sur le cluster mais toutes les commandes sont exécutées sous une seule balise PMF. Le cas échéant, la fonction PMF ne peut pas contrôler et redémarrer individuellement les arborescences de processus.

Réutilisation d'un travail terminé

Agent Builder vous permet d'exploiter un travail terminé de plusieurs façons :

- Vous pouvez cloner un type de ressources existant créé avec Agent Builder.
- Vous pouvez éditer le code source généré par Agent Builder puis recompiler le code pour créer un nouveau package.

Clonage d’un type de ressources existant

Procédez comme suit pour cloner un type de ressources existant généré par Agent Builder :

1. **Chargez un type de ressources existant dans Agent Builder. Pour ce faire, vous pouvez procéder de deux façons :**

   a. **Lancez Agent Builder à partir du répertoire de travail (dans lequel figure le fichier `rtconfig`) d’un type de ressources existant (créé avec Agent Builder). Agent Builder charge ensuite les valeurs de ce type de ressources dans les écrans Créer et Configurer.**

   b. **Utilisez la commande Charger type de ressources du menu Fichier.**

2. **Modifiez le répertoire de travail dans l’écran Créer.**

   Pour ce faire, sélectionnez le répertoire avec le bouton **Parcourir**, l’entrée du nom d’un nouveau répertoire n’étant pas suffisante. Une fois le répertoire sélectionné, Agent Builder active de nouveau le bouton **Créer**.

3. **Apportez les modifications requises.**

   Cette procédure vous permet notamment de modifier le type de code généré pour le type de ressources. Par exemple, si vous avez commencé par créer une version `ksh` d’un type de ressources avant de découvrir qu’il vous faut une version C, vous pouvez charger le type de ressources `ksh` existant, remplacer le langage de sortie par C, puis utiliser Agent Builder pour concevoir une version C du type de ressources.

4. **Créez le clone du type de ressources.**

   Sélectionnez **Créer** pour créer le type de ressources. Sélectionnez **Suivant** pour passer à l’écran Configurer. Sélectionnez **Configurer** pour configurer le type de ressources, puis **Annuler** pour terminer la procédure.

Édition du code source généré

Pour que le processus de création d’un type de ressources reste simple, Agent Builder limite le nombre d’entrées, ce qui restreint la portée du type de ressources généré. Par conséquent, si vous souhaitez ajouter des fonctions plus complexes (des contrôles de validation de propriétés supplémentaires, par exemple) ou régler des paramètres qu’Agent Builder n’affiche pas, vous devez modifier le code source généré ou le fichier RTR.

Les fichiers sources sont disponibles dans le répertoire *répertoire_installation* `/`*nom_type_res*`/src`. Agent Builder intègre des commentaires dans le code source là où vous pouvez ajouter un code. Exemple (pour un code C) :

```
/* Code ajouté par l’utilisateur -- BEGIN vvvvvvvvvvvvvvv */
/* Code ajouté par l’utilisateur -- END   ^^^^^^^^^^^^^^^ */
```

Remarque – ces commentaires sont identiques dans le code korn shell à la seule différence que le symbole # est inséré au début d'une ligne de commentaire.

Par exemple, *nom_typ_res*.h déclare toutes les routines utilitaires que les différents programmes utilisent. À la fin de la liste des déclarations, vous trouverez des commentaires vous permettant de déclarer les autres routines que vous aimeriez ajouter à n'importe quel code source.

Agent Builder génère également un fichier makefile dans le répertoire *répertoire_installation*/*nom_type_res*/src avec les cibles appropriées. Les commandes make et make pkg vous permettent respectivement de recompiler le code source et de générer de nouveau le package du type de ressources.

Le fichier RTR figure dans le répertoire *répertoire_installation*/ *nom_type_res*/etc. Vous pouvez éditer le fichier RTR au moyen d'un éditeur de texte standard (reportez-vous à la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 34 pour de plus amples informations sur le fichier RTR et à l'Annexe A pour de plus amples informations sur les propriétés).

Utilisation de la version ligne de commande d'Agent Builder

La version ligne de commande d'Agent Builder s'appuie sur la même procédure en deux étapes que l'interface utilisateur graphique. Au lieu d'entrer les données dans l'interface utilisateur graphique, vous fournissez les paramètres aux commandes scdscreate(1HA) et scdsconfig(1HA).

Procédez comme suit pour utiliser la version ligne de commande d'Agent Builder :

1. **Utilisez la commande `scdscreate` pour créer un modèle de type de ressources Sun Cluster permettant de concevoir une application hautement disponible ou évolutive.**
2. **Utilisez la commande `scdsconfig` pour configurer le modèle de type de ressources créé avec `scdscreate`.**
3. **Remplacez les répertoires par le sous-répertoire `pkg` dans le répertoire de travail.**
4. **Utilisez la commande `pkgadd`(1M) pour installer les packages que vous avez créés avec `scdscreate`.**
5. **Éditez le code source généré, si vous le souhaitez.**
6. **Exécutez le script de démarrage.**

Structure de répertoire

Agent Builder crée une structure de répertoire pour stocker tous les fichiers qu'il crée pour le type de ressources cible. Vous spécifiez le répertoire de travail (dans l'écran **Créer**). Vous devez indiquer des répertoires d'installation distincts pour chaque type de ressources que vous développez. Dans le répertoire de travail, Agent Builder crée un sous-répertoire dont le nom est une concaténation du nom du fournisseur et du nom du type de ressources (depuis l'écran **Créer**). Par exemple, si vous spécifiez `SUNW` comme nom de fournisseur et que vous créez un type de ressources que vous nommez `ftp`, Agent Builder crée le répertoire `SUNWftp` dans le répertoire de travail.

Dans ce sous-répertoire, Agent Builder crée et remplit les répertoires présentés dans le tableau suivant :

Nom du répertoire	Contenu
`bin`	Spécifique à une sortie C. Contient les fichiers binaires compilés à partir des fichiers sources. Spécifique à une sortie `ksh`. Contient les mêmes fichiers que le répertoire `src`.
`etc`	Contient le fichier RTR. Agent Builder concatène le nom du fournisseur et le nom d'application, en les séparant d'un point (.), pour constituer le nom du fichier RTR. Par exemple, si le fournisseur et le type de ressources s'appellent respectivement `SUNW` et `ftp`, le nom du fichier RTR est `SUNW.ftp`.
`man`	Contient les pages de manuel des scripts d'utilitaire `start`, `stop` et `remove`. Par exemple, `startftp`(1M), `stopftp`(1M) et `removeftp`(1M). Pour consulter les pages de manuel, spécifiez le chemin d'accès avec l'option `man -M`. Par exemple, `man -M` *répertoire_installation*`/SUNWftp/man removeftp`.
`pkg`	Contient le package final.
`src`	Contient les fichiers sources générés par Agent Builder.
`util`	Contient les scripts d'utilitaire `start`, `stop` et `remove` générés par Agent Builder. Reportez-vous à la rubrique "Scripts d'utilitaire et pages de manuel" à la page 182. Agent Builder ajoute le nom d'application à chaque nom de script, par exemple, `startftp`, `stopftp`, `removeftp`.

Sortie

Cette rubrique décrit la sortie générée par Agent Builder.

Fichiers sources et binaires

Le gestionnaire RGM (Resource Group Manager) permet de gérer les groupes de ressources et, en définitive, les ressources sur un cluster. Il utilise pour ce faire un modèle de rappel. Lorsqu'un événement spécifique se produit, comme un noeud tombant en panne par exemple, le gestionnaire RGM appelle les méthodes du type de ressources de toutes les ressources exécutées sur le noeud défectueux. Par exemple, le gestionnaire RGM appelle la méthode `Arrêt` pour arrêter une ressource fonctionnant sur le noeud défectueux et la méthode `Démarrage` de cette ressource pour la lancer sur un autre noeud. (Reportez-vous aux rubriques "Modèle RGM" à la page 21 et "Méthodes de rappel" à la page 24 et à la page de manuel `rt_callbacks`(1HA) pour de plus amples informations sur ce modèle).

Pour prendre ce modèle en charge, Agent Builder génère (dans le répertoire *répertoire_installation*/*nom_type_res*/`bin`) huit programmes (C) ou scripts (`ksh`) exécutables servant de méthodes de rappel.

Remarque – le programme *nom_type_res*`_sonde` mettant en oeuvre un système de détection des pannes, n'est pas à proprement parler un programme de rappel. Le gestionnaire RGM n'appelle pas directement *nom_type_res*`_sonde` ; il appelle *nom_type_res*`_démarrage_détecteur` et *nom_type_res*`_arrêt_détecteur` qui exécute et arrête respectivement le système de détection des pannes en appelant *nom_type_res*`_sonde`.

Les huit méthodes générées par Agent Builder sont :

- *nom_type_res*`_contrôle_détecteur;`
- *nom_type_res*`_démarrage_détecteur;`
- *nom_type_res*`_arrêt_détecteur;`
- *nom_type_res*`_sonde;`
- *nom_type_res*`_démarrage_svc;`
- *nom_type_res*`_arrêt_svc;`
- *nom_type_res*`_mise_à_jour;`
- *nom_type_res*`_validation`.

Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour de plus amples informations sur toutes ces méthodes.

Agent Builder génère les fichiers suivants dans le répertoire *répertoire_installation*/*nom_type_res*/`src` (sortie C) :

- fichier d'en-tête (*nom_type_res*`.h`) ;
- fichier source (*nom_type_res*`.c`) contenant le code commun à toutes les méthodes ;
- fichier d'objets (*nom_type_res*`.o`) pour le code commun ;
- des fichiers sources (`*.c`) pour chacune des méthodes ;
- des fichiers d'objets (`*.o`) pour chacune des méthodes.

Agent Builder lie le fichier *nom_type_res*`.o` à chaque fichier `.o` de méthode pour créer les exécutables dans le répertoire *répertoire_installation*/ *nom_type_res*/`bin`.

Avec une sortie `ksh`, les répertoires *répertoire_installation* /*nom_tr*/`bin` et *répertoire_installation*/*nom_tr*/`src` sont identiques : ils contiennent les huit scripts exécutables correspondant aux sept méthodes de rappel, plus la méthode de `SONDE`.

Remarque – la sortie `ksh` comprend deux programmes utilitaires compilés (`gettime` et `gethostnames`) que requièrent certaines méthodes de rappel pour extraire l'heure ou effectuer une détection.

Vous pouvez éditer le code source, exécuter la commande `make` pour recompiler le code, puis exécuter la commande `make pkg` pour générer un nouveau package. Pour faciliter la modification du code source, Agent Builder ajoute des commentaires au code source là où il est pertinent d'ajouter du code. Reportez-vous à la rubrique "Édition du code source généré" à la page 178.

Scripts d'utilitaire et pages de manuel

Vous avez généré un type de ressources et installé le package correspondant sur un cluster. Vous devez encore exécuter une instance (ressource) du type de ressources sur un cluster à l'aide, en règle générale, de commandes administratives ou de SunPlex Manager. Pourtant, par souci de simplicité, Agent Builder génère à cet effet un script d'utilitaire personnalisé (le script de démarrage), ainsi que des scripts d'arrêt et de suppression d'une ressource du type de ressources cible. Ces trois scripts, disponibles dans le répertoire *répertoire_installation*/*nom_type_res*/`util`, ont les fonctions suivantes :

- **Script de démarrage** : enregistre le type de ressources et crée les ressources et les groupes de ressources nécessaires. Il crée également les ressources d'adresse réseau (LogicalHostname ou SharedAddress) qui permettent à l'application de communiquer avec les clients sur le réseau.
- **Script d'arrêt** : arrête et désactive la ressource.

- **Script de suppression** : annule le travail du script de démarrage. Il arrête et supprime du système les ressources, les groupes de ressources et le type de ressources cible.

Remarque – vous ne pouvez utiliser le script de suppression qu'avec une ressource exécutée à l'aide du script de démarrage car ces scripts utilisent des conventions internes pour nommer les ressources et les groupes de ressources.

Agent Builder attribue un nom à ces scripts en ajoutant le nom d'application aux noms de script. Par exemple, si le nom d'application est `ftp`, les scripts sont appelés `startftp`, `stopftp` et `removeftp`.

Agent Builder contient des pages de manuel dans le répertoire *répertoire_installation*`/`*nom_type_res*`/man/man1m` de chaque script d'utilitaire. Vous devez les lire avant de lancer ces scripts car elles présentent les paramètres qu'il vous faut fournir au script.

Pour consulter ces pages de manuel, indiquez le chemin d'accès au répertoire correspondant en utilisant l'option `-M` avec la commande `man`. Par exemple, si le nom du fournisseur est `SUNW` et le nom d'application `ftp`, utilisez la commande suivante pour afficher la page de manuel `startftp`(1M) :

```
man -M répertoire_installation/SUNWftp/man startftp
```

Les scripts d'utilitaire des pages de manuel sont également disponibles auprès de l'administrateur du cluster. Lorsqu'un package généré par Agent Builder est installé sur un cluster, les pages de manuel des scripts d'utilitaire sont placées dans le répertoire `/opt/` *nom_type_res*`/man`. Par exemple, utilisez la commande suivante pour afficher la page de manuel `startftp`(1M) :

```
man -M /opt/SUNWftp/man startftp
```

Fichiers de prise en charge

Agent Builder place les fichiers de prise en charge, comme `pkginfo`, `postinstall`, `postremove` et `preremove`, dans le répertoire *répertoire_installation*`/`*nom_type_res*`/etc`. Ce répertoire contient également le fichier RTR (resource type registration), qui déclare les propriétés de ressources et de type de ressources disponibles pour le type de ressources cible et initialise les valeurs de propriétés au moment de la mise en ligne d'une ressource sur un cluster (reportez-vous à la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 34 pour de plus amples informations). Le fichier RTR s'appelle *nom_fournisseur*`.`*nom_type_ressources* : par exemple `SUNW.ftp`.

Vous pouvez modifier ce fichier à l'aide d'un éditeur de texte standard et apporter des modifications sans recompiler votre code source. Vous devez, cependant, reconstituer le package avec la commande `make pkg`.

Répertoire du package

Le répertoire *répertoire_installation*/*nom_type_res*/`pkg` contient un package Solaris. Le nom du package est une concaténation du nom du fournisseur et du nom d'application, par exemple `SUNWftp`. Le fichier `Makefile` situé dans le répertoire *répertoire_installation*/*nom_type_res*/`src` prend en charge la création d'un nouveau package. Par exemple, si vous modifiez les fichiers sources et recompilez le code ou si vous modifiez les scripts d'utilitaire du package, vous devez utiliser la commande `make pkg` pour créer un nouveau package.

La commande `pkgrm` peut échouer lors de la suppression d'un package d'un cluster si vous tentez de l'exécuter sur plusieurs noeuds simultanément. Vous pouvez pallier ce problème de l'une des deux façons suivantes :

- Exécutez le script `remove`*nom_type_res* sur l'un des noeuds du cluster avant d'exécuter `pkgrm` sur un autre noeud.
- Exécutez `pkgrm` sur l'un des noeuds du cluster (cette commande veille à éliminer toutes les données superflues), puis exécutez-la sur les noeuds restants, simultanément au besoin.

Si la commande `pkgrm` échoue car vous tentez de l'exécuter simultanément sur plusieurs noeuds, lancez-la de nouveau noeud par noeud, puis sur les noeuds restants.

Fichier `rtconfig`

Si vous générez un code source C ou `ksh`, dans le répertoire de travail, Agent Builder génère un fichier de configuration `rtconfig` contenant les données que vous avez entrées dans les écrans Créer et Configurer. Si vous lancez Agent Builder depuis le répertoire de travail d'un type de ressources existant (ou que vous chargez un type de ressources existant à l'aide de la commande **Charger type de ressources** du menu Fichier), Agent Builder lit le contenu du fichier `rtconfig` et renseigne les champs des écrans Créer et Configurer à l'aide des données que vous avez fournies pour le type de ressources existant. Cette fonction s'avère pratique si vous souhaitez cloner le type de ressources existant (reportez-vous à la rubrique "Clonage d'un type de ressources existant" à la page 178.

Navigation dans Agent Builder

La navigation dans Agent Builder est simple et intuitive. Agent Builder est un assistant en deux étapes constitué d'un écran par étape (les écrans Créer et Configurer). Vous renseignez les champs de chaque écran au moyen des actions suivantes :

- Entrée des données dans un champ.
- Parcours de la structure de répertoires et sélection d'un fichier ou d'un répertoire.
- Sélection d'un ensemble de cases d'option mutuellement exclusives, par exemple, **Évolutif** ou **Basculement**.
- Sélection d'une case d'activation/désactivation. Par exemple, si vous sélectionnez **Compatible réseau**, l'application de base est identifiée comme une application réseau tandis que si cette case n'est pas sélectionnée, l'application est identifiée comme une application non prévue pour être utilisée en réseau.

Les boutons en bas de chaque écran vous permettent de terminer la tâche, de passer à l'écran suivant ou précédent ou de quitter Agent Builder. Agent Builder active ou désactive ces boutons comme il se doit.

Par exemple, après avoir renseigné les champs et coché les options souhaitées dans l'écran Créer, cliquez sur le bouton **Créer** en bas de l'écran. Les boutons **Précédent** et **Suivant** sont désactivés car il n'y a pas d'écran précédent et vous ne pouvez pas accéder à l'écran suivant tant que vous n'avez pas terminé de renseigner l'écran actuel.

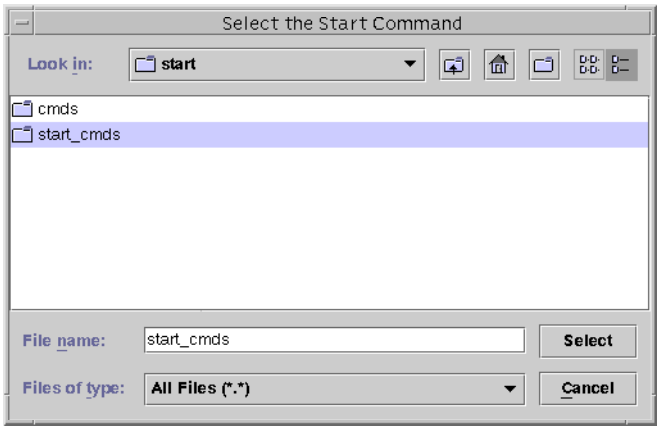
Agent Builder affiche des messages de progression dans la zone du journal située en bas de l'écran. Lorsqu'il a terminé, il affiche un message indiquant que l'opération a réussi ou un avertissement vous invitant à consulter le journal. Le bouton **Suivant** est mis en surbrillance. Si vous êtes dans le dernier écran, seul le bouton **Annuler** est mis en surbrillance.

Vous pouvez sélectionner **Annuler** à tout moment pour quitter Agent Builder.

Bouton Parcourir

Vous pouvez renseigner les champs d'Agent Builder en entrant des données ou en cliquant sur le bouton **Parcourir** pour parcourir votre structure de répertoire et sélectionner un fichier ou un répertoire.

Lorsque vous cliquez sur **Parcourir**, un écran similaire à l'écran suivant apparaît :



Cliquez deux fois sur un dossier pour l'ouvrir. Lorsque vous mettez un fichier en surbrillance, son nom apparaît dans le champ **Nom de fichier**. Cliquez sur **Sélectionner** lorsque vous avez localisé et mis en surbrillance le fichier approprié.

Remarque – si vous recherchez un répertoire, mettez-le en surbrillance et sélectionnez le bouton **Ouvrir**. S'il ne contient aucun sous-répertoire, Agent Builder ferme la fenêtre Parcourir et insère le nom du répertoire mis en surbrillance dans le champ approprié. S'il contient des répertoires, cliquez sur le bouton Fermer pour fermer la fenêtre et revenir à l'écran précédent. Agent Builder insère le nom du répertoire mis en surbrillance dans le champ approprié.

Fonction des icônes situées dans le coin supérieur droit de l'écran :

Cette icône permet de monter d'un niveau dans la structure de répertoire.

Cette icône permet de revenir au dossier personnel.

Cette icône crée un nouveau dossier sous le dossier actuellement sélectionné.

Cette icône permet de basculer entre différents affichages. Elle est dédiée à une utilisation ultérieure.

Menus

Agent Builder contient les menus Fichier et Édition.

Menu Fichier

Le menu Fichier comprend deux commandes :

- **Charger type de ressources :** ce menu permet de charger un type de ressources existant. Dans Agent Builder, un écran de recherche vous permet de sélectionner le répertoire de travail d'un type de ressources existant. Si le répertoire à partir duquel vous avez lancé Agent Builder contient un type de ressources, Agent Builder charge automatiquement ce type de ressources. La commande **Charger type de ressources** permet de lancer Agent Builder depuis n'importe quel répertoire et de sélectionner un type de ressources existant à utiliser comme modèle pour créer un nouveau type de ressources (reportez-vous à la rubrique "Clonage d'un type de ressources existant" à la page 178).
- **Quitter :** permet de quitter Agent Builder. Vous pouvez également quitter Agent Builder en cliquant sur **Annuler** dans l'écran Créer ou Configurer.

Menu Éditer

Le menu Éditer comprend des commandes permettant d'effacer et d'enregistrer le journal :

- **Effacer journal :** efface les données contenues dans le journal. Chaque fois que vous sélectionnez Créer ou Configurer, Agent Builder ajoute des messages d'état dans le journal. Lors d'un processus répétitif de modification de votre code source nécessitant de générer de nouveau la sortie dans Agent Builder, vous pouvez enregistrer et effacer le fichier journal avant chaque utilisation si vous souhaitez isoler les messages d'état.
- **Enregistrer fichier journal :** enregistre le journal dans un fichier. Dans Agent Builder, un écran de recherche vous permet de sélectionner le répertoire et d'entrer un nom de fichier.

Module Cluster Agent pour Agent Builder

Le module Cluster Agent pour Agent Builder est un module NetBeans™ permettant aux utilisateurs de Sun Java Studio (précédemment Sun ONE Studio) de créer des types de ressources ou des services de données pour le logiciel Sun Cluster à l'aide d'un environnement de développement intégré. Ce module se caractérise par une interface écran permettant de décrire chaque sorte de type de ressources que vous souhaitez créer.

Remarque – la documentation de Sun Java Studio présente les procédures de configuration, d'installation et d'utilisation de ce logiciel.

▼ Installation et configuration du module Cluster Agent

Le module Cluster Agent est installé en même temps que le logiciel Sun Cluster. L'outil d'installation de Sun Cluster enregistre le fichier du module Cluster Agent `scdsbuilder.jar` sous `/usr/cluster/lib/scdsbuilder`. Pour utiliser le module Cluster Agent avec le logiciel Sun Java Studio, vous devez créer un lien symbolique vers ce fichier.

Remarque – les logiciels Sun Cluster et Sun Java Studio et Java ™ 1.4 doivent être installés et disponibles sur le système sur lequel vous envisagez d'exécuter le module Cluster Agent.

---

1. **Configuration de l'accès multi-utilisateur ou mono-utilisateur au module Cluster Agent**

   - Pour permettre à tous les utilisateurs d'accéder au module Cluster Agent, connectez-vous en tant que superutilisateur ou équivalent et créez un lien symbolique dans le répertoire global du module :

     ```
     # cd /opt/s1studio/ee/modules
     # ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

Remarque – si vous installez le logiciel Sun Java Studio dans un autre répertoire que `/opt/s1studio/ee`, remplacez ce chemin d'accès par le chemin d'accès approprié.

- Pour être le seul à pouvoir accéder au module, créez le lien symbolique dans votre sous-répertoire `modules` :

```
% cd ~votre_rép_personnel/ffjuser40ee/modules
% ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

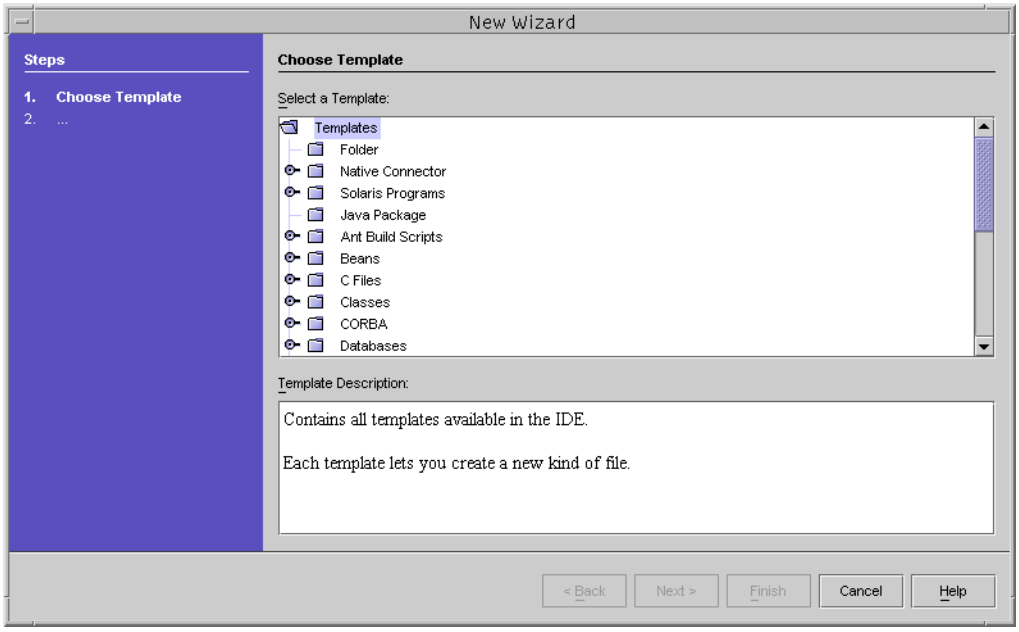
2. **Fermez et relancez le logiciel Sun Java Studio.**

▼ Démarrage du module Cluster Agent

La procédure suivante décrit comment démarrer le module Cluster Agent à partir du logiciel Sun Java Studio.

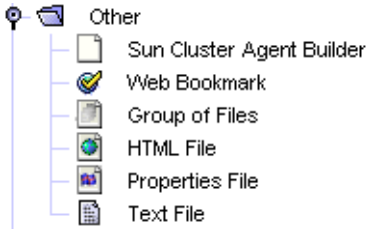
1. **Sélectionnez Nouveau dans le menu Fichier de Sun Java Studio ou cliquez sur l'icône suivante dans la barre d'outils :**

L'écran Assistant Nouveau apparaît.



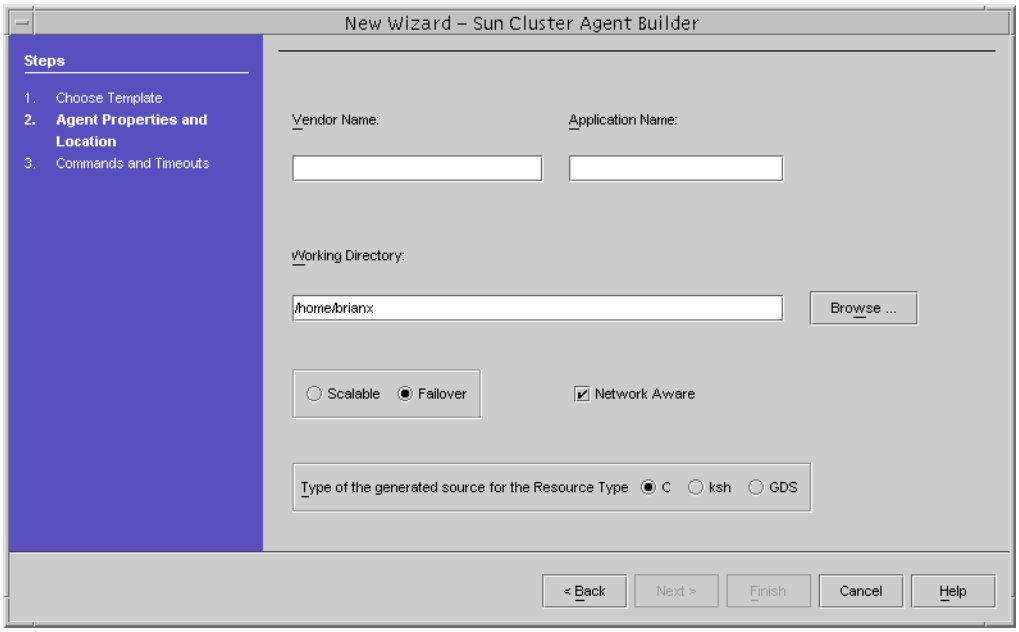
2. **Faites défiler la liste (si nécessaire) dans la fenêtre Sélectionner un modèle, puis cliquez sur la clé en regard du dossier Autre :**

Le menu Autre s'ouvre.



3. **Dans le menu Autre, sélectionnez Sun Cluster Agent Builder, puis cliquez sur Suivant.**

   Le module Cluster Agent pour Sun Java Studio démarre. Le premier écran Assistant Nouveau - Sun Cluster Agent Builder apparaît.



Utilisation du module Cluster Agent

Utilisez le module Cluster Agent de la même façon que le logiciel Agent Builder. Leurs interfaces sont identiques. Par exemple, les illustrations suivantes montrent que l'écran Créer du logiciel Agent Builder et le premier écran Assistant Nouveau - Sun Cluster Agent Builder du module Cluster Agent contiennent les mêmes champs et sélections.

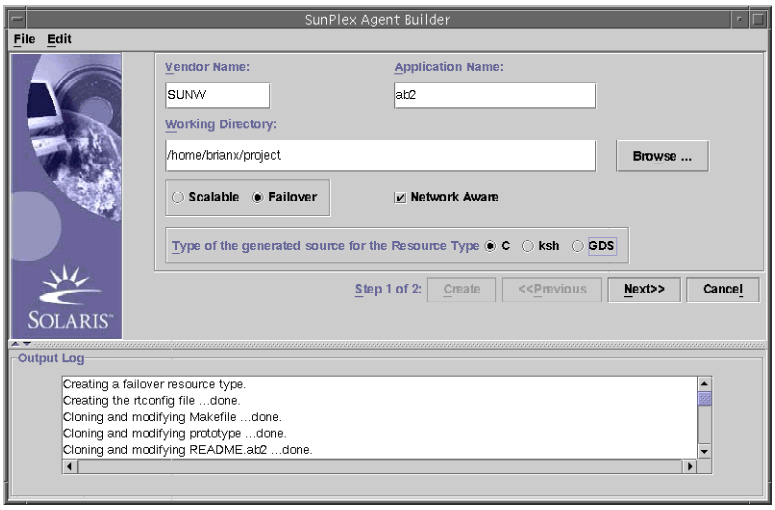


FIGURE 9–4 Écran Créer du logiciel Agent Builder

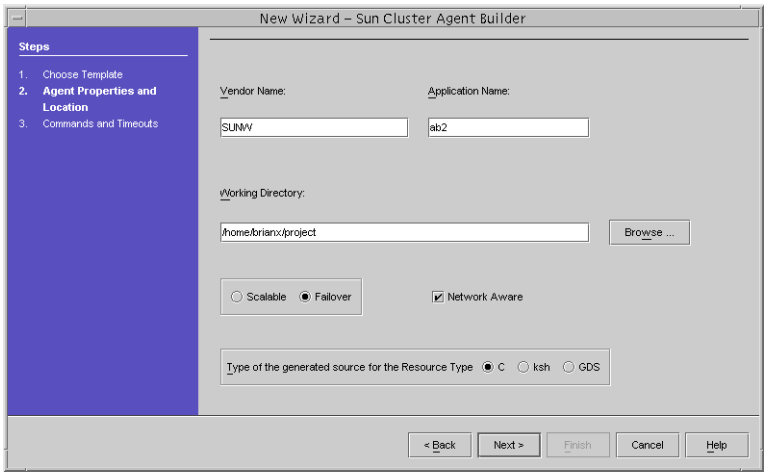


FIGURE 9–5 Écran Assistant Nouveau - Sun Cluster Agent Builder du module Cluster Agent

Différences entre le module Cluster Agent et Agent Builder

Malgré la grande similitude entre le module Cluster Agent et Agent Builder, il existe quelques petites différences :

- Dans le module Cluster Agent, le type de ressources n'est créé et configuré qu'après avoir cliqué sur Terminer dans le second Assistant Nouveau - Sun Cluster Agent Builder. Le type de ressources *n'*est *pas* créé lorsque vous cliquez sur Suivant dans le premier écran Assistant Nouveau - Sun Cluster Agent Builder.

  Dans Agent Builder, le type de ressources est créé immédiatement lorsque vous cliquez sur Créer dans l'écran Créer. Il est de même configuré immédiatement lorsque vous cliquez sur Configurer dans l'écran Configurer.
- Les données figurant dans la fenêtre Journal d'Agent Builder apparaissent dans une fenêtre de résultat dans le logiciel Sun Java Studio.

CHAPITRE 10

Services de données génériques

Ce chapitre présente le module de service de données générique (GDS) et décrit la procédure de création d'un service basé sur GDS, à l'aide de SunPlex Agent Builder ou des commandes standard d'administration de Sun Cluster.

Présentation générale des services de données génériques

Un service de données générique (GDS) est un mécanisme permettant de rendre hautement disponibles et évolutives des applications uniques et compatibles réseau en les connectant à la structure du gestionnaire RGM de Sun Cluster. Ce mécanisme évite l'utilisation d'un agent de programmation, procédure habituelle pour rendre une application hautement disponible ou évolutive.

Le module GDS est un service de données unique, précompilé. Vous ne pouvez pas modifier un service de données précompilé et ses composants, les mises en oeuvre de la méthode de rappel (`rt_callbacks`(1HA)) et le fichier d'enregistrement du type de ressources (`rt_reg`(4)).

Type de ressources précompilé

Le type de ressources du service de données générique `SUNW.gds` est inclus dans le package `SUNWscgds`. L'utilitaire `scinstall`(1M) installe ce package en même tant que le cluster. Le package `SUNWscgds` comprend les fichiers suivants :

```
# pkgchk -v SUNWscgds

/opt/SUNWscgds
/opt/SUNWscgds/bin
/opt/SUNWscgds/bin/gds_monitor_check
/opt/SUNWscgds/bin/gds_monitor_start
/opt/SUNWscgds/bin/gds_monitor_stop
/opt/SUNWscgds/bin/gds_probe
/opt/SUNWscgds/bin/gds_svc_start
/opt/SUNWscgds/bin/gds_svc_stop
/opt/SUNWscgds/bin/gds_update
/opt/SUNWscgds/bin/gds_validate
/opt/SUNWscgds/etc
/opt/SUNWscgds/etc/SUNW.gds
```

Pourquoi utiliser le module GDS

Le module GDS offre les avantages suivants par rapport à l'utilisation du modèle de code source généré par SunPlex Agent Builder (reportez-vous à `scdscreate`(1HA)) ou les commandes standard d'administration de Sun Cluster :

- le module GDS est simple à utiliser ;
- le module GDS et ses méthodes sont précompilés et ne sont donc pas modifiables ;
- SunPlex Agent Builder peut servir à générer des scripts moteurs pour votre application intégrés dans un package Solaris et réutilisable sur plusieurs clusters.

Processus de création d'un service utilisant le module GDS

Vous pouvez créer un service utilisant le module GDS de deux façons :

- à l'aide de SunPlex Agent Builder ;
- à l'aide des commandes standard d'administration de Sun Cluster.

Module GDS et SunPlex Agent Builder

Utilisez SunPlex Agent Builder et sélectionnez GDS comme type de code source généré. L'entrée de l'utilisateur sert à générer un ensemble de scripts moteurs qui configurent les ressources de l'application en question.

Module GDS et commandes standard d'administration de Sun Cluster

Cette méthode utilise le code de service de données précompilé dans `SUNWscgds` mais impose à l'administrateur système d'utiliser les commandes standard d'administration de Sun Cluster (`scrgadm`(1M) et `scswitch`(1M)) pour créer et configurer la ressource.

Sélection d'une méthode de création d'un service basé sur GDS

Comme l'indiquent les procédures "Utilisation des commandes d'administration de Sun Cluster permettant de créer un service à haut niveau de disponibilité basé sur le module GDS" à la page 206 et "Commandes d'administration standard de Sun Cluster permettant de créer un service évolutif basé sur le module GDS" à la page 207, l'exécution des commandes `scrgadm` et `scswitch` nécessite un effort conséquent de saisie.

L'utilisation du module GDS avec SunPlex Agent Builder simplifie le processus car ce module génère des scripts moteurs exécutant les commandes `scrgadm` et `scswitch` à votre place.

Cas dans lesquels le module GDS n'est pas approprié

Bien que le module GDS offre de nombreux avantages, son utilisation n'est pas toujours appropriée. Le module GDS *ne doit pas* être utilisé lorsque :

- Le type de ressources précompilé n'offre pas un contrôle suffisant (lorsque vous devez ajouter des propriétés d'extension ou modifier des valeurs par défaut par exemple).
- Le code source doit être modifié pour y ajouter des fonctions spéciales.
- Vous souhaitez utiliser plusieurs arborescences de processus.
- Vous souhaitez utiliser des applications non prévues pour être utilisées en réseau.

Propriétés requises pour le module GDS

Les propriétés suivantes sont indispensables :

- `Commande_démarrage` (propriété d'extension) ;
- `Liste_ports`.

Propriété d'extension `Commande_démarrage`

La commande de démarrage que vous spécifiez dans la propriété d'extension `Commande_démarrage` lance l'application. Cette commande UNIX et ses arguments peuvent être passés directement à un shell pour démarrer l'application.

Propriété `Liste_ports`

La propriété `Liste_ports` identifie la liste des ports que l'application écoute. La propriété `Liste_ports` doit figurer dans le script de démarrage créé par SunPlex Agent Builder ou dans la commande `scrgadm` si vous utilisez les commandes standard d'administration de Sun Cluster.

Propriétés facultatives pour le module GDS

Propriétés GDS facultatives :

- `Ressources_réseau_utilisées`
- `Commande_arrêt` (propriété d'extension) ;
- `Commande_sonde` (propriété d'extension) ;
- `Délai_démarrage` ;
- `Délai_arrêt` ;
- `Délai_sonde` (propriété d'extension) ;
- `Niveau_cont_fils` (propriété d'extension utilisée uniquement avec les commandes standard d'administration) ;
- `Basculement_activité` (propriété d'extension) ;
- `Signal_arrêt` (propriété d'extension).

Propriété `Ressources_réseau_utilisées`

La valeur par défaut de cette propriété est Null. Vous devez impérativement spécifier cette propriété dès lors que l'application doit être liée à une ou plusieurs adresses spécifiques. Si cette propriété est omise ou si elle est spécifiée comme Null, on considère que l'application écoute sur toutes les adresses.

Avant de créer une ressource GDS, la ressource `LogicalHostName` ou `SharedAddress` doit également avoir été configurée. Reportez-vous au document *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour de plus amples informations sur la procédure de configuration d'une ressource `LogicalHostname` ou `SharedAddress`.

Pour spécifier une valeur, spécifiez le nom d'une ou plusieurs ressources ; chaque nom pouvant contenir un ou plusieurs `LogicalHostname` ou une ou plusieurs `SharedAddress`. Reportez-vous à `r_properties`(5) pour de plus amples informations.

Propriété `Commande_arrêt`

La commande d'arrêt doit arrêter l'application et n'être retournée qu'une fois l'application totalement arrêtée. Il doit impérativement s'agir d'une commande UNIX complète qui peut être transmise directement à un shell pour arrêter l'application.

En présence de la `Commande_arrêt`, la méthode d'arrêt du module GDS lance la commande d'arrêt avec 80 % du délai imparti à l'arrêt. Quel que soit le résultat de la commande d'arrêt, la méthode d'arrêt du module GDS envoie la commande `SIGKILL` avec 15% du délai imparti à l'arrêt. Les 5% restants sont réservés au temps système de gestion interne.

Si la commande d'arrêt est omise, le module GDS tente d'arrêter l'application en utilisant le signal spécifié dans `Signal_arrêt`.

Propriété `Commande_sonde`

La commande de détection contrôle périodiquement l'état de l'application en question. Cette commande UNIX et ses arguments peuvent être transmis directement à un shell pour sonder l'application. La commande de détection renvoie un statut de sortie égal à 0 si l'application est en bon état.

L'état de sortie de la commande de détection permet de déterminer le degré de gravité de la panne qui touche l'application. Cet état de sortie, appelé statut de détection, doit être un nombre entier compris entre 0 (réussite) et 100 (panne intégrale). Le statut de détection peut également correspondre à une valeur spéciale de 201 qui entraîne un basculement immédiat de l'application sauf si `Basculement_activé` est défini sur False. L'algorithme de détection du module GDS (reportez-vous à `scds_fm_action`(3HA) 3HA)) se base sur le statut de détection pour décider de redémarrer l'application en local ou de la basculer sur un autre noeud ; si le statut de sortie est 201, le basculement de l'application est immédiat.

Si la commande de détection est omise, le module GDS effectue sa propre détection et se connecte à l'application sur l'ensemble des adresses IP dérivées de la propriété `Ressources_réseau_utilisées` ou de la sortie de `scds_get_netaddr_list`(3HA). Si la connexion réussit, le module GDS se déconnecte immédiatement. Si la connexion et la déconnexion réussissent, on considère que l'application fonctionne correctement.

Remarque – la détection effectuée par le module GDS n'est qu'un simple substitut à la détection complète de l'application.

Propriété `Délai_démarrage`

Cette propriété spécifie le délai d'attente au démarrage de la commande de démarrage (reportez-vous à la rubrique "Propriété d'extension `Commande_démarrage`" à la page 198 pour de plus amples informations). La valeur par défaut de `Délai_démarrage` est de 300 secondes.

Propriété `Délai_arrêt`

Cette propriété spécifie le délai d'attente à l'arrêt de la commande d'arrêt (reportez-vous à la rubrique "Propriété `Commande_arrêt`" à la page 199 pour de plus amples informations. La valeur par défaut de `Délai_arrêt` est de 300 secondes.

Propriété `Délai_sonde`

Cette propriété spécifie le délai d'attente de la commande de détection (reportez-vous à la rubrique "Propriété `Commande_sonde`" à la page 199 pour de plus amples informations). La valeur par défaut de `Délai_sonde` est de 30 secondes.

Propriété `Niveau_cont_fils`

Cette propriété permet de contrôler les processus gérés par la fonction PMF. Elle indique le niveau auquel les processus fils sont surveillés. Cette propriété est similaire à l'argument `-C` de la commande `pmfadm`(1M).

Omettre cette propriété, ou lui conférer la valeur par défaut de `-1`, revient à omettre l'option `-C` de la commande `pmfadm` ; c'est-à-dire que tous les fils (et leurs descendants) seront surveillés.

Remarque – cette option peut être spécifiée à l'aide des commandes d'administration standard de Sun Cluster. Vous ne pouvez pas spécifier cette option si vous utilisez SunPlex Agent Builder.

Propriété Basculement_activé

Cette propriété booléenne d'extension contrôle le comportement de basculement de la ressource. Si cette propriété d'extension est définie sur `vrai`, l'application est basculée dès lors que le nombre de redémarrages dépasse la valeur `nombre_nouvelles_tentatives` au cours du délai en secondes `intervalle_nouvelles_tentatives`.

Si cette propriété d'extension est définie sur `faux`, l'application ne redémarre pas et ne bascule pas sur un autre noeud lorsque le nombre de redémarrages dépasse la valeur `nombre_nouvelles_tentatives` au cours du délai en secondes `intervalle_nouvelles_tentatives`.

Cette propriété d'extension peut être utilisée pour empêcher la ressource d'application de basculer un groupe de ressources. La valeur par défaut est `vrai`.

Propriété Signal_arrêt

Le module GDS utilise la valeur de cette propriété d'extension entière pour déterminer le signal utilisé pour arrêter l'application au moyen de la fonction PMF. Reportez-vous à `signal`(3HEAD) pour connaître la liste des nombres entiers qu'il est possible de spécifier comme valeur. La valeur par défaut est 15 (`SIGTERM`).

Utilisation de SunPlex Agent Builder pour créer un service basé sur le module GDS

SunPlex Agent Builder vous permet de créer le service utilisant le module GDS. SunPlex Agent Builder est présenté plus en détail dans le Chapitre 9.

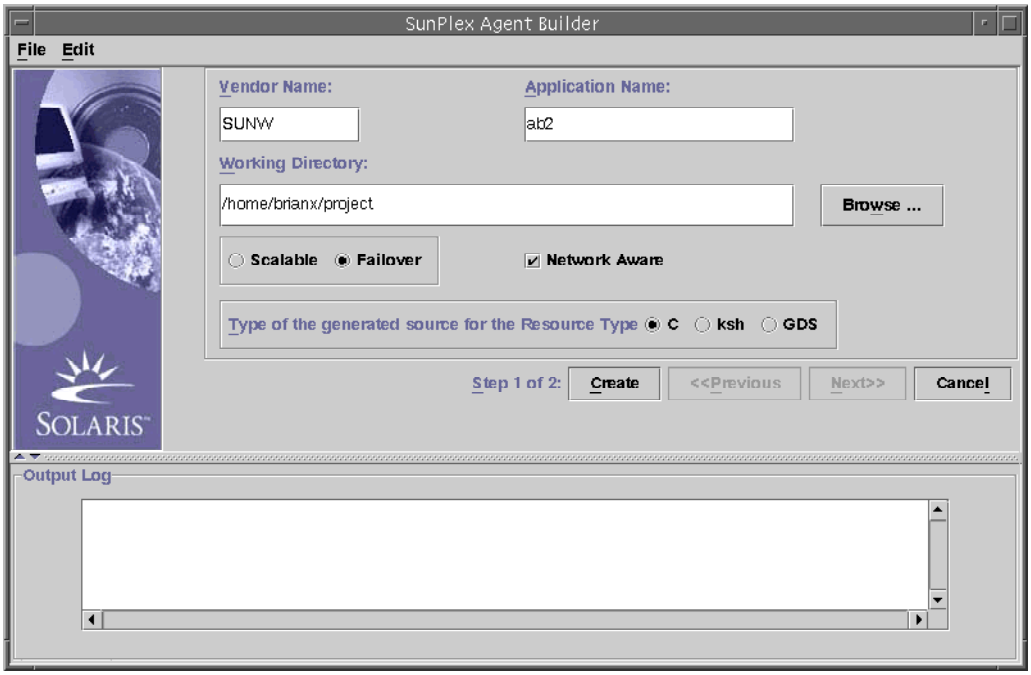
Création d'un service dans SunPlex Agent Builder à l'aide du module GDS

▼ Création d'un service basé sur le module GDS dans Agent Builder

1. **Lancez SunPlex Agent Builder.**

   ```
   # /usr/cluster/bin/scdsbuilder
   ```

2. **Le panneau SunPlex Agent Builder apparaît.**



3. **Renseignez le champ Nom du fournisseur.**

4. **Renseignez le champ Nom de l'application.**

Remarque – la combinaison du nom du fournisseur et de celui de l'application ne doit pas excéder neuf caractères. Cette combinaison sert de nom au package des scripts moteurs.

---

5. **Accédez au répertoire de travail.**

   Vous pouvez utiliser le bouton Parcourir pour sélectionner le répertoire de votre choix plutôt que d'entrer son chemin d'accès.

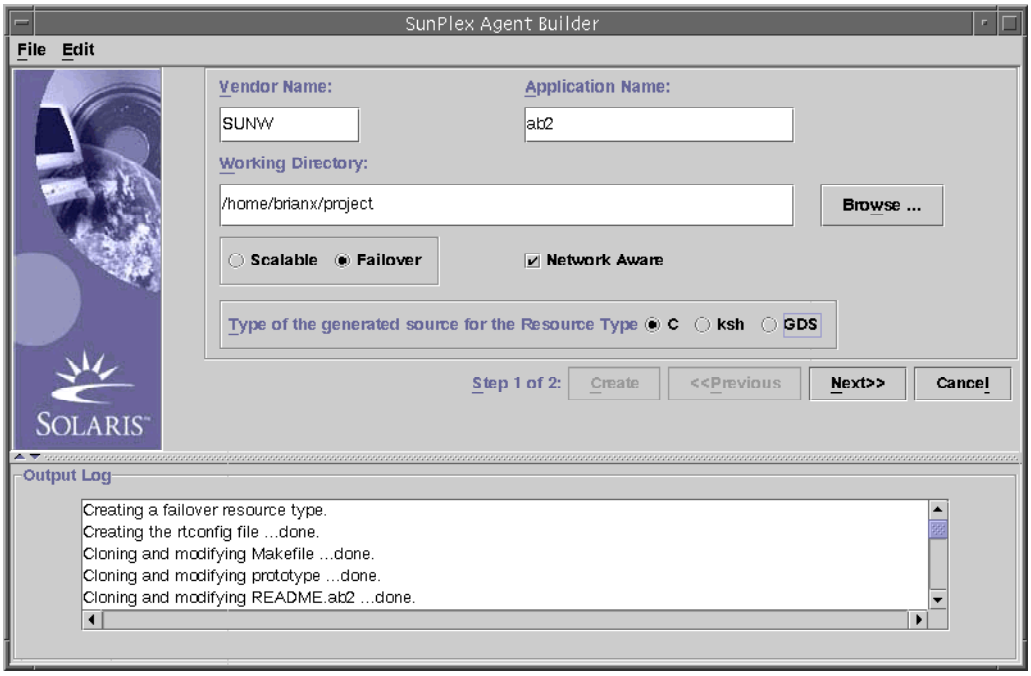
6. **Indiquez si le service de données est évolutif ou de basculement.**

   Vous n'êtes pas tenu de sélectionner l'option Compatible réseau, celle-ci étant celle attribuée par défaut lors de la création du module GDS.

7. **Sélectionnez GDS.**

8. **Cliquez sur le bouton Créer pour créer les scripts moteurs.**

9. **Le panneau SunPlex Agent Builder affiche les résultats de la création du service. Le bouton Créer est grisé. Vous pouvez à présent utiliser le bouton Suivant.**



▼ Configuration des scripts moteurs

Une fois que vous avez créé les scripts moteurs, vous devez utiliser SunPlex Agent Builder pour configurer le nouveau service.

1. **Cliquez sur le bouton Suivant pour afficher le panneau de configuration.**
2. **Entrez le chemin d'accès à la commande de démarrage ou utilisez le bouton Parcourir pour sélectionner la commande de démarrage.**
3. **(Facultatif) Entrez la commande d'arrêt ou utilisez le bouton Parcourir pour sélectionner la commande d'arrêt.**
4. **(Facultatif) Entrez la commande de détection ou utilisez le bouton Parcourir pour sélectionner la commande de détection.**
5. **(Facultatif) Spécifiez les valeurs de délai d'attente des commandes d'arrêt, de démarrage et de détection.**
6. **Cliquez sur Configurer pour démarrer la configuration des scripts moteurs.**

Remarque – le nom du package se compose d'une concaténation du nom du fournisseur et de celui de l'application.

Un package de scripts moteurs est ainsi créé et placé dans :

<rép_travail>/*<nom_fournisseur>**<application>*/pkg
Par exemple, /export/wdir/NETapp/pkg

7. Installez le package complet sur tous les noeuds du cluster.

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

8. Les fichiers suivants sont installés au cours de pkgadd :

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Remarque – les pages de manuel et les noms des scripts correspondront au nom de l'application que vous avez entré ci-dessus, précédé du nom du script ; par exemple, startapp.

Pour afficher les pages de manuel, vous devez spécifier leur chemin d'accès. Par exemple, pour afficher les pages de manuel startapp, utilisez :

```
# man -M /opt/NETapp/man startapp
```

9. Sur l'un des noeuds du cluster, configurez les ressources et démarrez l'application.

/opt/NETapp/util/startapp -h *<nom_hôte_logique>* **-p** *<liste_ports_et_protocoles>*

Les arguments du script de démarrage varient en fonction du type de ressources : de basculement ou évolutives. Vérifiez la page de manuel personnalisée ou exécutez le script de démarrage sans argument pour obtenir une déclaration d'utilisation.

```
# /opt/NETapp/util/startapp
Vous devez spécifier le nom de ressource de LogicalHostname
```

```
ou de SharedAddress.
Pour des services de basculement :
Utilisation : startapp -h <nom_hôte_logique>
-p <liste_ports_et_protocoles>
[-n <liste_groupes_ipmp/adaptateurs>] Pour des services évolutifs :
Utilisation : startapp
-h <nom_adresse_partagée>
-p <liste_ports_et_protocoles>
[-l <règle_équilibrage_charge>]
[-n <liste_groupes_ipmp/adaptateurs>] [-w <poids_équilibrage_charge>]
```

Sortie de SunPlex Agent Builder

SunPlex Agent Builder génère trois scripts de commande et un fichier de configuration en fonction des données que vous saisissez à la création du package. Le fichier de configuration spécifie les noms du groupe de ressources et du type de ressources.

Les scripts moteurs sont :

- Le script de démarrage : il sert à configurer les ressources et à démarrer l'application sous le contrôle du gestionnaire du groupe de ressources (RGM).
- Le script d'arrêt : il sert à arrêter l'application et à extraire les ressources et les groupes de ressources.
- Le script de suppression : il sert à supprimer les ressources et les groupes de ressources créés par le script de démarrage.

Ces scripts moteurs ont la même interface et se comportent de la même façon que les scripts utilitaires générés par SunPlex Agent Builder pour les agents qui ne sont pas basés sur GDS. Les scripts sont intégrés dans un package Solaris réutilisable sur plusieurs clusters.

Vous pouvez personnaliser le fichier de configuration pour y indiquer les noms de vos groupes de ressources ou d'autres paramètres qui correspondent généralement à des entrées de la commande `scrgadm`. Si vous ne personnalisez pas les scripts, SunPlex Agent Builder fournit des valeurs par défaut rationnelles aux paramètres `scrgadm`.

Utilisation des commandes d'administration standard de Sun Cluster pour créer un service basé sur GDS

Cette rubrique décrit la procédure d'introduction de ces paramètres dans le module GDS. Ce module est utilisé et administré à l'aide des commandes d'administration de Sun Cluster, telles que `scrgadm` et `scswitch`.

Les commandes d'administration de niveau inférieur, répertoriées dans cette rubrique, n'ont pas besoin d'être introduites si les scripts moteurs fournissent des fonctionnalités adéquates. Vous pouvez cependant les introduire si vous souhaitez affiner le contrôle des ressources basées sur le module GDS. Voici les commandes exécutées par les scripts moteurs.

▼ Utilisation des commandes d'administration de Sun Cluster permettant de créer un service à haut niveau de disponibilité basé sur le module GDS

1. **Enregistrez le type de ressources `SUNW.gds`**

   ```
   # scrgadm -a -t SUNW.gds
   ```

2. **Créez le groupe de ressources comportant la ressource LogicalHostname et le service de basculement.**

   ```
   # scrgadm -a -g haapp_rg
   ```

3. **Créez la ressource correspondant à la ressource LogicalHostname.**

   ```
   # scrgadm -a -L -g haapp_rs -l hhead
   ```

4. **Créez la ressource correspondant au service de basculement.**

   ```
   # scrgadm -a -j haapp_rs -g haapp_rg -t SUNW.gds \
           -y Scalable=false -y Start_timeout=120 \
           -y Stop_timeout=120 -x Probe_timeout=120 \
           -y Port_list="2222/tcp" \
           -x Start_command="/export/ha/appctl/start" \
           -x Stop_command="/export/ha/appctl/stop" \
           -x Probe_command="/export/app/bin/probe" \
           -x Child_mon_level=0 -y Network_resources_used=hhead \
           -x Failover_enabled=true -x Stop_signal=9
   ```

5. **Mettez en ligne le groupe de ressources `haapp_rg`.**

```
# scswitch -Z -g haapp_rg
```

▼ Commandes d'administration standard de Sun Cluster permettant de créer un service évolutif basé sur le module GDS

1. **Enregistrez le type de ressources SUNW.gds.**

```
# scrgadm -a -t SUNW.gds
```

2. **Créez le groupe de ressources correspondant à la ressource SharedAddress.**

```
# scrgadm -a -g sa_rg
```

3. **Créez la ressource SharedAddress sur `sa_rg`.**

```
# scrgadm -a -S -g sa_rg -l hhead
```

4. **Créez le groupe de ressources correspondant au service évolutif.**

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

5. **Créez le groupe de ressources correspondant au service évolutif.**

```
# scrgadm -a -j app_rs -g app_rg -t SUNW.gds \
       -y Scalable=true -y Start_timeout=120 \
       -y Stop_timeout=120 -x Probe_timeout=120 \
       -y Port_list="2222/tcp" \
       -x Start_command="/export/app/bin/start" \
       -x Stop_command="/export/app/bin/stop" \
       -x Probe_command="/export/app/bin/probe" \
       -x Child_mon_level=0 -y Network_resource_used=hhead \
       -x Failover_enabled=true -x Stop_signal=9
```

6. **Mettez en ligne le groupe de ressources comportant les ressources réseau.**

```
 # scswitch -Z -g sa_rg
```

7. **Mettez en ligne le groupe de ressources `app_rg`.**

```
# scswitch -Z -g app_rg
```

Interface de ligne de commande de SunPlex Agent Builder

SunPlex Agent Builder intègre une interface de ligne de commande dont les fonctionnalités sont identiques à celles de l'interface utilisateur graphique. Cette interface comprend les commandes `scdscreate`(1HA) et `scdsconfig`(1HA). La rubrique suivante exécute la même fonction que la procédure basée sur l'interface graphique utilisateur "Création d'un service utilisant le module GDS avec la version ligne de commande d'Agent Builder" à la page 208, sans utiliser cette dernière interface.

▼ Création d'un service utilisant le module GDS avec la version ligne de commande d'Agent Builder

1. **Créez le service.**

   Pour un service de basculement, utilisez :

   ```
   # scdscreate -g -V NET -T app -d /export/wdir
   ```

   Pour un service évolutif, utilisez :

   ```
   # scdscreate -g -s -V NET -T app -d /export/wdir
   ```

   Remarque – les paramètres *–d* sont facultatifs. À moins d'une indication contraire, le répertoire de travail est celui en cours d'utilisation.

   ---

2. **Configurez le service.**

   ```
   # scdsconfig -s "/export/app/bin/start' -t "/export/app/bin/stop" \
   -m "/export/app/bin/probe" -d /export/wdir
   ```

   Remarque – seule la commande de démarrage est requise. Tous les autres paramètres sont facultatifs.

   ---

3. **Installez le package complet sur tous les noeuds du cluster.**

   ```
   # cd /export/wdir/NETapp/pkg
   # pkgadd -d . NETapp
   ```

4. **Les fichiers suivants sont installés au cours de pkgadd :**

   ```
   /opt/NETapp
   /opt/NETapp/README.app
   ```

```
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Remarque – les pages de manuel et les noms des scripts correspondront au nom de l'application que vous avez entré ci-dessus, précédé du nom du script ; par exemple, `startapp`.

Pour afficher les pages de manuel, vous devez spécifier leur chemin d'accès. Par exemple, pour afficher les pages de manuel `startapp`, utilisez :

```
# man -M /opt/NETapp/man startapp
```

5. **Sur l'un des noeuds du cluster, configurez les ressources et démarrez l'application.**

   # **/opt/NETapp/util/startapp -h** *<nom_hôte_logique>* **-p** *<liste_ports_et_protocole>*

   Les arguments du script de démarrage varient en fonction du type de ressources : de basculement ou évolutives. Vérifiez la page de manuel personnalisée ou exécutez le script de démarrage sans argument pour obtenir une déclaration d'utilisation.

```
# /opt/NETapp/util/startapp
Vous devez spécifier le nom de ressource de LogicalHostname
ou de SharedAddress.
Pour des services de basculement :
Utilisation : startapp -h <nom_hôte_logique>
-p <liste_ports_et_protocoles>
[-n <liste_groupes_ipmp/adaptateurs>] Pour des services évolutifs :
Utilisation : startapp
-h <nom_adresse_partagée>
-p <liste_ports_et_protocoles>
[-l <règle_équilibrage_charge>]
[-n <liste_groupes_ipmp/adaptateurs>]
[-w <poids_équilibrage_charge>]
```

CHAPITRE 11

Référence à la bibliothèque de développement de services de données

Ce chapitre répertorie et décrit brièvement les fonctions API de la bibliothèque de développement de service de données (BDSD). Reportez-vous aux pages de manuel 3HA pour obtenir la description complète de chaque fonction BDSD. La BDSD définit uniquement une interface C. Il n'existe pas d'interface BDSD scriptable.

La BDSD intègre des fonctions dans les catégories suivantes :

Fonctions BDSD

Les rubriques suivantes présentent brièvement chaque catégorie de fonctions BDSD. Veuillez noter cependant que les pages de manuel 3HA représentent la principale référence en ce qui concerne les fonctions BDSD.

Fonctions générales

Les fonctions décrites dans cette rubrique couvrent de nombreuses fonctionnalités vous permettant de réaliser les tâches suivantes :

- initialiser l'environnement BDSD ;
- récupérer les noms de ressource, type de ressources et groupe de ressources, ainsi que les valeurs de propriété d'extension ;

- basculer et redémarrer un groupe de ressources et redémarrer une ressource ;
- convertir des chaînes d'erreur en message d'erreur ;
- exécuter une commande sans dépasser le délai d'attente.

Les fonctions suivantes initialisent la méthode d'appel :

- `scds_initialize` : alloue des ressources et initialise l'environnement BDSD.
- `scds_close` : libère les ressources allouées par la fonction `scds_initialize`.

Les fonctions suivantes récupèrent les données sur les ressources, les types de ressources, les groupes de ressources et les propriétés d'extension :

- `scds_get_resource_name` : récupère le nom de la ressource du programme d'appel.
- `scds_get_resource_type_name` : récupère le nom du type de ressources du programme d'appel.
- `scds_get_resource_group_name` : récupère le nom du groupe de ressources du programme d'appel.
- `scds_get_ext_property` : récupère la valeur de la propriété d'extension spécifiée.
- `scds_free_ext_property` : libère la mémoire allouée par la fonction `scds_get_ext_property`.

La fonction suivante récupère les informations relatives à l'état des ressources SUNW.HAStoragePlus utilisées par une ressource.

- `scds_hasp_check` : récupère les informations relatives à l'état des ressources `SUNW.HAStoragePlus` utilisées par une ressource. Vous obtiendrez cette information à partir de l'état (en ligne ou autre) de toutes les ressources `SUNW.HAStoragePlus` dont la ressource dépend au moyen des propriétés du système `Dépendances_ressource` ou `Dépendances_ressource_faibles` définies pour cette ressource.

  Reportez-vous à `SUNW.HAStoragePlus`(5) pour de plus amples informations sur `SUNW.HAStoragePlus`.

Les fonctions suivantes basculent ou redémarrent une ressource ou un groupe de ressources :

- `scds_failover_rg` : bascule un groupe de ressources.
- `scds_restart_rg` : redémarre un groupe de ressources.
- `scds_restart_resource` : redémarre une ressource.

Les deux fonctions suivantes exécutent une commande dans un délai imparti et convertissent un code d'erreur en message d'erreur :

- `scds_timerun` : exécute une commande sans dépasser un délai.
- `scds_error_string` : traduit le code d'erreur en chaîne de caractères constituant un message d'erreur.

Fonctions de propriété

Ces fonctions fournissent des API de convenance pour accéder aux propriétés spécifiques d'une ressource, d'un type de ressources ou d'un groupe de ressources approprié, y compris à quelques propriétés d'extension fréquemment utilisées. La BDSD propose la fonction `scds_initialize` pour analyser les arguments de la ligne de commande, puis elle *met en cache* les diverses propriétés de la ressource, du groupe de ressources ou du type de ressources approprié.

Toutes ces fonctions sont décrites sur une page de manuel unique, `scds_property_functions`(3HA). Cette rubrique présente les fonctions suivantes :

- `scds_get_rs_`*nom_propriété* ;
- `scds_get_rg_`*nom_propriété* ;
- `scds_get_rt_`*nom_propriété* ;
- `scds_get_ext_`*nom_propriété*.

Fonctions d'accès aux ressources en réseau

Les fonctions répertoriées dans cette rubrique récupèrent, impriment et libèrent les ressources réseau que les ressources et les groupes de ressources utilisent. Les fonctions `scds_get_*` décrites dans cette rubrique permettent de récupérer facilement les ressources réseau sans recourir à des propriétés spécifiques, telles que `Ressources_réseau_utilisées` et `Liste_ports`, au moyen des fonctions API GR. Les fonctions `scds_print_`*nom*`()` impriment les valeurs des structures de données retournées par les fonctions `scds_get_` *nom*`()`. Les fonctions `scds_free_` *nom*`()` libèrent la mémoire allouée par les fonctions `scds_get_`*nom*`()`.

Les fonctions suivantes sont en rapport avec les noms d'hôte :

- `scds_get_rg_hostnames` : récupère une liste de noms d'hôte utilisés par les ressources réseau au sein d'un groupe de ressources.
- `scds_get_rs_hostnames` : récupère une liste de noms d'hôte utilisés par la ressource.
- `scds_print_net_list` : imprime le contenu de la liste des noms d'hôte retournée par la fonction `scds_get_rg_hostnames` ou `scds_get_rs_hostnames`.
- `scds_free_net_list` : libère la mémoire allouée par les fonctions `scds_get_rg_hostnames` ou `scds_get_rs_hostnames`.

Les fonctions suivantes sont en rapport avec les listes de ports :

- `scds_get_port_list` : récupère une liste de paires port/protocole utilisées par une ressource.
- `scds_print_port_list` : imprime le contenu de la liste de paires port/protocole retournée par la fonction `scds_get_port_list`.

- `scds_free_port_list` : libère la mémoire allouée par la fonction `scds_get_port_list`.

Les fonctions suivantes sont en rapport avec les adresses réseau :

- `scds_get_netaddr_list` : récupère une liste d'adresses réseau utilisées par une ressource.
- `scds_print_netaddr_list` : imprime le contenu de la liste d'adresses réseau retournée par la fonction `scds_get_netaddr_list`.
- `scds_free_netaddr_list` : libère la mémoire allouée par la fonction `scds_get_netaddr_list`.

Détection des pannes à l'aide de connexions TCP

Les fonctions présentées dans cette rubrique activent la détection TCP. En règle générale, le système de détection des pannes utilise ces fonctions pour établir une connexion de prise unique à un service, lire et écrire des données relatives au service pour vérifier son état, puis se déconnecter du service.

Cette rubrique présente les fonctions suivantes :

- `scds_tcp_connect` : ouvre une connexion TCP sur un processus.
- `scds_tcp_read` : utilise une connexion TCP pour lire les données depuis un processus sous surveillance.
- `scds_tcp_write` : utilise une connexion TCP pour écrire des données dans un processus sous surveillance.
- `scds_simple_probe` : détecte un processus en ouvrant et fermant une connexion TCP sur ce processus.
- `scds_tcp_disconnect` : ferme la connexion à un processus sous surveillance.

Fonctions PMF

Ces fonctions encapsulent les fonctionnalités PMF. Le modèle BDSD de surveillance par le biais de la fonction PMF crée et utilise des valeurs *balise* implicites pour `pmfadm`(1M). La fonction PMF utilise également des valeurs implicites pour `Intervalle_redémarrage`, `nombre_nouvelles_tentatives` et `script_action` (les options `-t`, `-n` et `-a` de `pmfadm`). Ce qu'il faut surtout retenir, c'est que la BDSD lie l'historique des pannes de processus (détectées par la fonction PMF) à l'historique des pannes de l'application (détectées par le système de détection des pannes) pour choisir entre le redémarrage et le basculement.

Cette rubrique présente les fonctions suivantes :

- `scds_pmf_get_status` : détermine si l'instance spécifiée est surveillée par la fonction PMF.

- `scds_pmf_restart_fm` : redémarre le système de détection des pannes à l'aide de la fonction PMF.
- `scds_pmf_signal` : envoie le signal spécifié à une arborescence de processus fonctionnant sous le contrôle de la fonction PMF.
- `scds_pmf_start` : exécute un programme spécifié (y compris un système de détection des pannes) sous le contrôle de la fonction PMF.
- `scds_pmf_stop`: ferme un processus fonctionnant sous le contrôle de la fonction PMF.
- `scds_stop_monitoring` : arrête la surveillance d'un processus fonctionnant sous le contrôle de la fonction PMF.

Fonctions du système de détection des pannes

Les fonctions présentées dans cette rubrique proposent un modèle prédéterminé de détection des pannes en conservant l'historique des erreurs et en les évaluant avec les propriétés `Nombre_nouvelles_tentatives` et `Intervalle_nouvelles_tentatives`.

Cette rubrique présente les fonctions suivantes :

- `scds_fm_sleep` : attend un message du socket de contrôle du système de détection des pannes.
- `scds_fm_action` : effectue une action une fois la détection terminée.
- `scds_fm_print_probes` : complète le journal système avec les informations sur l'état de la détection.

Fonctions de l'utilitaire

Les fonctions présentées dans cette rubrique vous permettent d'écrire des messages et des messages de débogage dans le journal système. Cette rubrique présente les deux fonctions suivantes :

- `scds_syslog` : écrit des messages dans le journal système.
- `scds_syslog_debug` : écrit des messages de débogage dans le message système.

CHAPITRE 12

Protocole CRNP

Ce chapitre présente le protocole CRNP (Cluster Reconfiguration Notification Protocol). Le protocole CRNP permet aux applications de basculement et évolutives d'être "prises en charge sur le cluster". Il offre plus particulièrement un mécanisme permettant aux applications de s'enregistrer et d'être averties ultérieurement de façon asynchrone des événements de reconfiguration de Sun Cluster. Les services de données fonctionnant au sein du cluster et les applications exécutées à l'extérieur du cluster peuvent s'enregistrer pour être avertis des événements. Les événements sont générés lorsque l'adhésion au sein d'un cluster change et que l'état d'un groupe de ressources ou d'une ressource est modifié.

Présentation générale du protocole CRNP

Le protocole CRNP offre des mécanismes et des démons qui génèrent des événements de reconfiguration du cluster, les acheminent au sein du cluster et les transmettent aux clients intéressés.

Le démon `cl_apid` interagit avec les clients. Le gestionnaire de groupe de ressources (RGM) de Sun Cluster génère des événements de reconfiguration du cluster. Ces démons utilisent `syseventd`(1M) pour transmettre les événements à chaque noeud local. Le démon `cl_apid` communique avec les clients intéressés via le protocole TCP/IP au moyen du langage XML (Extensible Markup Language).

Le diagramme suivant présente de façon générale le flux d'événements entre les composants CRNP : un client est exécuté sur le noeud 2 du cluster tandis que l'autre fonctionne sur un ordinateur n'appartenant pas au cluster.

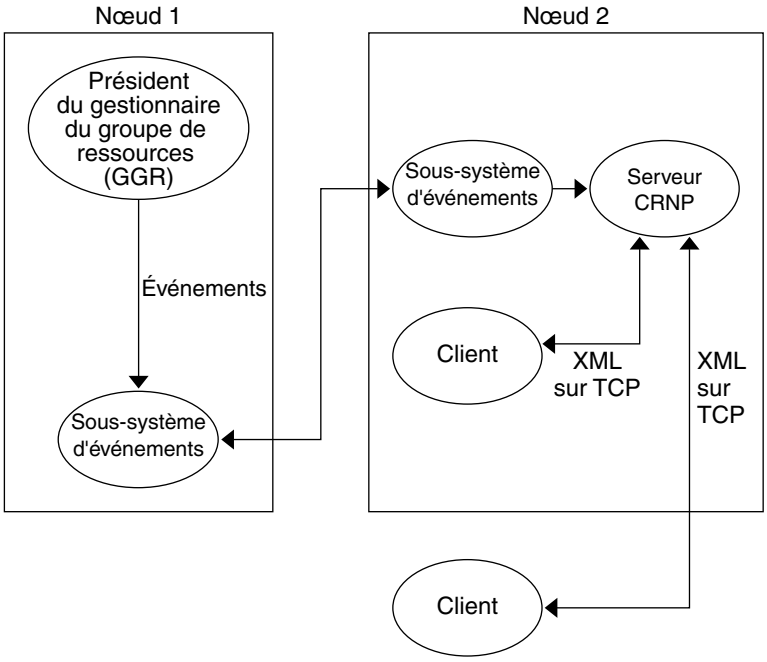


FIGURE 12–1 Fonctionnement du protocole CRNP

Présentation générale du protocole CRNP

Le protocole CRNP définit les couches application, présentation et session de la pile de protocoles de communication OSI (Open System Interconnect/interconnexion de systèmes ouverts) standard constituée de sept couches. La couche transport doit utiliser le protocole TCP et la couche réseau le protocole IP. Le protocole CRNP est indépendant des couches liaison de données et physique. Tous les messages de la couche application échangés au moyen du protocole CRNP utilisent le langage XML 1.0.

Sémantique du protocole CRNP

Les clients initient des communications en envoyant un message de connexion (`SC_CALLBACK_RG`) au serveur. Ce message spécifie le type d'événements dont les clients souhaitent être avertis, ainsi que le port auquel ces événements peuvent être transmis. L'IP source de la connexion d'enregistrement et le port spécifié constituent l'adresse de rappel.

Chaque fois qu'un événement susceptible d'intéresser un client est généré au sein du cluster, le serveur contacte le client sur son adresse de rappel (IP et port) et lui transmet l'événement (`SC_EVENT`). Le serveur est hautement disponible car il fonctionne sur le cluster lui-même. Il enregistre les connexions clients en mémoire et en conserve la trace même après la réinitialisation du cluster.

Les clients se déconnectent en envoyant un message de connexion (`SC_CALLBACK_RG` contenant un message `REMOVE_CLIENT`) au serveur. Lorsque ce dernier leur renvoie le message `SC_REPLY`, les clients ferment leur connexion.

Le diagramme suivant illustre le flux de communications entre un client et un serveur :

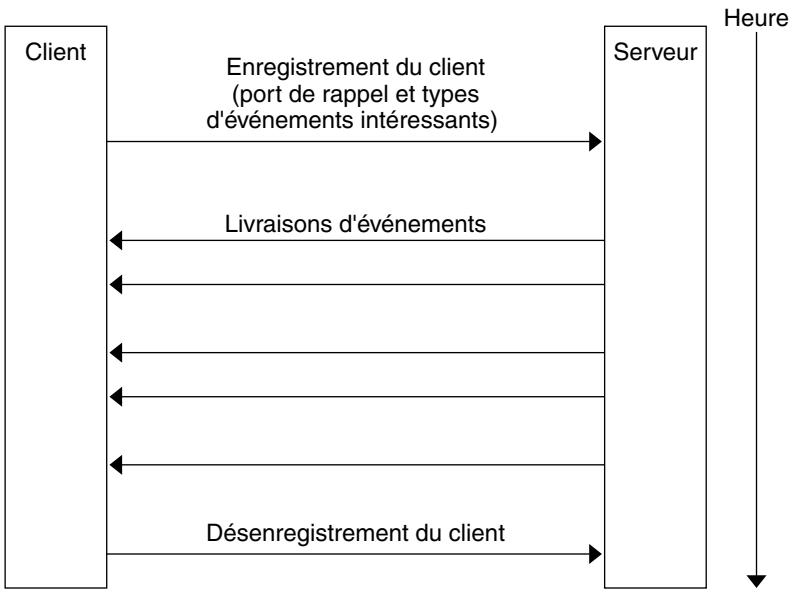


FIGURE 12–2 Flux de communications entre un client et un serveur

Types de messages utilisés par le protocole CRNP

Le protocole CRNP utilise trois types de messages basés sur le langage XML. Présentés brièvement dans le tableau suivant, ces trois types de messages sont également décrits plus en détail dans ce chapitre, de même que leur utilisation.

Type de message	Description
SC_CALLBACK_REG	Ce message se présente sous quatre formes : ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS et REMOVE_EVENTS. Elles contiennent toutes les informations suivantes : ■ version du protocole ; ■ port de rappel au format ASCII (et non au format binaire). Les formes ADD_CLIENT, ADD_EVENTS et REMOVE_EVENTS contiennent également une liste non bornée des types d'événements contenant les informations suivantes : ■ classe d'événement ; ■ sous-classe d'événement (facultative) ; ■ liste des paires nom/valeurs (facultative). La classe et la sous-classe d'événement définissent un "type d'événements" unique. La DTD (définition de type de document) permettant de générer les classes de SC_CALLBACK_REG est SC_CALLBACK_REG. Cette DTD est présentée plus en détail dans l'Annexe F.

SC_EVENT	Ce message contient les informations suivantes :
	■ version du protocole ;
	■ classe d'événement ;
	■ sous-classe d'événement ;
	■ fournisseur ;
	■ éditeur ;
	■ liste des paires nom/valeurs (0 ou plusieurs structures de données de paires nom/valeurs) :
	  ■ nom (chaîne de caractères) ;
	  ■ valeur (chaîne de caractères ou tableau de chaînes de caractères).
	
	Les valeurs d'un message SC_EVENT ne sont pas typées. La DTD (définition de type de document) permettant de générer les classes de SC_EVENT est SC_EVENT. Cette DTD est présentée plus en détail dans l'Annexe F.

Type de message	Description
SC_REPLY	Ce message contient les informations suivantes : ■ version du protocole ; ■ code d'erreur ; ■ message d'erreur. La DTD (définition de type de document) permettant de générer les classes de SC_REPLY est SC_REPLY. Cette DTD est présentée plus en détail dans l'Annexe F.

Processus de connexion d'un client au serveur

Cette rubrique décrit comment un administrateur configure le serveur, comment les clients sont enregistrés et les informations transmises aux couches application et session, ainsi que les conditions d'erreur.

Hypothèses sur la configuration du serveur par les administrateurs

L'administrateur système doit configurer le serveur à l'aide d'une adresse IP à haut niveau de disponibilité (c'est-à-dire une adresse IP qui n'est pas liée à une machine particulière du cluster) ni à un numéro de port. L'administrateur doit transmettre cette adresse réseau aux clients potentiels. Le protocole CRNP ne définit pas comment ce nom de serveur est rendu accessible aux clients. Les administrateurs utilisent un service d'attribution de noms qui permet aux clients de rechercher l'adresse réseau du serveur de façon dynamique ou ajoutent le nom du réseau au fichier de configuration que le client doit lire. Le serveur fonctionne au sein du cluster comme un type de ressources de basculement.

Processus d'identification d'un client par le serveur

Chaque client est identifié de façon unique au moyen de son adresse de rappel, c'est-à-dire son adresse IP et son numéro de port. Le port est spécifié dans les messages SC_CALLBACK_REG et l'adresse IP obtenue à partir de la connexion d'enregistrement TCP. Le protocole CRNP suppose que des messages SC_CALLBACK_REG ultérieurs, possédant la même adresse de rappel, proviendront du même client, même si le port source de transmission des messages sera différent.

Processus de transmission des messages SC_CALLBACK_REG entre un client et le serveur

Un client initie une connexion en ouvrant une connexion TCP sur l'adresse IP et le numéro de port du serveur. Une fois la connexion TCP établie et prête pour la lecture, le client doit envoyer son message de connexion. Il doit s'agir d'un message SC_CALLBACK_REG correctement formaté qui n'est ni suivi ni précédé d'octets supplémentaires.

Une fois tous les octets émis, le client doit rester connecté pour recevoir une réponse du client. Si ce dernier ne formate pas le message correctement, le serveur n'enregistre pas le client et lui transmet un message d'erreur. Si le client ferme la connexion de prise avant que le serveur n'envoie sa réponse, ce dernier enregistre le client comme d'habitude.

Un client peut contacter le serveur à tout moment. Chaque fois qu'un client contacte le serveur, il doit envoyer un message SC_CALLBACK_REG. Si le serveur reçoit un message malformé, en dérangement ou invalide, il renvoie un message d'erreur au client.

Un client ne peut pas transmettre un message ADD_EVENTS, REMOVE_EVENTS ou REMOVE_CLIENT avant d'avoir envoyé un message ADD_CLIENT. Un client ne peut pas transmettre un message REMOVE_CLIENT avant d'avoir envoyé un message ADD_CLIENT.

Si un client envoie un message ADD_CLIENT alors qu'il est déjà connecté, le serveur peut supporter ce message. Dans cette situation, le serveur remplace la connexion client antérieure par la nouvelle qui est spécifiée dans le second message ADD_CLIENT.

Mais en règle générale, un client se connecte une seule fois au serveur par l'envoi d'un message ADD_CLIENT. De même, il se déconnecte une seule fois en envoyant au serveur un message REMOVE_CLIENT. Le protocole CRNP offre une plus grande souplesse aux clients qui doivent modifier de façon dynamique leur liste de types d'événements.

Contenu d'un message SC_CALLBACK_REG

Chaque message ADD_CLIENT, ADD_EVENTS et REMOVE_EVENTS contient une liste d'événements. Le tableau indiqué ci-dessous présente les types d'événements que le protocole CRNP accepte, y compris les paires nom/valeurs requises.

Si un client :

- envoie un message REMOVE_EVENTS spécifiant un ou plusieurs types d'événements auxquels le client ne s'est pas précédemment connecté ;
- se connecte deux fois au même type d'événements,

le serveur ignore ces messages sans vous en avertir.

Classe et sous-classe	Paire nom/valeurs	Description
`EC_Cluster` `ESC_cluster_membership`	Requise : aucune Facultative : aucune	Connexion pour toutes les notifications d'événement sur la modification des membres du cluster (suppression ou ajout d'un noeud)
`EC_Cluster` `ESC_cluster_rg_state`	Une seule requise comme suit : `rg_name` Type de valeurs : chaîne de caractères Facultative : aucune	Connexion pour toutes les notifications d'événement sur la modification des états relatifs au *nom* des groupes de ressources
`EC_Cluster` `ESC_cluster_r_state`	Une seule requise comme suit : `r_name` Type de valeurs : chaîne de caractères Facultative : aucune	Connexion pour toutes les notifications d'événement sur la modification des états relatifs au *nom* des ressources
`EC_Cluster` Aucun	Requise : aucune Facultative : aucune	Connexion pour toutes les notifications d'événement de Sun Cluster

Processus de réponse du client au serveur

La connexion effectuée, le serveur envoie le message `SC_REPLY` sur la connexion TCP ouverte que le client a utilisée pour lui transmettre sa requête de connexion. Le client ferme ensuite la connexion. Le client doit conserver la connexion TCP ouverte jusqu'à ce que le serveur lui transmette le message `SC_REPLY`.

Par exemple, le client :

1. Ouvre une connexion TCP sur le serveur.
2. Attend que la connexion devienne "inscriptible".
3. Envoie un message `SC_CALLBACK_REG` (contenant un message `ADD_CLIENT`).
4. Attend de recevoir un message `SC_REPLY`.
5. Reçoit un message `SC_REPLY`.

6. Reçoit un message stipulant que le serveur a fermé la connexion (lecture de 0 octet du socket).
7. Ferme la connexion.

Puis, ultérieurement il :

1. Ouvre une connexion TCP sur le serveur.
2. Attend que la connexion devienne «inscriptible ».
3. Envoie un message `SC_CALLBACK_REG` (contenant un message `REMOVE_CLIENT`).
4. Attend de recevoir un message `SC_REPLY`.
5. Reçoit un message `SC_REPLY`.
6. Reçoit un message stipulant que le serveur a fermé la connexion (lecture de 0 octet du socket).
7. Ferme la connexion.

Chaque fois que le serveur reçoit un message `SC_CALLBACK_REG` d'un client, le serveur transmet un message `SC_REPLY` sur la même connexion ouverte indiquant si l'opération a réussi ou échoué. "DTD XML `SC_REPLY`" à la page 340 contient la DTD XML d'un message `SC_REPLY`, ainsi que les messages d'erreur éventuels que ce message peut contenir.

Contenu d'un message `SC_REPLY`

Un message `SC_REPLY` indique si une opération a réussi ou échoué. Il contient la version du message de protocole CRNP, un code d'état et un message d'état décrivant le code d'état de façon détaillée. Le tableau suivant présente les valeurs possibles du code d'état :

Code d'état	Description
`OK`	Le message a été traité avec succès.
`RETRY`	La connexion du client a été rejetée par le serveur à cause d'une erreur transitoire (le client doit tenter de se reconnecter en utilisant d'autres paramètres).
`LOW_RESOURCE`	Les ressources du cluster sont faibles et le client peut uniquement réessayer ultérieurement (l'administrateur système du cluster peut également augmenter les ressources sur le cluster).
`SYSTEM_ERROR`	Un incident grave s'est produit. Contactez l'administrateur système du cluster.
`FAIL`	Le refus d'une autorisation ou tout autre incident a provoqué l'échec de la connexion.

Code d'état	Description
MALFORMED	La requête XML est malformée et n'a pas pu être analysée.
INVALID	La requête XML est invalide (elle n'est pas conforme aux spécifications XML).
VERSION_TOO_HIGH	La version du message possède une résolution trop haute pour permettre de traiter le message avec succès.
VERSION_TOO_LOW	La version du message possède une résolution trop basse pour permettre de traiter le message avec succès.

Processus de gestion des conditions d'erreur par un client

Dans des conditions normales, un client envoyant un message SC_CALLBACK_REG reçoit une réponse précisant si la connexion a réussi ou échoué.

Pourtant, le serveur peut être confronté à une condition d'erreur l'empêchant de retransmettre le message SC_REPLY au client. Le cas échéant, la connexion a pu réussir avant que la condition d'erreur ne se produise, échouer ou tout simplement ne pas avoir encore été traitée.

Comme le serveur doit être exécuté sur le cluster en tant que serveur de basculement ou hautement disponible, cette condition d'erreur ne signifie pas que le service est interrompu. De fait, le serveur est rapidement en mesure d'envoyer des événements au client nouvellement connecté.

Pour pallier ces conditions, le client doit :

- Imposer un délai d'attente au niveau de l'application pour une connexion d'enregistrement en attente de la réception d'un message SC_REPLY, délai après lequel il doit retenter de se connecter.
- Commencer à écouter son adresse IP de rappel et son numéro de port en vue de recevoir des notifications d'événement avant d'être connecté pour la notification de rappel d'événement. Le client doit attendre de recevoir un message de confirmation de connexion et de notification d'événement en parallèle. S'il commence à recevoir des notifications d'événement avant de recevoir un message de confirmation, il doit fermer la connexion d'enregistrement.

Processus de notification d'événement au client par le serveur

Comme les événements sont générés dans le cluster, le serveur CRNP les notifie à tous les clients qui se sont connectés pour recevoir les notifications de ce type. La notification se concrétise par l'envoi d'un message `SC_EVENT` à l'adresse de rappel des clients. Chaque notification d'événement utilise une nouvelle connexion TCP.

Juste après qu'un client se soit connecté pour être averti d'un type d'événements à l'aide d'un message `SC_CALLBACK_REG` contenant un message `ADD_CLIENT` ou `ADD_EVENT`, le serveur lui transmet le dernier événement de ce type. Le client est ainsi informé de l'état actuel du système dont seront issus les événements ultérieurs.

Lorsque le serveur ouvre une connexion TCP sur le client, il envoie exactement un message `SC_EVENT` sur la connexion. Il procède ensuite à une fermeture bidirectionnelle.

Par exemple, le client :

1. Attend que le serveur initie une connexion TCP.
2. Accepte la connexion entrante du serveur.
3. Attend de recevoir un message `SC_EVENT`.
4. Lit un message `SC_EVENT`.
5. Reçoit un message stipulant que le serveur a fermé la connexion (lecture de 0 octet du socket).
6. Ferme la connexion.

Une fois tous les clients connectés, ils doivent écouter leur adresse de rappel (adresse IP et numéro de port) en permanence en vue d'une connexion entrante de notification d'événement.

Si le serveur ne parvient pas à contacter le client pour lui notifier un événement, il réessaie suivant l'intervalle et le nombre de fois que vous avez spécifié. Si toutes les tentatives échouent, le client est supprimé de la liste des clients du serveur. Il doit alors se reconnecter en envoyant un autre message `SC_CALLBACK_REG` contenant un message `ADD_CLIENT` avant de pouvoir recevoir d'autres notifications d'événement.

Mécanisme garantissant la notification des événements

Au sein du cluster, un ordre de génération des événements est préservé afin d'avertir chaque client. En d'autres termes, si l'événement A est généré sur le cluster avant l'événement B, le client X reçoit l'événement A avant l'événement B. Par contre, l'ordre de notification des événements à *tous* les clients *n'*est *pas* préservé. Cela signifie que le client Y peut recevoir les événements A et B avant que le client X ne soit averti de l'événement A. De cette façon, les clients dont la connexion est lente ne retardent pas la notification à tous les clients.

Tous les événements notifiés par le serveur (à l'exception du premier événement d'une sous-classe et des événements suivant des erreurs serveur) se produisent en réponse aux événements réels que génère le cluster, hormis lorsque le serveur est confronté à une erreur lui faisant ignorer les événements générés par le cluster. Le cas échéant, il génère un événement pour chaque type d'événements représentant l'état actuel du système. Chaque événement est transmis aux clients qui ont notifié leur souhait de recevoir ce type d'événements.

Chaque événement respecte la sémantique « au moins une fois », ce qui signifie que le serveur peut envoyer le même événement au client plus d'une fois. Cette faculté est indispensable lorsque le serveur redevient opérationnel après une interruption provisoire et qu'il ne peut déterminer si le client a reçu les dernières informations transmises.

Contenu d'un message `SC_EVENT`

Le message `SC_EVENT` contient le message exact généré au sein du cluster, retranscrit au format de message XML `SC_EVENT`. Le tableau indiqué ci-dessous présente les types d'événements que le protocole CRNP envoie, y compris les paires nom/valeurs requises, l'éditeur et le fournisseur.

Classe et sous-classe	Éditeur et fournisseur	Paire nom/valeurs	Notes
`EC_Cluster` `ESC_cluster_membership`	Éditeur : rgm Fournisseur : SUNW	Nom : `node_list` Type de valeurs : tableau de chaînes de caractères Nom : `state_list` Type de valeurs : tableau de chaînes de caractères	Les positions des éléments du tableau de `state_list` sont synchronisées avec celles de `node_list`. Cela signifie que l'état du noeud répertorié en première position dans le tableau `node_list` se trouve en première position dans le tableau `state_list`. `state_list` contient uniquement les numéros ASCII. Chaque numéro représente le numéro actuel du noeud dans le cluster. Si ce numéro correspond à celui reçu dans un message précédent, cela signifie que la relation qui lie le noeud au cluster n'a pas changé (disparu/connecté/déconnecté). Si ce numéro est -1, le noeud n'est pas un membre du cluster. Si ce numéro n'est pas un nombre négatif, le noeud est un membre du cluster. Les autres noms commençant par `ev_` et les valeurs connexes peuvent être présents mais ils ne sont pas destinés à être utilisés par les clients.

Classe et sous-classe	Éditeur et fournisseur	Paire nom/valeurs	Notes
EC_Cluster ESC_cluster_rg_state	Éditeur : rgm Fournisseur : SUNW	Nom : rg_name Type de valeurs : chaîne de caractères Nom : node_list Type de valeurs : tableau de chaînes de caractères Nom : state_list Type de valeurs : tableau de chaînes de caractères	Les positions des éléments du tableau de state_list sont synchronisées avec celles de node_list. Cela signifie que l'état du noeud répertorié en première position dans le tableau node_list se trouve en première position dans le tableau state_list. state_list contient les représentations de chaînes de caractères de l'état du groupe de ressources. Les valeurs valides correspondent à celles que vous pouvez rechercher et extraire à l'aide des commandes scha_cmds(1HA). Les autres noms commençant par ev_ et les valeurs connexes peuvent être présents mais ils ne sont pas destinés à être utilisés par les clients.

Classe et sous-classe	Éditeur et fournisseur	Paire nom/valeurs	Notes
`EC_Cluster` `ESC_cluster_r_state`	Éditeur : rgm Fournisseur : SUNW	Trois requises comme suit : Nom : `r_name` Type de valeurs : chaîne de caractères Nom : `node_list` Type de valeurs : tableau de chaînes de caractères Nom : `state_list` Type de valeurs : tableau de chaînes de caractères	Les positions des éléments du tableau de `state_list` sont synchronisées avec celles de `node_list`. Cela signifie que l'état du noeud répertorié en première position dans le tableau `node_list` se trouve en première position dans le tableau `state_list`. `state_list` contient les représentations de chaînes de caractères de l'état de la ressource. Les valeurs valides correspondent à celles que vous pouvez rechercher et extraire à l'aide des commandes `scha_cmds`(1HA). Les autres noms commençant par `ev_` et les valeurs connexes peuvent être présents mais ils ne sont pas destinés à être utilisés par les clients.

Processus d'authentification des clients et du serveur par le protocole CRNP

Le serveur authentifie un client en utilisant une forme de wrappers TCP. L'adresse IP source du message de connexion (qui sert également d'adresse IP de rappel à laquelle les événements sont envoyés) doit figurer dans la liste des clients autorisés sur le serveur. En outre, cette adresse et ce message ne peuvent pas figurer dans la liste des clients refusés. Si ces informations ne figurent pas dans la liste, le serveur refuse la requête et renvoie un message d'erreur au client.

Lorsque le serveur reçoit un message `SC_CALLBACK_REG ADD_CLIENT` d'un client, les messages `SC_CALLBACK_REG` ultérieurs de ce client doivent contenir une adresse IP source identique à celle figurant dans le premier message. Si le serveur CRNP reçoit un message `SC_CALLBACK_REG` qui n'est pas conforme à cette exigence, il :

- Ignore la requête et envoie un message d'erreur au client.
- Ou considère que la requête est émise par un nouveau client (suivant le contenu du message `SC_CALLBACK_REG`).

Ce mécanisme de sécurité permet de prévenir les attaques de refus de service tandis qu'une personne tente de déconnecter un client légitime.

Les clients doivent également authentifier le serveur de façon similaire. Les clients ne doivent accepter que les notifications d'événement d'un serveur dont l'adresse IP source et le numéro de port sont identiques à l'adresse IP de connexion et au numéro de port que le client a utilisé.

Comme il est prévu que les clients du service CRNP soient protégés par le pare-feu du cluster, le protocole CRNP n'intègre pas d'autres mécanismes de sécurité.

Création d'une application Java utilisant le protocole CRNP

L'exemple suivant illustre le processus de développement d'une application Java, `CrnpClient`, utilisant le protocole CRNP. Cette application enregistre les rappels d'événements avec le serveur CRNP sur le cluster, les écoute, puis traite les événements en imprimant leur contenu. Pour terminer, elle supprime les rappels d'événements.

Gardez ces informations à l'esprit pendant que vous étudiez cet exemple.

- Notre exemple d'application génère et analyse du code XML à l'aide de l'API Java pour l'analyse XML (JAXP). Il ne vous apprend pas à utiliser JAXP. JAXP est présenté plus en détail sur le site suivant : `http://java.sun.com/xml/jaxp/index.html`.
- Cet exemple reprend des passages d'une application complète que vous trouverez dans l'Annexe G. Il en est toutefois légèrement différent, afin d'illustrer plus efficacement des concepts particuliers.
- Pour plus de concision, les commentaires ont également été supprimés du code modèle. Vous les trouverez dans l'application complète à l'Annexe G.
- L'application proposée dans notre exemple gère la plupart des conditions d'erreur en quittant tout simplement l'application. Sachez que, dans la réalité, votre application doit offrir une gestion des erreurs plus robuste.

▼ Configuration de l'environnement

Vous devez commencer par configurer votre environnement.

1. **Téléchargez puis installez JAXP ainsi que la version appropriée du compilateur Java et de la machine virtuelle.**

   Vous trouvez de plus amples instructions à l'adresse suivante : `http://java.sun.com/xml/jaxp/index.html`.

   Remarque – cet exemple requiert Java 1.3.1 ou une version ultérieure.

   ---

2. **N'oubliez pas de spécifier un `classpath` sur la ligne de commande de compilation, afin que le compilateur puisse trouver les classes JAXP. Depuis le répertoire dans lequel figure le fichier source, entrez :**

   ```
   % javac -classpath RACINE_JAXP/dom.jar:RACINE_JAXPjaxp-api. \
   jar:RACINE_JAXPsax.jar:RACINE_JAXPxalan.jar:RACINE_JAXP/xercesImpl \
   .jar:RACINE_JAXP/xsltc.jar -sourcepath . NOM_FICHIER_SOURCE.java
   ```

   RACINE_JAXP correspond au chemin d'accès, relatif ou absolu, au répertoire dans lequel se trouvent les fichiers jar de JAXP et *NOM_FICHIER_SOURCE* au nom du fichier Java source.

3. **Lors de l'exécution de l'application, spécifiez le `classpath` pour que l'application puisse charger les fichiers de classe JAXP appropriés (veuillez noter que le premier chemin d'accès spécifié dans le `classpath` correspond au répertoire courant) :**

   ```
   java -cp .:RACINE_JAXP/dom.jar:RACINE_JAXPjaxp-api. \
   jar:RACINE_JAXPsax.jar:RACINE_JAXPxalan.jar:RACINE_JAXP/xercesImpl \
   .jar:RACINE_JAXP/xsltc.jar NOM_FICHIER_SOURCE ARGUMENTS
   ```

   L'environnement est à présent configuré. Vous pouvez développer l'application.

▼ Premiers pas

Dans cette première partie de notre exemple, vous allez créer une classe de base appelée `CrnpClient` à l'aide d'une méthode principale qui analyse les arguments de la ligne de commande et crée un objet `CrnpClient`. Cet objet transfère les arguments de la ligne de commande à la classe, attend que l'utilisateur arrête l'application, exécute la commande `shutdown` sur la classe `CrnpClient`, puis se ferme.

Le constructeur de la classe `CrnpClient` doit exécuter les tâches suivantes :

- définir les objets de traitement XML ;
- créer un thread contrôlant les rappels d'événement ;
- se connecter au serveur CRNP et s'enregistrer pour être averti des rappels d'événement.

- **Créez le code Java mettant en oeuvre la logique précédente.**

  L'exemple suivant présente le code du squelette de la classe `CrnpClient`. Les mises en oeuvre des quatre méthodes d'assistant référencées dans les méthodes d'arrêt et du constructeur sont décrites ultérieurement. Veuillez noter que le code important tous les packages requis est indiqué.

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

import java.net.*;
import java.io.*;
import java.util.*;


class CrnpClient
{
        public static void main(String []args)
        {
                InetAddress regIp = null;
                int regPort = 0, localPort = 0;

                try {
                        regIp = InetAddress.getByName(args[0]);
                        regPort = (new Integer(args[1])).intValue();
                        localPort = (new Integer(args[2])).intValue();
                } catch (UnknownHostException e) {
                        System.out.println(e);
                        System.exit(1);
                }

                CrnpClient client = new CrnpClient(regIp, regPort, localPort,
                    args);
                System.out.println("Hit return to terminate demo...");
                try {
                        System.in.read();
                } catch (IOException e) {
                        System.out.println(e.toString());
                }
                client.shutdown();
                System.exit(0);
        }

        public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
            String []clArgs)
        {
                try {
                        regIp = regIpIn;
                        regPort = regPortIn;
                        localPort = localPortIn;
```

```
                        regs = clArgs;

                        setupXmlProcessing();
                        createEvtRecepThr();
                        registerCallbacks();

                } catch (Exception e) {
                        System.out.println(e.toString());
                        System.exit(1);
                }
        }

        public void shutdown()
        {
                try {
                        unregister();
                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        private InetAddress regIp;
        private int regPort;
        private EventReceptionThread evtThr;
        private String regs[];

        public int localPort;
        public DocumentBuilderFactory dbf;
}
```

Les variables membres sont traitées plus en détail par la suite.

▼ Analyse des arguments de la ligne de commande

- **Pour découvrir comment analyser les arguments de la ligne de commande, reportez-vous au code de l'Annexe G.**

▼ Définition d'un thread de réception d'événements

Vous devez vous assurer, au niveau du code, que les événements sont reçus sur un thread distinct, afin que votre application puisse continuer d'exécuter d'autres tâches tandis que le thread d'événements attend la notification de rappel d'événements.

Remarque – la configuration XML est traitée ultérieurement.

---

1. **Dans votre code, définissez une sous-classe `Thread` appelée `EventReceptionThread` créant une classe `ServerSocket` et attendant que le socket reçoive des événements.**

   Dans cette partie du code, les événements ne sont ni lus ni traités. La lecture et le traitement des événements sont abordés ultérieurement. La classe `EventReceptionThread` crée une classe `ServerSocket` sur l'adresse générique du protocole interréseau. `EventReceptionThread` conserve également l'objet `CrnpClient` en référence, afin de pouvoir lui transmettre des événements.

```
class EventReceptionThread extends Thread
{
        public EventReceptionThread(CrnpClient clientIn) throws IOException
        {
                client = clientIn;
                listeningSock = new ServerSocket(client.localPort, 50,
                    InetAddress.getLocalHost());
        }

        public void run()
        {
                try {
                        DocumentBuilder db = client.dbf.newDocumentBuilder();
                        db.setErrorHandler(new DefaultHandler());

                        while(true) {
                                Socket sock = listeningSock.accept();
                                // Créer l'événement à partir du flux du sock et le traiter
                                sock.close();
                        }
                        // IMPOSSIBLE À ATTEINDRE

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* private member variables */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. **Maintenant que vous savez comment fonctionne la classe `EventReceptionThread`, créez un objet `createEvtRecepThr` :**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
}
```

▼ Connexion et déconnexion des rappels

La connexion recoupe les tâches suivantes :

- ouverture d'un socket TCP de base sur le port et le protocole interréseau de connexion ;
- création d'un message de connexion XML ;
- envoi d'un message de connexion XML au socket ;
- lecture d'un message de réponse XML sur le socket ;
- fermeture du socket.

1. **Créez le code Java mettant en oeuvre la logique précédente.**

   L'exemple suivant présente la mise en oeuvre de la méthode `registerCallbacks` de la classe `CrnpClient` (qui est appelée par le constructeur `CrnpClient`). Les appels des fonctions `createRegistrationString()` et `readRegistrationReply()` sont décrits ultérieurement.

   `regIp` et `regPort` sont des objets membres définis par le constructeur.

```
private void registerCallbacks() throws Exception
{
        Socket sock = new Socket(regIp, regPort);
        String xmlStr = createRegistrationString();
        PrintStream ps = new
                PrintStream(sock.getOutputStream());
        ps.print(xmlStr);
        readRegistrationReply(sock.getInputStream();
        sock.close();
}
```

2. **Mettez en oeuvre la méthode `unregister`. Cette méthode est appelée par la méthode `shutdown` de la classe `CrnpClient`. La mise en oeuvre de `createUnregistrationString` est présentée plus en détail ultérieurement.**

```
private void unregister() throws Exception
{
        Socket sock = new Socket(regIp, regPort);
        String xmlStr = createUnregistrationString();
        PrintStream ps = new PrintStream(sock.getOutputStream());
        ps.print(xmlStr);
        readRegistrationReply(sock.getInputStream());
        sock.close();
}
```

▼ Génération du langage XML

Maintenant que vous avez défini la structure de l'application et rédigé l'intégralité du code de gestion de réseaux, vous rédigez le code qui génère et analyse XML. Commencez par rédiger le code générant le message de connexion XML `SC_CALLBACK_REG`.

Un message `SC_CALLBACK_REG` est constitué d'un type de connexion (`ADD_CLIENT`, `REMOVE_CLIENT`, `ADD_EVENTS` ou `REMOVE_EVENTS`), d'un port de rappel et d'une liste d'événements intéressants. Chaque événement comprend une classe et une sous-classe, suivies d'une liste de paires nom/valeurs.

Dans cette partie de l'exemple, vous rédigez une classe `CallbackReg` qui enregistre le type de connexion, le port de rappel et la liste des événements de connexion. Cette classe peut également se sérialiser en message XML `SC_CALLBACK_REG`.

Cette classe comprend la méthode `convertToXml`. Cette méthode intéressante crée une chaîne de message XML `SC_CALLBACK_REG` à partir des membres de la classe. La documentation JAXP disponible à l'adresse suivante `http://java.sun.com/xml/jaxp/index.html` présente plus en détail le code de cette méthode.

La mise en oeuvre de la classe `Event` est présentée ci-après. Veuillez noter que la classe `CallbackReg` utilise une classe `Event` qui mémorise un événement et peut le convertir en `Element` XML.

1. Créez le code Java mettant en oeuvre la logique précédente.

```
class CallbackReg
{
        public static final int ADD_CLIENT = 0;
        public static final int ADD_EVENTS = 1;
        public static final int REMOVE_EVENTS = 2;
        public static final int REMOVE_CLIENT = 3;

        public CallbackReg()
        {
                port = null;
                regType = null;
                regEvents = new Vector();
        }

        public void setPort(String portIn)
        {
                port = portIn;
        }

        public void setRegType(int regTypeIn)
        {
                switch (regTypeIn) {
                case ADD_CLIENT:
                        regType = "ADD_CLIENT";
```

```
                break;
        case ADD_EVENTS:
                regType = "ADD_EVENTS";
                break;
        case REMOVE_CLIENT:
                regType = "REMOVE_CLIENT";
                break;
        case REMOVE_EVENTS:
                regType = "REMOVE_EVENTS";
                break;
        default:
                System.out.println("Error, invalid regType " +
                    regTypeIn);
                regType = "ADD_CLIENT";
                break;
        }
}

public void addRegEvent(Event regEvent)
{
        regEvents.add(regEvent);
}

public String convertToXml()
{
        Document document = null;
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                document = builder.newDocument();
        } catch (ParserConfigurationException pce) {
                // L'analyseur avec options spécifiques ne peut pas être créé
                pce.printStackTrace();
                System.exit(1);
        }

        // Créer l'élément racine
        Element root = (Element) document.createElement(
            "SC_CALLBACK_REG");

        // Ajouter les attributs
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Ajouter les événements
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(
                    document));
        }
        document.appendChild(root);
```

```
                // Tout convertir en chaîne
                DOMSource domSource = new DOMSource(document);
                StringWriter strWrite = new StringWriter();
                StreamResult streamResult = new StreamResult(strWrite);
                TransformerFactory tf = TransformerFactory.newInstance();
                try {
                        Transformer transformer = tf.newTransformer();
                        transformer.transform(domSource, streamResult);
                } catch (TransformerException e) {
                        System.out.println(e.toString());
                        return ("");
                }
                return (strWrite.toString());
        }

        private String port;
        private String regType;
        private Vector regEvents;
}
```

2. **Mettez en oeuvre les classes `Event` et `NVPair`.**

   Veuillez noter que la classe `CallbackReg` utilise une classe `Event` qui utilise elle-même une classe `NVPair`.

```
class Event
{

        public Event()
        {
                regClass = regSubclass = null;
                nvpairs = new Vector();
        }

        public void setClass(String classIn)
        {
                regClass = classIn;
        }

        public void setSubclass(String subclassIn)
        {
                regSubclass = subclassIn;
        }

        public void addNvpair(NVPair nvpair)
        {
                nvpairs.add(nvpair);
        }

        public Element createXmlElement(Document doc)
        {
                Element event = (Element)
                    doc.createElement("SC_EVENT_REG");
                event.setAttribute("CLASS", regClass);
                if (regSubclass != null) {
                        event.setAttribute("SUBCLASS", regSubclass);
```

```
                }
                for (int i = 0; i < nvpairs.size(); i++) {
                        NVPair tempNv = (NVPair)
                            (nvpairs.elementAt(i));
                        event.appendChild(tempNv.createXmlElement(
                            doc));
                }
                return (event);
        }

        private String regClass, regSubclass;
        private Vector nvpairs;
}

class NVPair
{
        public NVPair()
        {
                name = value = null;
        }

        public void setName(String nameIn)
        {
                name = nameIn;
        }

        public void setValue(String valueIn)
        {
                value = valueIn;
        }

        public Element createXmlElement(Document doc)
        {
                Element nvpair = (Element)
                    doc.createElement("NVPAIR");
                Element eName = doc.createElement("NAME");
                Node nameData = doc.createCDATASection(name);
                eName.appendChild(nameData);
                nvpair.appendChild(eName);
                Element eValue = doc.createElement("VALUE");
                Node valueData = doc.createCDATASection(value);
                eValue.appendChild(valueData);
                nvpair.appendChild(eValue);

                return (nvpair);
        }

        private String name, value;
}
```

▼ Création des messages de connexion et de déconnexion

Maintenant que vous avez créé les classes d'assistant générant les messages XML, vous pouvez écrire la mise en oeuvre de la méthode `createRegistrationString`. Cette méthode est appelée par la méthode `registerCallbacks` présentée dans la rubrique "Connexion et déconnexion des rappels" à la page 236.

`createRegistrationString` crée un objet `CallbackReg` et définit son type et son port de connexion. `createRegistrationString` crée divers événements à l'aide des méthodes d'assistant `createAllEvent`, `createMembershipEvent`, `createRgEvent` et `createREvent`. Chaque événement est ajouté à l'objet `CallbackReg` une fois cet objet créé. Enfin, `createRegistrationString` appelle la méthode `convertToXml` sur l'objet `CallbackReg` pour rechercher et extraire les messages XML au format `String`.

Veuillez noter que la variable membre `regs` enregistre les arguments de la ligne de commande qu'un utilisateur fournit à l'application. Le cinquième argument et les arguments ultérieurs spécifient les événements pour lesquels l'application doit se connecter. Le quatrième argument spécifie le type de connexion mais il est ignoré dans cet exemple. Le code complet présenté dans l'Annexe G montre comment utiliser ce quatrième argument.

1. **Créez le code Java mettant en oeuvre la logique précédente.**

```
private String createRegistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);

        cbReg.setRegType(CallbackReg.ADD_CLIENT);

        // ajouter les éléments
        for (int i = 4; i < regs.length; i++) {
                if (regs[i].equals("M")) {
                        cbReg.addRegEvent(
                            createMembershipEvent());
                } else if (regs[i].equals("A")) {
                        cbReg.addRegEvent(
                            createAllEvent());
                } else if (regs[i].substring(0,2).equals("RG")) {
                        cbReg.addRegEvent(createRgEvent(
                            regs[i].substring(3)));
                } else if (regs[i].substring(0,1).equals("R")) {
                        cbReg.addRegEvent(createREvent(
                            regs[i].substring(2)));
                }
        }

        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}
```

```
private Event createAllEvent()
{
        Event allEvent = new Event();
        allEvent.setClass("EC_Cluster");
        return (allEvent);
}

private Event createMembershipEvent()
{
        Event membershipEvent = new Event();
        membershipEvent.setClass("EC_Cluster");
        membershipEvent.setSubclass("ESC_cluster_membership");
        return (membershipEvent);
}

private Event createRgEvent(String rgname)
{
        Event rgStateEvent = new Event();
        rgStateEvent.setClass("EC_Cluster");
        rgStateEvent.setSubclass("ESC_cluster_rg_state");

        NVPair rgNvpair = new NVPair();
        rgNvpair.setName("rg_name");
        rgNvpair.setValue(rgname);
        rgStateEvent.addNvpair(rgNvpair);

        return (rgStateEvent);
}

private Event createREvent(String rname)
{
        Event rStateEvent = new Event();
        rStateEvent.setClass("EC_Cluster");
        rStateEvent.setSubclass("ESC_cluster_r_state");

        NVPair rNvpair = new NVPair();
        rNvpair.setName("r_name");
        rNvpair.setValue(rname);
        rStateEvent.addNvpair(rNvpair);

        return (rStateEvent);
}
```

2. **Créez la chaîne de caractères de déconnexion.**

   La création de la chaîne de caractères de déconnexion est plus facile à créer que la chaîne de caractères de connexion car vous n'avez pas à prendre en charge les événements :

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
```

```
        return (xmlStr);
}
```

▼ Configuration de l'analyseur XML

Maintenant que vous avez créé le code de génération XML et de gestion de réseaux de l'application, l'étape finale consiste à analyser et traiter la réponse de connexion et les rappels d'événement. Le constructeur `CrnpClient` appelle une méthode `setupXmlProcessing`. Cette méthode crée un objet `DocumentBuilderFactory` et définit plusieurs propriétés d'analyse sur cet objet. La documentation JAXP disponible à l'adresse suivante `http://java.sun.com/xml/jaxp/index.html` présente plus en détail cette méthode.

- **Créez le code Java mettant en oeuvre la logique précédente.**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        //Nous n'avons pas besoin de valider
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // Nous souhaitons ignorer les commentaires et les espaces
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Combiner les sections CDATA en noeuds TEXT.
        dbf.setCoalescing(true);
}
```

▼ Analyse de la réponse de connexion

Pour analyser le message XML `SC_REPLY` que le serveur CRNP transmet en réponse à un message de connexion ou de déconnexion, vous avez besoin d'une classe d'assistant `RegReply`. Vous pouvez créer cette classe à partir d'un document XML. Elle propose les mécanismes d'accès du code et du message d'état. Pour analyser le flux XML du serveur, vous devez créer un nouveau document XML et utiliser cette méthode d'analyse (la documentation JAXP disponible à l'adresse suivante `http://java.sun.com/xml/jaxp/index.html` présente plus en détail cette méthode).

1. **Créez le code Java mettant en oeuvre la logique précédente.**

   Veuillez noter que la méthode `readRegistrationReply` utilise la nouvelle classe `RegReply`.

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Créer le constructeur de document
```

```
                DocumentBuilder db = dbf.newDocumentBuilder();
                db.setErrorHandler(new DefaultHandler());

                // Analyser le fichier d'entrée
                Document doc = db.parse(stream);

                RegReply reply = new RegReply(doc);
                reply.print(System.out);
        }
```

2. **Mettez en oeuvre la classe `RegReply`.**

   Veuillez noter que la méthode `retrieveValues` parcourt l'arborescence DOM dans le document XML et extrait le code et le message d'état. La documentation JAXP disponible à l'adresse suivante `http://java.sun.com/xml/jaxp/index.html` contient de plus amples informations.

```
class RegReply
{
        public RegReply(Document doc)
        {
                retrieveValues(doc);
        }

        public String getStatusCode()
        {
                return (statusCode);
        }

        public String getStatusMsg()
        {
                return (statusMsg);
        }
        public void print(PrintStream out)
        {
                out.println(statusCode + ": " +
                    (statusMsg != null ? statusMsg : ""));
        }

        private void retrieveValues(Document doc)
        {
                Node n;
                NodeList nl;
                String nodeName;

                // Trouver l'élément SC_REPLY.
                nl = doc.getElementsByTagName("SC_REPLY");
                if (nl.getLength() != 1) {
                        System.out.println("Error in parsing: can't find "
                            + "SC_REPLY node.");
                        return;
                }

                n = nl.item(0);
```

```
            // Récupérer la valeur de l'attribut statusCode
            statusCode = ((Element)n).getAttribute("STATUS_CODE");

            // Trouver l'élément SC_STATUS_MSG
            nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
            if (nl.getLength() != 1) {
                    System.out.println("Error in parsing: can't find "
                        + "SC_STATUS_MSG node.");
                    return;
            }
            // Obtenir la section TEXT, le cas échéant.
            n = nl.item(0).getFirstChild();
            if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Not an error if there isn't one, so we just silently return.
                    return;
            }

            // Récupérer la valeur
            statusMsg = n.getNodeValue();
        }

        private String statusCode;
        private String statusMsg;
}
```

▼ Analyse des événements de rappel

L'étape finale consiste à analyser et traiter les événements de rappels réels. Pour faciliter cette tâche, modifiez la classe `Event` que vous avez créée à la rubrique "Génération du langage XML" à la page 237, afin qu'elle puisse créer un `Event` à partir d'un document XML et un `Element` XML. Cette modification requiert un constructeur supplémentaire (exécutant un document XML), une méthode `retrieveValues`, l'ajout de deux variables membres (`vendor` et `publisher`), des méthodes de mécanisme d'accès à tous les champs et une méthode d'impression.

1. **Créez le code Java mettant en oeuvre la logique précédente.**

   Veuillez noter que ce code est identique à celui de la classe `RegReply` décrite dans la rubrique "Analyse de la réponse de connexion" à la page 243.

```
public Event(Document doc)
{
        nvpairs = new Vector();
        retrieveValues(doc);
}
public void print(PrintStream out)
{
        out.println("\tCLASS=" + regClass);
        out.println("\tSUBCLASS=" + regSubclass);
        out.println("\tVENDOR=" + vendor);
        out.println("\tPUBLISHER=" + publisher);
```

```
        for (int i = 0; i < nvpairs.size(); i++) {
                NVPair tempNv = (NVPair)
                    (nvpairs.elementAt(i));
                out.print("\t\t");
                tempNv.print(out);
        }
}

private void retrieveValues(Document doc)
{
        Node n;
        NodeList nl;
        String nodeName;

        // Trouver l'élément SC_EVENT.
        nl = doc.getElementsByTagName("SC_EVENT");
        if (nl.getLength() != 1) {
           System.out.println("Error in parsing: can't find "
               + "SC_EVENT node.");
           return;
        }

        n = nl.item(0);

        //
        // Récupérer la valeur des attributs de CLASS, SUBCLASS,
        // VENDOR et PUBLISHER.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Récupérer toutes les paires nv
        for (Node child = n.getFirstChild(); child != null;
            child = child.getNextSibling())
        {
               nvpairs.add(new NVPair((Element)child));
        }
}

public String getRegClass()
{
        return (regClass);
}

public String getSubclass()
{
        return (regSubclass);
}

public String getVendor()
{
        return (vendor);
}
```

```
public String getPublisher()
{
        return (publisher);
}

public Vector getNvpairs()
{
        return (nvpairs);
}

private String vendor, publisher;
```

2. **Mettez en oeuvre les autres constructeurs et méthodes de la classe `NVPair` qui prennent en charge l'analyse XML.**

   Les modifications apportées à la classe `Event` (voir Étape 1) doivent être identiques aux modifications apportées à la classe `NVPair`.

```
public NVPair(Element elem)
{
        retrieveValues(elem);
}
public void print(PrintStream out)
{
        out.println("NAME=" + name + " VALUE=" + value);
}
private void retrieveValues(Element elem)
{
        Node n;
        NodeList nl;
        String nodeName;

        // Trouver l'élément NAME
        nl = elem.getElementsByTagName("NAME");
        if (nl.getLength() != 1) {
           System.out.println("Error in parsing: can't find "
               + "NAME node.");
           return;
        }
        // Obtenir la section TEXT
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
           System.out.println("Error in parsing: can't find "
               + "TEXT section.");
           return;
        }

        // Récupérer la valeur
        name = n.getNodeValue();

        // Obtenir maintenant l'élément de valeur
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
           System.out.println("Error in parsing: can't find "
               + "VALUE node.");
```

```
                return;
            }
            // Obtenir la section TEXT
            n = nl.item(0).getFirstChild();
            if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            System.out.println("Error in parsing: can't find "
                        + "TEXT section.");
                    return;
            }

            // Récupérer la valeur
            value = n.getNodeValue();
            }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3. **Mettez en oeuvre une boucle `while` dans la classe `EventReceptionThread` attendant les rappels d'événements (`EventReceptionThread` est présenté dans la rubrique "Définition d'un thread de réception d'événements" à la page 234).**

```
while(true) {
            Socket sock = listeningSock.accept();
            Document doc = db.parse(sock.getInputStream());
            Event event = new Event(doc);
            client.processEvent(event);
            sock.close();
        }
```

▼ Exécution de l'application

- **Exécutez votre application.**

  # **`java CrnpClient`** *hôte_crnp port_crnp port_local* **`...`**

  Le code complet de l'application `CrnpClient` figure à l'Annexe G.

ANNEXE **A**

Propriétés standard

Cette annexe présente les propriétés des types de ressources standard, des groupes de ressources et des ressources ainsi que les attributs des propriétés de ressources disponibles pour modifier les propriétés définies par le système et créer des propriétés d'extension.

Cette annexe comprend les rubriques suivantes :

Remarque – les valeurs de propriétés, comme `Vrai` et `Faux`, *ne* sont *pas* sensibles à la casse.

Propriétés des types de ressources

Le tableau suivant présente les propriétés des types de ressources définis par Sun Cluster. Les valeurs des propriétés sont classées comme suit (colonne Catégorie) :

- **Requise** : vous devez attribuer une valeur explicite à la propriété dans le fichier RTR (Resource Type Registration), sinon l'objet auquel elle appartient ne pourra pas être créé. Une chaîne de caractères blanche ou vide est proscrite.
- **Conditionnelle** : la propriété doit obligatoirement être déclarée dans le fichier RTR. Dans le cas contraire, le gestionnaire RGM ne la crée pas et elle n'est pas disponible au niveau des utilitaires d'administration. Une chaîne de caractères blanche ou vide est autorisée. Si la propriété est déclarée dans le fichier RTR mais qu'aucune valeur n'est spécifiée, le gestionnaire RGM lui attribue une valeur par défaut.

- **Conditionnelle/explicite** : la propriété doit obligatoirement être déclarée dans le fichier RTR avec une valeur explicite. Dans le cas contraire, le gestionnaire RGM ne la crée pas et elle n'est pas disponible au niveau des utilitaires d'administration. Une chaîne de caractères blanche ou vide est proscrite.
- **Facultative** : la propriété peut être déclarée dans le fichier RTR. Si elle n'est pas déclarée, le gestionnaire RGM la crée et lui attribue une valeur par défaut. Si elle est déclarée dans le fichier RTR sans qu'une valeur soit spécifiée, le gestionnaire RGM lui attribue la valeur par défaut qu'il lui aurait donnée si elle n'avait pas été déclarée.

Les utilitaires d'administration ne permettent pas de mettre à jour les propriétés des types de ressources à l'exception de `Noeuds_installés` qu'un administrateur doit définir. Vous ne pouvez pas déclarer cette propriété dans le fichier RTR.

TABLEAU A–1 Propriétés des types de ressources

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Autoriser_hôtes` (tableau de chaînes de caractères)	Cette propriété contrôle l'ensemble des clients autorisés à se connecter avec le démon `cl_apid` pour recevoir des événements de reconfiguration du cluster. Elle se présente généralement sous la forme `ipaddress/masklength` qui définit un sous-réseau à partir duquel les clients peuvent ouvrir une session. Par exemple, `129.99.77.0/24` permet aux clients du sous-réseau `129.99.77` de se connecter pour être avertis des événements tandis que `192.9.84.231/32` ne le permet qu'au client `192.9.84.231`. Cette propriété garantit la sécurité du protocole CRNP. Le démon `cl_apid` est décrit dans `SUNW.Event`(5). En outre, les mots clés spéciaux suivants sont reconnus. `LOCAL` fait référence à tous les clients appartenant aux sous-réseaux directement connectés du cluster. `ALL` permet à tous les clients d'ouvrir une session. Veuillez noter qu'un client ne peut pas ouvrir une session de mise en oeuvre s'il est référencé simultanément dans les propriétés `Autoriser_hôtes` et `Refuser_hôtes`. La valeur par défaut est `LOCAL`.	N	Facultative

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Version_API` (entier)	Version de l'API de gestion des ressources utilisée par cette mise en oeuvre du type de ressources. La valeur par défaut de Sun Cluster 3.1 4/04 est 2.	N	Facultative
`Initialisation` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute sur un noeud se connectant ou se reconnectant au cluster lorsqu'une ressource de ce type est déjà gérée. Les actions d'initialisation générées par cette méthode pour les ressources de ce type doivent normalement être identiques à celles de la méthode `Init`.	N	Conditionnelle/ explicite
`Nombre_nouvelles_ tentatives_client` (entier)	Cette propriété contrôle le nombre de fois (tentatives) que le démon `cl_apid` tente de communiquer avec des clients externes. Si un client ne répond pas au bout de `Nombre_nouvelles_tentatives` tentatives, son délai d'attente expire. Il est, par conséquent, supprimé de la liste des clients connectés susceptibles de recevoir des événements de reconfiguration du cluster. Il doit donc se reconnecter pour recevoir de nouveau des événements. Reportez-vous à la description de la propriété `Intervalle_ nouvelles_tentatives_client` pour obtenir plus d'informations sur la fréquence de ces essais par la mise en oeuvre. Le démon `cl_apid` est décrit dans `SUNW.Event`(5). La valeur par défaut est 3.	O	Facultative
`Intervalle_nouvelles_ tentatives_client` (entier)	Cette propriété définit le délai (en secondes) pris en compte par le démon `cl_apid` lorsqu'il tente de communiquer avec des clients externes qui ne répondent pas. Pendant ce laps de temps, jusqu'à `Nombre_nouvelles_tentatives_client` tentatives de communication avec le client sont exécutées. Le démon `cl_apid` est décrit dans `SUNW.Event`(5). La valeur par défaut est 1800.	O	Facultative

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Délai_client` (entier)	Cette propriété définit le délai d'attente (en secondes) pris en compte par le démon `cl_apid` lorsqu'il tente de communiquer avec des clients externes. Le démon `cl_apid` continue toutefois d'essayer de contacter le client pendant un nombre de fois ajustable. Reportez-vous à la description des propriétés `Nombre_nouvelles_tentatives_client` et `Intervalle_nouvelles_tentatives_client` pour plus d'informations sur les moyens dont vous disposez pour définir cette propriété. Le démon `cl_apid` est décrit dans `SUNW.Event`(5). La valeur par défaut est 60.	O	Facultative
`Refuser_hôtes` (tableau de chaînes de caractères)	Cette propriété contrôle l'ensemble des clients qui ne peuvent pas se connecter pour recevoir des événements de reconfiguration du cluster. Pour déterminer cet accès, les paramètres de cette propriété sont prioritaires sur ceux de la liste `Autoriser_hôtes`. Le format de cette propriété est identique à celui définit dans la propriété `Autoriser_hôtes`. Cette propriété garantit la sécurité du protocole CRNP. La valeur par défaut est `NULL`.	O	Facultative
`Basculement` (booléen)	`Vrai` indique qu'il est impossible de configurer les ressources de ce type dans un groupe pouvant être en ligne sur plusieurs noeuds à la fois. La valeur par défaut est `Faux`.	N	Facultative
`Fini` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute lorsqu'une ressource de ce type bascule en ressource non gérée.	N	Conditionnelle/ explicite
`Init` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute lorsqu'une ressource de ce type bascule en ressource gérée.	N	Conditionnelle/ explicite

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Noeuds_init` (énumération)	Valeurs disponibles : `Éléments_principaux_GR` (uniquement les noeuds pouvant gérer la ressource) ou `Noeuds_installés_TR` (tous les noeuds sur lesquels le type de ressource est installé). Cette propriété indique les noeuds sur lesquels le gestionnaire RGM exécute les méthodes `Init`, `Fini`, `Initialisation` et `Validation`. La valeur par défaut est `Éléments_principaux_GR`.	N	Facultative
`Noeuds_installés` (tableau de chaînes de caractères)	Liste des noms des noeuds du cluster sur lesquels le type de ressources est autorisé à fonctionner. Le gestionnaire RGM crée automatiquement cette propriété. La valeur peut être définie par l'administrateur du cluster. Cette propriété ne peut être déclarée dans le fichier RTR. Par défaut, elle concerne tous les noeuds du cluster.	O	Configuration possible par l'administrateur du cluster
`Max_clients` (entier)	Cette propriété contrôle le nombre maximum de clients pouvant ouvrir une session avec le démon `cl_apid` pour être avertis des événements du cluster. Par conséquent, l'application rejette les tentatives de connexion des autres clients. Comme chaque connexion client utilise des ressources du cluster, le réglage de cette propriété permet aux utilisateurs de contrôler l'utilisation par les clients externes des ressources du cluster. Le démon `cl_apid` est décrit dans `SUNW.Event`(5). La valeur par défaut est 1000.	O	Facultative
`Contrôle_détecteur` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme appelé par le gestionnaire RPM avant d'effectuer, sur demande du détecteur, le basculement d'une ressource de ce type.	N	Conditionnelle/ explicite
`Démarrage_détecteur` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute pour démarrer un système de détection des pannes pour une ressource de ce type.	N	Conditionnelle/ explicite

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Arrêt_détecteur` (chaîne de caractères)	Méthode de rappel requise si `Démarrage_détecteur` est défini : chemin d'accès au programme que le gestionnaire RGM exécute pour arrêter un système de détection des pannes pour une ressource de ce type.	N	Conditionnelle/ explicite
`Nombre_redémarrages_ ressource` sur chaque noeud du cluster (entier)	Cette propriété est définie par le gestionnaire RGM sur le nombre d'appels `scha_control RESTART` de cette ressource effectué sur ce noeud au cours des *n* secondes écoulées, *n* correspondant à la valeur de la propriété `Intervalle_nouvelles_tentatives` de la ressource. Si un type de ressources ne déclare pas la propriété `Intervalle_nouvelles_tentatives`, la propriété `Nombre_redémarrages_ressource` n'est pas disponible pour les ressources de ce type.	N	Interrogation uniquement
`Liste_packages` (tableau de chaînes de caractères)	Liste facultative des packages installés lors de l'installation du type de ressources.	N	Conditionnelle/ explicite
`Arrêt_après_réseau` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute après avoir appelé la méthode d'`Arrêt` de n'importe quelle ressource d'adresse réseau (`Ressources_réseau_utilisées`) dont une ressource de ce type dépend. Cette méthode d'arrêt doit normalement exécuter des actions d'`ARRÊT` après la configuration en aval des interfaces réseau.	N	Conditionnelle/ explicite
`Démarrage_avant_réseau` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute avant d'appeler d'appeler la méthode de `Démarrage` de n'importe quelle ressource d'adresse réseau (`Ressources_réseau_utilisées`) dont une ressource de ce type dépend. Cette méthode doit normalement exécuter des actions de `DEMARRAGE` avant la configuration en amont des interfaces réseau.	N	Conditionnelle/ explicite

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie

`Type_ressource` (chaîne de caractères)	Nom du type de ressources. Pour afficher les noms des types de ressources actuellement en ligne, utilisez :
	`scrgadm` `-p`
	`Sous Sun Cluster 3.1 et versions ultérieures, le nom d'un type de ressources comprend la version, ceci étant obligatoire :`
	`vendor_id.resource_type:version`
	`Les trois composants d'un nom de type de ressources correspondent aux propriétés spécifiées dans le fichier RTR file en tant que` *ID_fournisseur*, *Type_ressource* `et` *Version_TR*`. La commande scrgadm insère les délimiteurs (points et deux-points). Le suffixe Version_TR du nom du type de ressources correspond à la valeur de la propriété Version_TR. Pour avoir l'assurance que l'`*ID_fournisseur* `est unique, nous vous recommandons d'utiliser le symbole boursier de l'entreprise en tant que type de ressources. Les noms de type de ressources créés avec les versions antérieures à Sun Cluster 3.1 se présentent toujours sous la forme suivante :`
	`ID_fournisseur.Type ressource`
	`La chaîne de caractères est vide par défaut.`
		N	Requise

`Rép_base_TR` (chaîne de caractères)	Chemin d'accès au répertoire permettant de compléter les chemins d'accès relatifs des méthodes de rappel. Ce chemin doit être configuré conformément à l'emplacement où sont installés les packages du type de ressources. Il doit être complet, c'est-à-dire qu'il doit commencer par une barre oblique (/). Il n'est pas nécessaire de définir cette propriété si tous les noms de chemin d'accès des méthodes sont absolus.	N	Requise, à moins que les noms de chemin d'accès de toutes les méthodes ne soient absolus.
`Description_TR` (chaîne de caractères)	Brève description du type de ressources. La chaîne de caractères est vide par défaut.	N	Conditionnelle

TABLEAU A–1 Propriétés des types de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Version_TR` (chaîne de caractères)	Depuis Sun Cluster 3.1, une version obligatoire de cette mise en oeuvre du type de ressource. `Version_TR` représente le suffixe dans un nom de type de ressources complet. La propriété `Version_TR`, qui était facultative dans Sun Cluster 3.0, est désormais obligatoire sous Sun Cluster 3.1.	N	Conditionnelle/ explicite
`Instance_unique` (booléen)	Si cette propriété est définie sur `Vrai`, le cluster ne peut contenir qu'une seule ressource de ce type. Le gestionnaire RGM ne permet d'exécuter qu'une seule ressource de ce type à la fois à l'échelle du cluster. La valeur par défaut est `Faux`.	N	Facultative
`Démarrage` (chaîne de caractères)	Méthode de rappel : chemin d'accès au programme que le gestionnaire RGM exécute pour démarrer une ressource de ce type.	N	Requise à moins que le fichier RTR déclare une méthode `Démarrage_ avant_réseau`
`Arrêt` (chaîne de caractères)	Méthode de rappel : chemin d'accès au programme que le gestionnaire RGM exécute pour arrêter une ressource de ce type.	N	Requise à moins que le fichier RTR déclare une méthode `Arrêt_après_ réseau`
`Mise_à_jour` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute lorsque les propriétés d'une ressource en cours d'exécution de ce type sont modifiées.	N	Conditionnelle/ explicite
`Validation` (chaîne de caractères)	Méthode de rappel facultative : chemin d'accès au programme appelé pour contrôler les valeurs des propriétés des ressources de ce type.	N	Conditionnelle/ explicite
`ID_fournisseur` (chaîne de caractères)	Reportez-vous à la propriété `Type_ressource`.	N	Conditionnelle

Propriétés des ressources

Le Tableau A–2 décrit les propriétés des ressources définies par Sun Cluster. Les valeurs des propriétés sont classées comme suit (colonne Catégorie) :

- **Requise** : l'administrateur doit spécifier une valeur au moment de la création de la ressource à l'aide d'un utilitaire d'administration.
- **Facultative** : si l'administrateur ne spécifie pas de valeur au moment de la création d'un groupe de ressources, le système attribue une valeur par défaut.
- **Conditionnelle** : le gestionnaire RGM ne crée la propriété que si elle est déclarée dans le fichier RTR. Dans le cas contraire, la propriété n'existe pas et les administrateurs système ne peuvent y accéder. Une propriété conditionnelle déclarée dans le fichier RTR est soit facultative soit requise suivant qu'une valeur par défaut lui a été attribuée dans le fichier RTR ou non. Pour plus d'informations, reportez-vous à la description de toutes les propriétés conditionnelles.
- **Interrogation uniquement** : ne peut être directement définie par un outil d'administration.

Le Tableau A–2 précise également si et quand vous pouvez mettre à jour les propriétés des ressources (dans la colonne Mise à jour possible) comme suit :

`Aucune` ou `Faux`	Jamais
`Vrai` ou `À_tout_moment`	À tout moment
`À_la_création`	Lorsque la ressource est ajoutée à un cluster
`Lorsque_désactivée`	Lorsque la ressource est désactivée

TABLEAU A–2 Propriétés des ressources

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Délai_détermination_analogie` (entier)	Durée en secondes pendant laquelle les connexions de l'adresse IP d'un client donné pour n'importe quel service de la ressource sont transmises au même noeud du serveur. Cette propriété n'est appropriée que si `Règle_équilibrage_charge` est défini sur `Équilibrage_charge_sticky` ou `Équilibrage_charge_sticky_joker`. En outre, vous devez définir `Analogie_faible` sur le paramètre par défaut (Faux). Cette propriété n'est utilisée qu'avec les services évolutifs.	À tout moment	Facultative
`Intervalle_sonde_superficiel` (entier)	Durée en secondes entre les appels de détection rapide des pannes de la ressource. Cette propriété n'est créée que par le gestionnaire RGM. L'administrateur ne peut y accéder que si elle est déclarée dans le fichier RTR. Elle est facultative si une valeur par défaut est spécifiée dans le fichier RTR. Si l'attribut `Réglable` n'est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `Lorsque_désactivé`. Cette propriété est requise si l'attribut `Par_défaut` n'est pas spécifié dans la déclaration de propriétés dans le fichier RTR.	Lorsque désactivée	Conditionnelle
Propriétés d'extension	Propriétés d'extension telles qu'elles sont déclarées dans le fichier RTR du type de ressources. La mise en oeuvre du type de ressources définit ces propriétés. Pour plus d'informations sur les attributs individuels que vous pouvez configurer pour les propriétés d'extension, reportez-vous au Tableau A–4.	Dépend de chaque propriété	Conditionnelle

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Mode_basculement` (énumération)	Paramètres disponibles : `AUCUN`, `LOGICIEL` et `MATÉRIEL`. Cette propriété contrôle si le gestionnaire RGM transfère un groupe de ressources ou arrête un noeud après l'échec de l'exécution d'une méthode de `Démarrage`, d'`Arrêt` ou d'`Arrêt_détecteur` sur une ressource. `Aucun` indique que le gestionnaire RGM doit juste spécifier l'échec de la méthode au niveau de l'état de la ressource et attendre l'intervention de l'opérateur. `LOGICIEL` indique que, suite à l'échec d'une méthode de `Démarrage`, le gestionnaire RGM doit transférer le groupe de ressources sur un autre noeud. Par contre, suite à l'échec d'une méthode d'`Arrêt` ou d'`Arrêt_détecteur`, il doit basculer la ressource en mode `ÉCHEC_ARRÊT` et le groupe de ressources en mode `ERREUR_ÉCHEC_ARRÊT`, puis attendre l'intervention de l'opérateur. En cas d'échec d'`Arrêt` ou d'`Arrêt_détecteur`, les paramètres de `AUCUN` et `LOGICIEL` sont équivalents. `MATÉRIEL` indique que l'échec d'une méthode de `Démarrage` doit induire le transfert du groupe tandis que l'échec d'une méthode d'`Arrêt` ou d'`Arrêt_détecteur` doit induire l'arrêt forcé de la ressource par l'arrêt du noeud du cluster. La valeur par défaut est `Aucun`.	À tout moment	Facultative

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Règle_équilibrage_charge` (chaîne de caractères)	Chaîne de caractères définissant les règles d'équilibrage de la charge. Cette propriété n'est utilisée qu'avec les services évolutifs. Le logiciel RGM crée automatiquement cette propriété si la propriété `Évolutivité` est déclarée dans le fichier RTR. Valeurs possibles de Règle_équilibrage_charge : `Équilibrage_charge_pondéré` (par défaut). La charge est répartie entre plusieurs noeuds en fonction des poids définis dans la propriété `Poids_équilibrage_charge`. `Équilibrage_charge_sticky`. Un client donné (identifié par son adresse IP) du service évolutif est toujours envoyé au même noeud du cluster. `Équilibrage_charge_sticky_joker`. Un client donné (identifié par son adresse IP), connecté à l'adresse IP d'un service sticky à caractère joker est toujours envoyé sur le même noeud du cluster, indépendamment du port vers lequel il est dirigé. La valeur par défaut est `Équilibrage_charge_pondéré`.	À la création	Conditionnelle/ facultative

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Poids_équilibrage_charge` (tableau de chaînes de caractères)	Pour les ressources évolutives uniquement. Le logiciel RGM crée automatiquement cette propriété si la propriété `Évolutivité` est déclarée dans le fichier RTR. Format : *poids*@*noeud*, *poids*@*noeud*, *poids* correspondant à un nombre entier reflétant la part relative de la charge distribuée au *noeud* spécifié. La fraction de la charge distribuée à un noeud correspond au poids de ce noeud divisé par la somme de tous les poids. Par exemple, `1@1,3@2` indique que le noeud 1 reçoit 1/4 de la charge et que le noeud 2 en reçoit les 3/4. La chaîne de caractères vide (“”) définit une répartition uniforme. Il s’agit de la valeur par défaut. Les noeuds auxquels aucun poids explicite n’est attribué reçoivent un poids par défaut de 1. Si l’attribut `Réglable` n’est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `À_tout_moment`. En changeant cette propriété, vous modifiez uniquement la répartition des nouvelles connexions. La valeur par défaut est la chaîne de caractères vide (“”).	À tout moment	Conditionnelle/ facultative
Méthode`_délai_éxecution` de chaque méthode de rappel dans le Type (entier)	Intervalle de temps en secondes après lequel le gestionnaire RGM conclut que l’exécution d’une méthode a échoué. La valeur par défaut est 3600 (une heure) si la méthode est déclarée dans le fichier RTR.	À tout moment	Conditionnelle/ facultative
`Commutateur_contrôlé` (énumération)	Le gestionnaire RGM définit cette propriété sur `Activé` ou `Désactivé` si l’administrateur du cluster active ou désactive le détecteur à l’aide d’un utilitaire d’administration. Si `Désactivé` est sélectionné, la méthode de `Démarrage` du détecteur n’est pas exécutée tant que le détecteur n’a pas été réactivé. Si la ressource n’a pas de méthode de rappel du détecteur, cette propriété n’existe pas. La valeur par défaut est `Activé`.	Jamais	Interrogation uniquement

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Ressources_réseau_utilisées` (tableau de chaînes de caractères)	Liste des ressources (noms d'hôte logique et adresses partagées) utilisées par la ressource. Pour les services évolutifs, cette propriété doit se référer aux ressources d'adresse partagée qui se trouvent dans un groupe de ressources séparé. Pour les services à basculement, cette propriété se réfère aux noms d'hôte logique ou aux ressources d'adresse partagée qui se trouvent dans le même groupe de ressources. Le logiciel RGM crée automatiquement cette propriété si la propriété `Évolutivité` est déclarée dans le fichier RTR. Si elle n'est pas déclarée dans le fichier RTR, `Ressources_réseau_utilisées` n'est pas disponible à moins que cette propriété soit explicitement déclarée dans le fichier RTR. Si l'attribut `Réglable` n'est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `À_la_création`.	À la création	Conditionnelle/ requise
`Basculement_activé_desactivé` (énumération)	Le gestionnaire RGM définit cette propriété sur `Activé` ou `Désactivé` si l'administrateur du cluster active ou désactive la ressource à l'aide d'un utilitaire d'administration. Si elle est désactivée, aucun rappel ne peut être effectué pour la ressource ; il faut attendre qu'elle soit réactivée. La valeur par défaut est `Désactivé`.	Jamais	Interrogation uniquement

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Liste_ports` (tableau de chaînes de caractères)	Liste des numéros de port de réception du serveur. À la suite de chaque numéro de port se trouve le protocole utilisé par le port, par exemple, `Liste_ports=80/tcp`. Si la propriété `Évolutivité` est déclarée dans le fichier RTR, le gestionnaire RGM crée automatiquement `Liste_ports`. Dans le cas contraire, la propriété est indisponible à moins qu'elle ne soit explicitement déclarée dans le fichier RTR. Le paramétrage de cette propriété pour Apache est décrite dans le document *Sun Cluster Data Service for Apache Guide for Solaris OS*.	À la création	Conditionnelle/ requise
`Description_R` (chaîne de caractères)	Brève description de la ressource. La chaîne de caractères est vide par défaut.	À tout moment	Facultative
`Nom_ressource` (chaîne de caractères)	Nom de l'instance de la ressource. Ce nom doit être unique au sein de la configuration du cluster. Vous ne pouvez plus le modifier après avoir créé une ressource.	Jamais	Requise

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Nom_projet_ressource` (chaîne de caractères)	Nom de projet Solaris associé à la ressource. Cette propriété permet d'appliquer aux services de données du cluster, les fonctions de gestion de ressources de Solaris, telles que le partage d'unité centrale ou les pools de ressources. Lorsque le gestionnaire RGM connecte les ressources, il exécute les processus connexes sous ce nom de projet. Si cette propriété n'est pas spécifiée, le nom de projet est issu de la propriété `Nom_projet_GR` du groupe de ressources qui contient la ressource (reportez-vous à `rg_properties` (5)). Si aucune propriété n'est spécifiée, le gestionnaire RGM utilise le nom de projet prédéfini `par_défaut`. Le nom de projet spécifié doit se trouver dans la base de données des projets et le superutilisateur `root` doit être configuré comme membre du projet nommé. Cette propriété n'est prise en charge que si elle est exécutée sous Solaris 9. **Remarque –** toute modification apportée à cette propriété ne prend effet qu'après le redémarrage de la ressource. La valeur par défaut est Null.	À tout moment	Facultative
`État_ressource` sur chaque noeud du cluster (énumération)	État de la ressource déterminé par le gestionnaire RGM sur chaque noeud du cluster. États possibles : `En_ligne`, `Hors_ligne`, `Échec_arrêt`, `Échec_démarrage`, `Échec_contrôle` et `En_ligne_non_contrôlé`. Cette propriété n'est pas configurable par l'utilisateur.	Jamais	Interrogation uniquement

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Nombre_nouvelles_tentatives` (entier)	Nombre de fois qu'un détecteur tente de redémarrer une ressource si celle-ci échoue. Cette propriété n'est créée que par le gestionnaire RGM. L'administrateur ne peut y accéder que si elle est déclarée dans le fichier RTR. Elle est facultative si une valeur par défaut est spécifiée dans le fichier RTR. Si l'attribut `Réglable` n'est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `Lorsque_désactivé`. Cette propriété est requise si l'attribut `Par_défaut` n'est pas spécifié dans la déclaration de propriétés dans le fichier RTR.	Lorsque désactivée	Conditionnelle
`Intervalle_nouvelles_tentatives` (entier)	Intervalle de temps en secondes entre les tentatives de redémarrage d'une ressource qui a échoué. Le détecteur de ressources utilise cette propriété en association avec `Nombre_nouvelles_tentatives`. Cette propriété n'est créée que par le gestionnaire RGM. L'administrateur ne peut y accéder que si elle est déclarée dans le fichier RTR. Elle est facultative si une valeur par défaut est spécifiée dans le fichier RTR. Si l'attribut `Réglable` n'est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `Lorsque_désactivé`. Cette propriété est requise si l'attribut `Par_défaut` n'est pas spécifié dans la déclaration de propriétés dans le fichier RTR.	Lorsque désactivée	Facultative

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Évolutivité` (booléen)	Cette propriété indique si la ressource est évolutive. Si cette propriété est déclarée dans le fichier RTR, le logiciel RGM crée automatiquement les propriétés de service évolutif pour les ressources de ce type : `Ressources_réseau_utilisées`, `Liste_ports`, `Règle_équilibrage_charge` et `Poids_équilibrage_charge`. Ces propriétés sont définies sur leur valeur par défaut à moins qu'elles ne soient spécifiquement déclarées dans le fichier RTR. La valeur par défaut de la propriété `Évolutivité`, lorsqu'elle est déclarée dans le fichier RTR, est `Vrai`. Lorsque cette propriété n'est pas déclarée dans le fichier RTR, l'attribut `Réglable` doit être défini sur `À_la_création`, sinon la création de la ressource échoue. Si cette propriété n'est pas déclarée dans le fichier RTR, la ressource n'est pas évolutive, l'administrateur du cluster ne peut pas la paramétrer et aucune propriété de service évolutif n'est définie par le gestionnaire RGM. Vous pouvez cependant explicitement déclarer les propriétés `Ressources_réseau_utilisées` et `Liste_ports` dans le fichier RTR, si vous le souhaitez, car elles peuvent tout autant s'avérer utiles dans un service non évolutif qu'évolutif.	À la création	Facultative
`Statut` sur chaque noeud du cluster (énumération)	Cette propriété est définie par le détecteur de ressources. Valeurs disponibles : `OK`, `défectueux`, `par_défaut`, `inconnu` et `hors_ligne`. Le gestionnaire RGM définit la valeur sur `inconnu` à la mise en ligne de la ressource et sur `hors_ligne` à sa mise hors ligne.	Jamais	Interrogation uniquement
`Msg_statut` sur chaque noeud du cluster (chaîne de caractères)	Cette propriété est définie par le détecteur de ressources en même temps que la propriété `Statut`. Il est possible de la configurer par ressource et par noeud. Le logiciel RGM lui attribue une chaîne de caractères vide lorsque la ressource est mise hors ligne.	Jamais	Interrogation uniquement

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Intervalle_sonde_complet` (entier)	Durée en secondes entre les appels de détection des pannes nécessitant un temps système important de la ressource. Cette propriété n'est créée que par le gestionnaire RGM. L'administrateur ne peut y accéder que si elle est déclarée dans le fichier RTR. Elle est facultative si une valeur par défaut est spécifiée dans le fichier RTR. Si l'attribut `Réglable` n'est pas spécifié dans le fichier du type de ressources, la valeur `Réglable` de la propriété est `Lorsque_désactivé`. Cette propriété est requise si l'attribut `Par_défaut` n'est pas spécifié dans la déclaration de propriétés dans le fichier RTR.	Lorsque désactivée	Conditionnelle
`Type` (chaîne de caractères)	Type de ressource dont cette ressource est une instance.	Jamais	Requise
`Version_type` (chaîne de caractères)	Indique la version de type de ressources actuellement associée à cette ressource. Le gestionnaire RGM crée automatiquement cette propriété qui ne peut être déclarée dans le fichier RTR. La valeur de cette propriété correspond à la propriété `Version_TR` du type de ressources. La propriété `Version_type` n'est pas spécifiée explicitement à la création d'une ressource bien qu'elle puisse apparaître sous la forme d'un suffixe du nom du type de ressources. Lorsqu'une ressource est éditée, la valeur de `Version_type` peut être modifiée. Sa capacité de réglage provient de : ■ la version actuelle du type de ressources ; ■ l'instruction `#$upgrade_from` figurant dans le fichier RTR.	Voir la description	Voir la description

TABLEAU A–2 Propriétés des ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Analogie_UDP` (booléen)	Si cette propriété est définie sur Vrai, l'intégralité du trafic UDP provenant d'un client donné est envoyé au noeud de serveur qui gère actuellement tout le trafic TCP de ce client. Cette propriété n'est appropriée que si `Règle_équilibrage_charge` est défini sur `Équilibrage_charge_sticky` ou `Équilibrage_charge_sticky_joker`. En outre, vous devez définir `Analogie_faible` sur le paramètre par défaut (Faux). Cette propriété n'est utilisée qu'avec les services évolutifs.	Lorsque désactivée	Facultative
`Analogie_faible` (booléen)	Si cette propriété est définie sur Vrai, cette propriété active l'affinité faible du client. Ceci permet d'envoyer au même noeud de serveur les connexions provenant d'un client donné, hormis dans les cas suivants : ■ Démarrage du module d'écoute d'un serveur, suite par exemple, au redémarrage du système de détection des pannes, au basculement ou à la commutation des ressources ou à la reconnexion d'un noeud au cluster après une panne. ■ Lorsque la propriété `Poids_équilibrage_charge` d'une ressource évolutive change suite à une action administrative. Cette propriété offre une alternative de faible déperdition par opposition à la configuration par défaut, tant en termes d'utilisation de la mémoire que de cycles du processeur. Cette propriété n'est appropriée que si `Règle_équilibrage_charge` est défini sur `Équilibrage_charge_sticky` ou `Équilibrage_charge_sticky_joker`. Cette propriété n'est utilisée qu'avec les services évolutifs.	Lorsque désactivée	Facultative

Propriétés des groupes de ressources

Le tableau suivant présente les propriétés des groupes de ressources définis par Sun Cluster. Les valeurs des propriétés sont classées comme suit (colonne Catégorie) :

- **Requise** : l'administrateur doit spécifier une valeur au moment de la création du groupe de ressources à l'aide d'un utilitaire d'administration.
- **Facultative** : si l'administrateur ne spécifie pas de valeur au moment de la création d'un groupe de ressources, le système attribue une valeur par défaut.
- **Interrogation uniquement** : ne peut être directement définie par un outil d'administration.

La colonne Mise à jour possible indique si la propriété peut être mise à jour (O) ou non (N) après avoir été définie.

TABLEAU A–3 Propriétés des groupes de ressources

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Démarrage_automatique_sur_nouveau_cluster` (booléen)	Cette propriété interdit le démarrage automatique d'un groupe de ressources lorsqu'un nouveau cluster est formé. La valeur par défaut est `VRAI`. S'il est configuré sur `VRAI`, le gestionnaire RGM essaie de démarrer le groupe de ressources automatiquement afin d'obtenir les `Éléments_principaux_souhaités` lors de la réinitialisation du cluster. S'il est configuré sur `FAUX`, le groupe de ressources ne démarre pas automatiquement lors de la réinitialisation du cluster.	O	Facultative
`Éléments_principaux_souhaités` (entier)	Nombre de noeuds sur lesquels vous souhaitez que le groupe puisse être en ligne simultanément. La valeur par défaut est 1. Si la propriété `Mode_GR` est définie sur `Basculement`, sa valeur ne doit pas être supérieure à 1. Si la propriété `Mode_GR` est définie sur `Évolutivité`, vous pouvez lui octroyer une valeur supérieure à 1.	O	Facultative

TABLEAU A–3 Propriétés des groupes de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Basculement` (booléen)	Valeur booléenne indiquant s'il faut recalculer l'ensemble de noeuds sur lesquels le noeud est actif lorsque les membres du cluster changent. Un recalcul peut amener le gestionnaire RGM à déconnecter le groupe de noeuds n'étant pas des noeuds de prédilection et à les connecter aux noeuds de prédilection. La valeur par défaut est `Faux`.	O	Facultative
`Ressources_globales_utilisées` (tableau de chaînes de caractères)	Indique si les systèmes de fichiers du cluster sont utilisés par une ressource de ce groupe. L'administrateur peut utiliser deux valeurs : l'astérisque (*) pour indiquer toutes les ressources globales et la chaîne de caractère vide (“”) pour n'indiquer aucune ressource globale. La valeur par défaut est définie sur toutes les ressources globales.	O	Facultative
`Dépendances_réseau_implicites` (booléen)	Valeur booléenne indiquant, lorsque `Vrai` est défini, que le gestionnaire RGM doit appliquer les dépendances implicites fortes des ressources d'adresse non réseau aux ressources d'adresses réseau au sein du groupe. Les ressources d'adresse réseau comprennent les ressources de type nom d'hôte logique et adresse partagée. Dans un groupe de ressources évolutives, cette propriété n'a aucune incidence car un groupe de ressources évolutives ne contient pas de ressources d'adresse réseau. La valeur par défaut est `Vrai`.	O	Facultative
`Éléments_principaux_max.` (entier)	Nombre maximum de noeuds auxquels le groupe peut être connecté en même temps. La valeur par défaut est 1. Si la propriété `Mode_GR` est définie sur `Basculement`, sa valeur ne doit pas être supérieure à 1. Si la propriété `Mode_GR` est définie sur `Évolutivité`, vous pouvez lui octroyer une valeur supérieure à 1.	O	Facultative

TABLEAU A–3 Propriétés des groupes de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Liste_noeuds` (tableau de chaînes de caractères)	Liste des noeuds du cluster sur lesquels le groupe peut être mis en ligne suivant un ordre de prédilection. Ces noeuds correspondent aux noeuds principaux potentiels ou maîtres du groupe de ressources. La valeur par défaut correspond à la liste de tous les noeuds du cluster.	O	Facultative
`Préfixe_chemin_accès` (chaîne de caractères)	Répertoire du système de fichiers du cluster dans lequel les ressources du groupe peuvent écrire des fichiers d'administration essentiels. Certaines ressources demandent cette propriété. Faites en sorte que `Préfixe_chemin_accès` soit unique pour chaque groupe de ressources. La chaîne de caractères est vide par défaut.	O	Facultative

TABLEAU A–3 Propriétés des groupes de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Intervalle_transfert` (entier)	Nombre entier positif (en secondes) utilisé par le gestionnaire RGM pour déterminer où mettre en ligne un groupe de ressources après une reconfiguration ou l'exécution de la commande `scha_control -O GIVEOVER` ou de la fonction `scha_control()` avec l'argument `SCHA_GIVEOVER`. Dans le cadre d'une reconfiguration : si la mise en ligne du groupe de ressources échoue plus d'une fois au cours des `Intervalle_transfert` dernières secondes sur un noeud spécifique (car la méthode de `Démarrage` ou de `Démarrage_avant_réseau` de la ressource n'est pas définie sur zéro ou que son délai d'attente est dépassé), ce noeud ne peut pas héberger le groupe de ressources. Dans ce cas, le gestionnaire RGM recherche un autre noeud. Si l'exécution de la commande `scha_control` ou de la fonction `scha_control()` d'une ressource déconnecte le groupe de ressources d'un noeud particulier au cours des `Intervalle_transfert` dernières secondes, ce noeud ne peut pas héberger le groupe de ressources à cause de l'exécution ultérieure de la fonction `scha_control()` à partir d'un autre noeud. La valeur par défaut est 3600 (une heure).	O	Facultative
`Liste_ressources` (tableau de chaînes de caractères)	Liste des ressources contenues dans le groupe. L'administrateur ne peut pas définir directement cette propriété. En fait, le gestionnaire RGM met à jour cette propriété lorsque l'administrateur ajoute ou supprime des ressources du groupe de ressources. La liste est vide par défaut.	N	Interrogation uniquement
`Description_groupe_ressources` (chaîne de caractères)	Brève description du groupe de ressources. La chaîne de caractères est vide par défaut.	O	Facultative

TABLEAU A–3 Propriétés des groupes de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Mode_GR` (énumération)	Cette propriété indique si le groupe de ressources est un groupe évolutif ou de basculement. Si elle est définie sur `Basculement`, le gestionnaire RGM définit la propriété `Éléments_principaux_max.` du groupe sur 1. Par conséquent, le groupe de ressources ne peut être géré que par un seul noeud. Si elle est définie sur `Évolutivité`, le gestionnaire RGM permet à la propriété `Éléments_principaux_max.` d'être configurée sur une valeur supérieure à 1. Le cas échéant, le groupe peut être géré par plusieurs noeuds en même temps. Le gestionnaire RGM n'autorise pas l'ajout d'une ressource dont la propriété `Basculement` est définie sur `Vrai` dans un groupe de ressources dont la propriété `Mode_GR` est définie sur `Évolutivité`. Valeur par défaut : `Basculement` si `Éléments_principaux_max.` est défini sur 1 et `Évolutivité` si `Éléments_principaux_max.` possède une valeur supérieure.	N	Facultative
`Nom_groupe_ressources` (chaîne de caractères)	Nom du groupe de ressources. Ce nom doit être unique au sein du cluster.	N	Requise

TABLEAU A–3 Propriétés des groupes de ressources *(Suite)*

Nom de la propriété	Description	Mise à jour possible	Catégorie
`Nom_projet_GR` (chaîne de caractères)	Nom du projet Solaris associé au groupe de ressources. Cette propriété permet d'appliquer aux services de données du cluster, les fonctions de gestion de ressources de Solaris, telles que le partage d'unité centrale ou les pools de ressources. Lorsque le gestionnaire RGM connecte des groupes de ressources, il exécute des processus connexes sous ce nom de projet pour les ressources dont la propriété `Nom_projet_ressources` n'a pas été définie. Le nom de projet spécifié doit se trouver dans la base de données des projets et le superutilisateur `root` doit être configuré comme membre du projet nommé. Cette propriété n'est prise en charge que si elle est exécutée sous Solaris 9. **Remarque –** toute modification apportée à cette propriété ne prend effet qu'après le redémarrage de la ressource.	À tout moment	Requise
`État_GR` sur chaque noeud du cluster (énumération)	Cette propriété est définie par le gestionnaire RGM sur `En_ligne`, `Hors_ligne`, `En_ligne_en_attente`, `Hors_ligne_en_attente`, `En_ligne_en_attente_bloqué`, `Erreur_échec_arrêt` ou `En_ligne_par_défaut` pour décrire l'état du groupe sur chaque noeud du cluster. Cette propriété n'est pas configurable par l'utilisateur. Vous pouvez toutefois la configurer indirectement en exécutant `scswitch`(1M) (ou les commandes `scsetup`(1M) ou SunPlex Manager équivalentes).	N	Interrogation uniquement

Attributs des propriétés de ressources

Le tableau suivant présente les attributs des propriétés de ressources pouvant être utilisés pour modifier les propriétés définies par le système ou créer des propriétés d'extension.

Attention – vous ne pouvez pas définir les types `booléen`, `énum` ou `int` sur les valeurs par défaut suivantes : `NULL` ou une chaîne de caractères vide (“”) .

TABLEAU A–4 Attributs des propriétés de ressources

Propriété	Description
`Propriété`	Nom de la propriété de ressource.
`Extension`	Indique que l’entrée du fichier RTR déclare une propriété étendue définie par la mise en oeuvre du type de ressources. Si cet attribut n’apparaît pas, l’entrée correspond à une propriété définie par le système.
`Description`	Chaîne de caractères présentant une brève description de la propriété. L’attribut Description ne peut pas être utilisé dans le fichier RTR pour les propriétés définies par le système.
Type de propriété	Types permis : `string`, `booléen`, `int`, `énum` et `tableau_chaînes`. L’attribut type ne peut pas être utilisé dans le fichier RTR pour les propriétés définies par le système. Le type détermine les valeurs de propriété et les types d’attributs autorisés dans les entrées du fichier RTR. Le type `enum` correspond à un ensemble de valeurs de chaînes de caractères.
`Par_défaut`	Indique une valeur par défaut pour la propriété.
`Réglable`	Indique si l’administrateur du cluster peut lui-même définir la valeur de cette propriété dans une ressource. L’administrateur ne peut pas définir cette propriété si `Aucun` ou `Faux` est configuré. Valeurs autorisant l’administrateur à effectuer des réglages : `Vrai` ou `À_tout_moment` (à tout moment), `À_la_création` (uniquement à la création de la ressource) ou `Lorsque_désactivé` (lorsque la ressource est hors ligne). La valeur par défaut est `Vrai` (`À_tout_moment`).
`Enumlist`	Pour un type `énum`, ensemble de valeurs de chaînes de caractères permises pour la propriété.
`Min`	Pour un type `int`, valeur minimale permise pour la propriété.
`Max`	Pour un type `int`, valeur maximale permise pour la propriété.
`Longueur_min`	Pour les types `chaîne` et `tableau_chaîne`, longueur minimum permise.
`Longueur_max`	Pour les types `chaîne` et `tableau_chaîne`, longueur maximum permise.
`Taille_min_tableau`	Pour le type `tableau_chaîne`, nombre minimum d’éléments de tableau permis.
`Taille_max_tableau`	Pour le type `tableau_chaîne`, nombre maximum d’éléments de tableau permis.

ANNEXE **B**

Liste des codes de service de données échantillon

Cette annexe fournit le code complet de chaque méthode du service de données échantillon, ainsi que le contenu du fichier d'enregistrement du type de ressources.

Cette annexe présente les programmes suivants :

Liste de code du fichier RTR

Le fichier RTR (Resource Type Registration) contient les déclarations de propriétés de ressource définissant la configuration initiale du service de données au moment ou l'administrateur du cluster enregistre ce service.

EXEMPLE B–1 Fichier RTR `SUNW.Sample`

```
#
# Copyright (c) 1998-2004 par Sun Microsystems, Inc.
# Tous droits réservés.
#
# Données d'enregistrement du service de noms de domaine (DNS)
#
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

```
#pragma ident   "@(#)SUNW.sample   1.1   00/05/24 SMI"

TYPE_RESSOURCE = "sample";
ID_FOURNISSEUR = SUNW;
DESCRIPTION_TR = "Domain Name Service on Sun Cluster";

VERSION_TR ="1.0";
VERSION_API = 2;
BASCULEMENT = TRUE;

RÉP_BASE_TR=/opt/SUNWsample/bin;
LISTE_PACKAGES = SUNWsample;

DÉMARRAGE               = démarrage_svc_dns;
ARRÊT                   = arrêt_svc_dns;

VALIDATION              = validation_dns;
ACTUALISATION           = actualisation_dns;

DÉMARRAGE_DÉTECTEUR     = démarrage_détecteur_dns;
ARRÊT_DÉTECTEUR         = arrêt_détecteur_dns;
CONTRÔLE_DÉTECTEUR      = contrôle_détecteur_dns;

# Une liste des déclarations de propriétés de ressource entre crochets suit les
# déclarations du type de ressources. La déclaration du nom de propriété doit
# être le premier attribut suivant le crochet d'ouverture de chaque entrée.
#

# Les propriétés <délai_exécution_[méthode] définissent le délai en secondes passé lequel
# le gestionnaire RGM conclut que l'appel de la méthode a échoué.

#La valeur MIN du délai d'attente de toutes les méthodes est de 60 secondes.
# De cette manière, les administrateurs système ne peuvent pas définir de délais d'attente
# plus courts qui n'améliorent pas les performances de commutation/basculement et peuvent
# conduire le gestionnaire RGM à exécuter des actions non souhaitées (basculements
# erronés, réinitialisation de noeuds ou basculement du groupe de ressources en mode
# ERREUR_ÉCHEC_ARRÊT) nécessitant l'intervention de l'opérateur. La configuration
# de délais d'attente trop courts induit une *diminution* de la disponibilité globale
# du service de données.
{
   PROPRIÉTÉ = Délai_démarrage ;
   MIN=60 ;
  PAR DÉFAUT=300 ;
}

{
            PROPRIÉTÉ  = Délai_arrêt ;
            MIN=60 ;
           PAR DÉFAUT=300 ;
}
{
        PROPRIÉTÉ =  Délai_validation ;
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

```
        MIN=60 ;
        PAR DÉFAUT=300 ;
}
{
        PROPRIÉTÉ = Délai_mise_à_jour ;
        MIN=60 ;
        PAR DÉFAUT=300 ;
}
{
        PROPRIÉTÉ = Délai_démarrage_détecteur ;
        MIN=60 ;
        PAR DÉFAUT=300 ;
}
{
        PROPRIÉTÉ = Délai_arrêt_détecteur ;
        MIN=60 ;
        PAR DÉFAUT=300 ;
}
{
        PROPRIÉTÉ = Intervalle_sonde_complet ;
        MIN=1 ;
        MAX=3600 ;
        PAR DÉFAUT=60 ;
        RÉGLABLE = À_TOUT_MOMENT ;
}

# Nombre de relances à effectuer pendant un intervalle de temps prédéterminé avant de
# conclure qu'il est impossible de démarrer correctement l'application sur ce noeud.
{
        PROPRIÉTÉ = Nombre_nouvelles_tentatives;
        MIN=0 ;
        MAX=10 ;
        PAR DÉFAUT=2 ;
        RÉGLABLE = À_TOUT_MOMENT ;
}

# Définissez Intervalle_nouvelles_tentatives sur une valeur multiple de 60, ce paramètre
# configuré en secondes étant converti en minutes avec arrondissement au chiffre supérieur.
# Par exemple, 50 secondes est converti en 1 minute. Utilisez cette propriété pour
# programmer le nombre de relances (Nombre_nouvelles_tentatives).
{
       PROPRIÉTÉ  = Intervalle_nouvelles_tentatives ;
        MIN=60;
        MAX=3600;
        PAR DÉFAUT=300;
       RÉGLABLE = À_TOUT_MOMENT;
}

{
       PROPRIÉTÉ= Ressources_réseau_utilisées;
        RÉGLABLE = À_LA_CRÉATION;
        PAR DÉFAUT = "";
}
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

```
#
# Propriétés d'extension
#

# L'administrateur du cluster doit définir la valeur de cette propriété pour indiquer
# le répertoire contenant les fichiers de configuration utilisés par l'application.
# Pour cette application, DNS spécifie le chemin d'accès au fichier de configuration
# du service de noms de domaine sur le PXFS (en règle générale named.conf).
{
   PROPRIÉTÉ = Rép_conf;
   EXTENSION;
   CHAÎNE;
   RÉGLABLE = À_LA_CRÉATION;
   DESCRIPTION = "The Configuration Directory Path";
}

# Délai d'attente en secondes avant de déclarer que la détection a échoué.
{
        PROPRIÉTÉ = Délai_sonde;
        EXTENSION;
        INT;
        PAR DÉFAUT = 30;
        RÉGLABLE = À_TOUT_MOMENT;
        DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Méthode `Démarrage`

Le gestionnaire RGM appelle la méthode `Démarrage` sur un noeud du cluster lorsque le groupe de ressources contenant la ressource du service de données est connecté à ce noeud ou lorsque la ressource est activée. Dans l'application échantillon, la méthode `Démarrage` active le démon `in.named` (système de nom de domaine) sur ce noeud.

EXEMPLE B–2 Méthode `Démarrage_svc_dns`

```
#!/bin/ksh
#
# Méthode de démarrage de HA-DNS.
#
# Cette méthode démarre le service de données sous le contrôle de la fonction PMF.
# Avant de démarrer le processus in.named du service de noms de domaine, elle exécute
# quelques contrôles " sanitaires '. La balise de la fonction PMF pour le service de
# données est $RESOURCE_NAME.named. La fonction PMF tente de démarrer le service suivant
# un nombre de fois spécifié (Nombre_nouvelles_tentatives) et si le nombre de tentatives
# est supérieur à cette valeur dans l'intervalle de temps spécifié
# (Intervalle_nouvelles_tentatives), la fonction  PMF notifie un échec au démarrage
```

EXEMPLE B–2 Méthode `Démarrage_svc_dns` *(Suite)*

```
# du service. Nombre_nouvelles_tentatives et Intervalle_nouvelles_tentatives sont des
# propriétésde l'ensemble des ressources du fichier RTR.


#pragma ident   "@(#)dns_svc_start   1.1   00/05/24 SMI"

################################################################################
# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
        typeset opt

         while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}



################################################################################
# MAIN
#
################################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH
```

EXEMPLE B–2 Méthode `Démarrage_svc_dns` *(Suite)*

```
# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Accédez à la valeur de la propriété Rép_conf de la ressource afin de démarrer le
# service de noms de domaine. Entrez le nom de la ressource et le groupe de ressources,
# recherchez la valeur Rép_conf définie par l'administrateur du cluster lors de l'ajout
# de la ressource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get retourne le "type" et la "valeur" de la propriété d'extension.
# Accédez uniquement à la valeur de la propriété d'extension.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Vérifiez que $CONFIG_DIR est accessible.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} Directory $CONFIG_DIR missing or not mounted"
   exit 1
fi

# Configurez le répertoire $CONFIG_DIR si les fichiers de données contiennent
# des noms de chemin d'accès relatifs.
cd $CONFIG_DIR

# Vérifiez que le répertoire $CONFIG_DIR contient bien le fichier named.conf.
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} File $CONFIG_DIR/named.conf is missing or
 empty"
   exit 1
fi

# Accédez à la valeur de Nombre_nouvelles_tentatives à partir du fichier RTR.
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Accédez à la valeur Intervalle_nouvelles_tentatives à partir du fichier RTR.
# Convertissez cette valeur, à la base en secondes, en minutes pour le transfert
# vers pmfadm. Veuillez noter qu'il s'agit d'une conversion avec arrondissement :
# par exemple, 50 secondes devient 1 minute.
((RETRY_INTRVAL = `scha_resource_get -O Retry_Interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ 60))

# Démarrez le démon in.named sous le contrôle de la fonction PMF. Laissez-le
# s'interrompre et redémarrer jusqu'à ce que la valeur $RETRY_COUNT dans l'intervalle
# $RETRY_INTERVAL soit atteinte. En cas d'autre échec,la fonction PMF cesse d'essayer
# de le redémarrer. Si un processus est déjà enregistré sous la balise
```

EXEMPLE B–2 Méthode `Démarrage_svc_dns` *(Suite)*

```
# <$PMF_TAG>, la fonction PMF envoie un message d'alerte indiquant
# que le processus est toujours opérationnel.
echo "Retry interval is "$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Consignez un message indiquant que HA-DNS a démarré.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Méthode `Arrêt`

La méthode `Arrêt` est appelée sur un noeud du cluster lorsque le groupe de ressources contenant la ressource HA-DNS est déconnecté sur ce noeud ou que la ressource est désactivée. Cette méthode arrête le démon `in.named` (système de nom de domaine) sur ce noeud.

EXEMPLE B–3 Méthode `Arrêt_svc_dns`

```
#!/bin/ksh
#
# Méthode d'arrêt de HA-DNS
#
# Arrêtez le service de données à l'aide de la fonction PMF. Si le service ne fonctionne pas
# la méthode se ferme avec l'état 0, le retour de toute autre valeur basculant la ressource
# en mode ÉCHEC_ARRÊT.


#pragma ident   "@(#)dns_svc_stop   1.1   00/05/24 SMI"

################################################################################
# Arguments d'analyse du programme.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
```

EXEMPLE B–3 Méthode Arrêt_svc_dns *(Suite)*

```
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}


##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Obtenez la valeur Délai_arrêt à partir du fichier RTR.
DÉLAI_ARRÊT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME
-G \ $RESOURCEGROUP_NAMÈ

# Essayez d'arrêter le service de données de façon ordonnée à l'aide d'un signal SIGTERM
# au moyen de la fonction PMF. Attendez pendant 80% maximum de la valeur Délai_arrêt pour
# voir si SIGTERM parvient à arrêter le service de données correctement. Dans le
# cas contraire, envoyez un signal SIGKILL pour arrêter le service de données. Utilisez
# jusqu'à 15 % de la valeur Délai_arrêt pour voir si le signal SIGKILL offre les résultats
# escomptés. Dans le cas contraire, l'arrêt échoue et la méthode se ferme avec un état
# différent de zéro. Les 5% restant de Délai_arrêt sont dédiés à une autre utilisation.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))
```

EXEMPLE B–3 Méthode `Arrêt_svc_dns` *(Suite)*

```
((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))

# Vérifiez si in.named est exécuté. Le cas échéant, interrompez-le.
if pmfadm -q $PMF_TAG.named; then
   # Envoyez un signal SIGTERM au service de données et attendez pendant 80 % # de la
   # valeur totale du délai d'attente.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry
 with \
           SIGKILL"

      # Comme le service de données ne s'est pas arrêté avec le signal SIGTERM, utilisez
      # le signal SIGKILL et attendez pendant 15% de la valeur totale du délai d'attente.
      pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG]
 \
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL"

          exit 1
      fi
fi
else
   # Le service de données n'est plus en cours d'exécution. Consignez un message et
   # quittez avec succès.
   logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
           "HA-DNS is not started"

   # Même si HA-DNS n'est pas exécuté, quittez avec succès pour éviter de basculer
   # le service de données en mode ÉCHEC_ARRÊT.

   exit 0

fi

# Le service de noms de domaine se ferme correctement. Consignez un message et quittez
# avec succès.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

Utilitaire gettime

L'utilitaire gettime est un programme C utilisé par le programme de SONDE pour suivre l'intervalle de temps entre les redémarrages de la détection. Vous devez compiler ce programme et le placer dans le même répertoire que les méthodes de rappel, c'est-à-dire, le répertoire indiqué dans la propriété rép_base_TR.

EXEMPLE B–4 Programme utilitaire gettime.c

```
#
# Ce programme utilitaire, utilisé par la méthode de détection du service de données, suit
# l'intervalle de temps en secondes à partir d'un point de référence connu (point de période).
# Vous devez le compiler et le placer dans le même répertoire que les méthodes de rappel
# du service de données (rép_base_TR).


#pragma ident "@(#)gettime.c 1.1 00/05/24 SMI"


#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
 {
      printf("%d\n", time(0));
      exit(0);
}
```

Programme SONDE

Le programme SONDE vérifie la disponibilité du services de données à l'aide des commandes nslookup(1M). La méthode de rappel de démarrage_détecteur lance ce programme et la méthode de rappel de démarrage_détecteur l'arrête.

EXEMPLE B–5 Programme sonde_dns

```
#!/bin/ksh
#pragma ident   "@(#)dns_probe   1.1   00/04/19 SMI"
#
# Méthode de détection de HA-DNS.
#
# Ce programme vérifie la disponibilité du service de données à l'aide de nslookup, qui
# demande au serveur DNS de rechercher lui-même le serveur DNS. Si le serveur ne répond pas
# ou si un autre serveur répond à la requête, la méthode de détection conclut qu'il y a
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
# un problème au niveau du service de données et bascule ce service sur un autre noeud
# du cluster. La détection est réalisée suivant un intervalle spécifique défini par
# INTERVALLE_SONDE_COMPLET dans le fichier RTR.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"


################################################################################
# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel
                        # la ressourceest configurée.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}


################################################################################
# restart_service ()
#
# Cette fonction tente de redémarrer le service de données en appelant sa méthode
# d'arrêt, puis sa méthode de démarrage. Si le service de données a déjà été
# arrêté et qu'aucune balise n'est enregistrée pour ce service sous la fonction PMF,
# cette fonction bascule le service vers un autre noeud du cluster.
#
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
function restart_service
{

        # Pour redémarrer le service de données, commencez par vérifier qu'il
        # est toujours enregistré sous la fonction PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Comme la balise TAG du service de données est toujours enregistrée
                # sous la fonction PMF, commencez par arrêter le service de données,
                # puis redémarrez-le.


                # Obtenez le nom de la méthode d'arrêt et la valeur DÉLAI_ARRÊT
                # de cette ressource.
                 DÉLAI_ARRÊT=`scha_resource_get -O STOP_TIMEOUT
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                STOP_METHOD=`scha_resource_get -O STOP
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
 \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
 \
                                "${ARGV0} Stop method failed."
                        return 1
                fi

                # Obtenez le nom de la méthode de démarrage et la valeur DÉLAI_DÉMARRAGE
                # de cette ressource.
                DÉLAI_DÉMARRAGE=`scha_resource_get -O START_TIMEOUT
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                MÉTHODE_DÉMARRAGE=`scha_resource_get -O START
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD
 \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
 \
                        -T $RESOURCETYPE_NAME

                if [[ $? -ne 0 ]]; then
                        logger-p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
 \
                                "${ARGV0} Start method
 failed."
                        return 1
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
                fi


        else
                # L'absence de la balise TAG du service de données
                # implique que ce service a déjà dépassé
                # le nombre maximum de tentatives permises sous la fonction PMF.
                # Par conséquent, ne retentez pas de redémarrer
                # le service de données. Essayez de le basculer
                # sur un autre noeud du cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME
 \
                        -R $RESOURCE_NAME
        fi

        return 0
}



###############################################################################
# decide_restart_or_failover ()
#
# Cette fonction décide de l'action à entreprendre suite à l'échec d'une
# détection : Redémarrez le service de données localement ou basculez-le sur un
# autre noeud du cluster.
#
function decide_restart_or_failover
{

   # Vérifiez s'il s'agit de la première tentative de redémarrage.
   if [ $retries -eq 0 ]; then
         # Il s'agit de la première tentative de redémarrage. Notez la durée
         # de cette première tentative.
         start_time=`$RT_BASEDIR/gettimè
         retries=`expr $retries + 1`
         # Comme il s'agit de la première tentative, tentez de redémarrer
         # le service de données.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart data service."
            exit 1
         fi
   else
      # Ce n'est pas la première tentative
      current_time=`$RT_BASEDIR/gettimè
      time_diff=`expr $current_time - $start_timè
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # Cet échec survient une fois la fenêtre de temps
         # écoulée. Réinitialisez le compteur de tentatives,
         # faites glisser la fenêtre et recommencez.
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
         retries=1
         start_time=$current_time
         # Comme l'échec précédent est survenu il y a plus de
         # Intervalle_nouvelles_tentatives, tentez de redémarrer le service de données.

         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      elif [ $retries -ge $RETRY_COUNT ]; then
         # Toujours dans la fenêtre de temps,
         # le compteur de tentative a expiré. Basculez.
         retries=0
         scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
             -R $RESOURCE_NAME
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failover attempt failed."
            exit 1
         fi
      else
         # Toujours dans la fenêtre de temps,
         # le compteur de temps n'a pas expiré,
         # faites une autre tentative.
         retries=`expr $retries + 1`
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      fi
fi
}

################################################################################
# MAIN
################################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# L'intervalle suivant lequel la détection doit être effectuée est paramétré dans
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
# la propriété définie par le système INTERVALLE_SONDE_COMPLET.
# Obtenez la valeur de cette propriété avec scha_resource_get
INTERVALLE_SONDE=`scha_resource_get -O INTERVALLE_SONDE_COMPLET

# Obtenez la valeur du délai d'attente autorisé pour la détection qui est définie
# dans la propriété d'extension DÉLAI_SONDE dans le fichier RTR. Le délai d'attente
# par défaut de nslookup est de 1,5 minutes.
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAME Probe_timeout`
DÉLAI_SONDE=`echo $probe_timeout_info | awk `{print $2}'`

# Identifiez le serveur sur lequel le service de noms de domaine fonctionne en obtenant
# la valeur de la propriété RESSOURCES_RÉSEAU_UTILISÉES de la ressource.
HÔTE_DNS=`scha_resource_get -O NETWORK_RESOURCES_USED -R
$RESOURCE_NAME -G \$RESOURCEGROUP_NAMÈ

# Obtenez la valeur du compteur de tentative à partir de la propriété
# Nombre_nouvelles_tentatives définie par le système
NOMBRE_NOUVELLES_TENTATIVES=`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# Obtenez la valeur de l'intervalle entre les tentatives à partir de la propriété
# Intervalle_nouvelles_tentatives définie par le système
INTERVALLE_NOUVELLES_TENTATIVES=`scha_resource_get -O RETRY_INTERVAL -R
$RESOURCE_NAME -G \$RESOURCEGROUP_NAMÈ

# Obtenez le chemin complet de l'utilitaire gettime à partir de la propriété
# Rép_base_TR du type de ressources.
RÉP_BASE_TR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# La détection est exécutée dans une boucle sans fin, essayant les commandes nslookup.
# Définissez un fichier temporaire pour les réponses de nslookup.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
   # L'intervalle suivant lequel la détection doit être exécutée est spécifié dans la
   # propriété INTERVALLE_SONDE_COMPLET. Cependant, définissez la durée
   # <INTERVALLE_SONDE_COMPLET> pendant laquelle la détection est en sommeil.
   sleep $PROBE_INTERVAL

   # Exécutez la détection qui demande l'adresse IP sur laquelle
   # le service de noms de domaine fonctionne.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST
 \
            > $DNSPROBEFILE 2>&1

   retcode=$?
        if [ retcode -ne 0 ]; then
                probefail=1
```

EXEMPLE B–5 Programme sonde_dns *(Suite)*

```
        fi

   # Vérifiez que la réponse à la commande nslookup est émise par le serveur HA-DNS
   # et non par un autre serveur répertorié dans le fichier /etc/resolv.conf.
   if [ $probefail -eq 0 ]; then
      # Obtenez le nom du serveur qui a répondu à la requête de nslookup.
                  SERVER=` awk ` $1=="Server:" {
print $2 }' \
                  $DNSPROBEFILE | awk -F. ` { print $1 } ` `
               if [ -z "$SERVER" ];
                   then
                       probefail=1
               else
                       if [ $SERVER != $DNS_HOST ]; then
                               probefail=1
                       fi
               fi
        fi

   # Si la variable probefail n'est pas définie sur 0, cela signifie que le délai de
   # la commande nslookup est dépassé ou que la réponse à la requête a été émise par
   # un autre serveur (spécifié dans le fichier /etc/resolv.conf). Dans les deux cas,
   # le serveur DNS ne répond pas et la méthode appelle decide_restart_or_failover,
   # qui détermine si le service de données doit être redémarré ou basculé sur
   # un autre noeud.

   if [ $probefail -ne 0 ]; then
         decide_restart_or_failover
   else
         logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
         "${ARGV0} Probe for resource HA-DNS successful"
   fi
done
```

Méthode Démarrage_détecteur

Cette méthode démarre le programme de SONDE du service de données.

EXEMPLE B–6 Méthode Démarrage_détecteur_dns

```
#!/bin/ksh
#
# Méthode de démarrage du détecteur HA-DNS.
#
# Cette méthode démarre le détecteur (sonde) du service de données sous le contrôle
# de la fonction PMF. Le détecteur est un processus qui teste le service de données
# suivants des intervalles réguliers et qui, en cas de problème,
```

EXEMPLE B–6 Méthode Démarrage_détecteur_dns *(Suite)*

```
# le redémarre sur le même noeud ou le bascule sur un autre noeud du cluster.
# La balise PMF du détecteur est $RESOURCE_NAME.monitor.


#pragma ident   "@(#)dns_monitor_start   1.1   00/05/24 SMI"

################################################################################
# Arguments du programme d'analyse.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        NOM_RESSOURCE=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                        NOM_GROUPE_RESSOURCES=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        NOM_TYPE_RESSOURCES=$OPTARG
                        ;;
                *)
          logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
                     "ERROR: Option $OPTARG unknown"
                      exit 1
                      ;;
                esac
        done

}

################################################################################
# MAIN
#
################################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode
parse_args "$@"
```

EXEMPLE B–6 Méthode Démarrage_détecteur_dns *(Suite)*

```
PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Recherchez où réside la méthode de détection en obtenant la valeur de la
# propriété RÉP_BASE_TR du service de données.
RÉP_BASE_TR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# Lancez la détection du service de données sous la fonction PMF. Utilisez l'option
# de tentative illimitée pour lancer la détection. Passez le nom de la ressource,
# son groupe et son type à la méthode de détection.
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
 \
    -T $RESOURCETYPE_NAME

# Consignez un message indiquant que le détecteur de HA-DNS a démarré.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} Monitor for HA-DNS successfully started"
fi
exit 0
```

Méthode Arrêt_détecteur

Cette méthode interrompt le programme de SONDE du service de données.

EXEMPLE B–7 Méthode Arrêt_détecteur_dns

```
#!/bin/ksh
#
# Méthode d'arrêt du détecteur de HA-DNS
#
# Arrête le détecteur fonctionnant avec la fonction PMF.


#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"


################################################################################
# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
        typeset opt
```

EXEMPLE B–7 Méthode `Arrêt_détecteur_dns` *(Suite)*

```
        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        NOM_RESSOURCE=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                        NOM_GROUPE_RESSOURCES=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        NOM_TYPE_RESSOURCES=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}


##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Vérifiez si le détecteur fonctionne. Le cas échéant, interrompez-le.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
          "${ARGV0} Could not stop monitor for resource " \
```

EXEMPLE B–7 Méthode Arrêt_détecteur_dns *(Suite)*

```
          $RESOURCE_NAME
           exit 1
   else
      # Possible d'arrêter le détecteur avec succès. Consignez un message.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
          "${ARGV0} Monitor for resource " $RESOURCE_NAME
 \
          " successfully stopped"
   fi
fi

exit 0
```

Méthode Contrôle_détecteur

Cette méthode vérifie l'existence du répertoire spécifié par la propriété Rép_conf. Le gestionnaire RGM appelle Contrôle_détecteur chaque fois que la méthode SONDE bascule le service de données sur un nouveau noeud et contrôle les noeuds qui sont potentiellement des noeuds maîtres.

EXEMPLE B–8 Méthode Contrôle_détecteur_dns

```
#!/bin/ksh
#
# Méthode de contrôle du détecteur du service de noms de domaine.
#
# Le gestionnaire RGM appelle cette méthode chaque fois que le système de détection
# des pannes bascule le service de données vers un nouveau noeud. Contrôle_détecteur
# appelle la méthode Validation pour vérifier
# que le répertoire et les fichiers de configuration sont disponibles sur le nouveau noeud.


#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

################################################################################
# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
   typeset opt

   while getopts `R:G:T:' opt
   do
      case "$opt" in

      R)
      # Nom de la ressource DNS.
```

EXEMPLE B–8 Méthode `Contrôle_détecteur_dns` *(Suite)*

```
       NOM_RESSOURCE=$OPTARG
      ;;

      G)
      # Nom du groupe de ressources dans lequel la ressource est
      # configurée.
      NOM_GROUPE_RESSOURCES=$OPTARG
      ;;

      T)
      # Nom du type de ressources.
     NOM_TYPE_RESSOURCES=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
 \
      "ERROR: Option $OPTARG unknown"
      exit 1
      ;;

      esac
   done

}

##############################################################################
# MAIN
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode.
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Obtenez le nom complet de la méthode de validation à partir de la
# propriété RÉP_BASE_TR du type de ressources.
RÉP_BASE_TR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
 \
-G $RESOURCEGROUP_NAMÈ

# Obtenez le nom de la méthode de validation de cette ressource.
MÉTHODE_VALIDATION=`scha_resource_get -O VALIDATE \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtenez la valeur de la propriété Rép_conf afin de démarrer le
```

EXEMPLE B–8 Méthode `Contrôle_détecteur_dns` *(Suite)*

```
# service de données. Utilisez le nom de ressource et le groupe de ressources
# entrés pour obtenir la valeur Rép_conf définie au moment de l'ajout  de la ressource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get renvoie le type et la valeur des propriétés d'extension.
# Utilisez awk pour obtenir uniquement la valeur de la propriété d'extension.
# RÉP_CONFIG=`echo $config_info | awk `{print $2}'`

# Appelez la méthode de validation pour pouvoir basculer le service de données
# vers le nouveau noeud.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
 \
-T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

# Consignez un message indiquant que le contrôle du détecteur a réussi.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS successful."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS not successful."
   exit 1
fi
```

Méthode `Validation`

Cette méthode vérifie l'existence du répertoire spécifié par la propriété `Rép_conf`. Le gestionnaire RGM appelle cette méthode lorsque l'administrateur du cluster crée le service de données et met à jour ses propriétés. La méthode `Contrôle_détecteur` appelle cette méthode chaque fois que le système de détection des pannes bascule le service de données vers un nouveau noeud.

EXEMPLE B–9 Méthode `Validation_dns`

```
#!/bin/ksh
#
# Méthode Validation de HA-DNS.
# Cette méthode valide la propriété Rép_conf de la ressource. La méthode Validation
# est appelée à la création et à la mise à jour de la ressource. À la création de la
# ressource, cette méthode est appelée avec l'indicateur -c. En outre, toutes les
# propriétés définies par le système et d'extension sont passées en tant qu'arguments
# de la ligne de commande. Lorsqu'une propriété de ressource est mise à jour, la méthode
# Validation est appelée avec l'indicateur -u, et seule la paire propriété/valeur de la
# propriété mise à jour est passée en tant qu'argument de la ligne de commande.
```

EXEMPLE B–9 Méthode `Validation_dns` *(Suite)*

```
#
# exemple : si la ressource est en cours de création, les arguments de la ligne de
# commande sont
# dns_validate -c -R <..> -G <...> -T <..>
# -r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# Si la propriété de ressource est en cours de mise à jour, les arguments sont
#
# dns_validate -u -R <..> -G <...> -T <..>
# -r <sys-prop_being_updated=value>
#   OU
# dns_validate -u -R <..> -G <...> -T <..>
# -x <extn-prop_being_updated=value>
#

#pragma ident   "@(#)dns_validate   1.1   00/05/24 SMI"

################################################################################

# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
   typeset opt

   while getopts  `cur:x:g:R:T:G:' opt
   do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                        NOM_RESSOURCE=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                         NOM_GROUPE_RESSOURCES=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        NOM_TYPE_RESSOURCES=$OPTARG
                        ;;

                r)
                        # La méthode n'accède à aucune propriété définie par le
                        # système, il s'agit donc d'un no-op.
                        ;;

                g)
                        # La méthode n'accède à aucune propriété de groupe de ressources,
                        # il s'agit donc d'un no-op.
                        ;;
```

EXEMPLE B–9 Méthode `Validation_dns` *(Suite)*

```
                c)
                        # Indique que la méthode de validation est appelée tandis que
                        # la ressource est en cours de création. Cet indicateur est
                        #  donc un no-op.
                        ;;

                u)
                        # Indique la mise à jour d'une propriété lorsque la ressource
                        # existe déjà. Si la mise à jour concerne la propriété Rép_conf,
                        # Rép_conf doit apparaître dans les arguments de la ligne de
                        # commande. Dans le cas contraire, la méthode doit rechercher
                        # spécifiquement cette propriété à l'aide de scha_resource_get.
                        UPDATE_PROPERTY=1
                        ;;

                x)
                        # Liste des propriétés d'extension. Séparez les paires
                        # propriété/valeur en utilisant le séparateur "=".
                        PROPERTY=`echo $OPTARG | awk -F= `{print $1}'`
                        VAL=`echo $OPTARG | awk -F= `{print $2}'`

                        # Si la propriété d'extension Rép_conf figure sur la ligne de
                        # commande, notez sa valeur.
                        if [ $PROPERTY == "Confdir" ];
                        then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                        fi
                        ;;

                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        "ERROR: Option $OPTARG unknown"
                        exit 1
                        ;;
                esac
   done
}

##############################################################################
# MAIN
#
##############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Définissez la valeur de RÉP_CONF comme étant Null. Ultérieurement, cette méthode
# recherche et extra it la valeur de la propriété Rép_conf de la ligne de commande
# ou à l'aide de scha_resource_get.
```

EXEMPLE B–9 Méthode `Validation_dns` *(Suite)*

```
RÉP_CONF=""
PROPRIÉTÉ_ACTUALISATION=0
REP_CONF_TROUVÉ=0

# Analysez les arguments qui ont été transmis à cette méthode.
parse_args "$@"

# Si la méthode de validation est appelée du fait de la mise à jour des propriétés,
# essayez de récupérer la valeur de la propriété d'extension Rép_conf à partir de la
# ligne de commande.
# Sinon, obtenez la valeur de Rép_conf avec scha_resource_get.
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
   config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
 \
       -G $RESOURCEGROUP_NAME Confdir`
   CONFDIR=`echo $config_info | awk `{print $2}'`
fi

# Vérifiez que la propriété Rép_conf possède une valeur. Si elle n'en a pas, il en
# résulte un échec et une sortie avec l'état 1.
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
       "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
   exit 1

fi

# Validez désormais la valeur réelle de la propriété Rép_conf.

# Vérifiez que $CONFDIR est accessible.
if [ ! -d $CONFDIR ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
            "${ARGV0} Directory $CONFDIR missing or not mounted"

        exit 1
fi

# Vérifiez que le fichier named.conf figure dans le répertoire Rép_conf.

if [ ! -s $CONFDIR/named.conf ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
 \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"

        exit 1
fi

# Consignez un message indiquant que la méthode de validation a réussi.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME
 \
   " completed successfully"
```

EXEMPLE B–9 Méthode `Validation_dns` *(Suite)*

```
exit 0
```

Méthode `Mise_à_jour`

Le gestionnaire RGM appelle la méthode `Mise_à_jour` pour notifier à une ressource en cours d'exécution que ses propriétés ont été modifiées.

EXEMPLE B–10 Méthode `Mise_à_jour_dns`

```
#!/bin/ksh
#
# Méthode de mise à jour de HA-DNS.
#
# La mise à jour réelle des propriétés est réalisée par le gestionnaire RGM. Les mises
# à jour affectent uniquement le système de détection des pannes.
# Par conséquent, cette méthode doit redémarrer le système de détection des pannes.


#pragma ident   "@(#)dns_update   1.1   00/05/24 SMI"

################################################################################
# Analysez les arguments du programme.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nom de la ressource DNS.
                         NOM_RESSOURCE=$OPTARG
                        ;;
                G)
                        # Nom du groupe de ressources dans lequel la ressource
                        # est configurée.
                        NOM_GROUPE_RESSOURCES=$OPTARG
                        ;;
                T)
                        # Nom du type de ressources.
                        NOM_TYPE_RESSOURCES=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
```

EXEMPLE B–10 Méthode `Mise_à_jour_dns` *(Suite)*

```
 \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}



###############################################################################
# MAIN
#
###############################################################################

export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtenez la fonction syslog à utiliser pour consigner les messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analysez les arguments qui ont été transmis à cette méthode
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Recherchez où réside la méthode de détection en obtenant la valeur de
# la propriété RÉP_BASE_TR du service de données.
RÉP_BASE_TR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Lorsque la méthode de mise_à_jour est appelée, le gestionnaire RGM met à jour
# la valeur de la propriété en cours de mise à jour.
# Cette méthode doit vérifier que le système de détection des pannes (sonde) fonctionne.
# Le cas échéant, interrompez-le et redémarrez-le.
if pmfadm -q $PMF_TAG.monitor; then

# Interrompez le détecteur qui fonctionne déjà
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
 \
                    "${ARGV0} Could not stop the monitor"
                exit 1
        else
                # Possible d'arrêter le service de noms de domaine avec succès.
                # Consignez un message.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]
 \
                    "Monitor for HA-DNS successfully stopped"
```

EXEMPLE B–10 Méthode `Mise_à_jour_dns` *(Suite)*

```
        fi

   # Redémarrez le détecteur.
   pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
      -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCETYPE_NAME
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                  "${ARGV0} Could not restart monitor for HA-DNS "
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully restarted"
   fi
fi
exit 0
```

ANNEXE **C**

Liste des codes du type de ressources échantillon de la BDSD

Cette annexe fournit le code complet de chaque méthode du type de ressources `SUNW.xfnts`. Elle comprend la liste de `xfnts.c` contenant le code des sous-routines appelées par les méthodes de rappel. Programmes figurant dans cette annexe :

`xfnts.c`

Ce fichier met en oeuvre les sous-routines appelées par les méthodes `SUNW.xfnts`.

EXEMPLE C–1 `xfnts.c`

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * Cet utilitaire comprend les méthodes Validation, Démarrage et Arrêt du
 * service de données et du système de détection des pannes. Il contient également
 * la méthode de détection de l'état du service de données. La détection précise
 * simplement le succès ou l'échec. L'action est entreprise en fonction de la
 * valeur retournée dans laméthode figurant dans le fichier xfnts_probe.c
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 *
 */

#pragma ident ?@(#)xfnts.c 1.47 01/01/18 SMI?

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <scha.h>
#include <rgm/libdsdev.h>
#include <errno.h>
#include "xfnts.h"

/*
 * Délai d'attente initial autorisé pour que le service de données HAXFS
 * soit pleinement opérationnel. Nous attendrons pendant 3 % (SVC_WAIT_PCT)
 * du délai_démarrage avant de détecter le service.
 */
#define   SVC_WAIT_PCT      3

/*
 * Nous devons utiliser 95 % du délai_sonde pour la connexion au port, le
 * temps restant servant à la déconnexion de ce port dans la fonction svc_probe.
 */
#define   SVC_CONNECT_TIMEOUT_PCT      95

/*
 * SVC_WAIT_TIME sert uniquement lors du démarrage dans svc_wait().
 * Dans svc_wait(), nous devons nous assurer que le service est opérationnel
 * avant de revenir au mode de fonctionnement antérieur, puis nous devons appeler
 * svc_probe() pour contrôler le service. SVC_WAIT_TIME correspond à la durée entre
 * ces détections.
 */

#define   SVC_WAIT_TIME      5

/*
 * Cette valeur sera utilisée en tant que délai de déconnexion, si le délai
 * à partir de délai_sonde est écoulé.
 */

#define   SVC_DISCONNECT_TIMEOUT_SECONDS      2


/*
 * svc_validate():
 *
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 * Effectuez une validation HA-XFS spécifique de la configuration de la ressource.
 *
 * svc_validate vérifie :
 * 1. La propriété d'extension Liste_rép_conf
 * 2. Le fichier fontserver.cfg
 * 3. La valeur xfs binaire
 * 4. La propriété liste_ports
 * 5. Les ressources réseau
 * 6. Les autres propriétés d'extension
 *
 * Si l'une des validations ci-dessus échoue, Return> 0 ou
 * retournez 0 pour obtenir un succès
 */

int
svc_validate(scds_handle_t scds_handle)
{
   char   xfnts_conf[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_net_resource_list_t *snrlp;
   int rc;
   struct stat statbuf;
   scds_port_list_t   *portlist;
   scha_err_t   err;

   /*
    * Obtenez le répertoire de configuration du service de données XFS à partir
    * de la propriété d'extension/
   confdirs = scds_get_ext_confdir_list(scds_handle);

   /* Une erreur est retournée en l'absence de la propriété d'extension liste_rép_conf
*/
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }

   /*
    * Définissez le chemin d'accès au fichier de configuration à partir de la
    * propriété d'extension liste_rép_conf. Comme HA-XFS possède une seule
    * configuration, nous devons utiliser la première entrée de la propriété
    * liste_rép_conf.
    */
   (void) sprintf(xfnts_conf, ?%s/fontserver.cfg?, confdirs->str_array[0]);
confdirs->str_array[0]);

   /*
    * Vérifiez que le fichier de configuration de HA-XFS figure au bon emplacement.
    * Essayez d'accéder au fichier de configuration de HA-XFS et vérifiez que les
    * permissions d'accès sont correctement définies
    */
   if (stat(xfnts_conf, &statbuf) != 0) {
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
      /*
       * Supprimez l'erreur lint car le prototype errno.h
       * ne possède pas d'argument nul
       */
      scds_syslog(LOG_ERR,
          "Failed to access file <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }

   /*
    * Vérifiez que la valeur xfs binaire existe et que les permissions d'accès
    * sont correctes. La valeur XFS binaire doit figurer dans le système de
    * fichiers local et non dans le système de fichiers global
    */
   if (stat("/usr/openwin/bin/xfs", &statbuf)
!= 0) {
      scds_syslog(LOG_ERR,
          "Cannot access XFS binary : <%s> ",
strerror(errno));
      return (1);
   }

   /* HA-XFS possède uniquement un port */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list: %s.",
         scds_error_string(err));
      return (1); /* Echec de validation */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
      scds_syslog(LOG_ERR,
          "Property Port_list must have only one value.");
      scds_free_port_list(portlist);
      return (1); /* Echec de validation */
   }
#endif

   /*
    * Retournez une erreur en cas d'erreur lors de la tentative d'obtention
    * de ressources d'adresses réseau disponibles pour cette ressource
    */
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
      != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group: %s.",
         scds_error_string(err));
      return (1); /* Echec de validation */
   }
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
   /* Retournez une erreur en l'absence de ressources d'adresses réseau */
   if (snrlp == NULL || snrlp->num_netresources == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      rc = 1;
      goto finished;
   }

   /* Vérifiez que les autres propriétés d'extension importantes sont définies */
   if (scds_get_ext_monitor_retry_count(scds_handle) <= 0)
   {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_count is not set.");
      rc = 1; /* Échec de validation */
      goto finished;
   }
   if (scds_get_ext_monitor_retry_interval(scds_handle) <= 0) {
0) {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_interval is not set.");
      rc = 1; /* Échec de validation */
      goto finished;
   }

   /* Tous les contrôles de validation ont réussi */
   scds_syslog(LOG_INFO, "Successful validation.");
   rc = 0;

finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* retour des résultats de la validation */
}


/*
 * svc_start():

 * Démarre le serveur de polices X
 * Retourne 0 si succès,> 0 si échec.
 *
 * Le service XFS est démarré en exécutant la commande
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS est démarré via la fonction PMF. XFS est démarré en tant que service
 * d'instances unique. La balise PMF du service de données se présente sous
 * la forme <resourcegroupname,resourcename,instance_number.svc>. Dans le cas
 * de XFS, comme il n'y a qu'une seule instance, instance_number, dans
 * la balise est de 0.
 */

int
svc_start(scds_handle_t scds_handle)
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
{
   char    xfnts_conf[SCDS_ARRAY_SIZE];
   char   cmd[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_port_list_t    *portlist;
   scha_err_t   err;

   /* Obtenez le répertoire de configuration à partir de la propriété liste_rép_conf */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   (void) sprintf(xfnts_conf, "%s/fontserver.cfg",
confdirs->str_array[0]);

   /* Obtenez le port à utiliser par XFS à partir de la propriété Liste_ports */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }

   /*
    * Créez la commande pour démarrer HA-XFS.
    * REMARQUE : le démon XFS imprime le message suivant tout en arrêtant XFS
    * "/usr/openwin/bin/xfs notice: terminating"
    * Afin de supprimer le message du démon,
    * la sortie est redirigée vers /dev/null.
    */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);

   /*
    * Démarrez HA-XFS via la fonction PMF. Notez que HA-XFS est démarré en tant que
    * service d'instances unique. Le dernier argument de la fonction scds_pmf_start
    * indique le niveau enfant à contrôler. Si ce paramètre possède une valeur de -1,
    * tous les enfants et le processus original doivent être contrôlés.
    */
   scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

    if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
   }

          scds_free_port_list(portlist);
   return (err); /* retourne un état succès/échec */
   }
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
/*
 * svc_stop():
 *
 * Arrête le serveur XFS
 * Retourne 0 si succès,> 0 si échec.
 *
 * svc_stop arrête le serveur en appelant la fonction de la boîte à outils :
 * scds_pmf_stop.
 */
int
svc_stop(scds_handle_t scds_handle)
{
   scha_err_t   err;

   /*
    * La valeur du délai d'attente de la méthode d'arrêt pour réussir est définie
    * dans la propriété Délai_arrêt (définie par le système)
    */
   scds_syslog(LOG_ERR, "Issuing a stop request.");
   err = scds_pmf_stop(scds_handle,
       SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
       scds_get_rs_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Successfully stopped HA-XFS.");
   return (SCHA_ERR_NOERR); /* Arrêt réussi*/
}

/*
 * svc_wait():
 *
 * Attendez que le service de données démarre complètement et vérifiez qu'il
 * fonctionne correctement
 */

int
svc_wait(scds_handle_t scds_handle)
{
   int rc, svc_start_timeout, probe_timeout;
   scds_netaddr_list_t   *netaddr;

   /*  Obtenez la ressource réseau à utiliser pour la détection */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
      return (1);
   }

   /* Retournez une erreur en l'absence de ressources réseau */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }

   /*
    * Obtenez le délai d'attente de la méthode Démarrage, le numéro de port
    * sur lequel effectuer la détection, la valeur du délai d'attente de détection
    */
   délai_démarrage_svc = scds_get_rs_start_timeout(scds_handle);
   délai_sonde = scds_get_ext_probe_timeout(scds_handle);

   /*
    * patientez pendant le pourcentage de SVC_WAIT_PCT du délai_démarrage
    * avant de vraiment détecter le service de données. Ceci permet
    * au service de données d'être complètement opérationnel pour répondre à la
    * détection. REMARQUE : la valeur de SVC_WAIT_PCT peut être différente
    * pour différents services de données.
    * Au lieu d'appeler sleep(),
    * appelez scds_svc_wait() de sorte que si le service échoue également
    * plusieurs fois, vous pouvez abandonner et revenir rapidement.
    */
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

   do {
      /*
       * Détectez le service de données sur l'adresse IP de la
       * ressource réseau et du nom de port
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Succès. Libérez les ressources et retournez */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

      /*
       * Le service de données essaie toujours de s'afficher. Attendez quelques
       * instants avant de relancer la détection. Au lieu d'appeler sleep(),
       * appelez scds_svc_wait() de sorte que si le service échoue également
       * plusieurs fois, vous pouvez abandonner et revenir rapidement.
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
       */
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }

   /* Nous comptons sur le gestionnaire RGM pour temporiser et terminer le programme */
   } while (1);

}

/*
 * Cette fonction démarre le système de détection des pannes d'une ressource HA-XFS.
 * Pour ce faire, il suffit de lancer la détection via fonction PMF. La balise PMF
 * est dérivée en tant que <RG-name,RS-name,instance_number.mon>.
 * L'option de redémarrage de la fonction PMF est utilisée mais pas le
 * "redémarrage illimité".

 * interval/retry_time est obtenu à la place à partir du fichier RTR.
 */

int
mon_start(scds_handle_t scds_handle)
{
   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
     "Calling MONITOR_START method for resource <%s>.",
      scds_get_resource_name(scds_handle));

   /*
    * La détection xfnts_probe est supposée être disponible dans le même
    * sous-répertoire que celui dans lequel les autres méthodes de rappel du type de
    * ressources sont installées. Le dernier paramètre de scds_pmf_start indique le
    * niveau de contrôle enfant. Comme nous commençons la détection via la fonction PMF,
    * nous devons seulement contrôler le processus de détection, par conséquent nous
    * utiliserons une valeur de 0.
    */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe",
0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Started the fault monitor.");

   return (SCHA_ERR_NOERR); /* Détecteur démarré avec succès */
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
}


/*
 * Cette fonction arrête le système de détection des pannes d'une ressource HA-XFS.
 * Cette action est réalisée par le biais de la fonction PMF. La balise PMF du système
 * de détection des pannes est créée sur le modèle <RG-name_RS-name,instance_number.mon>.
 */

int
mon_stop(scds_handle_t scds_handle)
{

   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");

   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Détecteur arrêté avec succès */
}

/*
 * svc_probe() : exécute la détection spécifique d'un service de données. Retourne
 * une valeur flottantecomprise entre 0 (succès) et 100 (échec total).
 *
 * La détection se connecte simplement au serveur XFS sur le port spécifié qui est
 * configuré en tant que propriété d'extension de la ressource (Liste_ports) et
 * pingue le service de données. Si la détection ne parvient pas à se connecter au port,
 * nous retournons une valeur de 100 indiquant un échec total. Si la connexion
 * réussit et la déconnexion échoue, une valeur de 50 est retournée, indiquant
 * un échec partiel.
 *
 */
int
svc_probe(scds_handle_t scds_handle, char *hostname, int port, int
timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
char   testcmd[2048];
int    time_used, time_remaining;
time_t      connect_timeout;


/*
 * Détectez le service de données par une connexion au port
 * spécifié dans la propriété liste_ports de l'hôte qui prend en charge
 * le service de données XFS. Si le service XFS qui est configuré
 * pour écouter le port spécifié répond à la connexion, la détection
 * est réussie. Autrement, nous attendons pendant le temps spécifié dans
 * la propriété délai_sonde avant de conclure à l'échec de la détection.
 */

/*
 * Utilisez le pourcentage SVC_CONNECT_TIMEOUT_PCT du délai d'attente
 * pour vous connecter au port
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * La détection se connecte au nom d'hôte et au port spécifiés.
 * La connexion est temporisée sur 95 % du délai_sonde réel.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "Failed to connect to port <%d> of resource <%s>.",
       port, scds_get_resource_name(scds_handle));
   /* Il s'agit d'un échec total */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Calculez le temps réel requis pour la connexion. Ce délai doit être
 * inférieur ou égal au délai_connexion alloué pour se connecter.
 * Si la connexion requiert l'intégralité du délai alloué,
 * la valeur restante du délai_sonde, passée à la fonction est
 * utilisée en tant que délai de déconnexion. Sinon, le délai de
 * connexion restant est également ajouté au délai de déconnexion
 *
 */

time_used = (int)(t2 - t1);

/*
 * Utilisez le délai restant (timeout - time_took_to_connect) pour
 * la déconnexion
 */
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
time_remaining = timeout - (int)time_used;

/*
 * Si le délai est intégralement utilisé, utilisez un délai d'attente codé
 * dans un petit programme spécifique pour tenter de nouveau une déconnexion.
 * Cette action évite la fuite fd.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
       "%d seconds during connect operation and exceeded the "
       "timeout by %d seconds. Attempting disconnect with timeout"
       " %d ",
       connect_timeout,
       abs(time_used),
       SVC_DISCONNECT_TIMEOUT_SECONDS);

   time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Retournez un échec partiel en cas d'échec de déconnexion.
 * Motif : l'appel de connexion a réussi ce qui signifie que
 * l'application fonctionne. Un échec de déconnexion peut survenir
 * si une application est interrompue ou une charge volumineuse.
 * Dans ce dernier cas, ne déclarez pas l'application comme
 * bloquée en retournant un échec total. Déclarez plutôt
 * un échec partiel. Si cette situation persiste, l'appel
 * de déconnexion échoue de nouveau et l'application est
 * redémarrée.
 */
rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
if (rc != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "Failed to disconnect to port %d of resource %s.",
       port, scds_get_resource_name(scds_handle));
   /* Il s'agit d'un échec partiel */
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

t2 = (hrtime_t)(gethrtime()/1E9);
time_used = (int)(t2 - t1);
time_remaining = timeout - time_used;

/*
 * En l'absence de temps restant, n'exécutez pas le test complet avec
 * fsinfo. Retournez plutôt SCDS_PROBE_COMPLETE_FAILURE/2.
 * Vous avez ainsi l'assurance que si ce délai d'attente
 * persiste, le serveur est redémarré.
 */
if (time_remaining <= 0) {
   scds_syslog(LOG_ERR, "Probe timed out.");
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }

   /*
    * La connexion au port et la déconnexion du port sont réussies.
    * Exécutez la commande fsinfo pour réaliser un contrôle complet
    * de l'état du serveur.
    * Redirigez stdout, sinon la sortie de fsinfo
    * se termine sur la console.
    */
   (void) sprintf(testcmd,
      "/usr/openwin/bin/fsinfo -server %s:%d> /dev/null",
       hostname, port);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Checking the server status with %s.", testcmd);
   if (scds_timerun(scds_handle, testcmd, time_remaining,
      SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

      scds_syslog(LOG_ERR,
         "Failed to check server status with command <%s>",
         testcmd);
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }
   return (0);
}
```

Méthode Contrôle_détecteur_xfnts

Cette méthode vérifie que la configuration de base du type de ressources est valide.

EXEMPLE C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_monitor_check.c - méthode de contrôle du détecteur de HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"
```

EXEMPLE C–2 xfnts_monitor_check.c *(Suite)*

```
/*
 * Effectuez un simple contrôle de validation du service
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;

   /* Traitez les arguments passés au gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "monitor_check method "
       "was called and returned <%d>.", rc);

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Retournez le résultat de la méthode de validation exécutée dans le cadre /*
   /* du contrôle du détecteurreturn (rc); /*
}
```

Méthode Démarrage_détecteur_xfnts

Cette méthode exécute la méthode Sondage_ xfnts.

EXEMPLE C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_monitor_start.c - Méthode Démarrage du détecteur de HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"
```

EXEMPLE C–3 xfnts_monitor_start.c *(Suite)*

```
#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Cette méthode démarre le système de détection des pannes d'une ressource HA-XFS.
 * Cette action est réalisée en lançant la détection sous la fonction PMF.
 * La balise PMF est dérivée en tant que RG-name,RS-name.mon. L'option de redémarrage
 * de la fonction PMF est utilisée mais pas le "redémarrage illimité".
 * interval/retry_time est obtenu à la place à partir du fichier RTR.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;

   /* Traitez les arguments passés par le gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = mon_start(scds_handle);

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Retournez le résultat de la méthode de démarrage_détecteur */
   return (rc);
}
```

Méthode Arrêt_détecteur_xfnts

Cette méthode arrête la méthode Sonde_xfnts.

EXEMPLE C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_monitor_stop.c - Méthode d'arrêt du détecteur de HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"
```

EXEMPLE C–4 xfnts_monitor_stop.c *(Suite)*

```
#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Cette méthode arrête le système de détection des pannes d'une ressource HA-XFS.
 * Cette action est réalisée via la fonction PMF. La balise PMF du système de détection
 * des pannes est créée sur le modèle RG-name_RS-name.mon.
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int    rc;

   /* Traitez les arguments passés par le gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = mon_stop(scds_handle);

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Retournez le résultat de la méthode d'arrêt du détecteur */
   return (rc);
}
```

Méthode Sonde_xfnts

La méthode Sonde_xfnts contrôle la disponibilité de l'application et choisit de basculer ou de redémarrer le service de données. La méthode de rappel Démarrage_détecteur_xfnts lance ce programme tandis que la méthode de rappel Arrêt_détecteur_xfnts l'arrête.

EXEMPLE C–5 xfnts_probe.c+

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_probe.c - Probe for HA-XFS
 */
```

EXEMPLE C–5 xfnts_probe.c+ *(Suite)*

```
#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"


/*
 * main():
 * Boucle illimitée qui attend (sleep()s) pendant un laps de temps que le script
 * d'action de la fonction PMF interrompe sleep(). Une fois qu'il est interrompu
 * elle appelle la méthode de démarrage de HA-XFS pour le redémarrer.
 *
 */

int
main(int argc, char *argv[])
{
   int         timeout;
   int         port, ip, probe_result;
   scds_handle_t      scds_handle;

   hrtime_t      ht1, ht2;
   unsigned long      dt;

   scds_netaddr_list_t *netaddr;
   char   *hostname;

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }


   /* Obtenez les adresses IP disponibles pour cette ressource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Retournez une erreur en l'absence de ressources réseau */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
```

EXEMPLE C–5 xfnts_probe.c+ *(Suite)*

```
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }


   /*
    * Définissez le délai d'attente des propriétés X. Cela signifie que chaque
    * itération de détection obtient un délai d'attente complet sur chaque ressource
    * réseau sans supprimer le délai d'attente entre toutes les ressources réseau
    * configurées pour cette ressource.
    */
   timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {

      /*
       * Attendez que s'écoule l'intervalle_sonde_complet entre les
       * tentatives réussies.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));

      /*
       * À présent, détectez toutes les adresses IP utilisées. Itérez
       * 1. Toutes les ressources réseau utilisées.
       * 2. Toutes les adresses IP d'une ressource donnée.
       * Chaque adresse IP testée,
       * Calculez l'historique d'échec.
       */
      probe_result = 0;
      /*
       * Itérez toutes les ressources pour obtenir chaque
       * adresse IP à utiliser pour appeler svc_probe()
       */
      for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Saisissez le nom d'hôte et le port
          * dont l'état doit être contrôlé.
          */
         nom_hôte  = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS ne prend en charge qu'un port, par conséquent
          * obtenez la valeur du port à partir de la
          * première entrée dans l'ensemble des ports.
          */
         ht1 = gethrtime(); /* Verrouillez le délai de démarrage de détection */
         scds_syslog(LOG_INFO, "Probing the service on "
             "port: %d.", port);

        résultat_sonde =
         svc_probe(scds_handle, hostname, port, timeout);
```

EXEMPLE C–5 xfnts_probe.c+ *(Suite)*

```
        /*
         *Mettez à jour l'historique de détection du service,
         * effectuez les actions requises si nécessaire.
         * Verrouillez le délai de fin de détection.
         */
        ht2 = gethrtime();

        /* Conversion en millisecondes */
        dt = (ulong_t)((ht2 - ht1) / 1e6);

        /*
         * Calculez l'historique d'échec et effectuez les actions
         * requises le cas échéant
         */
        (void) scds_fm_action(scds_handle,
            probe_result, (long)dt);
      }   /* Chaque ressource réseau */
   }    /* Conserver la détection */
}
```

Méthode Démarrage_xfnts

Le gestionnaire RGM appelle la méthode Démarrage sur un noeud du cluster lorsque le groupe de ressources contenant la ressource du service de données est connecté à ce noeud ou lorsque la ressource est activée. La méthode Démarrage_xfnts active le démon xfs sur ce noeud.

EXEMPLE C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_svc_start.c - Méthode Démarrage de HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Méthode de démarrage de HA-XFS. Effectue quelques contrôles sanitaires sur
 * les paramètres de ressource, puis démarre HA-XFS via la fonction PMF avec un
 * script d'action.
 */
```

EXEMPLE C–6 xfnts_start.c *(Suite)*

```
int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int rc;

   /*
    * Traitez tous les arguments qui nous ont été passés à partir du gestionnaire
    * RG et effectuez des actions d'initialisation pour syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Validez l'initialisation et retournez en arrière en cas d'erreur */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }

   /* Démarrez le service de données et retournez une erreur en cas d'erreur */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

   /* Attendez que le service démarre complètement */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Calling svc_wait to verify that service has started.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "Successfully started the service.");
   } else {
      scds_syslog(LOG_ERR, "Failed to start the service.");
   }


finished:
   /* Libérez les ressources Environnement qui ont été allouées */
   scds_close(&scds_handle);

   return (rc);
```

EXEMPLE C–6 xfnts_start.c *(Suite)*

```
}
```

Méthode Arrêt_xfnts

Le gestionnaire RGM appelle la méthode Arrêt sur un noeud du cluster lorsque le groupe de ressources contenant la ressource HA-XFS est déconnecté de ce noeud ou lorsque la ressource est désactivée. Cette méthode arrête le démon xfs sur ce noeud.

EXEMPLE C–7 xfnts_stop.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_svc_stop.c - Méthode d'Arrêt de HA-XFS
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Arrête le processus HA-XFS via la fonction PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

   /* Traitez les arguments passés par le gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = svc_stop(scds_handle);

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Retournez le résultat de la méthode arrêt_svc */
   return (rc);
```

EXEMPLE C–7 xfnts_stop.c *(Suite)*

```
}
```

Méthode Mise_à_jour_xfnts

Le gestionnaire RGM appelle la méthode Mise_à_jour pour notifier à une ressource en cours d'exécution que ses propriétés ont été modifiées. Il appelle Mise_à_jour après l'exécution réussie d'une action administrative de paramétrage des propriétés d'une ressource ou de son groupe.

EXEMPLE C–8 xfnts_update.c

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droits réservés.
*
* xfnts_update.c - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Certaines des propriétés de ressource peuvent avoir été mises à jour. Toutes
 * ces propriétés qu'il est possible de mettre à jour sont liées au système de
 * détection des pannes. Par conséquent, il suffit normalement de redémarrer
 * le détecteur.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

   /*Traitez les arguments passés par le gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /*
    * Vérifiez si le système de détection des pannes fonctionne déjà et arrêtez-le
```

EXEMPLE C–8 xfnts_update.c *(Suite)*

```
    * le cas échéant, puis redémarrez-le. Le second paramètre de scds_pmf_restart_fm()
    * identifie uniquement l'instance du système de détection des pannes qui doit être
    * redémarré.
    */

   scds_syslog(LOG_INFO, "Restarting the fault monitor.");
   result = scds_pmf_restart_fm(scds_handle, 0);
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to restart fault monitor.");
      /* Libérez toute la mémoire allouée par scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Completed successfully.");

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   return (0);
}
```

Liste des codes de la méthode Validation_xfnts

Cette méthode vérifie l'existence du répertoire spécifié par la propriété Liste_rép_conf. Le gestionnaire RGM appelle cette méthode lorsque l'administrateur du cluster crée le service de données et met à jour ses propriétés. La méthode Contrôle_détecteur appelle cette méthode chaque fois que le système de détection des pannes bascule le service de données vers un nouveau noeud.

EXEMPLE C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2003 par Sun Microsystems, Inc.
* Tous droit réservés.
*
* xfnts_validate.c - Méthode de validation de HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
```

EXEMPLE C–9 xfnts_validate.c *(Suite)*

```
#include "xfnts.h"

/*
 * Vérifiez que toutes les propriétés ont été correctement définies.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   int   rc;

   /* Traitez les arguments passés par le gestionnaire RGM et initialisez syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = svc_validate(scds_handle);

   /* Libérez toute la mémoire allouée par scds_initialize */
   scds_close(&scds_handle);

   /* Retournez le résultat de la mithode de validation */
   return (rc);

}
```

ANNEXE **D**

Noms et valeurs RGM légaux

Cette annexe répertorie les exigences en matière de caractères légaux pour les noms et valeurs RGM.

Noms légaux RGM

Les noms RGM sont regroupés en cinq catégories :

- noms de groupes de ressources ;
- noms de types de ressources ;
- noms de ressources ;
- noms de propriétés ;
- noms de libellés d'énumération.

Hormis les noms de types de ressources, tous les noms doivent être conformes aux règles suivantes :

- Ils doivent être au format ASCII.
- Ils doivent commencer par une lettre.
- Ils peuvent contenir des lettres majuscules et minuscules, des chiffres, des tirets (-) et des traits de soulignements (_).
- Ils ne doivent pas contenir plus de 255 caractères.

Un nom de type de ressources peut n'être qu'un nom simple (spécifié par la propriété `Resource_type` dans le fichier RTR) ou un nom complet (spécifié par les propriétés `Vendor_id` et `Resource_type` dans le fichier RTR). Lorsque vous spécifiez ces deux propriétés, le gestionnaire RGM insère un point entre `Id_fournisseur` et `Type_ressource`, pour constituer un nom complet. Par exemple, si `Vendor_id=SUNW` et `Resource_type=sample`, le nom complet est `SUNW.sample`. Il s'agit du seul cas où le point est accepté en tant que caractère légal dans un nom RGM.

Valeurs RGM

Les valeurs RGM sont regroupées en deux catégories : les valeurs de propriété et les valeurs de description. Ces deux catégories répondent aux mêmes règles :

- Les valeurs doivent être au format ASCII.
- La longueur d'une valeur ne peut excéder 4 mégaoctets moins 1, c'est-à-dire, 4 194 303 octets.
- Les valeurs ne peuvent contenir aucun des caractères suivants : nul, interligne, virgule ou point-virgule.

ANNEXE **E**

Exigences des applications ordinaires non prévues pour être utilisées avec un cluster

Une application ordinaire non prévue pour être utilisée avec un cluster doit satisfaire certaines exigences pour pouvoir prétendre devenir une application à haut niveau de disponibilité (HA). La rubrique “Analyse du caractère approprié de l’application” à la page 29 répertorie ces exigences tandis que cette annexe fournit de plus amples détails sur certains éléments de cette liste.

Une application obtient un haut niveau de disponibilité par la configuration de ses ressources en groupes de ressources. Les données de l’application sont ainsi placées dans un système de fichiers global à haut niveau de disponibilité, de sorte qu’elles sont accessibles par un serveur opérationnel dans le cas où un serveur tombe en panne. Pour de plus amples informations sur les systèmes de fichiers de cluster, reportez-vous au *Sun Cluster Concepts Guide for Solaris OS*.

Pour permettre aux clients d’accéder au réseau, une adresse IP réseau logique est configurée dans les ressources de nom d’hôte logique contenues dans le même groupe de ressources que la ressource du service de données. La ressource de service de données et les ressources d’adresse réseau basculent en même temps. Par conséquent, les clients réseau du service de données peuvent accéder à la ressource du service de données sur le nouvel hôte.

Données multi-hôtes

Les ensembles de disques de systèmes de fichiers globaux à haut niveau de disponibilité sont multi-hôtes de sorte qu’en cas de panne d’un hôte physique, l’un des autres hôtes toujours opérationnels peut accéder au disque. Pour qu’une application atteigne un haut niveau de disponibilité, ses données doivent être hautement disponibles. Par conséquent, elles doivent résider dans les systèmes de fichiers globaux à haut niveau de disponibilité.

Le système de fichiers global est monté sur des groupes de disques créés comme des entités indépendantes. L'utilisateur peut choisir d'utiliser avec un service de données comme HA Oracle des groupes de disques en tant que systèmes de fichiers globaux montés et d'autres en tant que périphériques en mode caractères.

Une application peut contenir des commutateurs de ligne de commande ou des fichiers de configuration indiquant l'emplacement des fichiers de données. Si l'application utilise des noms de chemin d'accès câblés, vous pouvez les modifier par des liens symboliques désignant un système de fichiers global, sans modifier le code de l'application. Reportez-vous à la rubrique "Utilisation de liens symboliques pour déterminer l'emplacement des données multi-hôtes" à la page 332 pour plus de détails sur l'utilisation de ces liens symboliques.

Dans le pire des cas, vous devez modifier le code source de l'application pour disposer de quelques mécanismes permettant de spécifier l'emplacement réel des données. Pour ce faire, vous pouvez mettre en oeuvre des commutateurs de ligne de commande supplémentaires.

Sun Cluster prend en charge l'utilisation des systèmes de fichiers UFS d'UNIX et les périphériques en mode caractères à haut niveau de disponibilité configurés dans un gestionnaire de volumes. Lors de l'installation et de la configuration, l'administrateur système doit spécifier les ressources disques à utiliser avec les systèmes de fichiers UFS et les périphériques en mode caractères. En règle générale, les périphériques en mode caractères ne sont utilisés que par les serveurs de base de données et les serveurs multimédia.

Utilisation de liens symboliques pour déterminer l'emplacement des données multi-hôtes

Il arrive parfois que les noms de chemin d'accès aux fichiers de données d'une application soient câblés et qu'aucun mécanisme ne permette de contourner ces noms. Le cas échéant, il est parfois possible d'utiliser des liens symboliques pour ne pas avoir à modifier le code de l'application.

Par exemple, supposons que l'application nomme son fichier de données en utilisant le nom de chemin d'accès câblé `/etc/mydatafile`. Vous pouvez remplacer ce nom par un lien symbolique dont la valeur désigne un fichier dans lequel figure les systèmes de fichiers de l'hôte logique. Par exemple, vous pouvez définir le lien symbolique suivant : `/global/phys-schost-2/mydatafile`.

Un problème peut survenir avec l'utilisation des liens symboliques si l'application, ou l'une de ses procédures administratives, modifie le nom du fichier de données, ainsi que son contenu. Par exemple, supposons que l'application effectue une mise à jour en commençant par créer un nouveau fichier temporaire (`/etc/mydatafile.new`), puis en le renommant pour obtenir le nom de fichier réel à l'aide de l'appel système

`rename(2)` (ou du programme `mv(1)`). En créant le fichier temporaire, puis en le renommant en fichier réel, le service de données tente de garantir que le contenu de son fichier de données est toujours bien formé.

Malheureusement, l'action `rename(2)` supprime le lien symbolique. Le nom `/etc/mydatafile` devient un fichier standard et se trouve dans le même système de fichiers que le répertoire `/etc` et non dans le système de fichiers global du cluster. Comme le système de fichiers `/etc` est propre à chaque hôte, les données ne sont pas disponibles après un basculement ou une commutation.

Le problème sous-jacent dans cette situation est que l'application existante ne prend pas en charge le lien symbolique et n'a pas été écrite en tenant compte de liens symboliques. Pour rediriger un accès aux données au sein des systèmes de fichiers de l'hôte logique à l'aide de liens symboliques, la mise en oeuvre de l'application doit adopter un comportement qui ne supprime pas les liens symboliques. Par conséquent, les liens symboliques ne résolvent pas complètement le problème de l'emplacement des données sur les systèmes de fichiers globaux du cluster.

Noms d'hôtes

Vous devez déterminer si le service de données doit toujours connaître le nom d'hôte du serveur sur lequel il est exécuté. Le cas échéant, il sera peut-être nécessaire de modifier le service de données pour utiliser un nom d'hôte logique (c'est-à-dire, un nom d'hôte configuré au sein d'une ressource de nom d'hôte logique résidant dans le même groupe de ressources que la ressource de l'application), plutôt que le nom d'un hôte physique.

Il arrive parfois qu'au sein du protocole client-serveur d'un service de données, le serveur retourne son propre nom d'hôte au client comme élément de contenu d'un message qui lui est destiné. Pour ce type de protocoles, le client peut dépendre du nom d'hôte retourné, ce dernier devant être utilisé lors de la connexion au serveur. Pour que le nom d'hôte retourné soit utilisable après un basculement ou une commutation, ce doit être un nom d'hôte logique du groupe de ressources et non le nom d'un hôte physique. Le cas échéant, vous devez modifier le code du service de données pour renvoyer le nom d'hôte logique au client.

Hôtes multihome

Le terme *hôte multihome* décrit un hôte appartenant à plusieurs réseaux publics. Ce type d'hôtes a plusieurs noms d'hôte et adresses IP. Il possède, en fait, une paire nom d'hôte/adresse IP par réseau. Sun Cluster est conçu pour permettre à un hôte d'apparaître sur de nombreux réseaux, y compris un seul (le cas « non multihome »). De la même manière que le nom d'hôte physique possède plusieurs paires nom d'hôte/adresse IP, chaque groupe de ressources peut également en avoir plusieurs ; en fait, une par réseau public. Lorsque Sun Cluster déplace un groupe de ressources d'un hôte physique vers un autre, toutes les paires nom d'hôte logique/adresse IP de ce groupe sont également déplacées.

L'ensemble de paires nom d'hôte/adresse IP d'un groupe de ressources est configuré en tant que ressource de nom d'hôte logique appartenant à ce groupe. Ces ressources d'adresse réseau sont spécifiées par l'administrateur système à la création et à la configuration du groupe de ressources. L'interface API du service de données de Sun Cluster intègre des fonctions permettant de demander ces paires nom d'hôte/adresse IP.

La plupart des démons de service de données disponibles dans le commerce et dédié à l'origine au système d'exploitation Solaris gèrent également correctement les hôtes multihome. La plupart des services de données effectuent toutes leurs communications réseau en se connectant à l'adresse générique de Solaris `INADDR_ANY`. Du fait de cette liaison, les services de données sont automatiquement en mesure de gérer toutes les adresses IP de toutes les interfaces réseau. `INADDR_ANY` se connecte de façon efficace à toutes les adresses IP actuellement configurées sur la machine. Vous n'avez pas besoin de modifier le démon du service de données utilisant généralement `INADDR_ANY` pour qu'il prenne en charge les adresses réseau logiques de Sun Cluster.

Établissement d'une liaison à `INADDR_ANY` par opposition à une liaison aux adresses IP spécifiques

Même lorsque vous utilisez des hôtes non multihome, le concept des adresses réseau logiques de Sun Cluster permet à la machine de posséder plusieurs adresses IP. De fait, elle a une adresse IP pour son propre hôte physique et une autre pour chaque ressource d'adresse réseau (nom d'hôte logique) qu'elle gère actuellement. Lorsqu'elle

devient la machine maître d'une ressource d'adresse réseau, elle acquiert de façon dynamique d'autres adresses IP et lorsqu'elle perd son statut de maître, elle abandonne de façon dynamique des adresses IP.

Certains services de données ne peuvent pas fonctionner correctement au sein d'un environnement Sun Cluster s'ils sont liés à `INADDR_ANY`. Ces services doivent ainsi modifier de façon dynamique l'ensemble d'adresses IP auquel ils sont liés suivant que le groupe de ressources est géré ou non. L'une des stratégies de reliaison consiste à interrompre les méthodes de démarrage et d'arrêt de ces services de données et à redémarrer les démons du service de données.

La propriété de ressources `Ressources_réseau_utilisées` permet à l'utilisateur final de configurer un ensemble spécifique de ressources d'adresse réseau auquel la ressource de l'application peut être liée. Vous devez déclarer la propriété `Ressources_réseau_utilisées` dans le fichier RTR du type de ressources concerné pour les types de ressources exigeant cette fonctionnalité.

Lorsque le gestionnaire RGM connecte ou déconnecte le groupe de ressources, il suit un ordre spécifique de plombage, de déplombage et de configuration en amont et en aval des adresses réseau suivant le moment où il appelle les méthodes de ressource du service de données. Reportez-vous à la rubrique "Choix des méthodes `Démarrage` et `Arrêt` à utiliser" à la page 46.

Lorsque la méthode `Arrêt` du service de données est retournée, ce service doit avoir cessé d'utiliser les adresses réseau du groupe de ressources. De même, lorsque la méthode `Démarrage` est retournée, il doit avoir commencé à utiliser les adresses réseau.

Si le service de données se connecte à `INADDR_ANY` plutôt qu'aux adresses IP individuelles, l'ordre dans lequel les méthodes de ressource du service de données sont appelées n'est pas approprié.

Si les méthodes d'arrêt et de démarrage du service de données interrompent et redémarrent respectivement comme il se doit les démons du service de données, ce service s'arrête et redémarre à l'aide des adresses réseau au moment opportun.

Relance d'un client

Un basculement ou une commutation est considéré comme une panne de l'hôte logique suivie d'un redémarrage rapide au niveau d'un client du réseau. Dans l'idéal, l'application cliente et le protocole client-serveur sont structurés pour effectuer un certain nombre de relances. Si l'application et le protocole prennent déjà en charge le redémarrage d'un serveur tombé en panne, ils peuvent également gérer le

basculement ou la commutation du groupe de ressources. Certaines applications peuvent effectuer des relances indéfiniment. Les applications plus complexes avertissent l'utilisateur qu'un long processus de relance est en cours et lui permettent de choisir s'il souhaite continuer.

ANNEXE **F**

Définitions de type de document pour le protocole CRNP

Cette annexe présente les DTD (définitions de type de document) du protocole CRNP (Cluster Reconfiguration Notification Protocol).

DTD XML SC_CALLBACK_REG

Remarque – la structure de données NVPAIR utilisée par les DTD SC_CALLBACK_REG et SC_EVENT n'est définie qu'une seule fois.

```
<!— Spécification du format XML de SC_CALLBACK_REG Copyright 2001-2003 Sun Microsystems, Inc.
Tous droits réservés.
Utilisation sous licence.

Utilisation :

Ce format XML permet à un client du protocole CRNP de se connecter initialement à un service,
de se connecter ultérieurement à d'autres événements et de se déconnecter de plusieurs
événements ou du service tout entier.

Un client n'est identifié que par son port et son IP de rappel. Le port est défini dans
l'élément SC_CALLBACK_REG et l'IP correspond à l'IP source de l'ouverture de session. L'attribut
final de l'élément racine SC_CALLBACK_REG est ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT ou
REMOVE_EVENTS, suivant la forme du message que le client utilise.

SC_CALLBACK_REG contient aucun ou plusieurs sous-éléments SC_EVENT_REG.

Un sous-élément SC_EVENT_REG spécifie un type d'événements. Un client peut ne spécifier que
le type CLASS (un attribut de l'élément SC_EVENT_REG) ou indiquer une sous classe SUBCLASS
(un attribut en option) pour bénéficier d'une précision accrue. Par ailleurs, SC_EVENT_REG
comprend aucun ou plusieurs sous-éléments NVPAIR servant à spécifier encore davantage
l'événement.
```

```
Par conséquent, le client peut spécifier jusqu'à quel niveau de précision il souhaite être
notifié des événements. Veuillez noter qu'un client ne peut, dans le même message, spécifier
vouloir et ne pas vouloir être notifié des événements. Par contre, un client peut se
connecter au service et à des événements dans le même message.

Note de version : l'attribut VERSION de chaque élément racine est défini comme étant fixe,
c'est-à-dire que la version doit être spécifiée pour tous les messages qui se conforment à
ces DTD. Lorsqu'une nouvelle version du protocole est créée, l'attribut VERSION fixe des
DTD modifiées possède une nouvelle valeur, de sorte que tous les messages se conformant
à la nouvelle version doivent posséder le nouveau numéro de version.
—>

<!— Définition de SC_CALLBACK_REG

L'élément racine du document XML est un message d'enregistrement qui comprend les ports de
rappel et la version du protocole comme attribut ainsi qu'un attribut ADD_CLIENT, ADD_EVENTS,
REMOVE_CLIENT ou REMOVE_EVENTS, indiquant le type d'enregistrement. Les types ADD_CLIENT,
ADD_EVENTS et REMOVE_EVENTS doivent inclure un ou plusieurs sous-éléments SC_EVENT_REG.
REMOVE_CLIENT ne doit pas être défini comme un sous-élément de SC_EVENT_REG.
ATTRIBUTS :
VERSION Version du protocole CRNP du message.
PORT Port de rappel.
REG_TYPE Type d'enregistrement :
                        ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENUS :
SOUS-ÉLÉMENTS : SC_EVENT_REG (0 ou plusieurs)
—>
<!ELEMENT SC_CALLBACK_REG (SC_EVENT_REG*)>
<!ATTLIST SC_CALLBACK_REG
       VERSION        NMTOKEN                                                  #FIXED
       PORT           NMTOKEN                                                  #REQUIRED
       REG_TYPE       (ADD_CLIENT|ADD_EVENTS|REMOVE_CLIENT|REMOVE_EVENTS)      #REQUIRED

>
<!— Définition de SC_EVENT_REG

SC_EVENT_REG définit un événement pour lequel le client a notifié son désir ou son refus de
recevoir des notifications d'événement. L'enregistrement peut concerner n'importe quel niveau
de précision (d'une seule classe d'événements à des paires nom/valeurs spécifiques dont la
présence est indispensable). Par conséquent, le seul attribut requis est CLASS. L'attribut
SUBCLASS et les sous-éléments NVPAIRS sont optionnels et permettent d'obtenir une plus
grande précision.

Les enregistrements spécifiant les paires nom/valeurs pour lesquelles le client souhaite
recevoir une notification indiquent l'acceptation des notifications de message de la
classe/sous-classe spécifiée comportant TOUTES les paires nom/valeurs présentes.
Les enregistrements spécifiant les paires nom/valeurs pour lesquelles le client ne souhaite
pas recevoir de notification indiquent le rejet des notifications qui comportent EXACTEMENT
les paires nom/valeurs de la précision spécifiée précédemment.
Les enregistrements ne spécifiant pas les paires nom/valeurs pour lesquelles le client ne
souhaite pas recevoir de notification indiquent le rejet de TOUTES les notifications de
la classe/sous-classe spécifiée.
```

```
        ATTRIBUTS :
               CLASS:        Classe d'événements pour laquelle l'élément est enregistré ou non.
               SUBCLASS:     Sous-classe d'un événement (facultatif).

        CONTENU :
               SUBELEMENTS: 0 ou plusieurs NVPAIR.
—>

<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA               #REQUIRED
       SUBCLASS       CDATA               #IMPLIED
>
```

DTD XML NVPAIR

```
<!— Spécification du format XML de NVPAIR

Copyright 2001-2003 Sun Microsystems, Inc. Tous droits réservés.
Utilisation sous licence.

Utilisation :
Un élément nvpair est utilisé dans un élément SC_EVENT ou SC_CALLBACK_REG.
—>

<!— Définition de NVPAIR

NVPAIR est une paire nom/valeurs servant à représenter des combinaisons arbitraires
de nom/valeurs.
Un élément nvpair représente la traduction directe et générique de la structure
nvpair_t de Solaris utilisée par la structure sysevent. Aucun type de données n'est
associé au nom et à la valeur (qui ne sont que du texte arbitraire) dans cet élément xml.

NVPAIR comprend uniquement un élément NAME et un ou plusieurs éléments VALUE.
Un élément VALUE représente une valeur scalaire, tandis que plusieurs éléments VALUE
représentent un ensemble de valeurs.

ATTRIBUTS :

CONTENU :
SUBELEMENTS: NAME(1), VALUE(1 ou plusieurs)
—>

<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— Définition de NAME

NAME représente simplement une chaîne dont la longueur est arbitraire.

ATTRIBUTS :
```

```
       CONTENU :
Données texte arbitraires à introduire entre <![CDATA[...]]> pour être
ignorées par l'analyse XML.
—>
<!ELEMENT NAME (#PCDATA)>

<!— Définition de VALUE
VALUE représente simplement une chaîne dont la longueur est arbitraire.

ATTRIBUTS :

CONTENU :
Données texte arbitraires à introduire entre <![CDATA[...]]> pour être
ignorées par l'analyse XML.
—>

<!ELEMENT VALUE (#PCDATA)>
```

DTD XML SC_REPLY

```
<!— Spécification du format XML de SC_REPLY

Copyright 2001-2003 Sun Microsystems, Inc. Tous droits réservés.
Utilisation sous licence.
—>

<!— Définition de SC_REPLY

L'élément racine du document XML représente une réponse à un message. Cette
 réponse contient un code et un message d'état.

        ATTRIBUTS :
                VERSION:        Version du protocole CRNP du message.
                STATUS_CODE:    Code de retour du message. Il peut s'agir de l'un des codes
 suivants : OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
 MALFORMED, INVALID_XML, VERSION_TOO_HIGH ou VERSION_TOO_LOW.

                CONTENU :
                        SUBELEMENTS: SC_STATUS_MSG(1)
—>

<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
        VERSION         NMTOKEN                                         #FIXED   "1.0"
        STATUS_CODE     OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
                        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>
```

```
<!— Définition de SC_STATUS_MSG
SC_STATUS_MSG est simplement une chaîne de texte arbitraire expliquant le code d'état.
À introduire entre <![CDATA[...]]> pour être ignoré par l'analyse XML.

        ATTRIBUTS :

        CONTENU :
                Chaîne arbitraire.
—>

<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

DTD XML SC_EVENT

Remarque – la structure de données NVPAIR utilisée par les DTD SC_CALLBACK_REG et SC_EVENT n'est définie qu'une seule fois.

```
<!— Spécification du format XML de SC_EVENT

Copyright 2001-2003 Sun Microsystems, Inc. Tous droits réservés.
Utilisation sous licence.

L'élément racine du document XML représente la traduction directe et générique
du format de message sysseventd de Solaris. Il a des attributs représentant la classe,
la sous-classe, le fournisseur et l'éditeur et contient un nombre illimité
d'éléments NVPAIR.

ATTRIBUTS :
        VERSION: Version du protocole CRNP du message.
        CLASS: Classe sysevent de l'événement
        SUBCLASS: Sous-classe de l'événement
        VENDOR: Fournisseur associé à l'événement
        PUBLISHER: Éditeur de l'événement
CONTENU :
       SUBELEMENTS: NVPAIR (0 ou plusieurs)
—>

<!ELEMENT SC_EVENT (NVPAIR*)>
<!ATTLIST SC_EVENT
        VERSION         NMTOKEN         #FIXED "1.0"
        CLASS           CDATA           #REQUIRED
        SUBCLASS        CDATA           #REQUIRED
        VENDOR          CDATA           #REQUIRED
        PUBLISHER       CDATA           #REQUIRED
>
```

ANNEXE **G**

Application `CrnpClient.java`

Cette annexe montre l'application `CrnpClient.java` dans son ensemble. Celle-ci est abordée en détails dans le Chapitre 12.

Contenu de `CrnpClient.java`

```
/*
 * CrnpClient.java
 * ================
 *
 * Remarque concernant l'analyse XML :
 *
 * Ce programme utilise l'API Sun Java Architecture for XML Processing (JAXP).
 * Reportez-vous à http://java.sun.com/xml/jaxp/index.html pour obtenir de la documentation
 * sur les API ainsi que des informations relatives à leur disponibilité
 * des informations relatives à leur disponibilité.
 *
 * Ce programme a été rédigé pour Java 1.3.1 ou supérieur.
 *
 * Présentation du programme :
 *
 * Le thread principal du programme crée un objet CrnpClient, attend que
 * l'utilisateur mette un terme à la démonstration, puis appelle la fermeture de
 * l'objet CrnpClient et quitte le programme.
 *
 * Le constructeur de CrnpClient crée un objet EventReceptionThread,
 * ouvre une connexion au serveur CRNP (à l'aide de l'hôte et du port spécifiés
 * dans la ligne de commande), construit un message d'enregistrement (sur la base
 * des spécifications de la ligne de commande), envoie un message d'enregistrement,
 * puis lit et analyse la réponse.
 *
 * L'objet EventReceptionThread crée un socket d'écoute lié au
 * nom d'hôte de la machine sur laquelle tourne le programme et au port
```

```
 * spécifié sur la ligne de commande. Il attend un rappel d'événement entrant,
 * moment auquel il construit un document XML sur la base du flux de socket entrant,
 * qu'il transmet à l'objet CrnpClient pour traitement.
 *
 * La méthode de fermeture de l'objet CrnpClient envoie simplement un message de
 * désenregistrement (REMOVE_CLIENT) SC_CALLBACK_REG au serveur crnp.
 *
 * Remarque concernant la gestion des erreurs : pour être bref, ce programme
 * se ferme purement et simplement dans la plupart des cas. Il est évident qu'une
 * véritable application tenterait de gérer les erreurs de diverses manières, par exemple
 * en lançant de nouvelles tentatives, le cas échéant.
 */

// Packages JAXP
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

// Packages standard
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * class CrnpClient
 * ----------------
 * Voir les commentaires présentés ci-dessus dans l'en-tête du fichier.
 */
class CrnpClient
{
    /*
     * main
     * ----
     * Point d'entrée de l'exécution, main vérifie surtout
     * le nombre d'arguments de ligne de commande et construit une instance
     * d'un CrnpClient chargé d'effectuer toutes les tâches.
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

            /* Vérifie le nombre d'arguments de ligne de commande */
        if (args.length < 4) {
            System.out.println(
                "Usage: java CrnpClient Hôte_crnp Port_crnp "
                + "Port_local (-ac | -ae | -re) "
                + "[(M | A | RG=nom | R=nom) [...]]");
            System.exit(1);
        }
```

```
    /*
     * Nous nous attendons à ce que la ligne de commande contienne l'adresse IP/le
     * port du serveur crnp, le port local d'écoute ainsi que des arguments
     * indiquant le type d'enregistrement.
     */
    try {
        regIp = InetAddress.getByName(args[0]);
        regPort = (new Integer(args[1])).intValue();
        localPort = (new Integer(args[2])).intValue();
    } catch (UnknownHostException e) {
        System.out.println(e);
        System.exit(1);
    }

    // Créez le CrnpClient
    CrnpClient client = new CrnpClient(regIp, regPort, localPort,
        args);

    // Attendez maintenant que l'utilisateur souhaite terminer le programme
    System.out.println("Hit return to terminate demo...");

    // la lecture sera bloquée jusqu'à ce que l'utilisateur entre quelque chose
    try {
        System.in.read();
    } catch (IOException e) {
        System.out.println(e.toString());
    }

        // fermez le client
    client.shutdown();
    System.exit(0);
}

/*
 * ======================
 * Méthodes publiques
 * ======================
 */

/*
 * Constructeur de l'objet CrnpClient
 * ----------------------
 * Ce constructeur analyse les arguments de ligne de commande, ce qui nous permet
 * de savoir comment contacter le serveur crnp, crée le thread de réception des
 * événements et le démarre, crée l'objet XML DocumentBuilderFactory et enfin,
 * s'enregistre auprès du serveur crnp pour les rappels.
 */
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
```

```
        localPort = localPortIn;
        regs = clArgs;

        /*
         * Configuration de l'usine du générateur de documents pour
         * le traitement du format xml.
         */
        setupXmlProcessing();

        /*
         * Création de l'objet EventReceptionThread, qui crée un
         * ServerSocket et le lie à une adresse IP et à un port locaux.
         */
        createEvtRecepThr();

        /*
         * Enregistrement auprès du serveur crnp.
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * ---------------
 * Rappel de l'objet CrnpClient, utilisé par EventReceptionThread
 * quand il reçoit des rappels d'événements.
 */
public void processEvent(Event event)
{
    /*
     * Pour les besoins de la démonstration, il vous suffit d'imprimer l'événement
     * sur System.out. Une véritable application utiliserait
     * l'événement d'une manière ou d'une autre.
     */
    event.print(System.out);
}

/*
 * shutdown
 * -------------
 *  Désenregistrement du serveur CRNP.
 */
public void shutdown()
{
    try {
        /* envoi d'un message de désenregistrement au serveur */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
```

```
    }
}


/*
 * ======================
 * Méthodes des assistants privés
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Il crée l'usine du générateur de documents permettant
 * l'analyse des événements et réponses xml.
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    //  Il n'est pas nécessaire de valider
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // Nous souhaitons ignorer les commentaires et les blancs
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // Combinez les sections CDATA et les noeuds TEXT.
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -------------------
 * Il crée un nouvel objet EventReceptionThread, enregistre l'adresse IP
 * et le port auxquels est lié le socket d'écoute et
 * démarre le thread.
 */
private void createEvtRecepThr() throws Exception
{
    /* Création de l'objet du thread */
    evtThr = new EventReceptionThread(this);

    /*
     * Démarrage du thread pour lancer l'écoute
     * des rappels d'envoi d'événements.
     */
    evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
```

```
 * Il établit une connexion de socket au serveur crnp et envoie
 * un message d'enregistrement d'événement.
 */
private void registerCallbacks() throws Exception
{
    System.out.println("About to register");

    /*
     * Création d'un socket connecté à l'adresse IP/au port d'enregistrement
     * du serveur crnp et envoi des informations d'enregistrement.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Lecture de la réponse
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Fermeture de la connexion de socket
     */
    sock.close();
}

/*
 * unregister
 * ----------
 * Comme dans le cas de registerCallbacks, une connexion de socket est établie avec
 * le serveur crnp, le message de désenregistrement est envoyé et le système
 * attend la réponse du serveur, puis ferme le socket.
 */
private void unregister() throws Exception
{
    System.out.println("About to unregister");

    /*
     *Création d'un socket connecté à l'adresse IP/au port d'enregistrement
     * du serveur crnp et envoi des informations de désenregistrement.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Lecture de la réponse
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Fermeture de la connexion de socket
     */
```

```
    sock.close();
}

/*
 * createRegistrationString
 * ------------------
 * Il construit un objet CallbackReg basé sur les arguments de ligne de commande
 * de ce programme, puis récupère la chaîne XML depuis l'objet
 * CallbackReg.
 */
private String createRegistrationString() throws Exception
{
    /*
     * Création de la classe CallbackReg à proprement parler et définition du port.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    // définissez le type d'enregistrement
    if (regs[3].equals("-ac")) {
        cbReg.setRegType(CallbackReg.ADD_CLIENT);
    } else if (regs[3].equals("-ae")) {
        cbReg.setRegType(CallbackReg.ADD_EVENTS);
    } else if (regs[3].equals("-re")) {
        cbReg.setRegType(CallbackReg.REMOVE_EVENTS);
    } else {
        System.out.println("Invalid reg type: " + regs[3]);
        System.exit(1);
    }

    // ajoutez les événements
    for (int i = 4; i < regs.length; i++) {
        if (regs[i].equals("M")) {
            cbReg.addRegEvent(
                createMembershipEvent());
        } else if (regs[i].equals("A")) {
            cbReg.addRegEvent(
                createAllEvent());
        } else if (regs[i].substring(0,2).equals("RG")) {
            cbReg.addRegEvent(createRgEvent(
                regs[i].substring(3)));
        } else if (regs[i].substring(0,1).equals("R")) {
            cbReg.addRegEvent(createREvent(
                regs[i].substring(2)));
        }
    }

    String xmlStr = cbReg.convertToXml();
    System.out.println(xmlStr);
    return (xmlStr);
}

/*
 * createAllEvent
 * ----------------
```

```
 * Il crée un événement d'enregistrement XML avec une classe EC_Cluster et aucune
 * sous-classe.
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
 * ---------------------
 * Il crée un événement d'enregistrement XML avec la classe EC_Cluster, sous-classe
 * ESC_cluster_memberhip.
 */
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Il crée un événement d'enregistrement XML avec la classe EC_Cluster,
 * sous-classe ESC_cluster_rg_state, et aucun "nom_rg" nvpair (sur la base
 * du paramètre d'entrée).
 */
private Event createRgEvent(String rgname)
{
    /*
     * Création d'un événement de changement d'état pour le
     * groupe de ressources nom_rg. Remarquez que nous fournissons
     * un couple nom/valeur (nvpair) pour ce type d'événement, pour
     * spécifier le groupe de ressources qui nous intéresse.
     */
    /*
     * Construction de l'objet d'événement et paramétrage des classes et sous-classes.
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * Création de l'objet nvpair et ajout de celui-ci à l'événement
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
```

```
}

/*
 * createREvent
 * ----------------
 * Il crée un événement d'enregistrement XML avec la classe EC_Cluster,
 * sous-classe ESC_cluster_r_state, et un nvpair "nom-r" (sur la base
 * du paramètre d'entrée).
 */
private Event createREvent(String rname)
{
    /*
     * Création d'un événement de changement d'état pour
     * la ressource nom_rg. Remarquez que nous fournissons
     * un couple nom/valeur (nvpair) pour ce type d'événement afin
     * de spécifier le groupe de ressources qui nous intéresse.
     */
    Event rStateEvent = new Event();
    rStateEvent.setClass("EC_Cluster");
    rStateEvent.setSubclass("ESC_cluster_r_state");

    NVPair rNvpair = new NVPair();
    rNvpair.setName("r_name");
    rNvpair.setValue(rname);
    rStateEvent.addNvpair(rNvpair);

    return (rStateEvent);
}

/*
 * createUnregistrationString
 * ------------------
 * Il construit un objet REMOVE_CLIENT CallbackReg, puis récupère
 * la chaîne XML de l'objet CallbackReg.
 */
private String createUnregistrationString() throws Exception
{
    /*
     * Création de l'objet CallbackReg.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
     * Classement de l'enregistrement dans l'OutputStream
     */
    String xmlStr = cbReg.convertToXml();

    // Imprimez la chaîne dans un but de débogage
    System.out.println(xmlStr);
    return (xmlStr);
}
```

```
    /*
     * readRegistrationReply
     * ------------------------
     * Il analyse le langage xml dans un document, construit un objet RegReply
     * à partir du document et imprime l'objet RegReply. Remarquez
     * qu'une véritable application agirait sur la base du status_code
     * de l'objet RegReply.
     */
    private void readRegistrationReply(InputStream stream)
        throws Exception
    {
        // Créez le constructeur de document
        DocumentBuilder db = dbf.newDocumentBuilder();

        //
        // Définissez un Gestionnaire d'erreur avant l'analyse
        // Utilisez le gestionnaire par défaut.
        //
        db.setErrorHandler(new DefaultHandler());

        // analysez le fichier d'entrée
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
    }

    /* variables des membres privés */
    private InetAddress regIp;
    private int regPort;
    private EventReceptionThread evtThr;
    private String regs[];

    /* variables des membres publics */
    public int localPort;
    public DocumentBuilderFactory dbf;
}

/*
 * class EventReceptionThread
 * ---------------------------
 * Reportez-vous aux commentaires ci-dessus dans l'en-tête du fichier.
 */
class EventReceptionThread extends Thread
{
    /*
     * Constructeur d'EventReceptionThread
     * ---------------------------------
     * Il crée un nouveau ServerSocket, associé au nom d'hôte local et à
     * un port générique.
     */
    public EventReceptionThread(CrnpClient clientIn) throws IOException
    {
        /*
         * Conservation d'une référence au client permettant de le rappeler
```

```
     * à la réception d'un événement.
     */
    client = clientIn;

    /*
     * Spécification de l'adresse IP à laquelle s'associer.
     * Il s'agit simplement de l'IP de l'hôte local. S'il existe plus
     * d'une interface publique configurée sur cette
     * machine, nous suivons celle proposée par
     * InetAddress.getLocalHost.
     *
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
        System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Appelé par la méthode Thread.Start.
 *
 * Boucle infinie, attendant des connexions entrantes sur le ServerSocket.
 *
 * Toutes les connexions entrantes étant acceptées, un objet Event
 * est créé depuis le flux xml. Il est ensuite transmis à
 * l'objet CrnpClient pour traitement.
 */
public void run()
{
    /*
     * Boucle infinie.
     */
    try {
        //
        // Créez le constructeur de document à l'aide de l'usine
        // du constructeur dans le CrnpClient.
        //
        DocumentBuilder db = client.dbf.newDocumentBuilder();

        //
        // Définissez un Gestionnaire d'erreur avant l'analyse
        // Utilisez le gestionnaire par défaut.
        //
        db.setErrorHandler(new DefaultHandler());

        while(true) {
            /* Attente d'un rappel du serveur */
            Socket sock = listeningSock.accept();

            // analysez le fichier d'entrée
            Document doc = db.parse(sock.getInputStream());

            Event event = new Event(doc);
            client.processEvent(event);
```

```
                /*Fermeture du socket */
                sock.close();
            }
            // IMPOSSIBLE À ATTEINDRE

        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }

    /* Variables des membres privés */
    private ServerSocket listeningSock;
    private CrnpClient client;
}

/*
 * classe NVPair
 * -----------
 * Cette classe enregistre une paire nom/valeur (deux chaînes). Elle sait comment
 * construire un message NVPAIR XML depuis ses membres et comment décomposer
 * un élément NVPAIR XML en ses membres.
 *
 * Remarquez que la spécification formelle d'un couple NVPAIR autorise plusieurs valeurs.
 * Pour simplifier, nous nous basons ici sur une seule valeur.
 */
class NVPair
{
    /*
     * Deux constructeurs : le premier crée un couple NVPair vide, le second
     * un couple NVPair depuis un élément NVPAIR XML.
     */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
     * Réglages publics.
     */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }
```

```
/*
 * Impression du nom et de la valeur sur une seule ligne.
 */
public void print(PrintStream out)
{
    out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Il construit un élément NVPAIR XML à partir des variables des membres.
 * Il considère le document comme un paramètre, de manière à pouvoir créer
 * l'élément.
 */
public Element createXmlElement(Document doc)
{
    // Créez l'élément.
    Element nvpair = (Element)
        doc.createElement("NVPAIR");
    //
    // Ajoutez le nom. Remarquez que le nom actuel est
    // une section CDATA séparée.
    //
    Element eName = doc.createElement("NAME");
    Node nameData = doc.createCDATASection(name);
    eName.appendChild(nameData);
    nvpair.appendChild(eName);
    //
    // Ajoutez la valeur. Remarquez que la valeur actuelle est
    // une section CDATA séparée.
    //
    Element eValue = doc.createElement("VALUE");
    Node valueData = doc.createCDATASection(value);
    eValue.appendChild(valueData);
    nvpair.appendChild(eValue);

    return (nvpair);
}

/*
 * retrieveValues
 * ----------------
 * Il analyse l'élément XML pour récupérer le nom et la valeur.
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // Trouvez l'élément NOM
    //
    nl = elem.getElementsByTagName("NAME");
```

```
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }

    //
    // Obtenez la section TEXT
    //
    n = nl.item(0).getFirstChild();
    if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        System.out.println("Error in parsing: can't find "
            + "TEXT section.");
        return;
    }

    // Récupérez la valeur
    name = n.getNodeValue();

    //
    // Définissez maintenant l'élément de valeur
    //
    nl = elem.getElementsByTagName("VALUE");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "VALUE node.");
        return;
    }

    //
    // Définissez la section TEXT
    //
    n = nl.item(0).getFirstChild();
    if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        System.out.println("Error in parsing: can't find "
            + "TEXT section.");
        return;
    }

    // Récupérez la valeur
    value = n.getNodeValue();
}


/*
 * Accès public
 */
public String getName()
{
    return (name);
}

public String getValue()
{
    return (value);
```

```
    }

    // Var membre privé
    private String name, value;
}


/*
 * class Event
 * -----------
 * Cette classe enregistre un événement se composant d'une classe, d'une sous-classe,
 * d'un fournisseur, d'un éditeur et d'une liste de couples nom/valeur. Elle sait
 * comment construire un élément SC_EVENT_REG XML à partir de ses membres et comment
 * décomposer un élément SC_EVENT XML en ses membres. Remarquez qu'il existe une
 * asymétrie ici : nous analysons des éléments SC_EVENT, mais construisons des
 * éléments SC_EVENT_REG. Ceci est dû au fait que les éléments SC_EVENT_REG sont
 * (que nous devons construire), alors que les éléments SC_EVENT sont utilisés dans
 * utilisés dans des messages d'enregistrement les envois d'événements (que nous
 * devons analyser). La seule différence réside dans le fait que les éléments SC_EVENT_REG
 * n'ont pas de fournisseurs ou d'éditeurs.
 */
class Event
{

    /*
     * Deux constructeurs : le premier crée un événement vide. Le second
     * crée un événement à partir d'un document SC_EVENT XML.
     */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // Convertissez le document en une chaîne à imprimer dans un but de
        // débogage.
        //
        DOMSource domSource = new DOMSource(doc);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return;
        }
        System.out.println(strWrite.toString());
```

```
    // Procédez à l'analyse actuelle.
    retrieveValues(doc);
}

/*
 *  Réglages publics.
 */
public void setClass(String classIn)
{
    regClass = classIn;
}

public void setSubclass(String subclassIn)
{
    regSubclass = subclassIn;
}

public void addNvpair(NVPair nvpair)
{
    nvpairs.add(nvpair);
}

/*
 * createXmlElement
 * ------------------
 * Il construit un élément SC_EVENT_REG XML à partir des variables des membres.
 * Il considère le document comme un paramètre afin de pouvoir créer
 * l'élément. Il se fonde sur la capacité NVPair createXmlElement.
 */
public Element createXmlElement(Document doc)
{
    Element event = (Element)
        doc.createElement("SC_EVENT_REG");
    event.setAttribute("CLASS", regClass);
    if (regSubclass != null) {
        event.setAttribute("SUBCLASS", regSubclass);
    }
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        event.appendChild(tempNv.createXmlElement(
            doc));
    }
    return (event);
}

/*
 * Impression des variables des membres sur plusieurs lignes.
 */
public void print(PrintStream out)
{
    out.println("\tCLASS=" + regClass);
    out.println("\tSUBCLASS=" + regSubclass);
    out.println("\tVENDOR=" + vendor);
```

```
    out.println("\tPUBLISHER=" + publisher);
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        out.print("\t\t");
        tempNv.print(out);
    }
}

 /*
 * retrieveValues
 * ----------------
 * Analyse du document XML pour récupérer la classe, la sous-classe, le fournisseur,
 * l'éditeur et les couples nvpair.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Trouvez l'élément SC_EVENT.
    //
    nl = doc.getElementsByTagName("SC_EVENT");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "SC_EVENT node.");
        return;
    }

    n = nl.item(0);

    //
    // Récupérez les valeurs des attributs de CLASS, SUBCLASS,
    // VENDOR et PUBLISHER.
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // Récupérez toutes les paires nv
    //
    for (Node child = n.getFirstChild(); child != null;
         child = child.getNextSibling())
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 *  Méthodes d'accès public.
 */
public String getRegClass()
```

```
    {
        return (regClass);
    }

    public String getSubclass()
    {
        return (regSubclass);
    }

    public String getVendor()
    {
        return (vendor);
    }

    public String getPublisher()
    {
        return (publisher);
    }

    public Vector getNvpairs()
    {
        return (nvpairs);
    }

    //Var membre privé.
    private String regClass, regSubclass;
    private Vector nvpairs;
    private String vendor, publisher;
}


/*
 * classe CallbackReg
 * -----------
 * Cette classe enregistre un port et un regType (deux chaînes) ainsi qu'une liste .
 * des événements. Elle sait comment construire un message SC_CALLBACK_REG XML à partir
 * de ses membres.
 *
 * Remarquez que cette classe ne doit pas être en mesure d'analyser les messages
 * SC_CALLBACK_REG parce que seul le serveur CRNP doit
 * le faire.
 */
class CallbackReg
{
    // Définition utile pour la méthode setRegType
    public static final int ADD_CLIENT = 0;
    public static final int ADD_EVENTS = 1;
    public static final int REMOVE_EVENTS = 2;
    public static final int REMOVE_CLIENT = 3;

    public CallbackReg()
    {
        port = null;
        regType = null;
        regEvents = new Vector();
```

```
}

/*
 * Réglages publics.
 */
public void setPort(String portIn)
{
    port = portIn;
}

public void setRegType(int regTypeIn)
{
    switch (regTypeIn) {
    case ADD_CLIENT:
        regType = "ADD_CLIENT";
        break;
    case ADD_EVENTS:
        regType = "ADD_EVENTS";
        break;
    case REMOVE_CLIENT:
        regType = "REMOVE_CLIENT";
        break;
    case REMOVE_EVENTS:
        regType = "REMOVE_EVENTS";
        break;
    default:
        System.out.println("Error, invalid regType " +
            regTypeIn);
        regType = "ADD_CLIENT";
        break;
    }
}

public void addRegEvent(Event regEvent)
{
    regEvents.add(regEvent);
}


/*
 * convertToXml
 * ------------------
 * Il construit un document SC_CALLBACK_REG XML à partir des variables
 * des membres. Il se fonde sur la capacité Event createXmlElement.
 */
public String convertToXml()
{
    Document document = null;
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.newDocument();
    } catch (ParserConfigurationException pce) {
        // L'analyseur avec des options spécifiques ne peut être créé
```

```
            pce.printStackTrace();
            System.exit(1);
        }
        Element root = (Element) document.createElement(
            "SC_CALLBACK_REG");
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("REG_TYPE", regType);
        for (int i = 0; i < regEvents.size(); i++) {
            Event tempEvent = (Event)
                (regEvents.elementAt(i));
            root.appendChild(tempEvent.createXmlElement(
                document));
        }
        document.appendChild(root);

        //
        // Convertissez maintenant le document en chaîne.
        //
        DOMSource domSource = new DOMSource(document);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return ("");
        }
        return (strWrite.toString());
    }

    // var membre privé
    private String port;
    private String regType;
    private Vector regEvents;
}

/*
 * class RegReply
 * -----------
 * Cette classe enregistre un code_état et un msg_état (deux chaînes).
 * Elle sait comment décomposer un élément SC_REPLY XML en ses membres.
 */
class RegReply
{

    /*
     * L'unique constructeur prend un document XML et l'analyse.
     */
    public RegReply(Document doc)
    {
        //
        // Convertissez maintenant le document en chaîne.
```

```
    //
    DOMSource domSource = new DOMSource(doc);
    StringWriter strWrite = new StringWriter();
    StreamResult streamResult = new StreamResult(strWrite);
    TransformerFactory tf = TransformerFactory.newInstance();
    try {
        Transformer transformer = tf.newTransformer();
        transformer.transform(domSource, streamResult);
    } catch (TransformerException e) {
        System.out.println(e.toString());
        return;
    }
    System.out.println(strWrite.toString());

    retrieveValues(doc);
}

/*
 * Accès public
 */
public String getStatusCode()
{
    return (statusCode);
}

public String getStatusMsg()
{
    return (statusMsg);
}

/*
 * Impression des informations sur une seule ligne.
 */
public void print(PrintStream out)
{
    out.println(statusCode + ": " +
        (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Analyse des documents XML pour récupérer le Code_état et le Msg_état.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Trouvez l'élément SC_REPLY.
    //
    nl = doc.getElementsByTagName("SC_REPLY");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
```

```
                + "SC_REPLY node.");
            return;
        }

        n = nl.item(0);

        // Récupérez la valeur du STATUS_CODE attribute
        statusCode = ((Element)n).getAttribute("STATUS_CODE");

        //
        // Trouvez l'élément SC_STATUS_MSG
        //
        nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "SC_STATUS_MSG node.");
            return;
        }

        //
        // Obtenez la section TEXT, le cas échéant.
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Aucune erreur s'il n'y en a pas, par conséquent nous
            // revenons simplement sans message.
            return;
        }

        // Récupérez la valeur
        statusMsg = n.getNodeValue();
    }

    // var membre privé
    private String statusCode;
    private String statusMsg;
}
```

Index

B

C

D

J

K

L

M

N

O

P

R

S

T

V