Guide du développeur de services de données Sun Cluster pour SE Solaris

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

Référence : 819–2067
Août 2005, Révision A

050817@12762

Table des matières

Préface

Le *Guide du développeur de 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 document

Le *Guide du développeur de services de données Sun Cluster pour SE Solaris* contient les chapitres et les annexes suivants :

Le Chapitre 1 présente les concepts nécessaires au développement d'un service de données.

Le Chapitre 2 apporte des informations détaillées sur le développement d'un service de données.

Le Chapitre 3 présente les fonctions d'accès et les méthodes de rappel constituant l'API de gestion des ressources (APIGR).

Le Chapitre 4 traite des problèmes à appréhender avant de modifier un type de ressource. Il comprend également des informations sur les différentes méthodes vous permettant d'autoriser un administrateur de clusters à mettre à niveau une ressource.

Le Chapitre 5 présente un exemple de service de données Sun Cluster pour l'application `in.named`.

Le Chapitre 6 présente les interfaces de programmation d'application constituant la bibliothèque DSDL (Data Services Development Library).

Le Chapitre 7 décrit l'utilisation habituelle de la bibliothèque DSDL lors de la conception et de l'implémentation des types de ressources.

Le Chapitre 8 donne un exemple de type de ressources implémenté via la bibliothèque DSDL..

Le Chapitre 9 décrit SunPlex™ Agent Builder.

Le Chapitre 10 explique comment créer un service de données générique.

Le Chapitre 11 définit les fonctions d'API de la bibliothèque DSDL.

Le Chapitre 12 fournit des informations sur le protocole CRNP, qui permet aux applications de basculement et évolutives d'être compatibles avec les clusters.

L'Annexe A décrit les propriétés standard des types de ressources, des ressources et des groupes de ressources.

L'Annexe B fournit le code complet de chaque méthode dans un exemple de service de données.

L'Annexe C présente le code complet de chaque méthode dans le type de ressource `SUNW.xfnts`.

L'Annexe D répertorie les exigences que doivent respecter les caractères légaux des valeurs et des noms du RGM.

L'Annexe E définit les exigences que doivent respecter les applications normales, non compatibles avec les clusters, pour pouvoir être hautement disponibles.

L'Annexe F fait la liste des définitions de type de document pour le protocole CRNP.

L'Annexe G illustre l'application `CrnpClient.java` complète décrite dans le Chapitre 12.

Documentation connexe

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

Rubrique	Documentation
Présentation	*Présentation de Sun Cluster pour SE Solaris*
Concepts	*Guide des notions fondamentales de Sun Cluster pour SE Solaris*
Installation et administration matérielle	*Sun Cluster 3.0-3.1 Hardware Administration Manual for Solaris OS*
	Guides d'administration matérielle individuelle
Installation du logiciel	*Guide d'installation du logiciel Sun Cluster pour SE Solaris*
Installation et administration de 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	*Guide du développeur de services de données Sun Cluster pour SE Solaris*
Administration du système	*Guide d'administration système de Sun Cluster pour SE Solaris*
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*

Pour obtenir la liste complète de la documentation Sun Cluster, reportez-vous aux notes de version relatives à votre version du logiciel Sun Cluster à l'adresse `http://docs.sun.com`.

Accès à l'aide

Si vous n'arrivez pas à installer ou à utiliser le logiciel Sun Cluster, adressez-vous à votre fournisseur de services et communiquez-lui les renseignements suivants :

- votre nom et votre adresse de messagerie électronique ;
- 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 Solaris10) ;
- le numéro de version de Sun Cluster (par exemple Sun Cluster 3.1).

Pour obtenir des informations sur votre système, 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`.

Documentation, support et formation

Fonction Sun	URL	Description
Documentation	`http://www.sun.com/documentation/`	Télécharger des documents PDF et HTML, et commander des documents imprimés

Fonction Sun	URL	Description
Support et formation	`http://www.sun.com/supportraining/`	Obtenir une assistance technique, télécharger des patchs et recevoir des informations sur les cours Sun

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, fichiers, répertoires et 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`** `Mot de passe :`
aabbcc123	Paramètre substituable de ligne de commande à remplacer par un nom ou une valeur	La commande permettant de supprimer un fichier est `rm` *nom_fichier*.
AaBbCc123	Titres de manuels, termes nouveaux et mis en évidence.	Reportez-vous au chapitre 6 du *Guide de l'utilisateur*. Effectuez une *analyse de patchs*. *N'*enregistrez *pas* le fichier. [Notez que certains éléments mis en évidence s'affichent en gras sur le site.]

Invites de 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 du shell

Shell	Invite
Invite en C shell	`machine-name%`
Invite du superutilisateur en C shell	`machine-name#`
Invites 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 instructions pour la création d'un type de ressources destiné à une application telle qu'Oracle®, Sun Java™ System Web Server (anciennement Sun ONE Web Server) ou DNS. Il s'adresse aux développeurs de types de ressources.

Ce chapitre présente les concepts à comprendre pour développer un service de données. Il contient les rubriques suivantes :

Remarque – les expressions *type de ressource* et *service de données* sont utilisées de façon interchangeable dans ce manuel. Le terme *agent*, rarement utilisé dans ce manuel, désigne un *type de ressource* ou un *service de données*.

Environnement d'application Sun Cluster

Le système Sun Cluster vous permet de gérer et d'administrer vos applications comme des ressources hautement disponibles ou évolutives. Le gestionnaire de groupes de ressources (Resource Group Manager, RGM) fournit la structure de base de la haute disponibilité et de l'évolutivité. Son interface de programmation est constituée des éléments suivants :

- Un ensemble de méthodes de rappel à créer pour l'autoriser à contrôler une application du cluster.

- L'interface de programmation d'application de gestion des ressources (Resource Management API, APIGR), ensemble de fonctions et de commandes API de bas niveau qui vous permet de créer les méthodes de rappel. Ces API sont mises en œuvre dans la bibliothèque `libscha.so`.
- L'utilitaire de contrôle de processus (Process Monitor Facility, PMF), qui vous permet de contrôler et de redémarrer des processus du cluster.
- La bibliothèque de développement de services de données (Data Service Development Library, DSDL), ensemble de fonctions de bibliothèque qui encapsule les API de bas niveau et la fonction de gestion de processus à un niveau plus élevé. La bibliothèque DSDL fournit quelques fonctions destinées à faciliter la création de méthodes de rappel. Ces fonctions sont mises en œuvre dans la bibliothèque `libdsdev.so`.

La figure ci-dessous illustre les relations entre ces éléments.

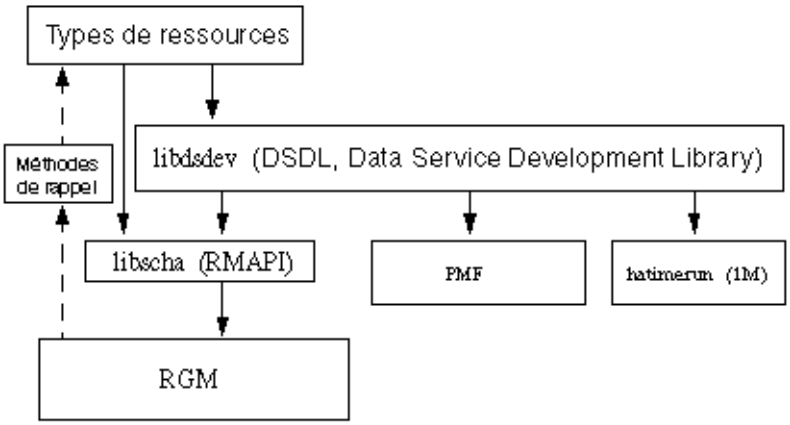


FIGURE 1–1 Architecture de programmation de l'environnement d'application Sun Cluster

SunPlex Agent Builder (décrit au Chapitre 9) est un outil du package Sun Cluster qui automatise la création des services de données. Il génère du code de service de données en langage C (s'il utilise les fonctions de la bibliothèque DSDL pour écrire les commandes de rappel) ou Korn (`ksh`) (s'il utilise les commandes API de bas niveau).

Le gestionnaire RGM est exécuté, en tant que démon, sur chaque nœud du cluster ; il démarre et arrête automatiquement les ressources des nœuds sélectionnés suivant des règles préconfigurées. Il assure la haute disponibilité d'une ressource, en cas de défaillance ou de redémarrage d'un nœud, en arrêtant cette ressource sur le nœud affecté et en la redémarrant sur un autre nœud. Il démarre et arrête automatiquement les détecteurs spécifiques à chaque ressource, qui détectent les défaillances de ressources et réaffectent les ressources défectueuses à d'autres nœuds ou qui surveillent d'autres aspects des performances des ressources.

Le RGM prend en charge aussi bien les ressources de basculement (qui ne peuvent être connectées qu'à un seul nœud à la fois) que les ressources évolutives (qui peuvent être connectées à plusieurs nœuds à la fois).

Modèle du gestionnaire de groupes de ressources

Cette rubrique définit certains termes fondamentaux et présente plus en détail le gestionnaire RGM et ses interfaces associées.

Le gestionnaire RGM gère trois grands types d'objets : types de ressources, ressources et groupes de ressources. Que sont ces objets ? Nous allons prendre un exemple pour le savoir.

Vous mettez en œuvre un type de ressources, `ha-oracle`, qui rend hautement disponible une application de base de données Oracle existante. Un utilisateur final crée une base de données de marketing, une base de données d'ingénierie et une base de données financière, chacune étant une ressource du type `ha-oracle`. L'administrateur du cluster place ces ressources dans des groupes de ressources différents, de sorte qu'elles soient exécutées sur des nœuds différents et basculent indépendamment les unes des autres. Vous créez un deuxième type de ressources, `ha-calendar`, pour mettre en œuvre un serveur de calendrier hautement disponible, qui nécessite une base de données Oracle. L'administrateur de cluster place la ressource dédiée au calendrier financier dans le groupe de ressources dans lequel figure déjà la base de données financière. Ainsi, ces deux ressources sont exécutées sur le même nœud et basculent en même temps.

Types de ressources

Un *type de ressources* comprend les éléments suivants :

- Une application à exécuter sur le cluster
- Des programmes de contrôle que le RGM utilise comme méthodes de rappel pour gérer l'application en tant que ressource de cluster
- Un ensemble de propriétés intégré à la configuration statique du cluster

Le RGM utilise des propriétés de type de ressources pour gérer les ressources d'un type spécifique.

Remarque – un type de ressources peut représenter des ressources système autres que des applications : par exemple, il peut représenter des adresses réseau.

Les propriétés du type de ressources et leurs valeurs sont définies dans un fichier d'enregistrement du type de ressources (fichier RTR). Ce fichier respecte le format décrit dans la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 35 et la page de manuel `rt_reg`(4). Pour voir une description d'un exemple de fichier RTR, consultez la rubrique "Définition du fichier d'enregistrement du type de ressource" à la page 84.

La rubrique "Propriétés des types de ressources" à la page 247 fournit une liste des propriétés des types de ressources.

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

Ressources

Chaque ressource hérite des propriétés et des valeurs de son type. En outre, il est possible de déclarer des propriétés de ressources dans le fichier RTR. La rubrique "Propriétés des ressources" à la page 255 fournit une liste des propriétés de ressources.

L'administrateur de clusters peut modifier les valeurs de certaines propriétés, selon la manière dont celles-ci sont définies dans le fichier RTR. Par exemple, les définitions de propriétés peuvent prévoir une plage de valeurs autorisées. Elles peuvent également indiquer à quels moments la propriété est réglable : jamais, à tout moment, à la création (lors de son ajout au cluster) ou lors de sa désactivation. Dans le cadre de ces définitions, l'administrateur du cluster peut modifier les propriétés en lançant des commandes d'administration.

Il peut créer un grand nombre de ressources du même type - chacune des ressources ayant son propre nom et son propre ensemble de valeurs de propriétés - afin d'exécuter plusieurs instances de l'application sous-jacente sur le même 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 RGM connecte et déconnecte simultanément toutes les ressources d'un groupe sur un même nœud. Il exécute alors les méthodes de rappel sur chacune des ressources du groupe.

Les nœuds auxquels un groupe de ressources est connecté sont appelés ses *nœuds principaux*. Un groupe de ressources est *géré* par chacun de ses nœuds principaux. Chaque groupe de ressources possède une propriété `Nodelist` qui identifie tous ses *nœuds principaux* ou *maîtres potentiels*. Cette propriété est définie par l'administrateur du cluster.

Un groupe de ressources possède également un ensemble 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 RGM définit deux types de groupes de ressources : les groupes de ressources de basculement et les groupes de ressources évolutifs. Un groupe de ressources de basculement ne peut être connecté qu'à un seul nœud à la fois tandis qu'un groupe de ressources évolutives peut être connecté simultanément à plusieurs nœuds. Le gestionnaire RGM fournit un ensemble de propriétés pour prendre en charge la création de chaque type de groupe de ressources. Pour plus d'informations sur ces propriétés, voir "Transfert d'un service de données sur un cluster" à la page 35 et "Mise en œuvre des méthodes de rappel" à la page 44.

La rubrique "Propriétés du groupe de ressources" à la page 271 fournit une liste des propriétés des groupes de ressources.

Gestionnaire de groupes de ressources

Le gestionnaire de groupes de ressources (RGM) est mis en œuvre sous la forme d'un démon, `rgmd`, exécuté sur chacun des nœuds d'un cluster. Tous les processus `rgmd` communiquent entre eux, ce qui leur permet de se comporter comme un utilitaire unique, à l'échelle du cluster.

Le gestionnaire RGM prend en charge les fonctions suivantes :

- Lorsqu'un nœud démarre ou s'arrête, le RGM tente de maintenir la disponibilité de tous les groupes de ressources gérés en les connectant automatiquement à des nœuds maîtres fonctionnels.
- 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 :
  - Changer le maître d'un groupe de ressources.
  - Activer ou désactiver une ressource dans un groupe.
  - Créer, modifier ou supprimer une ressource, un groupe ou un type de ressources.

Chaque fois que le RGM active des modifications de configuration, il coordonne ses actions sur tous les nœuds appartenant au cluster. On parle alors de *reconfiguration*. Pour modifier l'état d'une ressource individuelle, le RGM lui applique une méthode de rappel propre au type de ressources.

Méthodes de rappel

La structure Sun Cluster utilise un mécanisme de rappel pour assurer la communication entre un service de données et le RGM. Elle définit un ensemble de méthodes de rappel (avec leurs arguments et les valeurs renvoyées), ainsi que les circonstances dans lesquelles le RGM appelle chaque méthode.

Pour créer un service de données, on code un ensemble de méthodes de rappel et on met en œuvre chacune de ces méthodes comme un programme de contrôle que le RGM peut appeler : le service de données n'est donc pas un exécutable unique, mais regroupe un certain nombre de scripts (`ksh`) ou de binaires exécutables (C), dont chacun peut être appelé directement par le gestionnaire RGM.

Les méthodes sont enregistrées auprès du RGM via le fichier RTR. Celui-ci identifie le programme correspondant à chacune des méthodes mises en œuvre pour le service de données. Lorsqu'un administrateur de clusters enregistre le service de données sur un cluster, le RGM lit le fichier RTR, qui fournit l'identité des programmes de rappel, ainsi que d'autres informations.

Les seules méthodes de rappel indispensables (quel que soit le type de ressources) sont une méthode de démarrage (`Start` ou `Prenet_start`) et une méthode d'arrêt (`Stop` or `Postnet_stop`).

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

- Méthodes de contrôle et d'initialisation
  - Les méthodes `Start` et `Stop` démarrent et arrêtent les ressources d'un groupe lors de la connexion ou de la déconnexion de ce dernier.
  - Les méthodes `Init`, `Fini` et `Boot` exécutent du code d'initialisation ou d'arrêt sur une ressource.
- Méthodes d'assistance à l'administration
  - La méthode `Validate` contrôle les propriétés définies par les actions administratives.
  - La méthode `Update` met à jour les paramètres des propriétés d'une ressource en ligne.
- Méthodes relatives au réseau

  Les méthodes `Prenet_start` et `Postnet_stop` effectuent des actions de démarrage ou d'arrêt spécifiques avant que les adresses réseau d'un groupe de ressources soient configurées comme actives, ou après qu'elles ont été configurées

comme inactives.

- Méthodes de contrôle des détecteurs
  - Les méthodes `Monitor_start` et `Monitor_stop` lancent ou arrêtent le détecteur d'une ressource.
  - La méthode `Monitor_check` évalue la fiabilité d'un nœud avant qu'un groupe de ressources ne soit basculé sur ce nœud.

Pour plus d'informations sur les méthodes de rappel, consultez le Chapitre 3 et la page de manuel `rt_callbacks`(1HA). Pour connaître les méthodes de rappel de certains exemples de services de données, consultez le Chapitre 5 et le Chapitre 8.

Interfaces de programmation

Pour l'écriture du code des services de données, l'architecture de gestion des ressources fournit une API de bas niveau (ou de base), une bibliothèque de niveau plus élevé reposant sur l'API de base et l'outil SunPlex Agent Builder, qui peut créer automatiquement un service de données à partir de données de base saisies par l'utilisateur.

API de gestion des ressources

L'interface APIGR fournit un ensemble de fonctions de bas niveau qui permettent à un service de données d'accéder à des informations sur les ressources, les groupes et les types de ressources du système, de demander un redémarrage ou un basculement local et de définir le statut d'une ressource. Ces fonctions sont accessibles par le biais de la bibliothèque `libscha.so`. L'interface APIGR fournit ces méthodes de rappel sous la forme de commandes shell et de fonctions C. Pour plus d'informations sur les fonctions de l'interface APIGR, reportez-vous à la page de manuel `scha_calls`(3HA) et au Chapitre 3. Pour voir des exemples d'utilisation de ces fonctions dans des méthodes de rappel de services de données, consultez le Chapitre 5.

Bibliothèque de développement de services de données

Au niveau supérieur à celui de l'interface APIGR, la bibliothèque DSDL fournit une structure intégrée plus évoluée, tout en conservant le *modèle de rappel de méthodes* du RGM. Ses fonctions sont hébergées par la bibliothèque `libdsdev.so`. La bibliothèque DSDL réunit plusieurs utilitaires de développement de services de données, notamment :

- `libscha.so`, qui fournit les API de gestion de ressources de bas niveau.
- **PMF**, qui permet de contrôler les processus ainsi que leurs descendants et de les redémarrer lorsqu'ils s'arrêtent. Reportez-vous aux pages de manuel `pmfadm`(1M) et `rpc.pmfd`(1M).
- `hatimerun`, utilitaire qui permet d'exécuter les programmes respectant un délai d'attente. Reportez-vous à la page de manuel `hatimerun`(1M).

Avec la plupart des applications, la bibliothèque DSDL fournit toutes ou presque toutes les fonctionnalités dont vous avez besoin pour concevoir un service de données. Notez, toutefois, que la bibliothèque DSDL ne remplace pas l'interface API de bas niveau mais l'encapsule et l'étend. En fait, un grand nombre de fonctions de la bibliothèque DSDL appellent les fonctions du fichier `libscha.so`. C'est le cas, par exemple, lorsque vous utilisez la bibliothèque DSDL pour coder la majeure partie de votre service de données.

Pour plus d'informations sur la bibliothèque DSDL, reportez-vous au Chapitre 6 et à la page de manuel `scha_calls`(3HA).

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 qui inclut du code source et du code exécutable (shell Korn ou C), 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 cas, vous pouvez utiliser Agent Builder pour créer la majeure partie du code et coder manuellement le reste. Vous pouvez n'utiliser Agent Builder que pour générer le package Solaris.

Interface administrative du gestionnaire de groupes de ressources

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

SunPlex Manager

SunPlex Manager est un outil basé sur le Web qui vous permet d'effectuer les tâches suivantes :

- Installer un cluster
- Administrer un cluster
- Créer et configurer des ressources et des groupes de ressources
- Configurer des services de données à l'aide du logiciel Sun Cluster

Pour savoir comment installer SunPlex Manager et comment l'utiliser pour installer le logiciel de cluster, voir *Guide d'installation du logiciel Sun Cluster pour SE Solaris*. SunPlex Manager dispose d'une aide en ligne sur la plupart des tâches administratives uniques.

Utilitaire `scsetup`

L'utilitaire `scsetup`(1M) vous permet de mener à bien la plupart des tâches d'administration de Sun Cluster.

`scsetup` vous permet d'administrer les éléments suivants de Sun Cluster :

- quorum ;
- groupes de ressources ;
- services de données ;
- interconnexion de cluster ;
- groupes de périphériques et volumes ;
- noms d'hôtes privés ;
- nouveaux nœuds ;
- autres propriétés du cluster.

Il vous permet également d'effectuer les opérations suivantes :

- Créer un groupe de ressources
- Ajouter des ressources réseau à un groupe de ressources
- Ajouter une ressource de services de données à un groupe de ressources
- Enregistrer un type de ressources
- Connecter ou déconnecter un groupe de ressources
- Basculer un groupe de ressources
- Activer ou désactiver une ressource
- Modifier les propriétés d'un groupe de ressources
- Modifier les propriétés d'une ressource
- Supprimer une ressource d'un groupe de ressources
- Supprimer un groupe de ressources

- Supprimer l'indicateur d'erreur `Stop_failed` d'une ressource

Commandes administratives

Pour administrer les objets du gestionnaire RGM, Sun Cluster propose les commandes suivantes : `scrgadm`, `scswitch` et `scstat -g`.

La commande `scrgadm` vous permet d'afficher, de créer, de configurer ou de supprimer un type de ressources, un groupe de ressources, ainsi que les objets de ressource utilisés par le gestionnaire RGM. Elle fait partie de l'interface administrative du cluster, mais ne doit pas être utilisée dans le même contexte de programmation que celui de l'interface d'application décrite 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. Pour plus d'informations sur les tâches administratives que permet d'exécuter cette commande, reportez-vous à la page de manuel `scrgadm`(1M).

La commande `scswitch` permet de connecter et de déconnecter des groupes de ressources sur des nœuds donnés. Elle permet également d'activer ou de désactiver une ressource ou son détecteur. Pour plus d'informations sur les tâches administratives que la commande `scswitch` permet d'effectuer, reportez-vous à la page de manuel `scswitch`(1M).

La commande `scstat -g` indique l'état dynamique courant de tous les groupes de ressources et de toutes les ressources. Pour plus d'informations sur les tâches administratives qu'elle permet d'effectuer, reportez-vous à la page de manuel `scstat`(1M).

CHAPITRE 2

Développement d'un service de données

Ce chapitre explique comment rendre une application hautement disponible et évolutive et fournit des informations détaillées relatives au 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 pour pouvoir être rendue hautement disponible et évolutive. Si l[0092]application ne satisfait pas toutes les conditions requises, vous pourrez modifier son code source pour la rendre hautement disponible et évolutive.

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 Annexe B.

Remarque – Pour être hautement disponible, un service évolutif doit remplir toutes les conditions suivantes, ainsi que certains critères supplémentaires indiqués à la suite de la liste.

- Dans l'environnement Sun Cluster, vous pouvez rendre hautement disponibles ou évolutives des applications réseau (modèle client/serveur) ou 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 dans 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 nœud 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 connections.
- L'application ne doit pas dépendre du nom d'hôte physique du nœud sur lequel elle est exécutée. Reportez-vous à la rubrique "Noms d'hôtes" à la page 339 pour obtenir de plus amples informations.
- L'application doit fonctionner correctement dans les environnements sous lesquels plusieurs adresses IP sont configurées comme actives, comme par exemple les environnements comportant des hôtes multihome (caractérisés par le fait que le nœud figure sur plusieurs réseaux publics) et les environnements comportant des nœuds sur lesquels plusieurs interfaces logiques sont configurées comme actives 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 338.

  Si l'application utilise un nom de chemin d'accès codé en dur pour localiser les données, vous pouvez le remplacer 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 338 pour obtenir de plus amples informations.
- Les codes binaires applicatifs et les bibliothèques peuvent résider localement sur chaque nœud 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 lorsque 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 gèrent également le basculement ou la commutation du groupe de ressources. Reportez-vous à la rubrique "Relance d'un client" à la page 341 pour obtenir de plus amples informations.
- L'application ne doit comporter ni socket de domaine UNIX® ni tube 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 nœuds.
- L'application doit mettre en œuvre 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 charge`Lb_weighted`, 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 charge `Lb_sticky` et `Lb_sticky_wild` transmettent de manière répétée toutes les requêtes d'un client à la même instance de l'application, où elles pourront exploiter la mémoire cache d'entrée. Notez que si plusieurs requêtes client proviennent de différents clients, le RGM les distribue entre les instances du service. Reportez-vous à la rubrique "Mise en œuvre d'une ressource de basculement" à la page 52 pour obtenir de plus amples informations sur le paramétrage de la règle d'équilibrage de 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 (APIGR), un ensemble de routines de bas niveau (dans la bibliothèque`libscha.so`)
- La bibliothèque de développement de services de données (DSDL). Cet ensemble de fonctions d'un niveau plus élevé (dans la bibliothèque`libdsdev.so`) encapsule les fonctionnalités de l'interface APIGR 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 DSDL qui fournit uniquement une interface C.
2. Exécuter Agent Builder, indiquer les entrées demandées et générer un service de données comprenant un code source et un code exécutable, un fichier RTR et un package.
3. Si le service de données généré doit être personnalisé, vous pouvez ajouter un code DSDL 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 APIGR 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 DSDL ou APIGR. Quelle que soit votre façon de procéder, il est généralement conseillé de commencer par utiliser Agent Builder pour les raisons suivantes :

- Le code généré par Agent Builder, bien que générique par 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 de 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 le travail nécessaire.
- Vous pouvez modifier le code généré.

Remarque – contrairement à l'interface APIGR fournissant un ensemble de fonctions C et un ensemble de commandes à utiliser dans les scripts, la bibliothèque DSDL 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 APIGR car il n'y a pas de commande DSDL `ksh`.

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

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

Remarque – Vous ne pouvez pas mélanger du code C++ compilé en mode compatibilité avec du code C++ compilé en mode standard dans le système d'exploitation Solaris et dans les produits Sun Cluster. Par conséquent, si vous souhaitez créer un service de données basé sur C++ à utiliser sur Sun Cluster, vous devez compiler ce service de données comme suit :

- Pour Sun Cluster 3.0 et versions précédentes, utilisez le mode compatible.
- À partir de Sun Cluster 3.1, utilisez le mode standard.

À la fin du développement (sur un nœud non-cluster), vous pouvez transférer le service de données terminé sur un cluster pour le tester.

Remarque – Veillez à utiliser une version de développement ou le groupe de logiciels complet de distribution du système d'exploitation Solaris 8 ou une version ultérieure.

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

- installation du package de développement 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.

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Changez de répertoire en allant sur le CD-ROM dans le répertoire voulu.**

   # **cd** *cd-rom-directory*
3. **Installez le package `SUNWscdev` dans le répertoire courant.**

   - **Pour le système d'exploitation Solaris 10 dans un environnement de zones, en tant qu'administrateur général de la *zone globale*, entrez la commande suivante :**

     # **pkgadd -G -d . SUNWscdev**

     Le package `SUNWscdev` s'ajoute à la zone globale à condition que le contenu de `SUNWscdev` n'ait aucune incidence sur les parties de la zone globale partagées avec une zone non globale.
   - **Pour toute autre version du système d'exploitation Solaris ou pour le système d'exploitation Solaris 10 sans environnement de zones, entrez la commande suivante :**

     # **pkgadd -d . SUNWscdev**
4. **Dans le fichier `makefile`, indiquez les options du compilateur et de l'éditeur de liens identifiant les fichiers inclus et de bibliothèque du code de votre 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 dans 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

Une fois le service de données terminé sur une machine de développement, vous devez transférer le service de données sur un cluster afin de 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 nœuds 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 nœuds du cluster. Remarque : Agent Builder crée automatiquement ce package.

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 ressources, la version, la version de l'API, ainsi que les chemins d'accès aux méthodes de rappel. "Propriétés des types de ressources" à la page 247 répertorie toutes les propriétés de types de ressources.

Les propriétés de ressources, telles que `Failover_mode`, `Thorough_probe_interval` et les délais d'attente de la méthode, définissent également la configuration statique de la ressource. Les propriétés de ressources dynamiques, telles que `Resource_state` et `Status`, reflètent l'état actif d'une ressource gérée. "Propriétés des ressources" à la page 255 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 un service de données. Par exemple, certaines propriétés, telles que `Resource_type`, 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 à 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 – Seul un administrateur du cluster peut configurer la propriété de type de ressources `Installed_nodes`. Vous ne pouvez pas déclarer `Installed_nodes` dans le fichier RTR.

Syntaxe des déclarations de type de ressources :

property-name = *value*;

Remarque – Les noms de propriété des groupes de ressources, des ressources et des types de ressources *ne* sont *pas* sensibles à la casse. Vous pouvez associer les majuscules et les minuscules dans le nom des propriétés.

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

```
# Sun Cluster Data Services Builder template version 1.0
# Enregistrement de données et de ressources for smpl
#
#Remarque : les mots clés sont insensibles à la casse. Cela signifie
#que vous pouvez utiliser les minuscules ou les majuscules au choix.
#
Resource_type = "smpl";
Vendor_id = SUNW;
RT_description = "Exemple de service sur Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;

Init_nodes = RG_PRIMARIES;
```

```
RT_basedir=/opt/SUNWsmpl/bin;

Start           =    smpl_svc_start;
Stop            =    smpl_svc_stop;

Validate        =    smpl_validate;
Update          =    smpl_update;

Monitor_start   =    smpl_monitor_start;
Monitor_stop    =    smpl_monitor_stop;
Monitor_check   =    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 .

`Resource_type` et `Vendor_id`
Indique le nom du type de ressources. Vous pouvez spécifier le nom du type de ressources de deux façons : au moyen de la propriété `Resource_type` (`smpl`) uniquement ou en utilisant le préfixe `Vendor_id` suivi d'un point "." comme séparateur et du nom du type de ressources (`SUNW.smpl`), comme dans notre exemple. Si vous utilisez `Vendor_id`, utilisez-le comme symbole boursier de la société qui définit le type de ressources. Le nom du type de ressources doit être unique sur le cluster.

Remarque – Les noms de type de ressources (*vendoridApplicationname*) sont traditionnellement utilisés comme nom de package. À commencer par le nom du système d'exploitation Solaris 9, l'association de l'ID fournisseur et du nom de l'application peut dépasser neuf caractères. Toutefois, si vous utilisez une version antérieure du système d'exploitation Solaris, l'association de l'ID fournisseur et du nom de l'application ne doit pas dépasser neuf caractères, 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.

`RT_description`
Décrit en quelques mots le type de ressources.

`RT_version`
Identifie la version du service de données échantillon.

`Version_API`
Identifie la version de l'interface API. Par exemple, `API_version = 2` signifie que le service de données fonctionne sous Sun Cluster, version 3.0. `API_version = 5`

indique que le service de données fonctionne sous Sun Cluster, à commencer par la version Sun Cluster 3.1 9/04. Toutefois, `API_version = 5` indique également que le service de données ne peut pas fonctionner sur n'importe quelle version de Sun Cluster diffusée avant Sun Cluster 3.1 9/04. Cette propriété est décrite en détail sous l'entrée `API_version` dans "Propriétés des types de ressources" à la page 247.

`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 nœuds car il s'agit d'un service de données de basculement. Cette propriété est décrite en détail sous l'entrée `Failover` dans "Propriétés des types de ressources" à la page 247.

`Start`, `Stop` et `Validate`
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 `RT_basedir`.

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

`Init_nodes = RG_PRIMARIES`
Indique que le gestionnaire RGM n'appelle les méthodes `Init`, `Boot`, `Fini` et `Validate` que sur les nœuds pouvant gérer le service de données. Les nœuds spécifiés par `RG_PRIMARIES` constituent un sous-ensemble de tous les nœuds sur lesquels le service de données est installé. Définissez la valeur sur `RT_INSTALLED_NODES` pour indiquer que le gestionnaire RGM appelle ces méthodes sur tous les nœuds sur lesquels le service de données est installé.

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

`Start`, `Stop`, et `Validate`
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 `RT_basedir`.

Déclaration des propriétés de ressource

Comme pour les propriétés de type de ressources, vous déclarez des propriétés de ressource 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 :

```
{
    attribute = value;
    attribute = value;
```

```
                .
                .
                .
    attribute = value;
}
```

Vous pouvez modifier, dans le fichier RTR, les attributs spécifiques aux propriétés de ressource 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.

VVous 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. "Attributs des propriétés de ressources" à la page 281 répertorie les attributs permettant de modifier et de définir des propriétés de ressource. 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 ressources définies par le système indique le délai d'attente des méthodes de rappel.

```
...

# Resource property declarations appear as a list of bracketed
# entries after the resource type declarations. The property
# name declaration must be the first attribute after the open
# curly bracket of a resource property entry.
#
# Set minimum and default for method timeouts.
{
        PROPERTY = Start_timeout;
        MIN=60;
        DEFAULT=300;
}

{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Update_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Start_timeout;
        MIN=60;
```

```
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Stop_timeout;
        MIN=60;
        DEFAULT=300;
{
        PROPERTY = Monitor_Check_timeout;
        MIN=60;
        DEFAULT=300;
}
```

Le premier attribut de chaque déclaration de propriétés de ressource doit être le nom de la propriété (`PROPERTY` = *value*). Vous pouvez configurer des propriétés de ressource 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 à "Attributs des propriétés de ressources" à la page 281 pour consulter la liste complète des attributs de propriétés de ressource.

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

```
{
        PROPERTY = Failover_mode;
        DEFAULT=SOFT;
        TUNABLE = ANYTIME;
}
{
        PROPERTY = Thorough_Probe_Interval;
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Set Retry_interval as a multiple of 60 since it is converted from seconds
# to minutes, rounding up. For example, a value of 50 (seconds)
# is converted to 1 minute. Use this property to time the number of
# retries (Retry_count).
{
        PROPERTY = Retry_interval;
        MAX=3600;
        DEFAULT=300;
```

```
        TUNABLE = ANYTIME;
}

{
        PROPERTY = Network_resources_used;
        TUNABLE = WHEN_DISABLED;
        DEFAULT = "";
}
{
        PROPERTY = Scalable;
        DEFAULT = FALSE;
        TUNABLE = AT_CREATION;
}
{
        PROPERTY = Load_balancing_policy;
        DEFAULT = LB_WEIGHTED;
        TUNABLE = AT_CREATION;
}
{
        PROPERTY = Load_balancing_weights;
        DEFAULT = "";
        TUNABLE = ANYTIME;
}
{
        PROPERTY = Port_list;
        TUNABLE = ANYTIME;
        DEFAULT = ;
}
```

Ces déclarations de propriétés de ressource créent l'attribut `TUNABLE` qui restreint les occasions permettant à l'administrateur système de modifier leurs valeurs. Par exemple, la valeur `AT_CREATION` 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 reporter à la rubrique "Propriétés des ressources" à la page 255 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 nœud en cas d'échec d'une méthode `Start` ou `Stop`.

`Thorough_probe_interval`, `Retry_count` , et `Retry_interval`
: Propriétés utilisées dans le détecteur de pannes. `Tunable` et `ANYTIME` sont identiques de sorte que l'administrateur du cluster peut les ajuster si le détecteur de pannes ne fonctionne pas de façon optimale.

`Network_resources_used`
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 `FALSE` pour indiquer que la ressource correspondante n'utilise pas la fonction de gestion de réseaux du cluster (adresse partagée). Si vous réglez cette propriété sur `FALSE`, la propriété de type de ressources `Failover` doit être réglée sur `TRUE` 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 35 et "Mise en œuvre des méthodes de rappel" à la page 44 pour obtenir de plus amples informations sur l'utilisation de cette propriété.

`Load_balancing_policy` et `Load_balancing_weights`
Déclare automatiquement ces propriétés, sinon elles ne sont pas utilisées dans un type de ressources de basculement.

`Port_list`
Identifie la liste des ports de réception du serveur. Agent Builder déclare cette propriété, afin que l'administrateur du cluster puisse identifier une liste de ports lorsqu'il configure le 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.

```
# Extension Properties
#

# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, smpl, specify the path of the configuration file on
# PXFS (typically named.conf).
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "The Configuration Directory Path(s)";
}

# The following two properties control restart of the fault monitor.
{
        PROPERTY = Monitor_retry_count;
        EXTENSION;
        INT;
        DEFAULT = 4;
        TUNABLE = ANYTIME;
```

```
        DESCRIPTION = "Number of PMF restarts allowed for fault monitor.";
}
{
        PROPERTY = Monitor_retry_interval;
        EXTENSION;
        INT;
        DEFAULT = 2;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time window (minutes) for fault monitor restarts.";
}
# Time out value in seconds for the probe.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 120;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time out value for the probe (seconds)";
}

# Child process monitoring level for PMF (-C option of pmfadm).
# Default of -1 means to not use the -C option of pmfadm.
# A value of 0 or greater indicates the desired level of child-process.
# monitoring.
{
        PROPERTY = Child_mon_level;
        EXTENSION;
        INT;
        DEFAULT = -1;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Child monitoring level for PMF";
}
# User added code -- BEGIN VVVVVVVVVVV
# User added code -- END   ^^^^^^^^^^^^
```

Agent Builder crée les propriétés d'extension suivantes, utiles pour la plupart des services de données.

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

`Monitor_retry_count`, `Monitor_retry_interval` , et `Probe_timeout`
: Contrôlent les redémarrages du détecteur de pannes et non du démon serveur.

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

Vous pouvez créer d'autres propriétés d'extension dans la zone délimitée par les commentaires `User added code`.

Mise en œuvre des méthodes de rappel

Cette rubrique fournit des informations d'ordre général sur la mise en œuvre des méthodes de rappel.

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

En règle générale, les méthodes de rappel doivent accéder aux propriétés de la ressource. L'interface APIGR 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 man `scha_resource_get`(1HA) et `scha_resource_get`(3HA).

La bibliothèque DSDL fournit un ensemble de fonctions C (une par propriété) permettant d'accéder aux propriétés définies par le système et une fonction d'accès aux propriétés d'extension. Reportez-vous aux pages man `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 ressource (à l'exception de `Status` et `Status_msg`). 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 ressource à l'aide de la commande `scrgadm` ou par l'intermédiaire d'une interface administrative graphique. Cependant, 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 `Start` échoue, le gestionnaire RGM peut appeler une méthode `Stop` sur une ressource même si cette dernière n'a jamais été démarrée. De même, le gestionnaire RGM peut exécuter la méthode `Stop` sur le démon d'une ressource, même si celui-ci s'était déjà arrêté de lui-même. Les mêmes scénarios s'appliquent aux méthodes `Monitor_start` et `Monitor_stop`.

C'est pourquoi vous devez intégrer le principe d'idempotence dans vos méthodes `Stop` et `Monitor_stop`. Les appels répétés de `Stop` ou `Monitor_stop` sur la même ressource avec les mêmes arguments fournissent les mêmes résultats qu'un appel unique.

L'idempotence se caractérise notamment par le fait que l'`Arrêt` et l'`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`, `Boot` et `Update` 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 ne nécessite pas le codage d'un service de données, 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 obtenir 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 nœuds sont liés au processus d'entrée/sortie du cluster.

Démarrage et arrêt d'une ressource

La mise en œuvre d'un type de ressources requiert au minimum une méthode `Start` et une méthode `Stop`. Le gestionnaire RGM appelle les programmes de méthode du type de ressources aux moments opportuns et sur les nœuds appropriés pour connecter ou déconnecter les groupes de ressources. Par exemple, après la défaillance

d'un nœud de cluster, le gestionnaire RGM transfère les groupes de ressources gérés par ce nœud sur un autre nœud. Vous devez mettre en œuvre une méthode `Start` pour permettre au RGM de redémarrer chaque ressource sur le nœud hôte restant.

Une méthode `Start` ne doit pas être retournée tant que la ressource n'a pas été exécutée et n'est pas disponible sur le nœud local. Veillez à ce que les types de ressources nécessitant une longue période d'initialisation disposent d'un délai d'attente suffisant dans leur méthode `Start` (configurez la propriété `Start_timeout` dans le fichier RTR).

Vous devez mettre en œuvre une méthode `Arrêt` lorsque le gestionnaire RGM déconnecte un groupe de ressources. Par exemple, supposons qu'un groupe de ressources est déconnecté du Nœud1 puis reconnecté au Nœud2. Tout en déconnectant le groupe de ressources, le gestionnaire RGM appelle la méthode `Stop` sur les ressources du groupe dont vous souhaitez arrêter toute activité sur le Nœud1. Une fois que les méthodes `Stop` ont été exécutées sur toutes les ressources du Nœud1, le gestionnaire RGM connecte le groupe de ressources au Nœud2.

Une méthode `Stop` ne doit pas être retournée tant que la ressource n'a pas cessé totalement toute activité au niveau du nœud local et qu'elle n'est pas complètement arrêtée. La mise en œuvre la plus sûre d'une méthode `Stop` met fin, sur le nœud local, à tous les processus liés 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 `Stop`. Définissez la propriété `Stop_timeout` dans le fichier RTR.

Lorsqu'une méthode `Stop` échoue ou dépasse le délai imparti, le groupe de ressources bascule dans un état d'erreur requérant l'intervention de l'administrateur du cluster. Pour éviter cela, les mises en œuvre des méthodes d'`Arrêt` et d'`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 nœud local.

Choix des méthodes `Start` et `Stop` à utiliser

Cette rubrique présente des astuces permettant de savoir quand il est préférable d'utiliser les méthodes `Start` et `Stop` par opposition aux méthodes `Prenet_start` et `Postnet_stop`. Pour déterminer les méthodes à utiliser, vous devez avoir une connaissance approfondie du client et du protocole réseau 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 de `Prenet_start` et `Postnet_stop` permettent à la mise en œuvre d'un type de ressources d'effectuer des actions de démarrage et d'arrêt spéciales avant et après la configuration comme actives ou inactives 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 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 `Postnet_stop` du service de données (le cas échéant).
4. Déplombage des adresses réseau.

Pour choisir les méthodes `Start`, `Stop`, `Prenet_start` ou `Postnet_stop` 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 afin de configurer l'activation des adresses réseau avant d'appeler les méthodes `Start` des ressources du service de données. Par conséquent, si un service de données requiert des adresses configurées en amont à son démarrage, utilisez la méthode `Start` 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 afin de configurer la désactivation des adresses réseau avant d'appeler les méthodes de ressources d'`Arrêt` du service de données. Par conséquent, si un service de données requiert que des adresses réseau soient actives à son arrêt, utilisez la méthode `Stop` 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 `Start` et `Stop`.

Par contre, vous devez démarrer ou arrêter le service de données à l'aide des méthodes `Prenet_start` et `Postnet_stop` s'il requiert que les adresses réseau passent à l'état désactivé à son démarrage ou à son arrêt. 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`, soit une commutation avec `scswitch`). Par exemple, la mise en œuvre de clients peut nécessiter un minimum de tentatives, l'abandon survenant rapidement une fois l'indisponibilité du port du service de données détectée.

Si le service de données ne requiert pas qu'une adresse réseau soit activée, démarrez-le avant d'activer l'interface réseau. Vous avez ainsi l'assurance que le service de données peut répondre immédiatement aux requêtes des clients dès que l'adresse réseau a été activée. 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 de `Prenet_start` plutôt qu'avec la méthode `Start`.

Si vous utilisez la méthode `Postnet_stop`, la ressource de service de données se trouve encore en amont lors de la configuration en aval de l'adresse réseau. La méthode `Postnet_stop` n'est appelée qu'après la configuration de la désactivation de l'adresse réseau. Par conséquent, 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.

Remarque – Si vous installez un service RPC dans le cluster, le service ne doit pas utiliser les numéros de programme suivants : 100141, 100142 et 100248. Ces numéros sont réservés aux démons Sun Cluster `rgmd_receptionist`, `fed` et `pmfd`. Si le service RPC que vous installez utilise l'un de ces numéros, modifiez le numéro de programme de ce service RPC.

La décision d'utiliser les méthodes `Start` et `Stop` plutôt que les méthodes `Prenet_start` et `Postnet_stop` ou de combiner les deux, doit tenir compte des exigences et du comportement du client et du serveur.

Méthodes `Init`, `Fini` et `Boot`

Les trois méthodes facultatives `Init`, `Fini` et `Boot` 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,
- soit parce qu'elle est créée dans un groupe de ressources déjà géré.

Le gestionnaire RGM appelle la méthode `Fini` à des fins de nettoyage lorsque la ressource devient non gérée

- soit parce que le groupe de ressources auquel elle appartient bascule d'un état géré à un état non géré,

- soit parce qu'elle est supprimée d'un groupe de ressources géré.

Le nettoyage doit être idempotent. Autrement dit, si le nettoyage a déjà été effectué, le résultat de `Fini` est 0 (succès).

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

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

Contrôle d'une ressource

En règle générale, vous mettez en œuvre 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 APIGR `scha_control()` ou DSDL `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 détecteur de 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 offre un mécanisme permettant de démarrer automatiquement des détecteurs pour les ressources. Lors de la déconnexion d'une ressource, le gestionnaire RGM appelle la méthode d'`Arrêt_détecteur` pour arrêter le détecteur de la ressource sur les nœuds locaux avant d'arrêter la ressource elle-même. Lors de la connexion d'une ressource en ligne, le gestionnaire RGM appelle la méthode `Monitor_start` après le démarrage de la ressource.

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

Le détecteur de ressource peut définir les propriétés `Status` et `Status_msg` pour refléter l'affichage de l'état de la ressource sur le détecteur. Définissez ces propriétés à l'aide de la fonction APIGR `scha_resource_setstatus()`, de la commande `scha_resource_setstatus` ou de la fonction DSDL `scds_fm_action()`.

Remarque – Bien que l'utilisation de `Status` et `Status_msg` 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 101 présente un exemple de détecteur de pannes mis en œuvre avec l'interface APIGR. la rubrique "Détecteur de pannes `SUNW.xfnts`" à la page 149 présente un exemple de détecteur de pannes mis en œuvre avec la bibliothèque DSDL. Reportez-vous au *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour obtenir de plus amples informations sur les détecteurs de pannes intégrés aux services 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 APIGR et la DSDL offrent des fonctions de gestion des processus pour mettre en œuvre des détecteurs et des rappels de contrôle des ressources. L'interface APIGR définit les fonctions suivantes :

Process Monitor Facility (PMF) : `pmfadm` et `rpc.pmfd`
: 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`.

La DSDL fournit un ensemble de fonctions (précédées par le nom `scds_pmf_`) de mise en œuvre de la fonctionnalité PMF. Reportez-vous à la rubrique "Fonctions PMF" à la page 213. Elle présente brièvement la fonctionnalité PMF et contient une liste des fonctions individuelles.

Les pages man `pmfadm`(1M) et `rpc.pmfd`(1M) fournissent une description détaillée de cette commande et de ce démon.

`halockrun`
Programme d'exécution d'un programme enfant sans déverrouiller le fichier. Cette commande est pratique à utiliser dans les scripts shell.

La page man `halockrun`(1M) fournit une description détaillée de cette commande.

`hatimerun`
Programme d'exécution d'un programme enfant suivant un délai d'attente. Cette commande est pratique à utiliser dans les scripts shell.

La DSDL intègre la fonction `scds_hatimerun()` permettant de mettre en œuvre la fonctionnalité `hatimerun`.

La page man `hatimerun`(1M) fournit une description détaillée de cette commande.

Support administratif d'une ressource

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

Le gestionnaire RGM appelle la méthode `Validate` lorsqu'une ressource est créée ou 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 `Validate` pour l'ensemble des nœuds du cluster spécifiés par la propriété `Init_nodes` du type de ressources (reportez-vous à la rubrique "Propriétés des types de ressources" à la page 247 ou à la page man `rt_properties`(5) pour obtenir de plus amples informations sur `Init_nodes`. Le gestionnaire RGM appelle `Validate` avant que la création ou la mise à jour ne soit appliquée. Par conséquent, un code de sortie d'échec de la méthode sur n'importe quel nœud entraîne l'échec de la création ou de la mise à jour.

Le gestionnaire RGM n'appelle `Validate` qu'en cas de modification des propriétés d'une ressource ou de son groupe au moyen d'une action administrative. Cette méthode n'est donc pas appelée lorsqu'il définit des propriétés ou lorsqu'un détecteur configure les propriétés de ressource `Status` et `Status_msg`.

Le RGM appelle la méthode `Update` facultative pour notifier à une ressource en cours d'exécution que des propriétés ont été modifiées. Il appelle `Update` 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 nœuds 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 œuvre d'une ressource de basculement

Un groupe de ressources de basculement contient des adresses réseau (par exemple, `LogicalHostname` et `SharedAddress` 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 nœud 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 œuvre d'une ressource de basculement.

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

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

La propriété de groupe de ressources `RG_mode` permet à l'administrateur du cluster d'identifier un groupe de ressources comme étant évolutif ou de basculement. Si `RG_mode` est défini sur `FAILOVER`, tle gestionnaire RGM configure la propriété `Maximum_primaries` du groupe sur 1 et limite la gestion du groupe de ressources à un seul nœud. Le gestionnaire RGM ne permet pas de créer une ressource dont la propriété `Failover` est définie sur `TRUE` dans un groupe de ressources dont la propriété `RG_mode` est définie sur `SCALABLE`.

La propriété de groupe de ressources `Implicit_network_dependencies` 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 (`LogicalHostname` et `SharedAddress`) dans le groupe. Cela signifie que les méthodes `Start` 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 pour passer à l'état actif. La propriété `Implicit_network_dependencies` est définie par défaut sur `TRUE`.

Mise en œuvre d'une ressource évolutive

Une ressource évolutive peut être en ligne sur plusieurs nœuds 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 œuvre d'une ressource évolutive.

Définissez la propriété booléenne du type de ressources `Failover` sur `FALSE` pour autoriser la configuration d'une ressource dans un groupe pouvant être en ligne sur plusieurs nœuds simultanément.

La propriété de ressource `Scalable` détermine si la ressource utilise la fonction d'adresse partagée du cluster. Définissez cette propriété sur `TRUE` car une ressource évolutive utilise une ressource d'adresse partagée (de sorte 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 `RG_mode` est définie sur `SCALABLE`, le gestionnaire RGM permet de configurer `Maximum_primaries` sur une valeur supérieure à 1, ce qui signifie que le groupe peut être géré par plusieurs nœuds simultanément. Le gestionnaire RGM permet d'instancier une ressource dont la propriété `Failover` est définie sur `FALSE` dans un groupe de ressources dont la propriété `RG_mode` est définie sur `SCALABLE`.

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 nœud 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 `RG_dependencies` (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 `Scalable` d'une ressource dans le fichier RTR, le gestionnaire RGM crée automatiquement l'ensemble des propriétés évolutives suivantes pour cette ressource.

`Network_resources_used`
: 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.

`Load_balancing_policy`
: Spécifie la règle d'équilibrage de charge de la ressource. Vous pouvez explicitement définir cette règle dans le fichier RTR (ou activer la valeur par défaut `LB_WEIGHTED`). 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 `Tunable` de la propriété `Load_balancing_policy` sur `NONE` ou `FALSE` dans le fichier RTR). Valeurs admises :

  `Équilibrage_charge_pondéré`
  : La charge est répartie entre plusieurs nœuds en fonction des poids définis dans la propriété `>Load_balancing_weights`.

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

  `Équilibrage_charge_sticky_étendu`
  : Un client donné (identifié par l'adresse IP du client), qui se connecte à une adresse IP d'un service "sticky", est toujours envoyé au même nœud du cluster, quel que soit le numéro de port sur lequel il arrive.

  Dans le cadre d'un service évolutif avec une propriété `Load_balancing_policy` définie sur `LB_STICKY` ou `LB_STICKY_WILD`, la modification de la propriété `Load_balancing_weights` alors que le service est en ligne peut réinitialiser les affinités des clients existants. Le cas échéant, un autre nœud peut prendre en charge une requête ultérieure du client, même si ce dernier était précédemment géré par un autre nœud 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 nœud. Format : *poids*`@`*nœud*`,` *poids*`@`*nœud*`,` *poids* correspondant à un nombre entier reflétant la part relative de la charge distribuée au *nœud* spécifié. Cette part correspond au poids du nœud divisé par la somme de tous les poids des instances actives. Par exemple, `1@1,3@2` indique que le nœud 1 reçoit 1/4 de la charge et que le nœud 2 en reçoit les 3/4.

`Port_list`
: 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, définissez a propriété de type de ressources `Failover` et la propriété de ressource `Scalable` sur `FALSE` dans le fichier RTR du service de données. Spécifiez à la création que la propriété `Évolutivité` est réglable.

La valeur de la propriété `Failover` `FALSE` 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 `Scalable` sur `TRUE` lors de la création de la ressource, créant ainsi un service évolutif.

D'un autre côté, même si la propriété `Failover` est définie sur `FALSE`, l'administrateur peut configurer la ressource dans un groupe de ressources de basculement afin de mettre en œuvre un service de basculement. L'administrateur du cluster ne modifie pas la valeur de la propriété `Scalable` (déjà définie sur `FALSE`). Pour prendre en charge cette éventualité, vous devez fournir un contrôle de la propriété `Scalable` dans la méthode `Validate`. Si la propriété `Scalable` est définie sur `FALSE`, 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 *Guide des notions fondamentales de Sun Cluster pour SE Solaris*.

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 ressource. 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 `Network_resources_used` ne doit pas être vide. Elle doit contenir le nom des ressources d'adresse partagée existantes. Chaque nœud de la `Nodelist` du groupe de ressources contenant la ressource évolutive doit apparaître dans la propriété `NetIfList` ou `AuxNodeList` de chaque ressource d'adresse partagée nommée.
- La propriété `RG_dependencies` du groupe de ressources contenant la ressource évolutive doit inclure les groupes de ressources de toutes les ressources d'adresse partagée répertoriées dans la propriété `Network_resources_used` de cette ressource évolutive.
- La propriété `Port_list` ne doit pas être vide. Elle doit contenir une liste des paires port/protocole. Vous devez ajouter une barre oblique (`/`) à chaque numéro de port, suivi du protocole utilisé par le port en question. Exemple :

```
Port_list=80/tcp6,40/udp6
```

Vous pouvez indiquer les valeurs de protocole suivantes :

- `tcp`, pour TCP IPv4 ;
- `tcp6`, pour TCP IPv6 ;
- `udp`, pour UDP IPv4 ;
- `udp6`, pour UDP IPv6.

Écriture et test des services de données

Utilisation du mécanisme Keep-Alive TCP pour protéger le serveur

Côté serveur, l'utilisation du mécanisme Keep-Alive TCP protège le serveur contre les gaspillages de ressources système d'un client hors service (ou partitionné en réseau). Si ces ressources ne sont pas nettoyées (sur un serveur en fonction suffisamment longtemps), elles finissent par s'étendre indéfiniment à mesure des arrêts inopinés, puis des réinitialisations des clients.

Si les communications client-serveur passent par un flux TCP, le serveur et le client doivent activer le mécanisme Keep-Alive 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 du mécanisme Keep-Alive TCP permet au client d'être notifié 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.

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 à 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-Alive TCP, l'application cliente doit également exécuter son propre Keep-Alive périodique à son niveau. De fait, le mécanisme keep-alive TCP n'est pas parfait dans tous les cas de limite possibles. L'utilisation d'un mécanisme Keep-Alive 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 œuvre d’un service de données dans un environnement à haut niveau de disponibilité. Les cas présentés sont suggestifs et non exhaustifs. Vous devez pouvoir accéder à une configuration de test 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 haute disponibilité se comporte de façon adéquate dans tous les cas où un groupe de ressources est déplacé d’un hôte physique à 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 que dans le cadre du traitement d’une requête pour un client, un service de données client-serveur transmette des requêtes à un autre service de données client-serveur. Par exemple, un service de données A dépend d’un service de données B si, 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 man `scrgadm`(1M) pour obtenir de plus amples informations.

Si les ressources de votre type de ressources dépendent de ressources d’un autre type, vous devez indiquer à l’administrateur du cluster de configurer les ressources et 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 nœud que la ressource dont elle “dépend”, 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 tester la disponibilité du ou des autres services de données dans le code de votre service de données à haute disponibilité. Si la ressource dépendante/dont d’autres dépendent est exécutable sur différents nœuds, 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 des 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 de calendrier client-serveur hypothétique conservant toutes ses données dans une base de données SQL de type 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 de calendrier. L'administrateur du cluster enregistre le type de ressource de calendrier avec Sun Cluster.

Si l'application de calendrier doit fonctionner sur le même nœud que la base de données Oracle, l'administrateur du cluster configure la ressource de calendrier dans le même groupe de ressources que la ressource HA-ORACLE et rend la ressource de calendrier dépendante de la ressource HA-ORACLE. Cette dépendance est spécifiée à l'aide de la balise de propriété `Resource_dependencies` dans `scrgadm`.

Si la ressource HA-ORACLE peut fonctionner sur un autre nœud que la ressource de calendrier, l'administrateur du cluster les configure dans deux groupes de ressources distincts. L'administrateur du cluster peut définir une dépendance du groupe de ressources de calendrier vis-à-vis du 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 nœud. Par conséquent, après avoir été démarré, le démon du service de données de calendrier doit attendre que la base de données Oracle devienne disponible. Dans ce cas, la méthode `Start` du type de ressources de calendrier renvoie généralement un code de succès. Cependant, si elle est indéfiniment bloquée, son groupe de ressources bascule à l'état occupé et toute autre modification d'état devient impossible (édition, basculement ou commutation sur le groupe notamment). En outre, si la méthode `Start` de la ressource de calendrier dépasse le délai d'attente ou se termine avec un état différent de zéro, le groupe de ressources risque de basculer continuellement d'un nœud à un autre tant que la base de données Oracle reste indisponible.

CHAPITRE 3

Référence concernant l'API de gestion des ressources

Ce chapitre répertorie et présente brièvement les fonctions d'accès et les méthodes de rappel qui constituent l'interface APIGR (API de gestion des ressources). Toutefois, la référence faisant foi pour ces fonctions et méthodes se trouve dans les pages de manuel consacrées à l'interface APIGR.

Ce chapitre contient les rubriques suivantes :

- "Méthodes d'accès de l'interface APIGR" à la page 59 : commandes de script shell et fonctions C
- "Méthodes de rappel de l'interface APIGR" à la page 65 décrites dans la page de manuel `rt_callbacks`(1HA)

Méthodes d'accès de l'interface APIGR

L'API fournit des fonctions d'accès aux propriétés des ressources, des groupes et des types de ressources, ainsi qu'à d'autres informations sur le cluster, sous la forme de commandes shell et de fonctions C, qui vous permettent de mettre en œuvre des programmes de contrôle en tant que scripts shell ou programmes C.

Commandes shell de l'interface APIGR

Les commandes shell s'utilisent dans les mises en œuvre par scripts shell des méthodes de rappel des types de ressource qui représentent des services contrôlés par le gestionnaire RGM du cluster. Elles permettent d'effectuer les tâches suivantes :

- Accéder à des informations sur les ressources, les groupes et les types de ressources, ainsi que sur les clusters
- Définir les propriétés `Status` et `Status_msg` d'une ressource, à l'aide d'un détecteur

- Demander le redémarrage ou le déplacement d'un groupe de ressources

---

Remarque – cette rubrique ne fournit que de brèves descriptions des commandes shell ; la référence faisant foi en matière de commandes shell se trouve dans les pages de manuel 1HA. Sauf indication contraire, chaque commande est associée à une page de manuel du même nom.

Commandes APIGR relatives aux ressources

Les commandes ci-dessous vous permettent d'accéder aux informations sur une ressource ou d'en définir les propriétés `Status` et `Status_msg`.

`scha_resource_get`
Donne accès aux informations sur une ressource ou un type de ressource contrôlé par le RGM. Cette commande renvoie les mêmes données que la fonction C `scha_resource_get()`. Pour obtenir plus d'informations, reportez-vous à la page de manuel `scha_resource_get`(1HA).

`scha_resource_setstatus`
Définit les propriétés `Status` et `Status_msg` d'une ressource contrôlée par le RGM. C'est la commande que le détecteur de la ressource utilise pour indiquer l'état de cette dernière. Elle remplit les mêmes fonctions que `scha_resource_setstatus()`. Pour plus d'informations sur cette commande, reportez-vous à la page de manuel `scha_resource_setstatus`(1HA).

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

Commande relative aux types de ressources

`scha_resourcetype_get`
Donne accès à des informations sur un type de ressource enregistré dans le RGM. Cette commande fournit la même fonctionnalité que la fonction C `scha_resourcetype_get ()`. Elle est décrite plus en détail dans la page de manuel `scha_resourcetype_get`(1HA).

Commandes relatives aux groupes de ressources

Les commandes ci-dessous vous donnent accès aux informations sur un groupe de ressources ou vous permettent de redémarrer un groupe de ressources.

`scha_resourcegroup_get`
Présente des informations sur un groupe de ressources contrôlé par le RGM. Cette commande fournit la même fonctionnalité que la fonction C

`scha_resourcetype_get ()`. Pour plus d'informations sur cette commande, reportez-vous à la page de manuel `scha_resourcegroup_get`(1HA).

`scha_control`
Demande le redémarrage d'un groupe de ressources contrôlé par le RGM ou son déplacement vers un autre nœud. Cette commande est équivalente à la fonction C `scha_control()`. Elle est décrite plus en détail dans la page de manuel `scha_control`(1HA).

Commande du cluster

`scha_cluster_get`
Fournit des informations sur un cluster : nom du cluster, noms des noeuds, ID, états et groupes de ressources. Cette commande renvoie les mêmes données que la fonction C `scha_cluster_get()`. Elle est décrite plus en détail dans la page de manuel `scha_cluster_get`(1HA).

Fonctions C

Les fonctions C sont utilisées pour mettre en œuvre, à l'aide de programmes en langage C, les méthodes de rappel des types de ressources qui représentent les services contrôlés par le RGM du cluster. Elles permettent d'effectuer les tâches suivantes :

- Accéder à des informations sur les ressources, les groupes et les types de ressources, ainsi que sur les clusters
- Définir les propriétés `Status` et `Status_msg` d'une ressource, à l'aide d'un détecteur
- Demander le redémarrage ou le déplacement d'un groupe de ressources
- Convertir un code d'erreur en message

Remarque – cette rubrique ne fournit que de brèves descriptions des fonctions C ; la référence faisant foi en matière de fonctions C se trouve dans les pages de manuel 3HA. Sauf indication contraire, chaque fonction est associée à une page de manuel du même nom. Pour obtenir plus d'informations sur les arguments de sortie et les codes de retour des fonctions C, reportez-vous à la page de manuel `scha_calls`(3HA).

Fonctions relatives aux ressources

Les fonctions ci-dessous renvoient des informations sur une ressource gérée par le gestionnaire RGM ou présentent son état selon son détecteur.

`scha_resource_open()`, `scha_resource_get ()` et `scha_resource_close()`
Ces fonctions fournissent des informations sur une ressource gérée par le RGM. `scha_resource_open()` initialise l'accès à la ressource et renvoie un

identificateur destiné à la fonction `scha_resource_get()`, qui accède aux informations sur la ressource. `scha_resource_close()` annule la validité de l'identificateur et libère la mémoire affectée aux valeurs de retour de `scha_resource_get ()`.

Une ressource peut être modifiée (à la suite d'une reconfiguration du cluster ou d'une action administrative) après le renvoi de son identificateur par la fonction `scha_resource_open()`. Dans ce cas, `scha_resource_get()` risque d'obtenir des informations inexactes. Si un cluster a été reconfiguré ou qu'une action administrative a été effectuée sur une ressource, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resource_get()` pour indiquer que les informations sur la ressource ont peut-être changé. Ce message ne constitue pas une erreur fatale : il ne bloque pas la fonction. Vous pouvez ignorer le message et accepter les informations renvoyées. Vous pouvez également fermer l'identificateur actuel et ouvrir un nouvel identificateur pour accéder aux informations sur la ressource.

Ces trois fonctions sont décrites dans une même page de manuel, accessible à l'aide de la fonction `scha_resource_open`(3HA), `scha_resource_get`(3HA) ou `scha_resource_close`(3HA).

`scha_resource_setstatus()`
Définit les propriétés `Status` et `Status_msg` d'une ressource contrôlée par le 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 propre aux détecteurs de ressources, tout programme peut l'appeler.

Fonctions relatives au type de ressources

Les fonctions suivantes fournissent des informations sur un type de ressources enregistré dans le gestionnaire RGM.

`scha_resourcetype_open()`, `scha_resourcetype_get ()` et `scha_resourcetype_close()`
La fonction `scha_resourcetype_open()` initialise l'accès à une ressource et renvoie un identificateur destiné à la fonction `scha_resourcetype_get()`, qui accède aux informations sur le type de ressources.
`scha_resourcetype_close()` annule la validité de l'identificateur et libère la mémoire affectée aux valeurs de retour de `scha_resourcetype_get ()`.

Un type de ressource peut être modifié (à la suite d'une reconfiguration du cluster ou d'une action administrative) après le renvoi de son identificateur par la fonction `scha_resourcetype_open()`. Dans ce cas, `scha_resourcetype_get()` risque d'obtenir des informations inexactes. Si un cluster a été reconfiguré ou qu'une action administrative a été effectuée sur un type de ressource, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resourcetype_get()` pour

indiquer que les informations sur le type de ressource ont peut-être changé. Ce message ne constitue pas une erreur fatale : il ne bloque pas la fonction. Vous pouvez ignorer le message et accepter les informations renvoyées. Vous pouvez également fermer l'identificateur actuel et en ouvrir un autre pour accéder aux informations sur le type de ressource.

Ces trois fonctions sont décrites dans une même page de manuel, accessible à l'aide de la fonction `scha_resourcetype_open`(3HA), `scha_resourcetype_get`(3HA) ou `scha_resourcetype_close`(3HA).

Fonctions relatives au groupe de ressources

Les fonctions suivantes vous donnent accès aux informations sur un groupe de ressources ou vous permettent de redémarrer un groupe de ressources.

`scha_resourcegroup_open()`, `scha_resourcegroup_get ()` et `scha_resourcegroup_close()`
Ces fonctions présentent des informations sur un groupe de ressources contrôlé par le RGM. La fonction `scha_resourcegroup_open()` initialise l'accès au groupe de ressources et renvoie un identificateur destiné à la fonction `scha_resourcegroup_get ()`, qui accède aux informations sur le groupe de ressources. `scha_resourcegroup_close()` annule la validité de l'identificateur et libère la mémoire affectée aux valeurs de retour de `scha_resourcegroup_get()`.

Un groupe de ressources peut être modifié (à la suite d'une reconfiguration du cluster ou d'une action administrative) après le renvoi de son identificateur par la fonction `scha_resourcegroup_open()`. Dans ce cas, `scha_resourcegroup_get()` risque d'obtenir des informations inexactes. Si un cluster a été reconfiguré ou qu'une action administrative a été effectuée sur un groupe de ressources, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_resourcegroup_get()` pour indiquer que les informations sur le groupe de ressources ont peut-être changé. Ce message ne constitue pas une erreur fatale : il ne bloque pas la fonction. Vous pouvez ignorer le message et accepter les informations renvoyées. Vous pouvez également fermer l'identificateur actuel et en ouvrir un autre pour accéder aux informations sur le groupe de ressources.

Ces trois fonctions sont décrites dans une même page de manuel, accessible à l'aide de la fonction `scha_resourcegroup_open`(3HA), `scha_resourcegroup_get`(3HA) ou `scha_resourcegroup_close`(3HA).

`scha_control()`
Demande le redémarrage d'un groupe de ressources contrôlé par le RGM ou son déplacement vers un autre nœud. Cette fonction est décrite plus en détail dans la page de manuel `scha_control`(3HA).

Fonctions du cluster

Les fonctions ci-dessous fournissent des informations sur un cluster.

`scha_cluster_open()`, `scha_cluster_get ()` et `scha_cluster_close()`
Ces fonctions fournissent des informations sur un cluster : nom du cluster, noms des noeuds, ID, états et groupes de ressources.

Un cluster peut être modifié (lors d'une reconfiguration ou d'une action administrative) après le renvoi de son identificateur par la fonction `scha_cluster_open()`. Dans ce cas, `scha_cluster_get()` risque d'obtenir des informations erronées. Si un cluster a été reconfiguré ou a subi une action administrative, le RGM renvoie le code d'erreur `scha_err_seqid` à `scha_cluster_get()` pour indiquer que les informations sur le cluster ont peut-être changé. Ce message ne constitue pas une erreur fatale : il ne bloque pas la fonction. Vous pouvez ignorer le message et accepter les informations renvoyées. Vous pouvez également fermer l'identificateur actuel et en ouvrir un autre pour accéder aux informations sur le cluster.

Ces trois fonctions sont décrites dans une même page de manuel, accessible à l'aide de la fonction `scha_cluster_open`(3HA), `scha_cluster_get`(3HA) ou `scha_cluster_close`(3HA).

`scha_cluster_getlogfacility()`
Renvoie le numéro de l'utilitaire de journalisation système utilisé comme journal du cluster. Cette fonction utilise la valeur renvoyée, avec la fonction `syslog()` de Solaris, pour enregistrer les événements et les messages de statut dans le journal du cluster. Elle est décrite plus en détail dans la page de manuel `scha_cluster_getlogfacility`(3HA).

`scha_cluster_getnodename()`
Renvoie le nom du noeud du cluster sur lequel est appelée la fonction. Cette fonction est décrite plus en détail dans la page de manuel `scha_cluster_getnodename`(3HA).

Fonction de l'utilitaire

Cette fonction convertit les codes d'erreur en messages d'erreur équivalents.

`scha_strerror()`
Convertit un code d'erreur renvoyé par l'une des fonctions `scha_` en un message d'erreur. Vous pouvez associer cette fonction à la commande `logger` afin d'enregistrer les messages dans le journal système de Solaris (`syslog`). Cette fonction est décrite plus en détail dans la page de manuel `scha_strerror`(3HA).

Méthodes de rappel de l'interface APIGR

Les méthodes de rappel sont les éléments essentiels que l'API fournit pour la mise en œuvre d'un type de ressource. Elles permettent au RGM de gérer les ressources du cluster en cas de modification des membres du cluster (par exemple, initialisation ou arrêt brutal d'un nœud).

Remarque – pour exécuter les méthodes de rappel, le RGM utilise des droits de superutilisateur ou de rôle équivalent, car le programme client contrôle les services à haute disponibilité du système du cluster. Installez et gérez ces méthodes avec des membres et des autorisations de fichiers restrictifs. Vous devez notamment leur attribuer un propriétaire privilégié (par exemple, `bin` ou `root`) et les rendre inaccessibles en écriture.

Cette rubrique décrit les arguments et les codes de sortie des méthodes de rappel. Elle décrit les méthodes de rappel des catégories suivantes :

- Méthodes de contrôle et d'initialisation
- Méthodes de prise en charge administrative
- Méthodes relatives au réseau
- Méthodes de contrôle des détecteurs

Remarque – cette rubrique décrit brièvement les méthodes de rappel ; elle indique leur point d'exécution et leur effet escompté. Cependant, la référence faisant foi en matière de méthodes de rappel est la page de manuel `rt_callbacks`(1HA).

Arguments destinés aux méthodes de rappel

Pour exécuter les méthodes de rappel, le RGM utilise la syntaxe de commande suivante :

method `-R` *resource-name* `-T` *type-name* `-G` *group-name*

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 sous la forme des valeurs marquées suivantes :

- `-R` : nom de l'instance de la ressource

- `-T` : type de la ressource
- `-G` : groupe dans lequel la ressource est configurée

Ayez recours aux arguments disposant de fonctions d'accès pour récupérer des informations sur la ressource.

L'appel de la méthode `Validate` utilise des arguments supplémentaires, notamment les valeurs des propriétés de la ressource et le groupe sur lequel la méthode est appelée.

Pour obtenir plus d'informations, reportez-vous à la page de manuel `scha_calls`(3HA).

Codes de sortie des méthodes de rappel

Toutes les méthodes de rappel utilisent les mêmes codes de sortie. Ces codes définissent l'effet de l'appel de la méthode sur l'état de la ressource. Ils sont décrits plus en détail dans la page de manuel `scha_calls`(3HA). Les codes de sortie se classent en deux catégories principales :

- `0` : la méthode a réussi
- Toute valeur différente de 0 : la méthode a échoué

Le RGM gère également les arrêts anormaux lors de l'exécution des méthodes de rappel, notamment les dépassements des délais d'attente et les core dumps.

Les mises en œuvre de méthodes doivent consigner les informations sur les échecs en utilisant la fonction `syslog()` sur chaque nœud. En effet, les sorties écrites dans `stdout` ou `stderr` ne sont parfois pas communiquées à l'utilisateur, même si elles sont affichées sur la console du nœud local.

Méthodes de rappel de contrôle et d'initialisation

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

`Start`
: La méthode Start, exécutée par le gestionnaire RGM sur un nœud de cluster lorsque le groupe qui contient une ressource donnée est connecté à ce nœud, active cette ressource sur ce nœud.

  La méthode `Start` ne doit être arrêtée que lorsque la ressource qu'elle active a démarré et est disponible sur le nœud local. Par conséquent, avant sa fermeture, la méthode de `Démarrage` doit interroger la ressource afin de déterminer si elle a démarré. En outre, son délai d'attente doit être suffisamment long, car certaines ressources (par exemple, les démons de bases de données) mettent un certain temps à démarrer.

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

Le délai d'attente de la méthode `Start` d'une ressource est déterminé par la propriété `Start_timeout` du fichier RTR.

`Stop`
: La méthode Stop est la méthode obligatoire que le RGM exécute sur un nœud de cluster lorsque le groupe qui contient une ressource donnée est déconnecté de ce nœud. Cette méthode désactive la ressource si elle est active.

  Elle ne doit être fermée que lorsque la méthode qu'elle contrôle a été totalement désactivée et a fermé tous ses descripteurs de fichiers. Si elle était fermée trop tôt, le RGM supposerait à tort que la ressource a été désactivée (même si cette ressource était encore active) et des données risqueraient d'être corrompues. Pour prévenir toute corruption de données, la meilleure méthode consiste à mettre fin à tous les processus exécutés sur le nœud local associé à la ressource.

  Avant sa fermeture, la méthode `Arrêt` doit interroger la ressource afin de déterminer si elle est arrêtée. En outre, son délai d'attente doit être suffisamment long, car certaines ressources (par exemple, les démons de bases de données) mettent un certain temps à démarrer.

  La manière dont le gestionnaire RGM réagit à l'échec de la méthode `Stop` dépend du paramétrage de la propriété `Failover_mode`. Pour plus d'informations, voir "Propriétés des ressources" à la page 255.

  Le délai d'attente de la méthode `Stop` associée à une ressource est défini par la propriété `Stop_timeout` du fichier RTR.

`Init`
: Init est une méthode facultative que le gestionnaire RGM exécute ponctuellement sur une ressource afin de l'initialiser, lorsque cette ressource devient gérée, c'est-à-dire lorsque cette ressource est créée (si son groupe est déjà géré) ou que son groupe passe de l'état non géré à l'état géré. Elle est appelée sur les nœuds identifiés par la propriété de ressource `Init_nodes`.

`Fini`
: Fini est une méthode facultative que le RGM appelle, à des fins de nettoyage, lorsqu'une ressource devient non gérée, c'est-à-dire lorsque le groupe qui contient la ressource passe à l'état non géré ou que la ressource elle-même est supprimée (si elle appartient à un groupe de ressources géré). Elle est appelée sur des nœuds identifiés par la propriété de ressource `Init_nodes`.

`Boot`
: Le RGM exécute cette méthode facultative, similaire à `Init`, pour initialiser une ressource sur des nœuds qui rejoignent le cluster alors que le groupe auquel appartient cette ressource est déjà géré par le RGM. Les nœuds sur lesquels est exécutée cette méthode sont identifiés par la propriété de ressource `Init_nodes` . La méthode `Boot` est appelée lorsque l'un des nœuds concernés démarre ou redémarre, donc se connecte ou se reconnecte au cluster.

Remarque – l'échec des fonctions `Init`, `Fini` ou `Boot` entraîne la création d'un message d'erreur par la fonction `syslog()` ; il n'a aucun autre effet sur 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. Pour les réaliser, les mises en œuvre des types de ressources utilisent les méthodes de rappel `Validate` et `Update`.

`Validate`
: Le RGM appelle la méthode facultative `Validate` lorsqu'une ressource est créée ou qu'une action administrative met à jour les propriétés de la ressource ou de son groupe. La méthode est appelée sur l'ensemble de nœuds de clusters identifiés par la propriété `Init_nodes` du type de la ressource. La méthode `Validate` est appelée avant la création ou la mise à jour effective. Par conséquent, un code de sortie d'échec de la méthode sur un nœud, quel qu'il soit, entraîne l'annulation de la création ou de la mise à jour.

  `Validate` n'est appelée que lorsque l'administrateur d'un cluster modifie les propriétés d'une ressource ou d'un groupe de ressources et non lorsque le RGM définit des propriétés, ni lorsqu'un détecteur règle les propriétés de ressource `Status` et `Status_msg`.

`Update`
: Le RGM exécute la méthode facultative `Update` pour signaler à une ressource en cours d'exécution que ses propriétés ont été modifiées, 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. Elle peut utiliser les fonctions d'accès de l'interface API pour lire les valeurs de propriété susceptibles d'affecter une ressource active et régler les ressources en cours d'exécution en conséquence.

  Remarque – en cas d'échec de la méthode `Update`, la fonction `syslog()` émet un message d'erreur ; cependant, cet échec n'a aucun autre effet sur la gestion de la ressource par le RGM.

Méthodes de rappel relatives au réseau

Pour certains services qui utilisent des ressources d'adresse réseau, les étapes de démarrage et d'arrêt devront être exécutées dans un ordre précis, dépendant de la configuration. Les méthodes de rappel facultatives `Prenet_start` et `Postnet_stop` permettent à la mise en œuvre d'un type de ressource d'effectuer des actions de démarrage et d'arrêt spéciales avant et après la configuration ou l'annulation de la configuration d'une adresse réseau liée.

`Prenet_start`
: Cette méthode facultative permet d'effectuer des actions de démarrage spéciales avant la configuration des adresses réseau d'un même groupe de ressources.

`Postnet_stop`
: Cette méthode facultative permet d'effectuer des actions d'arrêt spéciales après le retrait des adresses réseau d'un groupe.

Méthodes de rappel de contrôle des détecteurs

Une mise en œuvre de type de ressources peut comporter un programme destiné à contrôler les performances d'une ressource, à écrire un rapport sur son statut ou à intervenir en cas de défaillance d'une ressource. Les méthodes `Monitor_start`, `Monitor_stop` et `Monitor_check` permettent d'implémenter un détecteur de ressource dans une mise en œuvre de type de ressources.

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

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

`Monitor_check`
: Cette méthode facultative est appelée pour évaluer la fiabilité d'un nœud avant qu'un groupe de ressources ne soit réaffecté à ce nœud. Lorsque vous la mettez en œuvre, vous devez veiller à ce qu'elle n'entre en conflit avec aucune autre méthode exécutée au même moment.

CHAPITRE 4

Modification d'un type de ressources

Ce chapitre présente les aspects essentiels de la modification des types de ressources, ainsi que des informations sur la manière d'autoriser un administrateur de cluster à mettre à niveau une ressource.

Ce chapitre contient les rubriques suivantes :

Modification des types de ressources

Les administrateurs de clusters doivent être à même d'effectuer les tâches suivantes :

- Installer et enregistrer une nouvelle version d'un type de ressources existant
- Autoriser l'enregistrement de plusieurs versions d'un type de ressources donné
- Mettre à niveau une ressource existante vers une nouvelle version de son type de ressources, sans avoir à la supprimer et à la recréer

Les types de ressources que vous prévoyez de mettre à niveau sont appelés types de ressources *prêts pour la mise à niveau*. Dans un type de ressources, vous pouvez modifier les éléments suivants :

- les attributs des propriétés du type de ressources ;
- l'ensemble des propriétés de ressources déclarées (notamment les propriétés standard et les propriétés d'extension) ;

- les attributs des propriétés des ressources, notamment `default`, `min`, `max`, `arraymin`, `arraymax` et `tunability` ;
- l'ensemble des méthodes déclarées ;
- la mise en œuvre des méthodes ou des détecteurs.

Remarque – il n'est pas indispensable de modifier un type de ressources lorsque l'on modifie du code d'application.

Pour fournir aux administrateurs de clusters des outils qui leur permettront de mettre à niveau un type de ressources, vous devez tenir compte de certaines exigences. Ce chapitre vous apporte les informations nécessaires pour configurer ces outils.

Contenu du fichier d'enregistrement du type de ressources

Nom du type de ressources

Le nom d'un type de ressources se compose de trois propriétés, définies dans le fichier RTR comme *vendor-id*, *resource-type* et *rt-version*. La commande `scrgadm` insère les délimiteurs requis pour créer le nom du type de ressources (point et deux-points) :

vendor-id . *resource-type* : *rt-version*

Le préfixe *vendor-id* permet de distinguer deux fichiers d'enregistrement portant le même nom, mais fournis par des sociétés différentes. Pour éviter tout risque d'ambiguité de la propriété *vendor-id*, il est recommandé d'utiliser le symbole de la société lors de la création du type de ressources. La propriété *rt-version* permet de faire la distinction entre plusieurs versions enregistrées (mises à niveau) du même type de ressources.

La commande ci-dessous fournit le nom complet d'un type de ressources :

```
# scha_resource_get -O Type -R resource-name -G resource-group-name
```

Les noms de types de ressources enregistrés dans les versions antérieures à Sun Cluster 3.1 se présentent toujours sous la forme suivante :

vendor-id . *resource-type*

Le format des noms de types de ressources est décrit dans la rubrique "Format des noms de types de ressources " à la page 334.

Définition des directives #$upgrade et #$upgrade_from

Pour garantir que le type de ressources modifié sera prêt pour la mise à niveau, incluez la directive #$upgrade dans son fichier RTR. Ajoutez ensuite éventuellement une ou plusieurs directives #$upgrade_from pour chacune des versions antérieures à prendre en charge.

Dans le fichier RTR, les directives #$upgrade et #$upgrade_from doivent être placées entre les déclarations des propriétés du type de ressources et les déclarations de ressources. Pour plus d'informations, reportez-vous à la page de manuel rt_reg(4).

EXEMPLE 4–1 Directive #$upgrade_from dans un fichier RTR

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$upgrade_from   "1.2"   WHEN_OFFLINE
#$upgrade_from   "1.3"   WHEN_OFFLINE
#$upgrade_from   "2.0"   WHEN_UNMONITORED
#$upgrade_from   "2.1"   ANYTIME
#$upgrade_from   ""      WHEN_UNMANAGED
```

La directive #$upgrade_from a le format suivant :

`#$upgrade_from` *version tunability*

version
RT_version. Si le type de ressources n'a pas de version ou que sa version est différente de celle que vous avez définie dans le fichier RTR, saisissez une chaîne vide ("").

tunability
Cet argument définit les conditions selon lesquelles l'administrateur du cluster est autorisé à mettre à niveau la version RT_version indiquée.

Dans les directives #$upgrade_from, les valeurs de capacité de réglage sont les suivantes :

ANYTIME
Utilisez cette option lorsqu'il n'existe aucune restriction ou que l'administrateur peut mettre la ressource à niveau. La ressource peut rester connectée pendant la mise à niveau.

WHEN_UNMONITORED
Utilisez cette option lorsque les méthodes de la version du nouveau type de ressources remplissent les conditions suivantes :

- Les méthodes Update, Stop, Monitor_check et Postnet_stop sont compatibles avec les méthodes de démarrage de l'ancienne version du type de ressources (Prenet_stop et Start).
- La méthode Fini est compatible avec les méthodes Init des anciennes versions.

L'administrateur du cluster doit seulement arrêter le détecteur de la ressource avant de procéder à la mise à niveau.

`WHEN_OFFLINE`
Utilisez cette option lorsque la méthode `Update`, `Stop`, `Monitor_check` ou `Postnet_stop` du nouveau type de ressources est :

- compatible avec la méthode `Init` d'une version antérieure,
- incompatible avec les méthodes de démarrage (`Prenet_stop` et `Start`) d'une version antérieure du type de ressources.

L'administrateur du cluster doit déconnecter la ressource avant de procéder à la mise à niveau.

`WHEN_DISABLED`
Cette option est similaire à `WHEN_OFFLINE`. Cependant, l'administrateur du cluster doit désactiver la ressource avant d'effectuer la mise à niveau.

`WHEN_UNMANAGED`
Utilisez cette option si la méthode `Fini` de la nouvelle version du type de ressources est incompatible avec la méthode `Init` d'une version antérieure. L'administrateur du cluster doit mettre le groupe de ressources existant à l'état non géré avant de procéder à la mise à niveau.

Si une version du type de ressources n'apparaît pas dans la liste des directives `#$upgrade_from`, le RGM lui impose l'option de capacité de réglage `WHEN_UNMANAGED`, par défaut.

`AT_CREATION`
Cette option empêche la mise à niveau des ressources vers la nouvelle version du type de ressources. Si elle est utilisée, l'administrateur du cluster doit supprimer et recréer la ressource.

Modification de la propriété `RT_version` d'un fichier RTR

Ne modifiez la propriété `RT_version` d'un fichier RTR que lorsque le contenu du fichier RTR change. Attribuez-lui une valeur qui indique clairement qu'il s'agit de la dernière version du type de ressources.

N'incluez pas les caractères ci-dessous dans la chaîne `RT_version` du fichier RTR, sinon l'enregistrement du type de ressources échouera :

- Espace
- Tabulation
- Barre oblique (`/`)
- Barre oblique inverse (`\`)
- Astérisque (`*`)
- Point d'interrogation (`?`)

- Virgule (`,`)
- Point virgule (`;`)
- Crochet gauche (`[`)
- Crochet droit (`]`)

La propriété `RT_version`, facultative dans Sun Cluster 3.0, est obligatoire depuis Sun Cluster 3.1.

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

Dans Sun Cluster 3.0, les noms des types de ressources ne contiennent pas de suffixe de version :

vendor-id . *resource-type*

Les noms de types de ressources enregistrés initialement dans Sun Cluster 3.0 conservent cette syntaxe même si le logiciel de cluster a été mis à niveau vers Sun Cluster 3.1 ou une version ultérieure. De même, un type de ressources dont le fichier RTR ne contient pas la directive `#$upgrade` aura un nom au format Sun Cluster 3.0 (sans suffixe de version) même s'il est enregistré sur un cluster fonctionnant sous Sun Cluster 3.1 ou une version ultérieure.

L'administrateur du cluster peut enregistrer des fichiers RTR à l'aide de la directive `#$upgrade` ou de la directive `#$upgrade_from` dans Sun Cluster 3.0. Cependant, la mise à niveau des ressources existantes vers un nouveau type de ressources dans Sun Cluster 3.0 n'est pas prise en charge.

Étapes et effets d'une mise à niveau effectuée par un administrateur de clusters

Lorsqu'un administrateur de clusters met à niveau un type de ressources :

- Si les attributs des propriétés de ressource ne remplissent pas les conditions de validité de la nouvelle version du type de ressources, l'administrateur de clusters doit fournir des valeurs valides. C'est le cas, notamment :
  - lorsque la nouvelle version du type de ressources n'a pas de valeur par défaut et utilise une propriété non déclarée dans la version antérieure ;
  - lorsqu'une ressource utilise une propriété dont la valeur n'est pas déclarée ou n'est pas valide dans la nouvelle version. Les propriétés déclarées dans une ancienne version du type de ressources, mais non dans sa nouvelle version,

sont supprimées de la ressource.

- Toute tentative de mise à niveau d'une version du type de ressource non prise en charge échoue.
- Après la mise à niveau, les ressources héritent des attributs de propriétés de ressource pour toutes les propriétés de la nouvelle version du type de ressources.
- Si vous changez la valeur par défaut d'un type de ressources dans le fichier RTR, les ressources existantes héritent de la nouvelle valeur par défaut, même si la capacité de réglage de la propriété est définie sur `AT_CREATION` ou `WHEN_DISABLED`. Si l'administrateur de clusters crée une propriété du même type, celle-ci hérite également de la valeur par défaut. Cependant, s'il attribue une nouvelle valeur à la propriété, la nouvelle valeur remplace la valeur par défaut définie dans le fichier RTR.

Remarque – les ressources créées dans Sun Cluster 3.0 n'héritent pas des attributs par défaut des propriétés de ressource lorsqu'elles sont mises à niveau vers une nouvelle version de Sun Cluster. Cette restriction s'applique uniquement aux clusters Sun Cluster 3.0 mis à niveau vers Sun Cluster 3.1 ; en outre, les administrateurs de clusters peuvent la surmonter en attribuant des valeurs aux propriétés (les valeurs définies remplaçant alors les valeurs par défaut).

Utilisation du code de contrôle du type de ressources

Il est toujours possible à l'administrateur du cluster d'enregistrer un type de ressources prêt à la mise à niveau dans Sun Cluster 3.0. Cependant, Sun Cluster enregistrera le nom du type de ressources sans le suffixe de version. Pour fonctionner correctement dans Sun Cluster 3.0 et Sun Cluster 3.1, les codes de contrôle de ces types de ressources doivent pouvoir gérer les deux conventions d'attribution de noms :

vendor-id . *resource-type* : *rt-version*
vendor-id . *resource-type*

Le format des noms de types de ressources est décrit dans la rubrique "Format des noms de types de ressources " à la page 334.

Il est impossible d'enregistrer deux fois la même version du type de ressources sous deux noms différents. Pour que le code de contrôle puisse identifier le bon nom, l'administrateur du cluster doit appeler les commandes ci-dessous dans ce code :

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

Il doit ensuite comparer les valeurs de sortie avec la valeur de `vers`. Pour chaque valeur de `vers`, une seule de ces commandes fonctionne.

Détermination des exigences relatives à l'installation et choix du type de package

Pour déterminer les exigences d'installation et le type de package à utiliser, tenez compte des points suivants :

- Pour que l'on puisse enregistrer un nouveau type de ressources, le fichier RTR correspondant doit être accessible sur le disque.
- Lors de la création d'une ressource du nouveau type, tous les chemins d'accès des méthodes, ainsi que le détecteur du nouveau type de ressources, doivent se trouver 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 savoir quel type de package utiliser, posez-vous les questions suivantes :

- 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 contrôle change-t-il ?
- Le code de méthode change-t-il ?
- Les nouvelles méthodes et/ou le code de contrôle sont-ils compatibles avec les versions antérieures ?

Les réponses à ces questions vous aideront à identifier le type de package à utiliser pour le nouveau type de ressources.

Étapes préalables à la modification du fichier RTR

Il n'est pas nécessaire de créer du nouveau code de contrôle ou de méthode lorsque vous modifiez un type de ressources : par exemple, vous pouvez changer uniquement la valeur par défaut ou la capacité de réglage d'une propriété de ressource. Dans ce cas, vous aurez uniquement besoin d'un nouveau chemin valide vers un fichier RTR lisible.

Si vous n'avez pas besoin de ré-enregistrer le type de ressources, vous pouvez remplacer l'ancienne version du fichier RTR par la nouvelle ; si vous devez ré-enregistrer le type de ressources, placez le nouveau fichier RTR dans un nouveau chemin.

Si la mise à niveau change la valeur par défaut ou la capacité de réglage d'une propriété, utilisez la méthode `Validate` correspondant à la nouvelle version du type de ressources en vue de vérifier que les attributs de propriété existants sont valides pour le nouveau type de ressources. S'ils ne le sont pas, l'administrateur du cluster pourra modifier les propriétés d'une ressource existante et leur attribuer les valeurs appropriées. Si la mise à niveau modifie les attributs `min`, `max` ou `type` d'une propriété, la commande `scrgadm` valide automatiquement ces contraintes lorsque l'administrateur du cluster met à niveau le type de ressources.

Si elle ajoute ou supprime une propriété, vous devrez probablement modifier les méthodes de rappel ou le code de contrôle.

Modification du code de contrôle

Si vous modifiez uniquement le code de contrôle d'un type de ressources, l'installation du package peut remplacer les binaires de contrôle.

Modification du code de méthode

Lorsque vous modifiez uniquement le code de méthode d'un type de ressources, vous devez déterminer si le nouveau code est compatible avec l'ancien. La réponse à cette question vous aidera à déterminer si vous devez enregistrer le nouveau code de méthode dans un nouveau chemin ou si vous pouvez remplacer les anciennes méthodes par les nouvelles.

Si vous pouvez appliquer les nouvelles méthodes `Stop`, `Postnet_stop` et `Fini` (si elles sont déclarées) à des ressources initialisées ou démarrées à l'aide des anciennes versions de `Start`, `Prenet_stop` ou `Init`, vous pouvez remplacer les anciennes méthodes par les nouvelles.

Si l'application d'une nouvelle valeur par défaut à une propriété provoque l'échec d'une méthode telle que `Stop`, `Postnet_stop` ou `Fini`, l'administrateur du cluster devra restreindre l'état de la méthode en conséquence, lors de la mise à niveau du type de ressources.

Pour l'autoriser à le faire, limitez la capacité de réglage de la propriété `Type_version`.

Une bonne méthode consiste à inclure toutes les versions précédentes d'un type de ressources, si elles sont prises en charge, dans le package. Il est ainsi possible de remplacer l'ancienne version du package par la nouvelle, sans remplacer ni supprimer les anciens chemins des méthodes. Vous devez décider du nombre d'anciennes versions à prendre en charge.

Choix du type de package à utiliser

Le tableau ci-dessous vous permet de choisir le type de package à utiliser pour vos nouveaux types de ressources.

TABLEAU 4–1 Type de package à utiliser

Type de modification	Valeur de la capacité de réglage	Type de package
Changer les propriétés dans le fichier RTR uniquement.	`ANYTIME`	Fournissez uniquement le nouveau fichier RTR.
Mettre à jour les méthodes.	`ANYTIME`	Placez les méthodes à jour dans un chemin différent de celui des anciennes méthodes.
Installer le nouveau programme de contrôle.	`WHEN_UNMONITORED`	Remplacez uniquement l'ancienne version du programme de contrôle.
Mettre à jour les méthodes. Les nouvelles méthodes `Update` et `Stop` sont incompatibles avec les anciennes méthodes `Start`.	`WHEN_OFFLINE`	Placez les méthodes à jour dans un chemin différent de celui des anciennes méthodes.
Mettre à jour les méthodes et ajouter des propriétés au fichier RTR. Les nouvelles méthodes requièrent de nouvelles propriétés. Cette modification vise à empêcher la ressource de se connecter si son groupe passe du mode hors ligne au mode en ligne sur un nœud, sans obliger le groupe de ressources à se déconnecter.	`WHEN_DISABLED`	Écrasez les versions précédentes des méthodes.
Mettre à jour les méthodes et ajouter des propriétés au fichier RTR. Les nouvelles méthodes ne requièrent pas de nouvelles propriétés.	`ANYTIME`	Écrasez les versions précédentes des méthodes.
Mettre à jour les méthodes. La nouvelle méthode `Fini` est incompatible avec la méthode `Init` antérieure.	`WHEN_UNMANAGED`	Placez les méthodes à jour dans un chemin différent de celui des anciennes méthodes.
Mettre à jour les méthodes. Le fichier RTR n'est pas modifié.	Sans objet. Le fichier RTR n'est pas modifié.	Écrasez les versions précédentes des méthodes. Puisque vous n'avez pas modifié le fichier RTR, il est inutile d'enregistrer ou de mettre à niveau la ressource.

Documentation requise pour un type de ressources modifié

Les administrateurs de clusters trouveront des instructions de mise à niveau des ressources dans la rubrique “Upgrading a Resource Type” du *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*. Afin de les aider à mettre à niveau un type de ressources modifié par vos soins, vous devez en outre leur communiquer les informations répertoriées dans cette section.

En règle générale, lorsque vous créez un nouveau type de ressources, vous devez fournir des documents qui remplissent les fonctions suivantes :

- Décrire les propriétés ajoutées, modifiées ou supprimées
- Indiquer comment rendre les propriétés compatibles avec les nouvelles exigences
- Présenter les contraintes de capacité de réglage appliquées aux ressources
- Signaler les nouveaux attributs de propriétés par défaut
- Informer les administrateurs de clusters qu’ils peuvent régler la valeur des propriétés de ressources existantes, si cela est nécessaire

Informations sur la préparation de l’installation de la mise à niveau

Expliquez aux administrateurs de clusters les procédures à suivre avant d’installer le package de mise à niveau sur un nœud :

- Si le package de mise à niveau remplace les méthodes existantes, l’administrateur doit redémarrer le nœud en mode non-cluster.
- Si le package de mise à niveau ne met à jour que le code de contrôle et ne modifie pas le code de méthode, l’administrateur doit laisser le nœud fonctionner en mode cluster. Il doit en outre désactiver la fonction de contrôle de tous les types de ressources.
- Si le package de mise à niveau met uniquement à jour le fichier RTR, sans modifier les codes de contrôle et de méthode, l’administrateur du cluster doit laisser le nœud fonctionner en mode cluster. Il doit également laisser la fonction de contrôle activée sur tous les types de ressources.

Moment opportun pour effectuer la mise à niveau

Expliquez aux administrateurs de clusters à quel moment ils peuvent mettre à niveau les ressources. Le moment opportun pour mettre la ressource à niveau dépend de l’option `#$upgrade_from` du fichier RTR - plus précisément, de l’option de capacité de réglage associée à chaque version de la ressource :

- à n'importe quel moment (`ANYTIME`) ;
- uniquement lorsque le contrôle de la ressource est désactivé (`WHEN_UNMONITORED`) ;
- uniquement lorsque la ressource est déconnectée (`WHEN_OFFLINE`) ;
- uniquement lorsque la ressource est désactivée (`WHEN_DISABLED`) ;
- uniquement lorsque le groupe de ressources est non géré (`WHEN_UNMANAGED`).

EXEMPLE 4–2 Moment où la mise à niveau est possible en fonction de la propriété `#$upgrade_from`

Cet exemple montre la relation entre la capacité de réglage de la directive `#$upgrade_from` et les circonstances dans lesquelles un administrateur de clusters peut faire passer une ressource à une version ultérieure de son type de ressources.

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$upgrade_from   "1.2"   WHEN_OFFLINE
#$upgrade_from   "1.3"   WHEN_OFFLINE
#$upgrade_from   "2.0"   WHEN_UNMONITORED
#$upgrade_from   "2.1"   ANYTIME
#$upgrade_from   ""      WHEN_UNMANAGED
```

Version	Moments où la mise à niveau est possible
1.1, 1.2 ou 1.3	Uniquement lorsque la ressource est déconnectée
2.0	Uniquement lorsque le contrôle de la ressource est désactivé
2.1	À tout moment
Autres versions	Uniquement lorsque le groupe de ressources est à l'état non géré

Informations sur les modifications apportées aux propriétés de ressources

Décrivez toutes les modifications que vous avez apportées au type de ressources et qui obligent les administrateurs de clusters à modifier les propriétés des ressources existantes lors de la mise à niveau. Ces modifications peuvent être des types suivants :

- Modification des paramètres par défaut d'une ou de plusieurs propriétés du type de ressources
- Création de nouvelles propriétés d'extension du type de ressources
- Suppression de propriétés du type de ressources
- Modification de l'ensemble des propriétés standard déclarées pour le type de ressources
- Modification des attributs des propriétés de ressource, notamment `min`, `max`, `arraymin`, `arraymax`, `default` et `tunability`

- Modification de l'ensemble des méthodes déclarées
- Modification des méthodes ou du détecteur de pannes

CHAPITRE 5

Service de données modèle

Ce chapitre présente un modèle de service de données Sun Cluster, HA-DNS, pour l'application `in.named`. Le démon `in.named` est la mise en œuvre 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.

Ce chapitre contient les rubriques suivantes :

Présentation du service de données modèle

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

Le redémarrage de l'application est géré par le gestionnaire de processus. Si le nombre d'échecs d'applications dépasse le nombre d'échecs accepté dans l'intervalle considéré, le détecteur de pannes bascule le groupe de ressources contenant la ressource d'application vers un autre nœud.

Le service de données modèle fournit une surveillance de panne sous forme d'une méthode `PROBE` qui utilise la commande `nslookup` pour s'assurer que l'application fonctionne correctement. Si la sonde détecte un service DNS bloqué, elle tente de corriger la situation en redémarrant localement l'application DNS. Si le redémarrage local de l'application DNS n'améliore pas la situation et que la sonde détecte régulièrement des problèmes au niveau du service, elle tente de basculer celui-ci vers un autre nœud du cluster.

Le service de données modèle inclut les éléments suivants :

- Un fichier d'enregistrement du type de ressource identifiant les propriétés statiques du service de données.
- Une méthode de rappel `Start` exécutée par le gestionnaire RGM pour démarrer le démon `in.named` lorsque le groupe de ressources qui contient le service de données HA-DNS est mis en ligne.
- Une méthode de rappel `Stop` exécutée par le gestionnaire RGM pour arrêter le démon `in.named` lorsque le groupe de ressources qui contient HA-DNS est mis 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 œuvre grâce à une méthode `PROBE` définie par l'utilisateur ; il est démarré et arrêté par les méthodes de rappel `Monitor_start` et `Monitor_stop`.
- Une méthode de rappel `Validate` exécutée par le gestionnaire RGM pour s'assurer que le répertoire de configuration du service est accessible.
- Une méthode de rappel `Update` exécutée par le gestionnaire RGM pour redémarrer le détecteur de pannes lorsque l'administrateur du cluster change la valeur de la propriété d'une 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.

Le gestionnaire RGM (Resource Group Manager) lit les informations du fichier RTR 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 man `rt_reg`(4) et à la rubrique "Paramétrage des propriétés de ressources et de types de ressources" à la page 35 pour obtenir plus d'informations.

Les sections suivantes décrivent les propriétés spécifiques dans le fichier RTR modèle. Ces sections proposent des listes de programme de différentes parties du fichier. Pour obtenir une liste de programme complète du contenu du fichier RTR modèle, reportez-vous à la rubrique "Liste de code du fichier RTR" à la page 283.

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 :

Remarque – les noms de propriété des groupes de ressources, des ressources et des types de ressources *ne* sont *pas* sensibles à la casse. Vous pouvez associer les majuscules et les minuscules dans le nom des propriétés.

```
#
# Copyright (c) 1998-2005 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#

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

Resource_type = "sample";
Vendor_id = SUNW;
RT_description = "Domain Name Service on Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;

RT_basedir=/opt/SUNWsample/bin;
Pkglist = SUNWsample;

Start         =   dns_svc_start;
Stop          =   dns_svc_stop;
```

```
Validate      =   dns_validate;
Update        =   dns_update;

Monitor_start =   dns_monitor_start;
Monitor_stop  =   dns_monitor_stop;
Monitor_check =   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.

Les informations suivantes décrivent ces propriétés :

- Vous pouvez indiquer le nom du type de ressource par la propriété `Resource_type` seule (`sample`) ou en utilisant *vendor-id* comme préfixe, suivi d'un point (.), puis de la propriété du type de ressource (`SUNW.sample`).

  Si vous indiquez *vendor-id*, utilisez le symbole boursier de la société qui définit le type de ressource. Le nom du type de ressources doit être unique sur le cluster.
- La propriété `RT_version` identifie la version du service de données modèle, telle qu'indiquée par le fournisseur.
- La propriété `Version_API` identifie la version de Sun Cluster. Par exemple, `API_version = 2` signifie que le service de données fonctionne sur Sun Cluster à partir de Sun Cluster 3.0. `API_version = 5` indique que le service de données fonctionne sur Sun Cluster, à partir de la version 3.1 9/04. Toutefois, `API_version = 5` indique également que le service de données ne peut pas être installé sur une version de Sun Cluster publiée la version 3.1 9/04. Cette propriété est décrite en détail sous l'entrée `API_version` dans "Propriétés des types de ressources" à la page 247.
- `Failover = TRUE` indique qu'il est impossible d'exécuter le service de données dans un groupe de ressources pouvant être en ligne simultanément sur plusieurs nœuds.
- `RT_basedir` pointe vers `/opt/SUNWsample/bin`, utilisé comme chemin d'accès pour compléter les chemins relatifs, tels que les chemins d'accès aux méthodes de rappel.
- `Start`, `Stop` et `Validate` fournissent les chemins d'accès aux programmes des méthodes de rappel correspondantes, exécutés par le gestionnaire RGM. Ces chemins d'accès dépendent du répertoire spécifié par `RT_basedir`.
- `Pkglist` identifie `SUNWsample` comme le package qui contient l'installation du service de données modèle.

Les propriétés des types de ressource non spécifiées dans ce fichier RTR, telles que `Single_instance`, `Init_nodes` et `Installed_nodes` , sont définies à leurs valeurs par défaut. "Propriétés des types de ressources" à la page 247 contient une liste complète des propriétés des types de ressource, y compris leurs valeurs par défaut.

L'administrateur du cluster ne peut modifier les valeurs des propriétés des types de ressource dans le fichier RTR.

Propriétés de ressource dans le fichier RTR modèle

Par convention, déclarez les propriétés de ressource après les propriétés des types de ressource dans le fichier RTR. Les propriétés de ressource incluent les propriétés définies au niveau du système fournies par le logiciel Sun Cluster et les propriétés d'extension que vous définissez. Pour les deux types, vous pouvez spécifier différents attributs de propriété fournis par le logiciel Sun Cluster, tels que les valeurs minimum, maximum et par défaut.

Propriétés définies par le système dans le fichier RTR

La liste de programme suivante montre les propriétés définies au niveau du système dans un fichier RTR modèle.

```
# Une liste de déclarations de propriétés de ressource entre accolades suit
# les déclarations des types de ressource. La déclaration du nom de propriété
# doit être le premier attribut après l'accolade d'ouverture de chaque entrée.

# Les propriétés <method>_timeout définissent le délai en secondes avant
# que le gestionnaire RGM ne considère que l'appel de la méthode a échoué.

# La valeur MIN de tous les délais d'attente de méthode est définie à 60
# secondes. Cela empêche les administrateurs de définir des délais plus
# courts, qui n'améliorent pas la performance de commutation/basculement et
# peuvent conduire à des actions RGM indésirables (erreurs de basculement,
# réinitialisation de nœud, ou déplacement d'un groupe de ressources en mode
# ERROR_STOP_FAILED, nécessitant l'intervention de l'opérateur). Définir des
# délais de méthode trop courts conduit à une *diminution* de la disponibilité
# globale du service de données.
{
   PROPERTY = Start_timeout;
   MIN=60;
   DEFAULT=300;
}

{
   PROPERTY = Stop_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Validate_timeout;
   MIN=60;
   DEFAULT=300;
}
{
```

```
   PROPERTY = Update_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Monitor_Start_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Monitor_Stop_timeout;
   MIN=60;
   DEFAULT=300;
}
{
   PROPERTY = Thorough_Probe_Interval;
   MIN=1;
   MAX=3600;
   DEFAULT=60;
   TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
   PROPERTY = Retry_count;
   MIN=0;
   MAX=10;
   DEFAULT=2;
   TUNABLE = ANYTIME;
}

# Définissez Retry_interval sur un multiple de 60 puisque cette valeur en
# secondes est convertie en minutes (arrondie à la valeur supérieure). Par
# exemple une valeur de 50 (secondes) est convertie à 1 minute. Utilisez cette
# propriété pour définir le nombre de tentatives dans le temps (Retry_count).
{
   PROPERTY = Retry_interval;
   MIN=60;
   MAX=3600;
   DEFAULT=300;
   TUNABLE = ANYTIME;
}

{
   PROPERTY = Network_resources_used;
   TUNABLE = AT_CREATION;
   DEFAULT = "";
}
```

Bien que le logiciel Sun Cluster fournisse les propriétés définies au niveau du système, vous pouvez leur attribuer des valeurs par défaut différentes en utilisant les attributs des propriétés de ressource. Reportez-vous à la rubrique "Attributs des propriétés de ressources" à la page 281 pour obtenir une liste complète des attributs que vous pouvez appliquer aux propriétés de ressource.

Notez les points suivants sur les propriétés des ressources définies au niveau système dans le fichier RTR modèle :

- Sun Cluster fournit une valeur minimale (1 seconde) et une valeur par défaut (3600 secondes ou une heure) pour tous les délais d'attente. Le fichier RTR modèle change le délai d'attente minimal à 60 secondes et la valeur par défaut à 300 secondes. Un administrateur du cluster peut accepter cette valeur par défaut ou modifier le délai d'attente avec une autre valeur, par exemple 60 ou plus. Sun Cluster n'a pas de valeur autorisée maximale.
- L'attribut `TUNABLE` des propriétés `Thorough_probe_interval`, `Retry_count` et `Retry_interval` est défini sur `ANYTIME`. Ce paramètre indique que l'administrateur du cluster peut modifier la valeur de ces propriétés, même quand le service de données fonctionne. Celles-ci sont utilisées par le détecteur de pannes mis en œuvre par le service de données modèle. Le service de données modèle met en œuvre 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 "Fonctionnement de la méthode `Update`" à la page 116.
- Les propriétés de ressource sont classées de la façon suivante :
  - **Requise**. L'administrateur du cluster doit spécifier une valeur lors de la création de la ressource.
  - **Facultatif**. Si l'administrateur du cluster ne précise pas de valeur, le système en fournit une par défaut.
  - **Conditionnelle**. Le gestionnaire 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 `Thorough_probe_interval` , `Retry_count`, `Retry_interval` et `Network_resources_used` ; vous devez donc 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 255 pour obtenir des informations sur la classification des propriétés.

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 pour pointer
# vers le répertoire contenant les fichiers de configuration utilisés par
# l'application. Pour cette application, DNS, spécifiez le chemin d'accès du fichier
# de configuration DNS sur PXFS (généralement named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
```

```
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "Chemin d'accès au répertoire de configuration";
}

# Valeur de délai d'attente en secondes avant de déclarer un échec de la sonde
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 120;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Valeur de délai d'attente de la sonde (secondes)";
}
```

Le fichier RTR modèle définit deux propriétés d'extension, `Confdir` et `Probe_timeout`. La propriété `Confdir` spécifie le chemin d'accès au répertoire de configuration de DNS. Ce répertoire contient le fichier `in.named` requis par le DNS pour fonctionner correctement. Les méthodes `Start` et `Validate` 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 `Start` vérifie que le nouveau répertoire est accessible.

La méthode `PROBE` du service de données modèle n'est pas une méthode de rappel Sun Cluster mais une méthode définie par l'utilisateur. Par conséquent, Sun Cluster ne lui fournit pas de propriété `Probe_timeout`. Vous devez définir une propriété d'extension dans le fichier RTR pour permettre à un administrateur du cluster de configurer une valeur `Probe_timeout`.

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. Chaque script de méthode dans le service de données modèle identifie l'interpréteur de commandes de la façon suivante :

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

Tous les scripts de méthode de l'application modèle exportent le chemin d'accès vers les binaires et les bibliothèques Sun Cluster plutôt que de compter sur la définition de la variable d'environnement `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 `Validate`, utilisent `pmfadm` pour démarrer ou arrêter le service de données ou le détecteur, en lui passant le nom de la ressource. Chaque script définit une variable, `PMF_TAG`, qui peut être passée à `pmfadm` pour identifier le service de données ou le détecteur.

De même, chaque script de méthode utilise la commande `logger` pour journaliser les messages dans le journal système. Chaque script définit une variable, `SYSLOG_TAG`, qui peut être passée à la commande `logger` avec l'option `-t` pour identifier le type, le nom et le groupe de ressource de la ressource pour laquelle le message est consigné.

Toutes les méthodes définissent `SYSLOG_TAG` de la même manière, comme cela est indiqué dans l'exemple de code suivant. Les méthodes `dns_probe`, `dns_svc_start` , `dns_svc_stop`, et `dns_monitor_check` définissent `PMF_TAG` comme suit (les commandes `pmfadm` et `logger` sont utilisées à partir de la méthode `dns_svc_stop`).

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

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # Envoie un signal SIGTERM au service de données et attend pendant 80% de
   # la valeur de délai totale.
   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 `dns_monitor_start`, `dns_monitor_stop` et `dns_update` définissent `PMF_TAG` de la façon suivante (la commande `pmfadm` est utilisée à partir de la méthode `dns_monitor_stop`) :

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

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
...
# Voir si le détecteur est en cours d'exécution et si oui, le tuer.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

Analyse des arguments de la fonction

Le gestionnaire RGM exécute toutes les méthodes de rappel, à l'exception de `Validate`, de la façon suivante :

method-name `-R` *resource-name* `-T` *resource-type-name* `-G` *resource-group-name*

Le nom de la méthode correspond au nom de chemin d'accès du programme mettant en œuvre 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 `RT_basedir`, 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 comme suit :

```
RT_basedir=/opt/SUNWsample/bin;
Start = dns_svc_start;
Stop =  dns_svc_stop;
...
```

Tous les arguments des méthodes de rappel sont passées en tant que valeurs marquées. L'argument `-R` indique le nom de l'instance de ressource. L'argument `-T` indique le type de la ressource. L'argument `-G` indique le groupe dans lequel la ressource est configurée. Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour obtenir plus d'informations sur les méthodes de rappel.

Remarque – la méthode `Validate` est appelée avec des arguments supplémentaires, à savoir les valeurs de propriété de la ressource et du groupe de ressources pour lesquels elle est appelée. Reportez-vous à la rubrique "Gestion des mises à jour des propriétés" à la page 111 pour obtenir plus d'informations.

Chaque méthode de rappel a besoin d'une fonction pour analyser les arguments transmis à la fonction. Tous les rappels étant 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 illustre le cas de la fonction `parse_args()`, utilisée pour les méthodes de rappel dans 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 lequel la ressource
                  # est configurée.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
             T)
                  # Nom du type de ressource.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
             *)
                  logger -p ${SYSLOG_FACILITY}.err \
                  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                  "ERROR: Option $OPTARG inconnue"
                  exit 1
                      ;;
             esac
    done
}
```

Remarque – bien que la méthode de 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 qu'utilisent les autres méthodes de rappel.

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

```
parse_args "$@"
```

Génération de messages d'erreur

Les méthodes de rappel doivent utiliser la fonction syslog() pour générer les messages d'erreur destinés aux utilisateurs finaux. Toutes les méthodes de rappel du service de données modèle utilisent la commande scha_cluster_get pour récupérer le numéro de la fonction syslog() utilisée pour le journal du cluster, comme cela est indiqué ci-après :

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

La valeur est stockée dans une variable de shell, SYSLOG_FACILITY, et peut être utilisée par la commande logger pour journaliser les messages dans le journal du cluster. Par exemple, la méthode Start du service de données modèle récupère la fonction syslog() et consigne un message indiquant que le service de données a été démarré de la façon 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 obtenir plus d'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.

Les propriétés définies au niveau système et les propriétés d'extension sont disponibles. Les premières sont prédéfinies et vous définissez les autres dans le fichier RTR.

Si vous utilisez `scha_resource_get()` pour obtenir la valeur d'une propriété définie au niveau système, spécifiez le nom de la propriété avec l'option `-O`. La commande ne renvoie que la *valeur* de la propriété. Par exemple, dans le service de données modèle, la méthode `Monitor_start` doit localiser le programme de sonde afin de pouvoir le démarrer. Le programme de sonde se trouve dans le répertoire de base du service de données, vers lequel pointe la propriété `RT_basedir`. La méthode `Monitor_start` récupère la valeur de `RT_basedir` et la place dans la variable `RT_BASEDIR`, comme suit :

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

Pour les propriétés d'extension, vous devez utiliser l'option `-O` pour indiquer que la propriété est une propriété d'extension. Vous devez également fournir le nom de la propriété comme dernier argument. Pour les propriétés d'extension, la commande renvoie à la fois le *type* et la *valeur* de la propriété. Par exemple, dans le service de données modèle, le programme de sonde récupère le type et la valeur de la propriété d'extension `Probe_timeout`, puis utilise la commande `awk` pour placer uniquement la valeur dans la variable de shell `PROBE_TIMEOUT`, comme suit :

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

Contrôle du service de données

Un service de données doit fournir une méthode de `Démarrage` ou de `Démarrage_avant_réseau` pour activer le démon d'application du cluster, ainsi qu'une méthode d'`Arrêt` ou d'`Arrêt_après_réseau` pour arrêter le démon d'application du cluster. Le service de données modèle met en œuvre des méthodes `Démarrage` et `Arrêt`. Reportez-vous à la rubrique "Choix des méthodes `Start` et `Stop` à utiliser" à la page 46 pour savoir dans quels cas il peut être préférable d'utiliser `Prenet_start` et `Postnet_stop` .

Fonctionnement de la méthode `Start`

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

Cette rubrique décrit les principaux éléments de la méthode de `Démarrage` pour l'application modèle. Cette rubrique ne décrit pas les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction `parse_args()`. Par ailleurs, elle ne décrit pas l'utilisation de la fonction `syslog()`. Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour une liste complète du code de la méthode `Start`, reportez-vous à la rubrique "Listing de code de la méthode `Start`" à la page 286.

Fonction de la méthode `Start`

Avant de tenter de démarrer le DNS, la méthode `Start` du service de données modèle s'assure que le répertoire de configuration et le fichier de configuration (`named.conf`) sont accessibles et disponibles. Les informations du fichier `named.conf` sont essentielles pour un bon fonctionnement du DNS.

Cette méthode de rappel utilise la fonction PMF (`pmfadm`) pour démarrer le démon DNS (`in.named()`. Si le DNS s'arrête brutalement ou ne parvient pas à démarrer, la fonction PMF tente de démarrer le démon DNS un certain nombre de fois (le nombre de tentatives étant paramétrable) pendant un intervalle spécifié. 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. La méthode `Start` effectue donc différents contrôles de validité pour vérifier que le répertoire et le fichier sont accessibles avant de tenter de démarrer 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. Cependant, l'administrateur du cluster spécifie l'emplacement réel lorsqu'il configure le service de données.

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

Remarque – `Confdir` étant une propriété d'extension, `scha_resource_get()` renvoie à la fois le type et la valeur. La commande `awk` récupère uniquement la valeur et place celle-ci dans une variable de shell, `CONFIG_DIR` .

```
# trouver la valeur de Confdir 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`

# Pour les propriétés d'extension, scha_resource_get renvoie le "type"
# et la "valeur". Récupérer uniquement la valeur de la propriété d'extension
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

La méthode `Start` utilise la valeur de `CONFIG_DIR` pour vérifier que le répertoire est accessible. S'il ne l'est pas, `Start` consigne un message d'erreur et se termine avec un état d'erreur. Reportez-vous à la rubrique "État de `Start` à la fermeture" à la page 98.

```
# 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 le fichier n'est pas présent, `Start` consigne un message d'erreur et se termine avec un état d'erreur.

```
# Se placer dans le répertoire $CONFIG_DIR au cas où il y aurait
# des chemins 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 fait appel aux services du gestionnaire de processus (`pmfadm`) pour démarrer l'application. La commande `pmfadm` vous permet de définir le nombre de tentatives de redémarrage de l'application pendant un intervalle spécifié. Le fichier RTR contient les propriétés suivantes : `Retry_count` spécifie le nombre de tentatives de redémarrage d'une application, et `Retry_interval` l'intervalle pendant lequel les tentatives doivent être effectuées.

La méthode `Start` récupère les valeurs de `Retry_count` et de `Retry_interval` en utilisant la fonction `scha_resource_get ()`, puis stocke leurs valeurs dans des variables de shell. La méthode `Start` passe ces valeurs à `pmfadm` en utilisant les options `-n` et `-t`.

```
# Récupère la valeur du nombre de tentatives dans le fichier RTR.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`
```

```
# Récupère la valeur de l'intervalle de tentative dans le fichier RTR. Cette valeur est en secondes
# et doit être convertie en minutes pour être passée à pmfadm. Remarquez que la
# conversion est arrondie à la valeur supérieure ; par ex., 50 secondes est arrondi à 1 minute.
((RETRY_INTRVAL=`scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME` / 60))

# Démarrer le démon in.named sous le controle de PMF. Il peut s'arrêter et redémarrer
# jusqu'à $RETRY_COUNT fois dans l'intervalle $RETRY_INTERVAL. S'il s'interrompt
# plus souvent, PMF cessera d'essayer de le redémarrer.
# Si un processus est déjà enregistré sous la marque
# <$PMF_TAG>, PMF envoie 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

# Consigne 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 `Start` à la fermeture

Une méthode `Start` ne doit pas se terminer avec un code de succès tant que l'application correspondante n'est pas effectivement en cours d'exécution et disponible, particulièrement si d'autres services de données en dépendent. L'un des moyens à votre disposition pour vérifier le succès de la méthode consiste à sonder l'application afin de vous assurer qu'elle fonctionne avant de quitter la méthode `Start`. Pour une application complexe, telle qu'une base de données, veillez à définir la propriété `Start_timeout` dans le fichier RTR sur une valeur suffisamment élevée pour permettre à l'application de s'initialiser et le cas échéant de récupérer après une panne.

Remarque – comme la ressource d'application (DNS) du service de données modèle démarre rapidement, la méthode de démarrage du service de données ne l'interroge pas pour vérifier son fonctionnement avant de se terminer avec succès.

Si cette méthode n'arrive pas à démarrer le DNS et se ferme en affichant un état d'échec, le RGM contrôle la propriété `Failover_mode`, qui détermine la réaction à adopter. Le service de données modèle ne définit pas explicitement la propriété `Failover_mode` qui a donc la valeur par défaut `NONE` (sauf si l'administrateur du cluster remplace la valeur par défaut par une autre valeur). Dans ce cas, le RGM ne prend pas de mesure autre que la définition de l'état du service de données. L'administrateur du cluster doit lancer un redémarrage sur le même nœud ou un basculement sur un autre nœud.

Fonctionnement de la méthode `Stop`

Le gestionnaire RGM exécute la méthode d'`Arrêt` sur un nœud du cluster lorsque le groupe de ressources contenant la ressource HA-DNS passe hors ligne sur ce nœud ou si le groupe de ressources est en ligne et la ressource désactivée. Cette méthode arrête le démon `in.named` (DNS) sur ce nœud.

Cette rubrique décrit les principaux éléments de la méthode d'`Arrêt` pour l'application modèle. Cette rubrique ne décrit pas les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction `parse_args()`. Par ailleurs, elle ne décrit pas l'utilisation de la fonction `syslog()`. Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour la liste complète du code de la méthode `Stop`, reportez-vous à la rubrique "Listing de code de la méthode `Stop`" à la page 289.

Fonction de la méthode `Stop`

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 façon d'effectuer un arrêt correct est d'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 solution pour placer le service de données dans cet état est d'envoyer un signal `SIGKILL` par l'intermédiaire de `pmfadm`.

La méthode `Stop` du service de données modèle tient compte de ces deux considérations. Elle envoie d'abord un signal `SIGTERM`. Si ce signal ne parvient pas à arrêter le service de données, elle envoie un signal `SIGKILL` .

Avant de tenter d'arrêter le DNS, cette méthode d'`Arrêt` vérifie si le processus tourne effectivement. Si le processus est en cours d'exécution, `Stop` utilise la fonction PMF (`pmfadm`) pour l'arrêter.

L'idempotence de cette méthode `Stop` est garantie. Bien que le gestionnaire RGM ne doive pas appeler deux fois une méthode `Stop` sans démarrer au préalable le service de données par un appel à sa méthode `Start`, le RGM pourrait appeler une méthode `Stop` sur une ressource alors que celle-ci n'a jamais été démarrée ou s'est déjà arrêtée d'elle-même. C'est pourquoi cette méthode `Arrêt` se ferme correctement même si le DNS ne tourne pas.

Arrêt de l'application

La méthode `Stop` fournit une approche à deux niveaux d'arrêt du service de données : une approche ordonnée ou en douceur utilisant un signal `SIGTERM` par l'intermédiaire de `pmfadm` et une approche brutale utilisant un signal `SIGKILL`. La

méthode `Stop` obtient la valeur `Stop_timeout` (le délai avant lequel la méthode `Stop` doit se terminer). `Arrêt` alloue alors 80 % de ce temps à un arrêt en douceur et 15 % à un arrêt abrupt (5 % sont réservés), de la manière indiquée dans l'exemple de code suivant.

```
STOP_TIMEOUT='scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME'
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))
((HARD_TIMEOUT=$STOP_TIMEOUT * 15/100))
```

La méthode d'`Arrêt` utilise `pmfadm -q` pour vérifier si le démon DNS fonctionne. Si le démon DNS est en cours d'exécution, `Stop` utilise d'abord `pmfadm -s` pour envoyer un signal `TERM` afin de terminer le processus DNS. Si ce signal ne parvient pas à terminer le processus après 80 % de la valeur de délai d'attente, `Stop` envoie un signal `SIGKILL`. Si ce signal ne parvient pas non plus à terminer le processus après 15 % de la valeur de délai d'attente, la méthode consigne un message d'erreur et se termine avec un état d'erreur.

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 `Stop` utilise `pmfadm` pour arrêter le processus DNS.

```
# Voir si in.named est en cours d'exécution, et si oui, le tuer.
if pmfadm -q $PMF_TAG; then
   # Envoyer un signal SIGTERM au service de données et attendre 80%
   # de la valeur de délai d'attente totale.
   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"

      # Comme le service de données ne s'est pas arrêté avec un signal SIGTERM, utiliser
      # maintenant SIGKILL et attendre encore 15% de la valeur de délai d'attente totale.
      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 UNSUCCESSFUL"
         exit 1
      fi
fi
else
   # Le service de données ne fonctionne pas pour l'instant. Consigner un message et
   # quitter avec un code de succès.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS is not started"

   # Même si HA-DNS ne fonctionne pas, quitter avec un code de succès afin d'éviter
```

```
  # de placer la ressource du service de données à l'état STOP_FAILED.
   exit 0
fi

# L'arrêt du DNS a réussi. Consigner un message et quitter avec un code de 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 `Stop` ne doit pas se terminer avec un code de succès tant que l'application correspondante n'est pas effectivement arrêtée, particulièrement 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 `Failover_mode` qui a donc la valeur par défaut `NONE` (sauf si l'administrateur du cluster remplace la valeur par défaut par une autre valeur). Dans ce cas, le gestionnaire RGM se contente de définir l'état du service de données sur `Stop_failed`. L'administrateur du cluster doit forcer l'arrêt de l'application et effacer l'état `Stop_failed`.

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 des éléments suivants :

- `dns_probe`, un programme défini par l'utilisateur qui utilise `nslookup` pour vérifier que 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.
- `dns_monitor_start`, une méthode de rappel que démarre `dns_probe`. Une fois le service de données modèle en ligne et si la surveillance est activée, le gestionnaire RGM appelle automatiquement `dns_monitor_start`.

- `dns_monitor_stop`, une méthode de rappel qui arrête `dns_probe`. Le gestionnaire RGM appelle automatiquement `dns_monitor_stop` avant de placer le service de données modèle hors ligne.
- `dns_monitor_check`, une méthode de rappel qui appelle la méthode `Validate` de manière à vérifier que le répertoire de configuration est disponible lorsque le programme `PROBE` bascule le service de données sur un autre noeud.

Fonctionnement du programme de sonde

Le programme `dns_probe` met en oeuvre un processus qui vérifie en permanence que la ressource DNS contrôlée par le service de données modèle fonctionne. Le programme `dns_probe` est démarré par la méthode `dns_monitor_start`, elle-même automatiquement exécutée par le RGM lorsque le service de données modèle a été mis en ligne. Le service de données est arrêté par la méthode `dns_monitor_stop`, que le RGM exécute 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 les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction `parse_args()`. Par ailleurs, elle ne décrit pas l'utilisation de la fonction `syslog()`. Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour une liste complète du code de la méthode `PROBE`, reportez-vous à la rubrique "Listing de code du programme `PROBE`" à la page 292.

Fonction du programme de sonde

Probe tourne en boucle infinie. Il utilise `nslookup` pour vérifier que la ressource DNS correcte fonctionne. Si le DNS fonctionne, la sonde passe en mode sommeil pendant un intervalle prédéfini (via la propriété système `Thorough_probe_interval`), puis vérifie à nouveau. 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 requiert les valeurs des propriétés suivantes :

- `Thorough_probe_interval` : pour définir le délai pendant lequel la sonde passe en mode de sommeil.
- `Probe_timeout` : pour mettre en oeuvre la valeur de délai d'attente de la sonde au niveau de la commande `nslookup`, qui effectue le test.
- `Network_resources_used` : pour obtenir l'adresse IP sur laquelle tourne le DNS.

- `Retry_count` et `Retry_interval` : pour déterminer le nombre de tentatives de redémarrage et la période pendant laquelle les effectuer.
- `RT_basedir` : 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 stocke dans des variables de shell, comme cela est indiqué ci-après :

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

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

DNS_HOST=‘scha_resource_get -O Network_resources_used -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘

RETRY_COUNT=‘scha_resource_get -O Retry_count -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

RETRY_INTERVAL=‘scha_resource_get -O Retry_interval -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

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

Remarque – pour les propriétés définies au niveau système, comme par exemple `Thorough_probe_interval` , la fonction `scha_resource_get()` renvoie la valeur uniquement. Pour les propriétés d'extension, comme par exemple `Probe_timeout`, la fonction `scha_resource_get()` retourne le type et la valeur. Utilisez la commande `awk` pour obtenir la valeur seule.

Contrôle de la fiabilité du service

La sonde elle-même est une boucle `while` infinie de commandes `nslookup`. Un fichier temporaire est défini avant la boucle `while` pour recevoir les réponses de `nslookup`. Les variables `probefail` et `retries` sont initialisées à 0.

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

La boucle `while` assure les tâches suivantes :

- Définit l'intervalle de sommeil de la sonde.
- Utilise `hatimerun` pour démarrer `nslookup`, passe la valeur `Probe_timeout`, 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 de `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 d'exécution de la sonde est spécifié dans la propriété
   # THOROUGH_PROBE_INTERVAL. Par conséquent, configurer la sonde en mode sommeil
  # pour une durée de THOROUGH_PROBE_INTERVAL.
   sleep $PROBE_INTERVAL

   # Exécute une commande nslookup de l'adresse IP servie par le DNS.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
   > $DNSPROBEFILE 2>&1

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

   # S'assurer que la réponse à nslookup vienne du serveur HA-DNS
   # et non d'un autre serveur de noms mentionné dans le fichier
   # /etc/resolv.conf.
   if [ $probefail -eq 0 ]; then
# Obtenir le nom du serveur qui a répondu à l'interrogation 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
```

Comparaison du redémarrage et du basculement

Si la variable `probefail` est différente de 0 (succès), le délai de la commande `nslookup` a expiré ou la réponse est venue d'un serveur autre que le serveur 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` est 0, un message est généré indiquant que la sonde a réussi.

```
   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 une fenêtre temporelle (`Retry_interval`) et un compteur d'échecs (`Retry_count`) afin de déterminer s'il convient ou non de redémarrer le DNS localement ou de demander que le RGM déplace le service de données sur un autre noeud. Cette fonction met en oeuvre la logique conditionnelle suivante. La liste du code de `decide_restart_or_failover()` dans la rubrique "Listing de code du programme `PROBE`" à la page 292 contient le code utilisé.

- 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 écoulé mais que le compteur de tentatives a dépassé la valeur autorisée, basculez le service sur un autre noeud. Si le basculement échoue, consignez une erreur et quittez le programme de sonde avec l'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 situé dans le répertoire (`RT_basedir`).
- Les propriétés de ressource définies au niveau système `Retry_count` et `Retry_interval` déterminent le nombre de tentatives de redémarrage ainsi que l'intervalle pendant lequel le comptage est effectué. Ces propriétés sont définies par défaut à deux tentatives au cours d'une période de 5 minutes (300 secondes) dans le fichier RTR, bien que l'administrateur du cluster puisse 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 106, pour obtenir des informations sur cette fonction.
- La fonction API `scha_control()`, avec l'option `GIVEOVER`, place 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 redémarrer le service de données sur le même noeud. Cette fonction exécute la logique suivante :

- Détermine si le service de données est toujours enregistré sous le PMF. Si le service est toujours enregistré, la fonction exécute les actions suivantes :
  - Obtient le nom de la méthode `Stop` et la valeur `Stop_timeout` pour le service de données.
  - Utilise `hatimerun` pour démarrer la méthode `Stop` du service de données, en lui passant la valeur `Stop_timeout`.
  - Si le service de données est correctement arrêté, obtient le nom de la méthode `Start` et la valeur `Start_timeout` pour le service de données.
  - Utilise `hatimerun` pour démarrer la méthode `Start` du service de données, en lui passant la valeur `Start_timeout`.
- Si le service de données n'est plus enregistré sous le PMF, il est très probable qu'il a dépassé le nombre maximum de tentatives autorisées sous le PMF. La fonction `scha_control()` est appelée avec l'option `GIVEOVER` pour basculer le service de données sur un autre noeud.

```
function restart_service
{

        # Pour redémarrer le service de données, vérifiez d'abord que
        # le service de données lui-même est toujours enregistré sous le PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Comme le TAG pour le service de données est toujours enregistré
                # sous le PMF, arrêtez d'abord le service de données, puis redémarrez-le.

                # Récupérez le nom de la méthode d'arrêt et la valeur STOP_TIMEOUT
                # pour cette ressource.
                STOP_TIMEOUT=`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

                # Récupérez le nom de la méthode de démarrage et la valeur START_TIMEOUT
                # pour cette ressource.
                START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
```

```
                START_METHOD=`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 du TAG pour le service de données
                # implique le service de données a déjà dépassé
                # le nombre maximum de tentatives autorisées sous le PMF.
                # Par conséquent, ne tentez pas de redémarrer le
                # service de données, mais essayez de le basculer
                # sur un autre noeud dans le cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}
```

État de la sonde à la fermeture

Le programme `PROBE` du service de données modèle se termine avec un code d'échec si toutes les tentatives de redémarrage local échouent et que la tentative de basculement sur un autre noeud échoue également. Ce programme consigne le message `Failover attempt failed`.

Fonctionnement de la méthode `Monitor_start`

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

Cette rubrique décrit les principaux éléments de la méthode `Monitor_start` pour l'application modèle. Cette rubrique ne décrit pas les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction `parse_args()`. Par ailleurs, elle ne décrit pas l'utilisation de la fonction `syslog()`. Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour obtenir une liste complète du code de la méthode `Monitor_start`, reportez-vous à la rubrique "Listing de code de la méthode `Monitor_start`" à la page 297.

Action de la méthode Monitor_start

Cette méthode utilise le PMF (pmfadm) pour démarrer 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 démarre la sonde en utilisant l'option "essayer indéfiniment" de pmfadm (-n -1, -t -1), ce qui signifie que si le démarrage de la sonde échoue, le PMF essaiera de la démarrer un nombre infini de fois sans limites de temps.

```
# 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‘

# Démarre la sonde pour le service de données sous le PMF. Utilise l’option
# "essayer indéfiniment" pour démarrer la sonde. Passe le nom de la ressource,
# son type et son groupe 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
```

Fonctionnement de la méthode Monitor_stop

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

Cette rubrique décrit les principaux éléments de la méthode Monitor_stop pour l'application modèle. Cette rubrique ne décrit pas les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction parse_args(). Par ailleurs, elle ne décrit pas l'utilisation de la fonction syslog(). Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour obtenir une liste complète du code de la méthode Monitor_stop, reportez-vous à la rubrique "Listing de code de la méthode Monitor_stop" à la page 299.

Action de la méthode Monitor_stop

Cette méthode utilise le PMF (pmfadm) pour vérifier si la sonde fonctionne et, dans ce cas, l'arrête.

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.

Attention – Veillez à utiliser le signal KILL avec pmfadm pour arrêter la sonde et non un signal susceptible d'être masqué, comme par exemple TERM. Sinon, la méthode Monitor_stop peut se bloquer indéfiniment et son délai d'attente expirer. En effet, la méthode PROBE appelle scha_control() lorsqu'il est nécessaire de redémarrer ou de basculer le service de données. Lorsque scha_control() appelle Monitor_stop pendant le processus de mise hors ligne du service de données, si Monitor_stop utilise un signal qui peut être masqué, Monitor_stop se bloque en attendant la fin de scha_control(), tandis que scha_control() se bloque en attendant la fin de Monitor_stop.

```
# Voir si le détecteur fonctionne et si oui, le tuer.
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 réussi du détecteur. Consigner un message.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor for resource " $RESOURCE_NAME \
            " successfully stopped"
   fi
fi
exit 0
```

État d'Arrêt_détecteur à la fermeture

La méthode Monitor_stop consigne un message d'erreur si elle ne peut pas arrêter la méthode PROBE. Le RGM place le service de données modèle en mode MONITOR_FAILED sur le noeud primaire, ce qui peut entraîner une erreur grave du noeud.

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

Fonctionnement de la méthode Monitor_check

Le RGM appelle la méthode Monitor_check lorsque la méthode de 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 Monitor_check pour l'application modèle. Cette rubrique ne décrit pas les fonctionnalités communes à toutes les méthodes de rappel, comme par exemple la fonction parse_args(). Par ailleurs, elle ne décrit pas l'utilisation de la fonction syslog(). Les fonctionnalités communes sont décrites à la rubrique "Fonctionnalité commune à toutes les méthodes" à la page 90.

Pour obtenir une liste complète du code de la méthode `Monitor_check`, reportez-vous à la rubrique "Listing de code de la méthode `Monitor_check`" à la page 301.

La méthode `Monitor_check` doit être mise en oeuvre de manière à ne pas entraîner de conflits avec d'autres méthodes qui sont exécutées simultanément.

La méthode `Monitor_check` appelle la méthode `Validate` pour vérifier que le répertoire de configuration du DNS est disponible sur le nouveau noeud. La propriété d'extension `Confdir` pointe vers le répertoire de configuration du DNS. Par conséquent, `Monitor_check` obtient le chemin d'accès et le nom pour la méthode `Validate`, ainsi que la valeur de `Confdir`. Elle transmet cette valeur à `Validation` , comme le montre l'affichage suivant :

```
# Obtenez le chemin d'accès complet pour la méthode
# Validate à partir de la propriété RT_basedir du type de ressource.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAMÈ

# Obtenez le nom de la méthode Validate pour cette ressource.
VALIDATE_METHOD=`scha_resource_get -O Validate \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtenez la valeur de la propriété Confdir afin de démarrer le service
# de données. Utilisez le nom de ressource et le groupe de ressource
# entrés pour obtenir la valeur de Confdir définie 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 pour les propriétés
# d'extension. Utilisez awk pour obtenir uniquement la valeur de la
# propriété d'extension.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Appelez la méthode validate afin que le service de données puisse
# être correctement basculé sur le nouveau noeud.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

Reportez-vous à la rubrique "Fonctionnement de la méthode `Validate`" à la page 111 pour comprendre comment l'application modèle vérifie si un noeud peut convenir pour l'hébergement du service de données.

Gestion des mises à jour des propriétés

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

Fonctionnement de la méthode `Validate`

Le RGM appelle la méthode de `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.

Le RGM n'appelle `Validate` que si l'administrateur du cluster modifie les propriétés de la ressource ou du groupe de ressources, et non lorsque le RGM définit des propriétés ou qu'un détecteur définit les propriétés de ressource `Status` ou `Status_msg`.

Remarque – la méthode `Monitor_check` appelle aussi explicitement la méthode `Validate` à chaque fois que la méthode `PROBE` tente de basculer le service de données sur un nouveau noeud.

Actions de la méthode `Validate`

Le RGM appelle `Validate` avec des arguments supplémentaires par rapport à ceux passés à d'autres méthodes, et notamment avec les propriétés et valeurs qui sont mises à jour. Par conséquent, cette méthode du service de données modèle doit mettre en oeuvre une fonction `parse_args()` différente pour gérer les arguments supplémentaires.

La méthode `Validate` vérifie une seule propriété, la propriété d'extension `Confdir`. 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 fonctionne, la propriété `Confdir` est déclarée `TUNABLE = AT_CREATION` dans le fichier RTR. La méthode `Validate` n'est donc jamais appelée pour vérifier la propriété `Confdir` après une mise à jour, mais seulement lors de la création de la ressource du service de données.

Si `Confdir` est l'une des propriétés que le RGM passe à `Validate`, la fonction `parse_args()` récupère et enregistre sa valeur. `Validate` vérifie que le répertoire désigné par la nouvelle valeur de `Confdir` est accessible et qu'il existe bien dans ce répertoire un fichier `named.conf` qui contient des données.

Si la fonction `parse_args()` ne peut récupérer la valeur de `Confdir` à partir des arguments de ligne de commande que lui transmet le RGM, `Validate` essaie malgré tout de valider la propriété `Confdir`. `Validate` utilise `scha_resource_get()` pour obtenir la valeur de `Confdir` à partir de la configuration statique. `Validate` effectue les mêmes contrôles pour vérifier que le répertoire de configuration est accessible et contient un fichier `named.conf` non vide.

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

Fonction d'analyse de la méthode `Validation`

Le RGM passant à la méthode `Validate` une série d'arguments différente de celle transmise aux autres méthodes de rappel, `Validate` requiert une fonction d'analyse des arguments différente de celle des autres méthodes. Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour obtenir plus d'informations sur les arguments passés à `Validate` et aux autres méthodes de rappel. L'exemple de code suivant illustre la fonction `parse_args()` de `Validate` .

```
#############################################################################
# Analyse les arguments de Validate.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Nom de la ressource DNS.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Nom du groupe de ressources dans lequel la
                  # ressource est configurée.
```

```
                RESOURCEGROUP_NAME=$OPTARG
                ;;
        T)
                # Nom du type de ressource.
                RESOURCETYPE_NAME=$OPTARG
                ;;
        r)
                # La méthode n'accède à aucune propriété
                # définie au niveau système, d'où l'opération ineffective
                ;;
        g)
                # La méthode n'accède à aucune propriété de
                # groupe de ressources, d'où l'opération ineffective
                ;;
        c)
                # Indique que la méthode Validate est appelée lors
                # de la création de la ressource, l'indicateur est donc ineffectif.
                ;;
        u)
                # Indique la mise à jour d'une propriété lorsque la
                # ressource existe déjà. Si la mise à jour concerne la
                # propriété Confdir, alors Confdir doit apparaître dans les
                # arguments de ligne de commande. Sinon, la méthode doit
                # spécifiquement le rechercher avec scha_resource_get.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Liste de propriétés d'extension. Séparer les paires 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 Confdir est trouvée sur la ligne
                # de commande, enregistrer 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()` pour d'autres méthodes, cette fonction fournit 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 la ressource qui est passée par le RGM.

L'indicateur `r` (qui indique une propriété définie par le système), l'indicateur `g` (qui indique une propriété de groupe de ressource) et l'indicateur `c` (qui indique que la validation est effectuée pendant la création de la ressource) sont ignorés, car cette méthode est appelée pour valider une propriété d'extension lorsque la ressource est mise à jour.

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

Validation de `Confdir`

Dans sa fonction `MAIN`, `Validate` définit d'abord la variable `CONFDIR` sur la chaîne vide et `UPDATE_PROPERTY` et `CONFDIR_FOUND` sur `0`.

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

`Validate` appelle `parse_args()` pour analyser les arguments passés par le RGM.

```
parse_args "$@"
```

`Validate` vérifie si `Validate` est appelée en conséquence d'une mise à jour des propriétés. `Validate` vérifie également si la propriété d'extension `Confdir` était sur la ligne de commande. `Validate` s'assure que la propriété `Confdir` a une valeur et, dans le cas contraire, se termine avec un état d'échec et 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

# Vérifie que la propriété Confdir a une valeur. Sinon, c'est un échec
# et la fonction se termine 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 – spécifiquement, le code précédent vérifie si `Validate` est appelée en conséquence d'une mise à jour (`$UPDATE_PROPERTY == 1`) et si la propriété n'a *pas* été trouvée sur la ligne de commande (`CONFDIR_FOUND == 0`). Dans ce cas, le code récupère la valeur existante de `Confdir` en utilisant `scha_resource_get()`. Si `Confdir` est présent sur la ligne de commande (`CONFDIR_FOUND == 1`), la valeur de `CONFDIR` est obtenue par la fonction `parse_args()`, non par `scha_resource_get()`.

La méthode `Validate` utilise la valeur de `CONFDIR` pour vérifier que le répertoire est accessible. Si le répertoire n'est pas accessible, `Validate` consigne un message d'erreur et se termine avec un état d'erreur.

```
# 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 le fichier n'est pas présent, la méthode consigne un message d'erreur et se termine avec un état d'erreur.

```
# Vérifier si le fichier named.conf est présent dans le répertoire Confdir
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 la vérification finale réussit, `Validate` consigne un message indiquant le succès et se termine avec un état de succès.

```
# Consigner un message indiquant que la méthode Validate 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 `Validate` se termine avec succès (`0`), `Confdir` est créé avec la nouvelle valeur. Si `Validate` se termine avec un échec (`1`), `Confdir` et les autres propriétés éventuelles ne sont pas créées et un message en indiquant la cause est généré.

Fonctionnement de la méthode `Update`

Le RGM exécute la méthode de `Mise_à_jour` pour notifier à une ressource en cours d'exécution que ses propriétés ont changé. Le RGM exécute `Update` après que l'administrateur du cluster ait fini de définir les propriétés d'une ressource ou de son groupe. Cette méthode est appelée sur les noeuds sur lesquels la ressource est en ligne.

Action de la méthode `Update`

La méthode `Update` ne met pas à jour les propriétés. C'est le rôle du RGM. La méthode `Update` notifie certains processus en cours d'exécution qu'une mise à jour a été effectuée. Le seul processus, dans le service de données modèle, affecté par une mise à jour de propriété est le détecteur de pannes. Par conséquent, la méthode `Update` arrête et redémarre le processus du détecteur de pannes.

La méthode `Update` doit vérifier que le détecteur de pannes est en cours d'exécution, puis le tuer à l'aide de la commande `pmfadm`. La méthode obtient l'emplacement du programme de sonde qui met en oeuvre le détecteur de pannes, et le redémarre par l'intermédiaire de la commande `pmfadm`.

Arrêt du détecteur à l'aide de la méthode `Update`

La méthode `Update` utilise `pmfadm -q` pour vérifier que le détecteur fonctionne et, si tel est le cas, le tue avec `pmfadm -s TERM`. Si le détecteur se termine correctement, un message en ce sens est envoyé à l'administrateur du cluster. Si le détecteur ne peut être arrêté, `Update` se termine avec un code d'échec et envoie un message d'erreur à l'administrateur du cluster.

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

# Tuer le détecteur déjà 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 détecteur réussi. 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 se trouve dans le répertoire de base du service de données, vers lequel pointe la propriété `RT_basedir`. `Update` récupère la valeur de `RT_basedir` et l'enregistre dans la variable `RT_BASEDIR`, de la manière suivante :

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

`Update` utilise la valeur de `RT_BASEDIR` avec `pmfadm` pour redémarrer le programme `dns_probe`. En cas de succès, `Update` se termine avec un code de succès et envoie un message en ce sens à l'administrateur du cluster. Si `pmfadm` ne parvient pas à démarrer le programme de sonde, `Update` se termine avec un code d'échec et consigne un message d'erreur.

État de `Update` à la fermeture

L'échec de la méthode `Update` entraîne le passage de son état à “update failed” (“échec de la mise à jour”). Cet état est sans effet sur la gestion de la ressource par le RGM, mais indique l'échec de l'action de mise à jour aux outils d'administration via la fonction `syslog()`.

CHAPITRE 6

Bibliothèque de développement de services de données

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

Ce chapitre contient les rubriques suivantes :

Présentation générale de la DSDL

L'API de la DSDL est mise en couche au-dessus de l'interface de programmation de l'application de gestion des ressources (APIGR). À ce titre, elle ne remplace pas l'APIGR mais l'encapsule et étend ses fonctionnalités. La DSDL 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 la majeure partie du temps de développement aux problèmes de haute disponibilité et d'évolutivité intrinsèques à votre application. De cette manière, vous perdez moins de temps à intégrer dans Sun Cluster les procédures de démarrage, d'arrêt et de surveillance de l'application.

Gestion des propriétés de configuration

Toutes les méthodes de rappel nécessitent l'accès aux propriétés de configuration. La DSDL 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 RGM transfère à la méthode de rappel, ce qui vous évite d'avoir à écrire une fonction d'analyse de la ligne de commande.
- Configure les structures de données internes que les autres fonctions DSDL 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 la même manière, les valeurs de la ligne de commande, qui sont prioritaires sur les valeurs issues du gestionnaire RGM, sont enregistrées dans ces structures de données.
- Initialise l'environnement de connexion et valide les paramètres de contrôle du détecteur de pannes.

Remarque – Pour la méthode `Validate`, `scds_initialize ()` analyse les valeurs de propriété qui sont transmises sur la ligne de commande, ce qui vous évite d'avoir à écrire une fonction d'analyse pour `Validate`.

La DSDL offre un ensemble de fonctions permettant de rechercher et d'extraire les propriétés de types de ressource, de ressources et de groupes de ressources, ainsi que les propriétés d'extension couramment utilisées. Ces fonctions normalisent l'accès aux propriétés à l'aide des conventions suivantes :

- Chaque fonction ne prend en charge qu'un argument de gestion (renvoyé par `scds_initialize()`).
- Chaque fonction correspond à une propriété particulière. Le type de valeur renvoyée de la fonction correspond au type de la valeur de propriété récupérée.
- Les fonctions ne renvoient pas d'erreurs, les valeurs ayant été précalculées par `scds_initialize()`. Les fonctions récupèrent les valeurs dans le RGM à moins qu'une nouvelle valeur soit passée sur la ligne de commande.

Démarrage et arrêt d'un service de données

Une méthode de `Start` effectue les opérations nécessaires pour démarrer un service de données sur un noeud de cluster. En général, ces opérations incluent l'extraction des propriétés de ressource, la localisation des fichiers exécutables et de configuration spécifiques à l'application, et le démarrage 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 de `Start` peut utiliser les fonctions de convenance des propriétés pour extraire les valeurs de propriétés spécifiques, telles que `Confdir_list`, qui permettent d'identifier les répertoires et les fichiers de configuration de l'application à lancer.

Une méthode de `Start` peut appeler `scds_pmf_start()` pour lancer une application sous le contrôle du gestionnaire de processus (PMF). Le PMF vous permet de spécifier le niveau de surveillance à appliquer au processus et permet de redémarrer le processus en cas d'échec. La section "Méthode `Démarrage_xfnts`" à la page 140 présente un exemple de méthode de `Start` mise en oeuvre avec la DSDL.

Une méthode d'`Stop` doit être idempotente pour que l'on puisse la fermer avec succès, même si elle est appelée sur un noeud lorsque l'application ne fonctionne pas. Si la méthode d'`Stop` échoue, la ressource qui est arrêtée passe à l'état `STOP_FAILED`, ce qui peut entraîner le cluster à effectuer un redémarrage brutal.

Pour éviter que la ressource ne passe à l'état `STOP_FAILED`, la méthode d' `Stop` doit tout mettre en oeuvre pour arrêter la ressource. La fonction `scds_pmf_stop()` permet une tentative d'arrêt de la ressource par phase. Elle tente d'abord de l'arrêter à l'aide d'un signal `SIGTERM` et, en cas d'échec, elle utilise un signal `SIGKILL`. Pour plus d'informations, reportez-vous à la page de manuel `scds_pmf_stop`(3HA).

Mise en oeuvre d'un détecteur de pannes

La DSDL simplifie grandement la mise en oeuvre d'un détecteur de pannes en fournissant un modèle prédéfini. Une méthode `Monitor_start` démarre le détecteur de pannes, sous le contrôle du gestionnaire de processus, lorsque la ressource démarre sur un noeud. Le détecteur de pannes fonctionne en boucle tant que la ressource est exécutée sur le noeud. La logique de haut niveau d'un détecteur de pannes de la DSDL est la suivante :

- La fonction `scds_fm_sleep()` utilise la propriété `Thorough_probe_interval` pour déterminer l'intervalle de temps entre les détections. Tout échec du processus de l'application détecté par le gestionnaire de processus au cours de cet intervalle entraîne un redémarrage de la ressource.
- La sonde elle-même renvoie une valeur qui indique la gravité des échecs : de `0` en l'absence d'échec à `100` en cas d'échec total.
- Cette valeur est envoyée à la fonction `scds_action()`, qui conserve un historique des échecs cumulatifs dans l'intervalle de la propriété `Retry_interval`.
- La fonction `scds_action()` indique ce qu'il faut faire en cas d'échec :
  - Si la gravité de l'échec cumulatif est inférieure à `100`, ne faites rien.
  - Si la valeur 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 redémarrages est supérieur à la valeur de la propriété `Retry_count` dans l'intervalle spécifié par `Retry_interval`, procédez au basculement du service de données.

Accès aux données d'adresse réseau

La DSDL dispose de fonctions de convenance pour renvoyer les données d'adresse réseau des ressources et des groupes de ressources. Par exemple, `scds_get_netaddr_list()` recherche et extrait les ressources d'adresse réseau utilisées par une ressource, permettant ainsi à un détecteur de pannes d'analyser l'application.

La DSDL comprend également un ensemble de fonctions pour la surveillance TCP. En général, ces fonctions établissent une connexion de prise simple à un service, lisent et écrivent des données sur ce service, et se déconnectent du service. Le résultat de la détection peut être envoyé à la fonction `scds_fm_action()` de la DSDL pour déterminer l'action à mettre en oeuvre.

La section "Méthode xfnts_validate" à la page 155 présente un exemple de détection TCP des pannes.

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

La DSDL offre des fonctions intégrées pour vous aider à déboguer votre service de données.

L'utilitaire DSDL `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 (nombre compris entre 1 et 9) peut être défini de manière dynamique pour chaque mise en oeuvre de type de ressource sur chaque noeud de cluster. Un fichier nommé `/var/cluster/rgm/rt/`*rtname*`/loglevel`, qui ne contient qu'un nombre entier compris entre 1 et 9, est lu par toutes les méthodes de rappel du type de ressource. La fonction DSDL `scds_initialize()` lit ce fichier et définit le niveau de débogage en interne au niveau spécifié. Le niveau 0 de débogage par défaut indique que le service de données ne consigne pas les messages de débogage.

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

Vous pouvez convertir certains messages de débogage en messages d'information pour les opérations courantes du type de ressource (notamment au niveau de la priorité `LOG_INFO`) à l'aide de la fonction `scds_syslog()`. Notez que l'exemple d'application DSDL du Chapitre 8 utilise librement les fonctions `scds_syslog_debug()` et `scds_syslog()`.

Activation des systèmes de fichiers locaux à haut niveau de disponibilité

Vous pouvez utiliser le type de ressource `HAStoragePlus` pour rendre un système de fichiers locaux hautement disponible au sein d'un environnement Sun Cluster. Les partitions du système de fichiers local doivent se trouver sur les groupes de disques globaux. Les commutations d'affinité doivent être activées, et l'environnement Sun Cluster doit être configuré de manière à prendre en charge le basculement. Cette

configuration permet à l'administrateur du cluster de rendre tout système de fichiers situé sur des disques multi-utilisateur accessible par tout hôte directement connecté à ces disques. 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 des informations sur la configuration du type de ressource `HAStoragePlus` dans la section "Enabling Highly Available Local File Systems" du *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

CHAPITRE 7

Conception des types de ressource

Ce chapitre explique l'usage habituel de la DSDL (Bibliothèque de développement de services de données) dans la conception et la mise en oeuvre des types de ressource. 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. Enfin, il explique comment utiliser la DSDL pour mettre en oeuvre les méthodes de rappel associées au type de ressource.

Reportez-vous à la page de manuel `rt_callbacks`(1HA) pour en savoir plus.

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

Ce chapitre contient les rubriques suivantes :

Fichier RTR (Resource Type Registration)

Le fichier Resource Type Registration (RTR) indique les détails concernant le type de ressource dans le logiciel Sun Cluster. Ces détails comprennent des informations telles que :

- les propriétés nécessaires à la mise en oeuvre
- les types de données et les valeurs par défaut de ces propriétés
- le chemin du système de fichiers pour les méthodes de rappel associées à la mise en oeuvre du type de ressource
- différents paramètres pour les propriétés définies par le système

L'exemple de fichier RTR fourni avec la DSDL est suffisant pour la plupart des mises en oeuvre de types de ressource. Vous devez seulement modifier 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 en oeuvre le type de ressource, vous pouvez la déclarer comme propriété d'extension dans le fichier RTR de la mise en oeuvre du type de ressource, et y accéder à l'aide de l'utilitaire `scds_get_ext_property()` de la DSDL.

Méthode `Validate`

L'objectif de la méthode de rappel `Validate` de la mise en oeuvre d'un type de ressource est de vérifier que les paramètres de ressource proposés (tels que spécifiés par les paramètres de propriété proposés pour la ressource) sont acceptables pour le type de ressource.

La méthode `Validate` de mise en oeuvre d'un type de ressource est appelée par le gestionnaire RGM (Resource Group Manager) dans l'un des deux cas suivants :

- création d'une ressource du même type ;
- 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 `Validate` de la ressource.

La méthode `Validate` est appelée sur chaque élément d'un jeu de noeuds défini par la valeur de la propriété `Init_nodes` du type de ressource. Si `Init_nodes` a la valeur `RG_PRIMARIES` , `Validate` est appelé sur chaque noeud pouvant héberger (en tant que primaire) le groupe contenant la ressource. Si `Init_nodes` a la valeur `RT_INSTALLED_NODES`, `Validate` est appelé sur chaque noeud où est installé le logiciel de type de ressource, généralement tous les noeuds du cluster.

La valeur par défaut de `Init_nodes` est `RG_PRIMARIES` (voir la page de manuel `rt_reg`(4)). Au moment où la méthode `Validate` est appelée, le gestionnaire 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 des propriétés actualisées (dans le cas d'un rappel de mise à jour).

Remarque – Si vous utilisez des systèmes de fichiers locaux gérés par le type de ressource `HAStoragePlus`, vous utilisez la fonction `scds_hasp_check()` pour vérifier l'état de ce type de ressource. Ces informations sont obtenues à partir de l'état (en ligne ou autre) de toutes les ressources `SUNW.HAStoragePlus` dont dépend la ressource, à l'aide des propriétés système `Resource_dependencies` ou `Resource_dependencies_weak` définies pour cette ressource. Reportez-vous à la page de manuel `scds_hasp_check`(3HA) pour obtenir une liste complète des codes d'état renvoyés par la fonction `scds_hasp_check ()`.

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

- Dans le cas d'une création de ressource, `scds_initialize()` analyse les propriétés proposées pour la ressource, telles qu'elles sont transmises sur la ligne de commande. Les valeurs proposées pour les propriétés de ressources sont donc disponibles comme si la ressource avait déjà été créée dans le système.
- Dans le cas d'une mise à jour de la ressource ou du groupe de ressources, les valeurs proposées pour les propriétés qui sont mises à jour par l'administrateur du cluster sont lues depuis la ligne de commande. Les autres propriétés (dont les valeurs ne sont pas actualisées) sont lues depuis Sun Cluster à l'aide de l'API de gestion des ressources. Si vous utilisez la DSDL, vous n'avez pas besoin de vous préoccuper de ces tâches. Vous pouvez valider une ressource comme si toutes les propriétés de la ressource étaient disponibles.

Supposons que la fonction qui met en oeuvre la validation des propriétés d'une ressource s'appelle `svc_validate()`, et qu'elle utilise la famille de fonctions `scds_get_`*`name`*`()` 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 `Validate` du type de ressource peut alors 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);   /* Initialization Error */
   }
   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. Toutefois, pour aller à l'essentiel (vous trouverez au Chapitre 8 des informations plus réalistes sur la fonction de validation), un exemple plus 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);   /* Invalid resource property setting */
   }
   return (0);   /* Acceptable setting */
}
```

Vous ne devez donc vous préoccuper que de la mise en oeuvre de la fonction `svc_validate()`.

Méthode de `Démarrage`

La méthode de rappel de `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 `Start` effectue 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 fichiers exécutables et/ou de configuration spécifiques à l'application, ainsi que le démarrage de l'application avec les arguments de ligne de commande appropriés.

Avec la DSDL, 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 `Start` est le suivant :

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

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Initialization Error */
```

```
    }
    if (svc_validate(handle) != 0) {
    return (1);   /* Invalid settings */
    }
    if (svc_start(handle) != 0) {
    return (1);   /* Start failed */
    }
    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, cela signifie soit que la configuration de la ressource et la configuration de l'application ne correspondent pas, soit qu'il y a un problème lié au système sur ce noeud de cluster. Par exemple, un système de fichiers de cluster requis par la ressource peut ne pas être disponible sur ce noeud de cluster. Dans ce cas, il est inutile d'essayer de démarrer la ressource sur ce noeud de cluster. Il est préférable de laisser le RGM tenter de démarrer la ressource sur un autre noeud.

Notez toutefois que l'instruction précédente considère que `svc_validate ()` est suffisamment conservatrice, c'est-à-dire qu'elle ne vérifie que les ressources du noeud de cluster qui sont absolument indispensables à l'application. Si ce n'était pas le cas, la ressource risquerait de ne pas pouvoir démarrer sur tous les noeuds de cluster et, par conséquent, prendrait l'état `START_FAILED`. Reportez-vous à la page de manuel `scswitch`(1M) et au *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour en savoir plus 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 renvoyer une valeur différente de zéro. Si cette fonction échoue, le RGM tente de démarrer la ressource sur un autre noeud du cluster.

Pour tirer parti au maximum de la DSDL, la fonction `svc_start()` peut appeler l'utilitaire `scds_pmf_start()` pour démarrer l'application sous le gestionnaire de processus (PMF). Cet utilitaire utilise également l'action de rappel d'échec du PMF pour détecter l'échec du processus. Pour en savoir plus, reportez-vous à la description de l'argument d'action `-a` à la page de manuel `pmfadm`(1M).

Méthode `Stop`

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 `Stop` exige que les conditions suivantes soient remplies :

- La méthode `Stop` doit être *idempotente* parce qu'elle peut être appelée par le gestionnaire RGM, même si la méthode `Start` n'a pas fonctionné correctement sur le noeud. Par conséquent, la méthode `Stop` doit aboutir (valeur zéro), même si

l'application n'est pas en cours d'exécution sur le noeud de cluster et qu'elle n'a aucune tâche à accomplir.

- Si la méthode `Stop` du type de ressource échoue (valeur différente de zéro) sur un noeud de cluster, la ressource qui est arrêtée passe à l'état `STOP_FAILED`. En fonction du paramètre `Failover_mode` de la ressource, ceci peut entraîner un redémarrage brutal du noeud de cluster par le RGM.

  Vous devez donc concevoir la méthode `Stop` de manière à ce qu'elle arrête vraiment l'application. Vous devrez peut-être recourir à `SIGKILL` pour fermer l'application de manière brutale si la procédure de fermeture normale échoue.

  Vous devez également vous assurer que cette méthode arrête l'application dans les temps, car la structure traite l'expiration de la propriété `Stop_timeout` comme un échec de l'arrêt et, par conséquent, fait passer la ressource à l'état `STOP_FAILED`.

L'utilitaire de la DSDL `scds_pmf_stop()` devrait suffire pour la plupart des applications, dans la mesure où il essaie d'abord d'arrêter l'application normalement avec `SIGTERM` . Cette fonction envoie ensuite un `SIGKILL` au processus. Elle considère que l'application a été démarrée par le gestionnaire de processus à l'aide de `scds_pmf_start()`. Pour en savoir plus sur cet utilitaire, reportez-vous à la section "Fonctions PMF" à la page 213.

En supposant que la fonction spécifique à l'application qui doit arrêter cette dernière s'appelle `svc_stop()`, mettez en oeuvre la méthode `Stop` de la manière suivante :

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

Que la mise en oeuvre de la fonction précédente `svc_stop()` inclue ou non la fonction `scds_pmf_stop()` n'a pas d'importance. Cela dépend d'un éventuel démarrage de l'application par le gestionnaire de processus via la méthode `Start`.

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

Méthode `Monitor_start`

Le RGM appelle la méthode `Monitor_start` pour lancer un détecteur de pannes pour la ressource. Les détecteurs de pannes contrôlent l'état de l'application qui est gérée par la ressource. Les mises en oeuvre de types de ressource mettent généralement en oeuvre un détecteur de pannes sur un démon séparé exécuté en arrière-plan. La méthode de rappel `Monitor_start` est utilisée pour démarrer ce démon avec les arguments appropriés.

Le démon du détecteur étant sujet aux échecs (il peut mourir par exemple, laissant l'application sans surveillance), vous devez utiliser le gestionnaire de processus pour lancer le démon du détecteur. L'utilitaire de la DSDL `scds_pmf_start()` dispose d'une prise en charge intégrée pour le démarrage des détecteurs de pannes. Il utilise le nom du chemin d'accès relatif de `RT_basedir` pour l'emplacement des mises en oeuvre de la méthode de rappel du type de ressource de ce démon de détecteur. Cet utilitaire utilise les propriétés d'extension `Monitor_retry_interval` et `Monitor_retry_count` gérées par la DSDL pour éviter des redémarrages illimités du démon.

Il impose également la même syntaxe de ligne de commande que celle définie pour toutes les méthodes de rappel (c'est-à-dire `-R` *resource* `-G` *resource-group* `-T` *resource-type*) sur le démon du détecteur, bien que celui-ci ne soit jamais appelé directement par le RGM. Enfin, l'utilitaire autorise également la mise en oeuvre du démon du détecteur pour permettre à l'utilitaire `scds_initialize()` de configurer son propre environnement. L'effort principal consiste à concevoir le démon du détecteur lui-même.

Méthode `Monitor_stop`

Le RGM appelle la méthode `Monitor_stop` pour arrêter le démon du détecteur de pannes qui a été démarré à l'aide de la méthode `Monitor_start`. L'échec de la méthode de rappel est considéré exactement comme un échec de la méthode `Stop`, c'est pourquoi la méthode `Monitor_stop` doit être idempotente et aussi robuste que la méthode `Stop`.

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 de `Monitor_check`

Le RGM exécute la méthode de rappel `Monitor_check` sur une ressource spécifiée d'un noeud pour vérifier si le noeud de cluster est capable de maîtriser la ressource. En d'autres termes, il exécute cette méthode pour savoir si l'application qui est gérée par la ressource peut fonctionner sur le noeud.

En général, dans cette situation on doit s'assurer que toutes les ressources système requises par l'application sont effectivement disponibles sur le noeud du cluster. Comme expliqué à la section "Méthode `Validate`" à la page 126, c'est la fonction `svc_validate()` que vous mettez en oeuvre qui permet de s'en assurer.

Selon l'application spécifique qui est gérée par la mise en oeuvre du type de ressource, il est possible de rédiger la méthode `Monitor_check` de manière à effectuer des tâches supplémentaires. La méthode `Monitor_check` doit être mise en oeuvre pour ne pas entrer en conflit avec les autres méthodes exécutées en même temps. Si vous utilisez la DSDL, la méthode `Monitor_check` doit appeler la fonction `svc_validate()`, qui met en oeuvre la validation spécifique à l'application des propriétés de la ressource.

Méthode de `Mise_à_jour`

Le RGM appelle la méthode `Update` de mise en oeuvre d'un type de ressource pour appliquer les modifications apportées par l'administrateur du cluster 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.

Les modifications qui viennent d'être apportées à la configuration de la ressource sont nécessairement acceptables pour la mise en oeuvre du type de ressource, car le RGM exécute la méthode `Validate` du type de ressource avant d'exécuter la méthode `Update` . La méthode `Validate` est appelée avant que les propriétés de la ressource ou du groupe de ressources soient modifiées, et la méthode `Validate` peut s'opposer aux modifications proposées. La méthode de `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.

Vous devez être prudent lorsque vous déterminez les propriétés que vous souhaitez pouvoir mettre à jour de manière dynamique, et que vous les marquez avec le paramètre `TUNABLE = ANYTIME` dans le fichier RTR. De manière générale, vous pouvez spécifier que vous souhaitez pouvoir mettre à jour de manière dynamique les propriétés relatives à la mise en oeuvre d'un type de ressource utilisées par le démon du détecteur de pannes. Toutefois, la mise en oeuvre de la méthode `Update` doit au moins redémarrer le démon du détecteur.

Les propriétés que vous pouvez utiliser sont les suivantes :

- `Thorough_probe_interval`
- `Retry_count`
- `Retry_interval`
- `Nombre_nouvelles_tentatives_détecteur;`
- `Intervalle_nouvelles_tentatives_détecteur;`
- `Délai_sonde`.

Ces propriétés affectent la manière dont un démon de détecteur de pannes vérifie l'état du service, la fréquence à laquelle il effectue ces vérifications, l'intervalle qu'il utilise pour effectuer le suivi des erreurs dans l'historique, et les seuils de redémarrage définis par le gestionnaire de processus. Pour mettre à jour ces propriétés, l'utilitaire `scds_pmf_restart()` est fourni avec la DSDL.

Si vous avez besoin de pouvoir mettre à jour de manière dynamique une propriété de ressource, mais que la modification de cette propriété risque d'affecter l'application en cours d'exécution, vous devez mettre en oeuvre les actions appropriées. Vous devez vous assurer que les mises à jour de cette propriété sont correctement appliquées aux instances en cours d'exécution de l'application. Actuellement, vous ne pouvez pas utiliser la DSDL pour mettre à jour de manière dynamique une propriété de ressource de cette manière. `Update` ne vous permet pas de transmettre les propriétés modifiées à la ligne de commande (contrairement à `Validate`).

Description des méthodes `Init`, `Fini` et `Boot`

Il s'agit de méthodes à *action unique*, tel que défini par les spécifications de l'API de gestion des ressources. L'exemple d'implémentation fourni avec la DSDL n'illustre pas l'utilisation de ces méthodes. Toutefois, si vous avez besoin de ces méthodes, elles disposent de toutes les fonctions de la DSDL. Généralement, les méthodes `Init` et `Boot` sont exactement les mêmes pour un type de ressource qui mettrait en oeuvre une *action unique*. Quant à la méthode `Fini`, elle exécute une opération qui *annule* l'action des méthodes `Init` ou `Boot`.

Conception du démon du détecteur de pannes

Les mises en oeuvre du type de ressource qui utilisent la DSDL possèdent généralement un démon de détecteur de pannes qui prend en charge les opérations suivantes :

- Contrôle régulier de l'état de l'application gérée. Cette tâche dépend de l'application et peut varier considérablement d'un type de ressource à un autre. La DSDL contient des fonctions d'utilitaire intégrées qui procèdent à des vérifications d'état pour des services TCP simples. Vous pouvez utiliser ces utilitaires pour mettre en oeuvre des applications qui utilisent des protocoles ASCII, tels que HTTP, NNTP, IMAP et POP3.
- Consignation des problèmes rencontrés par l'application à l'aide des propriétés de ressource `Retry_interval` et `Retry_count` . Lorsque l'application est en échec total, le détecteur de pannes doit déterminer si le script d'action du gestionnaire de processus doit redémarrer le service ou si les échecs de l'application se sont

accumulés si rapidement qu'il faut procéder à un basculement. Les utilitaires de la DSDL `scds_fm_action()` et `scds_fm_sleep()` sont destinés à vous aider à mettre en oeuvre ce mécanisme.

- Prise de mesures appropriées, soit en redémarrant l'application, soit en procédant à une tentative de basculement du groupe de ressources. L'utilitaire `scds_fm_action ()` de la DSDL met en oeuvre cet algorithme. Il calcule l'accumulation actuelle des échecs de sonde au cours du délai `Retry_interval` (en secondes) défini dans ce but.
- Mise à jour de l'état de la ressource, de manière à ce que la commande `scstat` et l'interface utilisateur graphique de gestion du cluster puissent connaître l'état de l'application.

Les utilitaires de la DSDL sont conçus de manière à ce que la boucle principale du démon du détecteur de pannes puisse être représentée par le pseudo code que vous trouverez à la fin de cette section.

Gardez les éléments suivants à l'esprit lorsque vous mettez en oeuvre un détecteur de pannes avec la DSDL :

- La mort du processus de l'application est détectée assez rapidement par la fonction `scds_fm_sleep()`, car sa notification par le gestionnaire de processus est asynchrone. Le temps de détection des pannes s'en trouve réduit de manière significative, ce qui augmente la disponibilité du service. Autrement, le détecteur de pannes risquerait de se réactiver régulièrement pour vérifier l'état d'un service et constaterait que le processus de l'application est mort.
- Si le RGM rejette la tentative de basculement du service à l'aide de l'API `scha_control`, la fonction `scds_fm_action()` *réinitialise*, ou oublie, son historique des pannes actuel. La raison en est que son historique dépasse déjà `Retry_count`. Si le démon du détecteur s'active au cours de l'itération suivante et est incapable de contrôler l'état du démon, il tente à nouveau d'appeler la fonction `scha_control()`. Cet appel est probablement rejeté une fois de plus, étant donné que la situation qui a entraîné son rejet au cours de la dernière itération perdure. La réinitialisation de l'historique garantit que le détecteur de pannes tente au moins de remédier à 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 échecs de l'application en cas d'échec du redémarrage, étant donné que l'utilisateur tente généralement d'utiliser `scha_control()` rapidement si la situation ne s'améliore pas d'elle-même.
- L'utilitaire `scds_fm_action()` met à jour l'état de la ressource en le faisant passer à `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_DEGRADED` ou `SCHA_RSSTATUS_FAULTED` selon l'historique des pannes. Cet état est donc disponible pour la gestion du système de cluster.

Dans la plupart des cas, le contrôle d'état spécifique à l'application peut être mis en oeuvre dans un utilitaire autonome distinct (`svc_probe()`, par exemple). Vous pouvez l'intégrer à cette boucle principale générique :

```
for (;;) {
   /* sleep for a duration of thorough_probe_interval between
   *  successive probes.
   */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));
   /* Now probe all ipaddress we use. Loop over
   * 1. All net resources we use.
   * 2. All ipaddresses in a given resource.
   * For each of the ipaddress that is probed,
   * compute the failure history.
   */
   probe_result = 0;
   /* Iterate through the all resources to get each
   * IP address to use for calling svc_probe()
   */
   for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
   /* Grab the hostname and port on which the
   * health has to be monitored.
   */
   hostname = netaddr->netaddrs[ip].hostname;
   port = netaddr->netaddrs[ip].port_proto.port;
   /*
   * HA-XFS supports only one port and
   * hence obtaint the port value from the
   * first entry in the array of ports.
   */
   ht1 = gethrtime();
   /* Latch probe start time */
   probe_result = svc_probe(scds_handle, hostname, port, timeout);
   /*
   * Update service probe history,
   * take action if necessary.
   * Latch probe end time.
   */
   ht2 = gethrtime();
   /* Convert to milliseconds */
   dt = (ulong_t)((ht2 - ht1) / 1e6);
   /*
   * Compute failure history and take
   * action if needed
   */
   (void) scds_fm_action(scds_handle,
   probe_result, (long)dt);
   }       /* Each net resource */
   }       /* Keep probing forever */
```

CHAPITRE 8

Mise en oeuvre du type de ressource DSDL modèle

Ce chapitre décrit un exemple de type de ressource, `SUNW.xfnts`, qui est mis en oeuvre à l'aide de la DSDL (Bibliothèque de développement de services de données). Ce service de données est rédigé en C. L'application sous-jacente est le serveur X Font, un service basé sur le protocole TCP/IP. Vous trouverez le code complet de chaque méthode du type de ressource `SUNW.xfnts` à l'Annexe C.

Ce chapitre contient les rubriques suivantes :

Serveur X Font

Le serveur X Font est un service basé sur le protocole TCP/IP qui fournit 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 dans le

fichier /usr/openwin/bin/xfs. Il est normalement démarré à partir de inetd. Toutefois, dans l'exemple actuel, considérez que l'entrée appropriée du fichier /etc/inetd.conf a été désactivée (par exemple, à l'aide de la commande fsadmin -d) et que le démon est donc sous le seul contrôle du logiciel 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 polices que peut fournir le démon. L'administrateur du cluster peut placer les répertoires de police sur le système de fichiers du cluster. Cet emplacement optimise l'utilisation du serveur X Font sur Sun Cluster, car une seule copie de la base de données des polices est gérée sur le système. Si l'administrateur du cluster souhaite modifier l'emplacement, il doit modifier fontserver.cfg en indiquant les nouveaux chemins d'accès aux répertoires de police.

Pour faciliter la configuration, il peut également placer le fichier de configuration dans le système de fichiers du cluster. Le démon xfs propose des arguments de ligne de commande qui remplacent 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 du logiciel Sun Cluster.

/usr/openwin/bin/xfs -config *location-of-configuration-file*/fontserver.cfg \
-port *port-number*

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

Numéro du port TCP

Le numéro du port TCP sur lequel écoute le démon du serveur xfs est normalement le port "fs", (généralement défini comme 7100 dans le fichier /etc/services). Toutefois, l'option -port ajoutée à la commande xfs par l'administrateur du cluster permet à ce dernier d'ignorer la valeur par défaut.

Vous pouvez utiliser la propriété Port_list du type de ressource SUNW.xfnts pour définir la valeur par défaut et permettre à l'administrateur du cluster d'utiliser l'option -port avec la commande xfs. Vous définissez la valeur par défaut de cette propriété comme 7100/tcp dans le fichier RTR. Dans la méthode SUNW.xfnts Start, vous transmettez Port_list 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). L'administrateur du cluster peut spécifier une valeur différente pour la propriété Port_list lorsqu'il configure le type de ressource.

Fichier RTR SUNW.xfnts

Cette section décrit plusieurs propriétés clés du fichier RTR SUNW.xfnts. Elle ne décrit pas l'objectif de chaque propriété du fichier. En revanche, vous trouverez cette description à la section "Paramétrage des propriétés de ressources et de types de ressources" à la page 35.

La propriété d'extension Confdir_list identifie le répertoire de configuration (ou une liste de répertoires), de la manière suivante :

```
{
        PROPRIÉTÉ = Confdir_list;
        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 spécifier un nom de répertoire au moment de la création de la ressource. Cette valeur ne pourra pas être modifiée ultérieurement car son paramétrage est limité à AT_CREATION .

La propriété Port_list identifie le port sur lequel le démon du serveur écoute, de la manière suivante :

```
{
        PROPERTY = Port_list;
        DEFAULT = 7100/tcp;
        TUNABLE = ANYTIME;
}
```

Étant donné que la propriété déclare une valeur par défaut, l'administrateur du cluster peut spécifier une nouvelle valeur ou accepter la valeur par défaut au moment de la création de la ressource. Cette valeur ne pourra pas être modifiée ultérieurement car son paramétrage est limité à AT_CREATION .

Conventions d'attribution de noms pour les fonctions et les méthodes de rappel

Vous pouvez identifier les différentes parties de l'exemple de code grâce aux conventions suivantes :

- Les fonctions APIGR commencent par scha_.

- Les fonctions DSDL commencent par `scds_`.
- Les méthodes de rappel commencent par `xfnts_`.
- Les fonctions rédigées par l'utilisateur commencent par `svc_`.

Fonction `scds_initialize()`

La DSDL requiert que chaque méthode de rappel appelle la fonction `scds_initialize ()` au début de la méthode. Cette fonction exécute les opérations suivantes :

- Contrôle et traite les arguments de ligne de commande (`argc` et `argv`) que la structure transmet à la méthode de service de données. La méthode n'a pas besoin de traiter d'autres arguments de ligne de commande.
- Définit les structures de données internes pour une utilisation par les autres fonctions dans DSDL.
- Initialise l'environnement de consignation.
- Valide les paramètres de sondage du détecteur de pannes.

Utilisez la fonction `scds_close()` pour récupérer les ressources affectées par `scds_initialize()`.

Méthode `Démarrage_xfnts`

Le RGM exécute la méthode `Start` sur un noeud de cluster lorsque le groupe de ressources qui contient la ressource du service de données est mis en ligne sur ce noeud ou lorsque la ressource est activée. Dans l'exemple de type de ressource `SUNW.xfnts`, la méthode `xfnts_start` active le démon `xfs` sur ce noeud.

La méthode `xfnts_start` appelle la fonction `scds_pmf_start()` pour démarrer le démon sous le gestionnaire de processus (PMF). Le PMF envoie une notification d'échec automatique et redémarre les fonctionnalités, et il s'intègre au détecteur de pannes.

Remarque – Le premier appel de `xfnts_start` est destiné à `scds_initialize ()`, qui effectue certaines tâches de *gestion interne* nécessaires. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_initialize`(3HA) contiennent plus de détails.

Validation du service avant le démarrage du serveur X Font

Avant d'essayer de démarrer le serveur X Font, la méthode `xfnts_start` appelle `svc_validate()` pour vérifier si une configuration correcte est en place pour prendre en charge le démon `xfs`.

```
rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }
```

Reportez-vous à la section "Méthode `xfnts_validate`" à la page 155 pour plus de détails.

Démarrage du service avec `svc_start()`

La méthode `xfnts_start` appelle la méthode `svc_start()`, qui est définie dans le fichier `xfnts.c`, pour démarrer le démon `xfs`. Cette rubrique décrit `Démarrage_svc()`.

La commande permettant de démarrer le démon `xfs` est la suivante :

```
# xfs -config config-directory/fontserver.cfg -port port-number
```

La propriété d'extension `Confdir_list` identifie le *config-directory* alors que la propriété système `Port_list` identifie le *port-number*. L'administrateur du cluster fournit à ces propriétés des valeurs spécifiques lorsqu'il configure le service de données.

La méthode `xfnts_start` déclare ces propriétés en tant que tableaux de chaînes de caractères. Elle prend les valeurs que l'administrateur du cluster définit à l'aide des fonctions `scds_get_ext_confdir_list()` et `scds_get_port_list()`. Ces fonctions sont décrites dans la page de manuel `scds_property_functions`(3HA).

```
scha_str_array_t *confdirs;
scds_port_list_t    *portlist;
scha_err_t   err;

   /* obtention du répertoire de configuration depuis la propriété Confdir_list */
   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);
}
```

Notez que la variable `confdirs` pointe vers le premier élément (`0`) du tableau.

La méthode `xfnts_start` utilise la fonction `sprintf()` pour former la ligne de commande de `xfs`.

```
/* Construct the command to start the xfs daemon. */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);
```

Notez que la sortie est redirigée vers `/dev/null` pour supprimer les messages générés par le démon.

La méthode `xfnts_start` transmet la ligne de commande `xfs` à `scds_pmf_start()` pour démarrer le service de données sous le contrôle du gestionnaire de processus.

```
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 ");
   }
```

Notez les points suivants en ce qui concerne l'appel destiné à `scds_pmf_start()` :

- L'argument `SCDS_PMF_TYPE_SVC` identifie le programme à démarrer en tant qu'application de service de données. Cette méthode peut également lancer un détecteur de pannes ou un autre type d'application.
- L'argument `SCDS_PMF_SINGLE_INSTANCE` l'identifie comme une ressource à instance unique.
- L'argument `cmd` est la ligne de commande qui a été générée précédemment.
- Le dernier argument, `-1`, indique le niveau de surveillance des processus enfants. La valeur `-1` indique que le gestionnaire de processus surveille tous les processus enfants ainsi que le processus d'origine.

Avant de renvoyer un message, `svc_pmf_start()` libère la mémoire allouée pour la structure `portlist`.

```
scds_free_port_list(portlist);
return (err);
```

Retour de `svc_start()`

Même lorsque `svc_start()` renvoie un message de réussite, il se peut que l'application sous-jacente n'ait pas réussi 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 également tenir compte du fait que l'application peut ne pas être disponible immédiatement étant donné que son démarrage prend un certain temps. La méthode `svc_start()` appelle la fonction `svc_wait()`, qui est définie dans le fichier `xfnts.c`, pour vérifier que l'application est en cours d'exécution.

```
/* Wait for the service to start up fully */
   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 ()` pour obtenir les ressources d'adresse réseau nécessaires pour sonder l'application.

```
/* obtain the network resource to use for probing */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

La fonction `svc_wait()` prend les valeurs `Start_timeout` et `Stop_timeout`.

```
svc_start_timeout = scds_get_rs_start_timeout(scds_handle)
   probe_timeout = scds_get_ext_probe_timeout(scds_handle)
```

Pour prendre en compte le temps qu'il faut au serveur pour démarrer, `svc_wait()` appelle `scds_svc_wait()` et transmet une valeur de délai d'attente équivalente à trois pour cent de la valeur `Start_timeout`. La fonction `svc_wait()` appelle la fonction `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. Si elle n'arrive pas à se connecter au port, elle renvoie une valeur de 100, ce qui indique un échec total. Si la connexion se fait 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 `Retry_count` de la ressource dans la période spécifiée par la propriété `Retry_interval` de la ressource, la fonction `scds_svc_wait()` renvoie un échec. Dans ce cas, la fonction `svc_start()` renvoie également un échec.

```
#define    SVC_CONNECT_TIMEOUT_PCT    95
#define    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 {
      /*
       * probe the data service on the IP address of the
       * network resource and the portname
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Success. Free up resources and return */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

       /* Call scds_svc_wait() so that if service fails too
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }

   /* Rely on RGM to timeout and terminate the program */
   } while (1);
```

Remarque – Avant de quitter, la méthode `xfnts_start` appelle `scds_close()` pour récupérer les ressources affectées par `scds_initialize ()`. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_close`(3HA) contiennent plus de détails.

Méthode `Arrêt_xfnts`

La méthode `xfnts_start` utilisant la fonction `scds_pmf_start()` pour démarrer le service sous le gestionnaire de processus, `xfnts_stop` utilise la fonction `scds_pmf_stop()` pour l'arrêter.

Remarque – Le premier appel d'`xfnts_stop` est destiné à la fonction `scds_initialize ()`, qui effectue certaines tâches de *gestion interne* nécessaires. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_initialize`(3HA) contiennent plus de détails.

La méthode `xfnts_stop` appelle la méthode `svc_stop()`, qui est définie dans le fichier `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); /* Successfully stopped */
```

Notez les points suivants concernant l'appel de `svc_stop()` à la fonction `scds_pmf_stop()` :

- L'argument `SCDS_PMF_TYPE_SVC` identifie le programme à arrêter en tant qu'application de service de données. Cette méthode peut également arrêter un détecteur de pannes ou certains autres types d'application.
- L'argument `SCDS_PMF_SINGLE_INSTANCE` identifie le signal.

- L'argument `SIGTERM` identifie le signal à utiliser pour arrêter l'instance de 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 obtenir plus de détails.
- La valeur du délai imparti correspond à la propriété `Délai_arrêt` de la ressource.

Remarque – Avant de quitter, la méthode `xfnts_stop` appelle `scds_close()` pour récupérer les ressources allouées par `scds_initialize ()`. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_close`(3HA) contiennent plus de détails.

Méthode de `démarrage_détecteur_xfnts`

Le RGM appelle la méthode `Monitor_start` sur un noeud pour démarrer le détecteur de pannes après le démarrage d'une ressource sur ce noeud. La méthode `xfnts_monitor_start` utilise `scds_pmf_start()` pour démarrer le démon du détecteur sous le gestionnaire de processus.

Remarque – Le premier appel de `xfnts_monitor_start` est destiné à `scds_initialize()`, qui effectue certaines tâches de *gestion interne* nécessaires. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_initialize`(3HA) contiennent plus de détails.

La méthode `xfnts_monitor_start` appelle la méthode `mon_start` , qui est définie dans le fichier `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));

    /* Call scds_pmf_start and pass the name of the probe. */
   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); /* Successfully started Monitor */
}
```

Notez les points suivants concernant l'appel de `svc_mon_start()` destiné à la fonction `scds_pmf_start()` :

- L'argument `SCDS_PMF_TYPE_MON` identifie le programme à démarrer en tant que détecteur de pannes. Cette méthode peut également démarrer un service de données ou certains autres types d'application.
- L'argument `SCDS_PMF_SINGLE_INSTANCE` l'identifie comme une ressource à instance unique.
- L'argument `xfnts_probe` identifie le démon du détecteur à démarrer. Le démon du détecteur est supposé se trouver dans le même répertoire que les autres programmes de rappel.
- Le dernier argument, `0`, indique le niveau de surveillance des processus enfants. Dans ce cas, cette valeur indique que le gestionnaire de processus surveille uniquement le démon du détecteur.

Remarque – Avant de quitter, la méthode `xfnts_monitor_start` appelle `scds_close()` pour récupérer les ressources affectées par `scds_initialize ()`. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_close`(3HA) contiennent plus de détails.

Méthode `Arrêt_détecteur_xfnts`

La méthode `xfnts_monitor_start` utilisant `scds_pmf_start()` pour démarrer le démon du détecteur sous le gestionnaire de processus, `xfnts_monitor_stop` utilise `scds_pmf_stop()` pour arrêter le démon du détecteur.

Remarque – Le premier appel de `xfnts_monitor_stop` est destiné à `scds_initialize()`, qui effectue certaines tâches de *gestion interne* nécessaires. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_initialize`(3HA) contiennent plus de détails.

La méthode `xfnts_monitor_stop()` appelle la méthode `mon_stop` , qui est définie dans le fichier `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); /* Successfully stopped monitor */
}
```

Notez les points suivants concernant l'appel de `svc_mon_stop()` destiné à la fonction `scds_pmf_stop()` :

- L'argument `SCDS_PMF_TYPE_MON` identifie le programme à arrêter en tant que détecteur de pannes. Cette méthode peut également arrêter un service de données ou certains autres types d'application.
- L'argument `SCDS_PMF_SINGLE_INSTANCE` l'identifie en tant que ressource à instance unique.
- L'argument `SIGKILL` identifie le signal à utiliser pour arrêter l'instance de 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 plus de détails.
- La valeur de délai d'attente est celle de la propriété `Monitor_stop_timeout` de la ressource.

Remarque – Avant de quitter, la méthode `xfnts_monitor_stop` appelle `scds_close()` pour récupérer les ressources affectées par `scds_initialize ()`. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_close`(3HA) contiennent plus de détails.

Méthode `xfnts_monitor_check`

Le RGM appelle la méthode `Monitor_check` à chaque tentative du détecteur de pannes de basculer le groupe contenant la ressource sur un autre noeud. La méthode `xfnts_monitor_check` appelle la méthode `svc_validate()` pour vérifier qu'une

configuration correcte est en place pour prendre en charge le démon `xfs`. Reportez-vous à la section "Méthode `xfnts_validate`" à la page 155 pour en savoir plus. Le code de `xfnts_monitor_check` est le suivant :

```
/* Process the arguments passed by RGM and initialize 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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method run as part of monitor check */
   return (rc);
}
```

Détecteur de pannes `SUNW.xfnts`

Le RGM n'appelle pas directement la méthode `PROBE` : il appelle d'abord la méthode `Monitor_start` pour démarrer le détecteur après le démarrage d'une ressource sur un noeud. La méthode `xfnts_monitor_start` démarre le détecteur de pannes sous le contrôle du gestionnaire de processus. La méthode `xfnts_monitor_stop` arrête le détecteur de pannes.

Le détecteur de pannes `SUNW.xfnts` effectue les opérations suivantes :

- Il surveille régulièrement l'état du démon du serveur `xfs` à l'aide d'utilitaires spécialement conçus pour vérifier des services TCP simples, tels que `xfs`.
- Il consigne les problèmes rencontrés par l'application dans une fenêtre de temps (à l'aide des propriétés `Retry_count` et `Retry_interval`) et décide de redémarrer ou de basculer le service de données si l'application échoue complètement. Les fonctions `scds_fm_action()` et `scds_fm_sleep()` fournissent une assistance intégrée à ce mécanisme de suivi et de décision.
- Il applique la décision de basculement ou de redémarrage à l'aide de la fonction `scds_fm_action ()`.
- Il met à jour l'état de la ressource et le met à disposition des outils administratifs et des interfaces utilisateur graphiques.

Boucle principale xfonts_probe

La méthode xfonts_probe met en oeuvre une boucle. Mais avant cela, xfonts_probe effectue les opérations suivantes :

- Elle récupère les ressources d'adresse réseau de la ressource xfnts de la manière suivante :

```
/* Get the ip addresses available for this resource */
   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);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }
```

- Elle appelle scds_fm_sleep() et transmet la valeur de Thorough_probe_interval en tant que valeur de délai d'attente. La sonde passe en mode veille pendant la durée de l'Thorough_probe_interval entre les détections, de la manière suivante :

```
timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {
      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));
```

La méthode xfnts_probe met en oeuvre la boucle suivante :

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS supports only one port and
          * hence obtain the port value from the
          * first entry in the array of ports.
          */
         ht1 = gethrtime(); /* Latch probe start time */
         scds_syslog(LOG_INFO, "Probing the service on port: %d.", port);
```

```
        probe_result =
        svc_probe(scds_handle, hostname, port, timeout);

        /*
         * Update service probe history,
         * take action if necessary.
         * Latch probe end time.
         */
        ht2 = gethrtime();

        /* Convert to milliseconds */
        dt = (ulong_t)((ht2 - ht1) / 1e6);

        /*
         * Compute failure history and take
         * action if needed
         */
        (void) scds_fm_action(scds_handle,
            probe_result, (long)dt);
      }   /* Each net resource */
   }    /* Keep probing forever */
```

La fonction `svc_probe()` met la logique de sondage en oeuvre. La valeur de retour de `svc_probe()` est transmise à `scds_fm_action ()`, qui détermine s'il faut redémarrer l'application, basculer le groupe de ressources, ou ne rien faire du tout.

Fonction `svc_probe()`

La fonction `svc_probe ()` établit une connexion de prise simple au port spécifié en appelant `scds_fm_tcp_connect()`. Si la connexion échoue, `svc_probe()` renvoie une valeur de `100`, ce qui indique un échec total. Si la connexion est établie mais que la déconnexion échoue, `svc_probe()` renvoie une valeur de `50`, ce qui indique un échec partiel. Si la connexion et la déconnexion fonctionnent, `svc_probe()` renvoie une valeur de `0`, ce qui indique 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;


   /*
    * probe the data service by doing a socket connection to the port
    * specified in the port_list property to the host that is
```

```
 * serving the XFS data service. If the XFS service which is configured
 * to listen on the specified port, replies to the connection, then
 * the probe is successful. Else we will wait for a time period set
 * in probe_timeout property before concluding that the probe failed.
 */

/*
 * Use the SVC_CONNECT_TIMEOUT_PCT percentage of timeout
 * to connect to the port
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
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));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Compute the actual time it took to connect. This should be less than
 * or equal to connect_timeout, the time allocated to connect.
 * If the connect uses all the time that is allocated for it,
 * then the remaining value from the probe_timeout that is passed to
 * this function will be used as disconnect timeout. Otherwise, the
 * the remaining time from the connect call will also be added to
 * the disconnect timeout.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Use the remaining time(timeout - time_took_to_connect) to disconnect
 */

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
 */
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;
}

/*
 * Return partial failure in case of disconnection failure.
 * Reason: The connect call is successful, which means
 * the application is alive. A disconnection failure
 * could happen due to a hung application or heavy load.
 * If it is the later case, don't declare the application
 * as dead by returning complete failure. Instead, declare
 * it as partial failure. If this situation persists, the
 * disconnect call will fail again and the application will be
 * restarted.
 */
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));
    /* this is a partial failure */
    return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

t2 = (hrtime_t)(gethrtime()/1E9);
time_used = (int)(t2 - t1);
time_remaining = timeout - time_used;

/*
 * If there is no time left, don't do the full test with
 * fsinfo. Return SCDS_PROBE_COMPLETE_FAILURE/2
 * instead. This will make sure that if this timeout
 * persists, server will be restarted.
 */
if (time_remaining <= 0) {
    scds_syslog(LOG_ERR, "Probe timed out.");
    return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

/*
 * The connection and disconnection to port is successful,
 * Run the fsinfo command to perform a full check of
 * server health.
 * Redirect stdout, otherwise the output from fsinfo
 * ends up on the 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);
}
```

Ensuite, `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()`.

Détermination de l'action du détecteur de pannes

La méthode `xfnts_probe` appelle `scds_fm_action()` pour décider de la mesure à prendre. La logique de `scds_fm_action()` est la suivante :

- Gérez un historique des pannes cumulées dans la valeur de la propriété `Retry_interval`.
- 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 redémarrages dépasse la valeur de la propriété `Retry_count` dans l'intervalle spécifié par `Retry_interval`, basculez le service de données.

Par exemple, supposons que la sonde établisse une connexion au serveur `xfs`, mais qu'elle 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. L'échec de la déconnexion entraîne l'envoi d'un échec partiel (`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, au cours de la détection suivante, le serveur n'arrive toujours pas à se déconnecter, une valeur de `50` est ajoutée à l'historique des pannes géré par `scds_fm_action ()`. La valeur des pannes cumulées est à présent de `100`. Par conséquent, `scds_fm_action()` redémarre le service de données.

Méthode xfnts_validate

Le RGM appelle la méthode Validate quand une ressource est créée et quand un administrateur de cluster met à jour les propriétés de la ressource ou de son groupe. 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.

Le RGM n'appelle Validate que lorsqu'un administrateur de cluster modifie les propriétés de la ressource ou du groupe de ressources, ou lorsqu'un dispositif de surveillance définit les propriétés Status et Status_msg de la ressource. Le RGM n'appelle pas Validate lorsque le RGM définit les propriétés.

Remarque – La méthode Monitor_check appelle également explicitement la méthode Validate à chaque fois que la méthode PROBE tente de basculer le service de données sur un autre noeud.

Le RGM appelle Validate avec des arguments supplémentaires, en plus de ceux qui sont transmis à d'autres méthodes, dont les propriétés et les valeurs qui sont mises à jour. L'appel destiné à scds_initialize() au début de xfnts_validate analyse tous les arguments que le RGM transmet à xfnts_validate et enregistre les informations dans l'argument scds_handle. Les sous-routines appelées par xfnts_validate utilisent ces informations.

La méthode xfnts_validate appelle svc_validate(), qui vérifie les conditions suivantes :

- 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);

/* Return error if there is no confdir_list extension property */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }
```

- Le répertoire spécifié par Confdir_list contient le fichier fontserver.cfg.

```
(void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suppress lint error because errno.h prototype
```

```
         * is missing void arg
         */
        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 `Port_list` 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); /* Validation Failure */
    }

#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); /* Validation Failure */
    }
#endif
```

- Le groupe de ressources qui contient 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); /* Validation Failure */
    }

    /* Return an error if there are no network address resources */
    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); /* return result of validation */
```

Remarque – Avant de quitter, la méthode `xfnts_validate` appelle `scds_close()` pour récupérer les ressources affectées par `scds_initialize ()`. Les pages de manuel "Fonction `scds_initialize()`" à la page 140 et `scds_close`(3HA) contiennent plus de détails.

Méthode `Mise_à_jour_xfnts`

Le RGM appelle la méthode `Update` pour notifier une ressource en cours d'exécution que ses propriétés ont été modifiées. Les seules propriétés pouvant être modifiées par le service de données `xfnts` concernent le détecteur de pannes. Par conséquent, chaque fois qu'une propriété est mise à jour, la méthode `xfnts_update` appelle `scds_pmf_restart_fm()` pour redémarrer le détecteur de pannes.

```
  /* check if the Fault monitor is already running and if so stop
   * and restart it. The second parameter to scds_pmf_restart_fm()
   * uniquely identifies the instance of the fault monitor that needs
   * to be restarted.
   */

   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.");
      /* Free up all the memory allocated by scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
   "Completed successfully.");
```

Remarque – Le second argument destiné à `scds_pmf_restart_fm()` identifie uniquement l'instance du détecteur de pannes à redémarrer s'il y en a plusieurs. La valeur `0` dans l'exemple indique qu'il n'y a qu'une seule instance du détecteur de pannes.

CHAPITRE 9

SunPlex Agent Builder

Ce chapitre décrit SunPlex Agent Builder et le module Cluster Agent pour Agent Builder. Ces deux outils automatisent la création de types de ressources ou de services de données, qui fonctionnent sous le contrôle du Gestionnaire de groupe de ressources (RGM). Un *type de ressource* est un wrapper autour d'une application, qui lui permet de fonctionner dans un environnement clusterisé, sous le contrôle du RGM.

Ce chapitre contient les rubriques suivantes :

Présentation d'Agent Builder

Agent Builder fournit une interface utilisateur graphique (GUI) qui vous permet de préciser des informations concernant des applications et le type de ressource que vous souhaitez créer. Agent Builder prend en charge aussi bien des applications reconnaissant le réseau que celles ne le reconnaissant pas. *Les premières* utilisent le réseau pour communiquer avec les clients. *Les secondes* sont des applications autonomes.

Remarque – Si la version de l'interface utilisateur graphique de Agent Builder n'est pas accessible, vous pouvez accéder à Agent Builder via l'interface de ligne de commande. Voir "Utilisation de la version de ligne de commande d'Agent Builder " à la page 177.

Selon les informations que vous spécifiez, Agent Builder génère les logiciels suivants :

- Un ensemble d'interpréteurs de commande C ou korn shell (`ksh`), ou de fichiers source de services de données génériques (GDS) pour un type de ressource de basculement ou évolutif qui correspond aux rappels de méthode du type de ressource. Ces fichiers sont destinés aussi bien aux applications qui reconnaissent le réseau (modèle serveur-client) qu'à celles qui ne le reconnaissent pas (sans clients).
- Un fichier d'enregistrement de type de ressource (RTR) personnalisé (si vous générez du code source en C ou korn shell).
- Des scripts d'utilitaire personnalisés pour lancer, arrêter et supprimer une instance (ressource) du type de ressource, ainsi que des pages de manuel personnalisées qui expliquent comment utiliser chacun de ces fichiers.
- Un package Solaris comprenant les codes binaires (si vous générez du code source C), un fichier RTR (si vous générez du code source en C ou en korn shell) et les scripts d'utilitaire.

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.

Avant d'utiliser Agent Builder

Avant d'utiliser Agent Builder, vous devez savoir comment créer des types de ressource avec plusieurs arborescences de processus indépendantes.

Agent Builder peut créer des types de ressource pour des applications comportant plusieurs arborescences de processus indépendantes. Ces arborescences sont indépendantes dans la mesure où la fonction PMF les surveille et les démarre individuellement, et ce avec leur propre balise.

Remarque – Agent Builder vous permet de créer des types de ressources avec plusieurs arborescences de processus indépendantes, uniquement si le code source généré que vous indiquez est de type C ou GDS. Vous ne pouvez pas utiliser Agent Builder pour créer ces types de ressources avec le code source korn shell. Pour créer ces types de ressource pour le korn shell, 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 pas contenir de lignes vides. Vous devez le spécifier dans le champ Commande de démarrage de l'écran de configuration de Agent Builder.

Vérifiez que le fichier ne contient aucun droit d'exécution pour que Agent Builder puisse le reconnaître. Ce fichier permet de démarrer plusieurs arborescences de processus à partir d'un seul script exécutable contenant plusieurs commandes. Si ce fichier texte dispose de droits d'exécution, les ressources s'affichent correctement sur le cluster, mais toutes les commandes sont exécutées sous une seule balise PMF. La fonction PMF est alors incapable de surveiller et de redémarrer les arborescences de processus individuellement.

Utilisation d'Agent Builder

Cette section explique comment utiliser Agent Builder. Elle indique aussi les tâches que vous devez effectuer avant de pouvoir utiliser Agent Builder. Vous y découvrirez également comment profiter de cet outil après avoir généré le code du type de ressources.

Les rubriques traitées sont les suivantes :

Analyse de l'application

Avant d'utiliser Agent Builder, vous devez déterminer si l'application que vous souhaitez rendre hautement disponible ou évolutive répond aux critères requis. Agent Builder ne peut pas réaliser cette analyse qui s'appuie uniquement sur les caractéristiques du temps d'exécution de l'application. Pour plus d'informations à ce sujet, voir "Analyse du caractère approprié de l'application" à la page 29.

Agent Builder ne sera peut-être pas toujours en mesure de créer un type de ressource complet pour votre application. Toutefois, dans la plupart des cas, Agent Builder fournit au moins une solution partielle. Les applications plus complexes nécessitent parfois du code supplémentaire, que Agent Builder ne génère pas par défaut, comme le code qui permet d'ajouter des marques de validation pour les propriétés supplémentaires ou de régler les paramètres que Agent Builder n'affiche pas. Le cas échéant, vous devez apporter des modifications au code source généré ou au fichier RTR. Agent Builder est spécialement conçu pour fournir ce type de flexibilité.

Agent Builder place des commentaires en des endroits bien précis du code source généré, où vous pouvez ajouter votre propre code de type de ressource. 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 écrivez tout votre code de type de ressource sans utiliser de code généré par Agent Builder, vous pouvez utiliser le fichier makefile et la structure fournis par Agent Builder pour créer le package Solaris pour votre type de ressource.

Installation et configuration d'Agent Builder

Agent Builder ne nécessite aucune installation particulière. Agent Builder est inclus dans le package `SUNWscdev`, qui est installé par défaut avec le logiciel Sun Cluster. Pour plus d'informations, voir *Guide d'installation du logiciel Sun Cluster pour SE Solaris*.

Avant d'utiliser Agent Builder, vérifiez les points suivants :

- L'environnement d'exécution Java est inclus dans votre variable `$PATH`. Agent Builder fonctionne avec la version 1.3.1. ou une version ultérieure du kit de développement Java. Si ce kit n'est pas inclus dans votre variable `$PATH`, la commande de Agent Builder (`scdsbuilder`) renvoie et affiche un message d'erreur.
- Vous avez installé au minimum la version Solaris 8 du groupe de logiciels du Support système développeur.
- Le compilateur `cc` est inclus dans votre variable `$PATH` . Agent Builder se sert de la première occurrence de `cc` dans votre variable `$PATH` pour identifier le compilateur à utiliser pour générer le code binaire C du type de ressource. Si `cc` n'est pas inclus dans `$PATH`, Agent Builder désactive l'option permettant de générer le code C. Voir "Utilisation de l'écran Créer" à la page 168.

Remarque – Avec Agent Builder, vous pouvez utiliser un autre compilateur que le compilateur standard `cc`. Pour ce faire, créez un lien symbolique dans `$PATH` de `cc` vers un autre compilateur, tel que `gcc`. Vous pouvez également changer le compilateur spécifié dans le fichier makefile (actuellement `CC=cc`) en saisissant le chemin complet d'un autre compilateur. Par exemple, dans le fichier makefile généré par Agent Builder, remplacez `CC=cc` par `CC=`*pathname*`/gcc`. Si vous procédez à cette modification, vous ne pouvez pas exécuter Agent Builder directement. Au lieu de cela, vous devez utiliser les commandes `make` et `make pkg` pour générer du code de service de données et le package.

Écrans d'Agent Builder

Agent Builder est un assistant en deux étapes, chacune d'elles disposant d'un écran. Agent Builder comprend les deux écrans suivants qui vous guident tout au long du processus de création d'un nouveau type de ressource :

1. **L'écran Créer.** Cet écran vous permet d'entrer des informations de base sur le type de ressource à créer, telles que son nom et le répertoire de travail où placer les fichiers générés. Le répertoire de travail constitue l'emplacement de création et de configuration du modèle de type de ressource. Vous pouvez également préciser les informations suivantes :

   - Le type de ressource à créer (évolutive ou de basculement)
   - Si l'application de base reconnaît le réseau (c'est-à-dire si elle utilise le réseau pour communiquer avec les clients)
   - Le type de code à générer (C, korn shell (`ksh`) ou GDS)

   Pour plus d'informations sur GDS, voir Chapitre 10. Vous devez remplir tous les champs d'information de cet écran et sélectionner Créer pour générer la sortie correspondante. Ensuite, vous pouvez afficher l'écran Configurer.
2. **L'écran Configurer.** Sur cet écran, vous devez spécifier la ligne de commande complète qui peut être transmise à un shell UNIX pour démarrer votre application de base. Si vous le souhaitez, vous pouvez fournir des commandes permettant d'arrêter et d'analyser l'application. Si vous ne spécifiez pas ces deux commandes, la sortie générée utilise des signaux pour arrêter l'application et propose un mécanisme de sonde par défaut. Voir la description de la commande de la sonde à la section "Utilisation de l'écran Configurer" à la page 170. L'écran Configurer vous permet également de modifier les valeurs de délai d'attente pour chacune de ces trois commandes : démarrage, arrêt, sonde.

Démarrage de Agent Builder

Remarque – Si la version d'interface utilisateur graphique de Agent Builder n'est pas accessible, vous pouvez accéder à Agent Builder via l'interface de ligne de commande. Voir "Utilisation de la version de ligne de commande d'Agent Builder " à la page 177.

Si vous lancez Agent Builder à partir d'un répertoire de travail pour un type de ressource existant, Agent Builder initialise les écrans de création et de configuration en fonction des valeurs de ce type de ressource.

Lancez Agent Builder en entrant la commande suivante :

```
% /usr/cluster/bin/scdsbuilder
```

L'écran Créer apparaît.

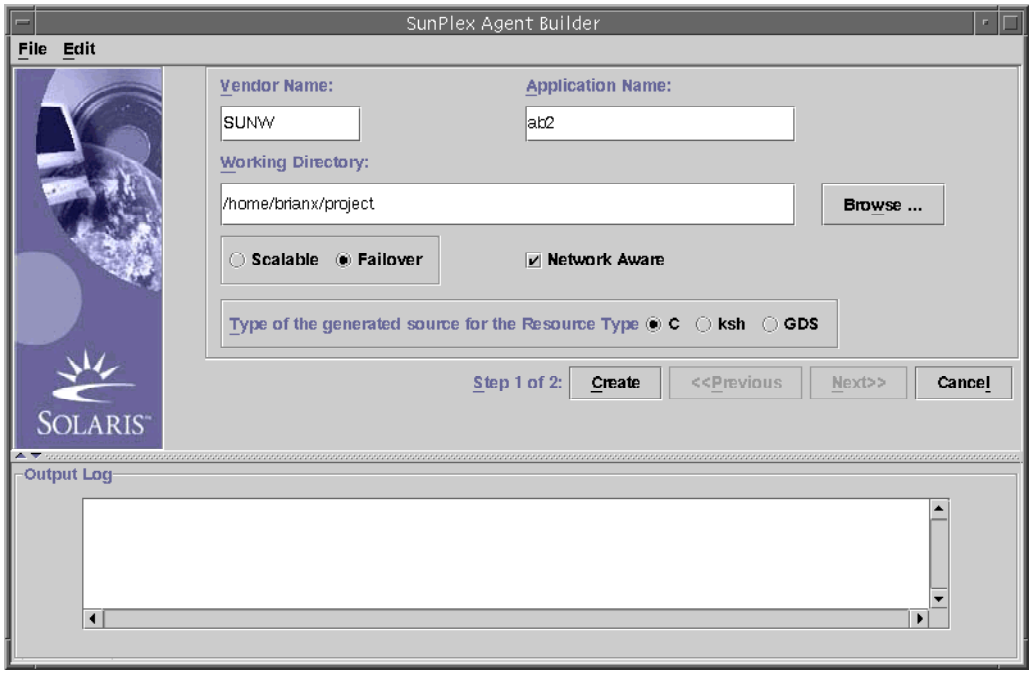


FIGURE 9–1 Écran Créer de Agent Builder

Navigation dans Agent Builder

Procédez de la manière suivante pour entrer des informations sur les écrans de création et de configuration :

- Tapez l'information dans le champ correspondant
- Parcourez votre arborescence et choisissez un fichier ou un répertoire
- Choisissez un bouton radio excluant tous les autres, par exemple le bouton correspondant à "évolutif" ou à "basculement"
- Cochez la case de reconnaissance du réseau pour caractériser l'application de base comme reconnaissant le réseau. Sinon, laissez cette case décochée

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 fait ressortir ces boutons ou les grise, le cas échéant.

Par exemple, lorsque vous avez rempli les champs et sélectionné les options de votre choix sur l'écran Créer, cliquez sur Créer en bas de l'écran. Les options Précédent et Suivant sont grisées car il n'existe aucun écran précédent et vous ne pouvez pas passer à l'étape suivante avant d'avoir effectué celle-ci.

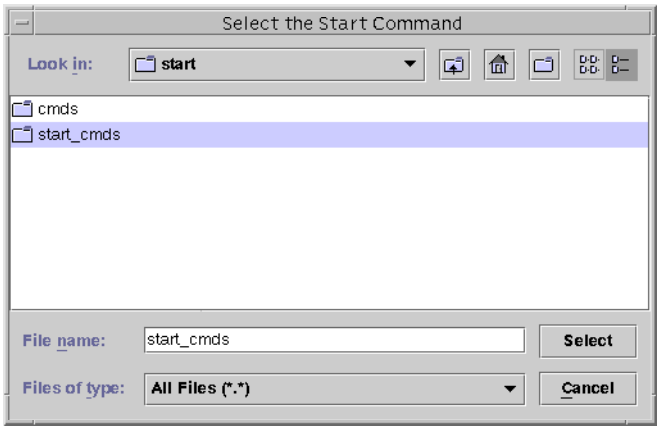
Agent Builder affiche des messages de progression dans la zone Journal de sortie située en bas de l'écran. Lorsque l'opération est terminée, il affiche un message de réussite ou un avertissement. Le bouton Suivant est activé ou, s'il s'agit du dernier écran, seul le bouton Annuler est mis en surbrillance.

Vous pouvez cliquer sur Annuler à tout moment pour quitter Agent Builder.

Commande Parcourir

Certains champs Agent Builder vous permettent d'entrer des informations. D'autres champs vous invitent à cliquer sur Parcourir pour naviguer dans une arborescence et sélectionner un fichier ou un répertoire.

Lorsque vous cliquez sur Parcourir, un écran semblable à celui-ci apparaît.



Double-cliquez sur un dossier pour l'ouvrir. Lorsque vous déplacez le curseur sur un fichier, son nom apparaît dans le champ correspondant au nom de fichier. Après avoir localisé le fichier souhaité et pointé le curseur dessus, cliquez sur Sélectionner.

Remarque – Si vous recherchez un répertoire, positionnez le curseur sur le répertoire souhaité et cliquez sur Ouvrir. Si le répertoire ne contient aucun sous-répertoire, Agent Builder ferme la fenêtre du navigateur et place le nom du répertoire sur lequel vous avez déplacé le curseur dans le champ approprié. S'il contient des sous-répertoires, cliquez sur Fermer pour faire disparaître la fenêtre et revenir à l'écran précédent. Agent Builder place le nom du répertoire sur lequel vous avez placé le curseur dans le champ approprié.

Les icônes situées dans le coin supérieur droit de l'écran Parcourir permettent d'effectuer les opérations suivantes :

Icône	Objet
	Cette icône permet de monter d'un niveau dans la structure de répertoire.
	Cette icône permet de revenir au dossier personnel.

Icône	Objet
	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 de Agent Builder

Agent Builder propose les menus déroulants Fichier et Édition.

Menu Fichier de Agent Builder

Le menu Fichier contient deux options :

- **Charger type de ressources** : Permet de charger un type de ressource existant. Dans Agent Builder, un écran de recherche vous permet de sélectionner le répertoire de travail d'un type de ressources existant. Si un type de ressource existe dans le répertoire à partir duquel vous démarrez Agent Builder, celui-ci charge automatiquement le type de ressource. L'option de chargement du type de ressource vous permet de lancer Agent Builder à partir de n'importe quel répertoire et de sélectionner un type de ressource existant à utiliser en tant que modèle pour créer un nouveau type de ressource. Voir "Réutilisation du code créé avec Agent Builder " à la page 175.
- **Quitter** : permet de quitter Agent Builder. Vous pouvez également quitter Agent Builder en cliquant sur Annuler dans l'écran Créer ou Configurer.

Menu Édition de Agent Builder

Le menu Édition contient deux options :

- **Effacer le journal de sortie** : permet de supprimer les informations du journal. Chaque fois que vous sélectionnez Créer ou Configurer, Agent Builder ajoute des messages d'état dans le journal. Si vous modifiez votre code source de manière itérative, si vous générez à nouveau la sortie dans Agent Builder et souhaitez distinguer les messages d'état, vous pouvez enregistrer et effacer le fichier journal avant chaque utilisation.
- **Enregistrer journal** : permet d'enregistrer la sortie du journal dans un fichier. Agent Builder fournit un écran de navigation qui vous permet de sélectionner le répertoire et de spécifier un nom de fichier.

Utilisation de l’écran Créer

Pour créer un type de ressource, vous devez en premier lieu remplir les champs de l’écran Créer, qui apparaît lorsque vous démarrez Agent Builder. La capture suivante représente l’écran Créer une fois que vous avez entré les informations dans les champs.

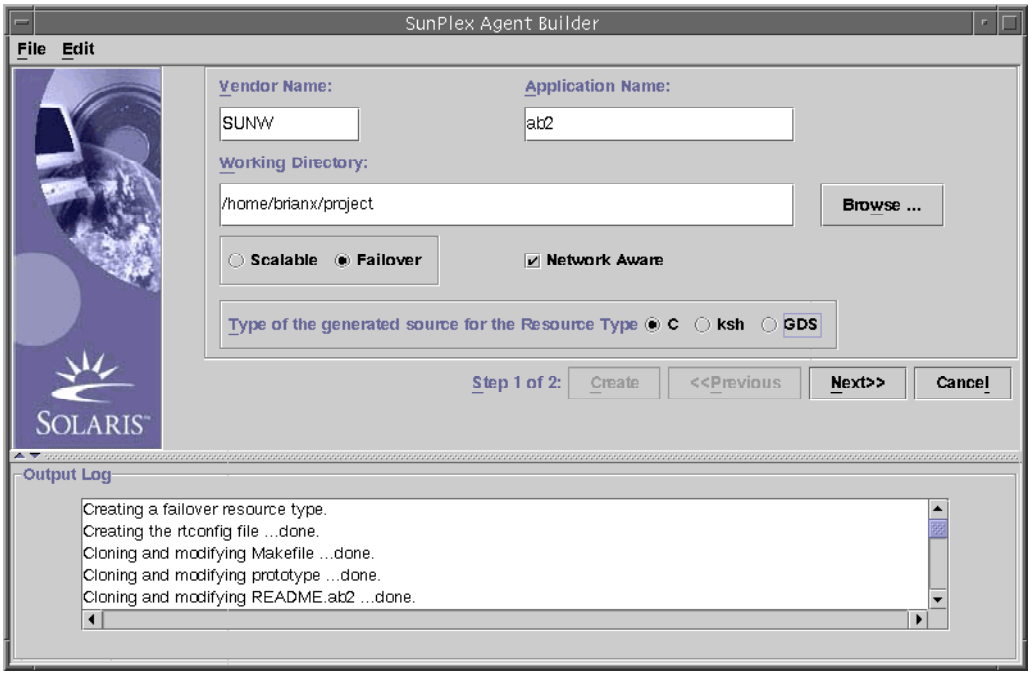


FIGURE 9–2 Écran Créer de Agent Builder, une fois que vous avez entré les informations

L’écran Créer contient les champs, les cases d’option et les cases à cocher suivantes :

- **ID du fournisseur** : nom identifiant le fournisseur du type de ressources. En général, on précise le symbole du fournisseur. Toutefois, tout nom permettant d’identifier le fournisseur de manière unique est considéré comme valable. N’utilisez que des caractères alphanumériques.
- **Nom de l’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. Avec le système d'exploitation Solaris 9, la combinaison du nom du fournisseur et du nom d'application peut dépasser neuf caractères. En revanche, elle ne doit pas dépasser neuf caractères si vous utilisez une version précédente du système d'exploitation Solaris.

- **Version TR** : version du type de ressources. Le champ Version TR fait la distinction entre les différentes versions enregistrées ou les mises à niveau d'un même type de ressources.

  Vous ne pouvez pas utiliser les caractères suivants dans le champ Version TR :

  - Espace
  - Onglet
  - Barre oblique (`/`)
  - Barre oblique inverse (`\`)
  - Astérisque (`*`)
  - Point d'interrogation (`?`)
  - Virgule (`,`)
  - Point virgule (`;`)
  - Crochet gauche (`[`)
  - Crochet droit (`]`)

- **Répertoire de travail** : Répertoire sous lequel Agent Builder crée une structure de répertoire destinée à contenir tous les fichiers qui sont créés pour le type de ressource cible. Quel que soit le répertoire de travail, vous ne pouvez y créer qu'un seul type de ressources. Agent Builder initialise ce champ avec le chemin du répertoire à partir duquel vous avez démarré Agent Builder. Toutefois, vous pouvez taper un autre nom ou utiliser l'option Parcourir pour rechercher un autre répertoire.

  Sous le répertoire de travail, Agent Builder crée un sous-répertoire contenant le nom du type de ressources. Par exemple, si `SUNW` est le nom du fournisseur et `ftp`, le nom de l'application, Agent Builder nomme ce sous-répertoire `SUNWftp`.

  Agent Builder place tous les répertoires et fichiers du type de ressources cible dans ce sous-répertoire. Voir "Structure de répertoire créée par Agent Builder " à la page 178.

- **Évolutif ou Basculement** : précisez le type de ressource cible.
- **Compatible réseau** : indique si l'application de base est sensible au réseau, c'est-à-dire si elle utilise le réseau pour communiquer avec ses clients. Cochez la case de reconnaissance du réseau pour indiquer que l'application reconnaît le réseau. Sinon, laissez cette case décochée.
- **C, ksh** : indique le langage du code source généré. Bien que ces options ne puissent pas être sélectionnées en même temps, Agent Builder vous permet de créer un type de ressource avec génération de code korn shell et de réutiliser les mêmes informations pour créer un code C généré. Voir "Réutilisation du code créé avec

Agent Builder " à la page 175.

- **GDS** : indique qu'il s'agit d'un service de données générique (Generic Data Service, GDS). Pour plus d'informations sur la création et la configuration d'un service de données générique, voir Chapitre 10.

Remarque – Si le compilateur `cc` n'est pas inclus dans votre variable `$PATH`, Agent Builder grise le bouton radio C et vous permet de sélectionner le bouton ksh. Pour spécifier un autre compilateur, reportez-vous à la remarque à la fin de la section "Installation et configuration d'Agent Builder" à la page 162.

Une fois que vous avez entré les informations requises, cliquez sur Créer. Située en bas de l'écran, la zone Journal de sortie montre les actions effectuées par Agent Builder. Vous pouvez choisir l'option Enregistrer journal du menu Éditer pour enregistrer les informations dans le journal.

Une fois l'opération terminée, Agent Builder affiche un message indiquant qu'elle a été correctement effectuée ou un avertissement.

- Si Agent Builder n'a pas pu effectuer cette opération, consultez le journal de sortie pour savoir pourquoi.
- Si l'opération a pu être effectuée correctement, cliquez sur Suivant pour afficher l'écran Configurer. L'écran Configurer vous permet de terminer le processus de génération du type de ressource.

Remarque – Bien que la génération d'un type de ressource complet se fasse en deux étapes, vous pouvez quitter Agent Builder après la première étape (création) sans perdre les informations que vous avez entrées, ni le travail effectué par Agent Builder. Voir "Réutilisation du code créé avec Agent Builder " à la page 175.

Utilisation de l'écran Configurer

L'écran Configurer, représenté sur la capture suivante, apparaît une fois que Agent Builder a terminé de créer le type de ressource et que vous avez cliqué sur Suivant sur l'écran Créer. Vous ne pouvez pas accéder à l'écran Configurer avant d'avoir créé le type de ressources.

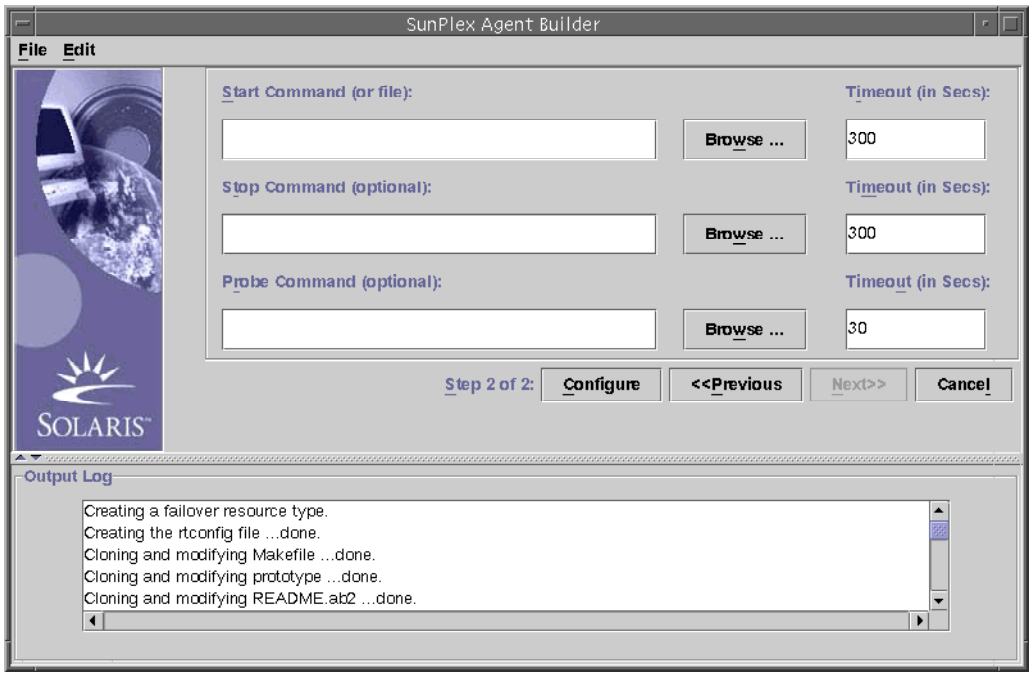


FIGURE 9–3 Écran Configurer de Agent Builder

L'écran Configurer contient les champs suivants :

- **Commande de démarrage** : ligne de commande complète qui peut être transmise à tout shell UNIX pour lancer l'application de base. Vous devez indiquer une commande de démarrage. Vous pouvez taper la commande dans le champ prévu à cet effet, ou utiliser Parcourir pour rechercher un fichier qui contienne la commande permettant de démarrer l'application.

  La ligne de commande complète doit inclure tous les éléments nécessaires pour lancer l'application, tels que les noms d'hôte, les numéros de port et un chemin d'accès aux fichiers de configuration. Vous pouvez également spécifier des variables de propriété, qui sont décrites dans "Utilisation des variables de propriété" à la page 174. Si l'application korn shell requiert la définition d'un nom d'hôte dans la ligne de commande, vous pouvez utiliser la variable `$hostnames` définie par Agent Builder. Voir "Utilisation de la variable `$hostnames` basée sur le korn shell de Agent Builder" à la page 173.

  Ne placez pas la commande entre guillemets ("").

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 (Process Monitor Facility), vous ne pouvez indiquer aucune commande. À la place, vous devez créer un fichier texte qui contienne les commandes individuelles permettant de lancer chaque arborescence de processus, puis spécifier le chemin d'accès à ce fichier dans le champ Commande de démarrage. Voir "Avant d'utiliser Agent Builder" à la page 160. Cette section énumère certains caractères spéciaux dont ce fichier a besoin pour fonctionner correctement.

- **Commande d'arrêt** : ligne de commande complète qui peut être transmise à tout shell UNIX pour arrêter l'application de base. Vous pouvez saisir la commande dans le champ prévu à cet effet, ou utiliser la fonction Parcourir pour rechercher un fichier qui contienne la commande permettant d'arrêter l'application. Vous pouvez également spécifier des variables de propriété, qui sont décrites dans "Utilisation des variables de propriété" à la page 174. Si l'application korn shell requiert la définition d'un nom d'hôte dans la ligne de commande, vous pouvez utiliser la variable `$hostnames` définie par Agent Builder. Voir "Utilisation de la variable `$hostnames` basée sur le korn shell de Agent Builder" à la page 173.

  Cette commande est facultative. Si vous n'indiquez pas de commande d'arrêt, le code généré utilise les signaux (de la méthode `Stop`) pour arrêter l'application, comme suit :

  - La méthode `Stop` 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 `Stop` 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` ne fonctionne pas, la méthode `Stop` ne permet pas de quitter l'application. Les 5 % restants du délai d'attente sont considérés comme temps système.

Attention – veillez à ce que la commande d'arrêt ne soit pas renvoyée avant l'arrêt complet de l'application.

- **Commande d'analyse** : Commande qui peut être exécutée périodiquement pour vérifier l'état de l'application et renvoyer un état de sortie compris entre `0` (réussite) et `100` (échec total). Cette commande est facultative. Vous pouvez saisir le chemin d'accès complet à la commande, ou utiliser l'option Parcourir pour rechercher un fichier qui contienne les commandes permettant de sonder l'application.

  En règle générale, vous indiquez un client unique de l'application de base. Si vous ne spécifiez aucune commande de sonde, le code généré se connecte et se déconnecte simplement du port utilisé par la ressource. Si la connexion et la

déconnexion fonctionnent, le code généré signale que l'application fonctionne parfaitement. Vous pouvez également spécifier des variables de propriété, qui sont décrites dans "Utilisation des variables de propriété" à la page 174. Si l'application korn shell requiert la définition d'un nom d'hôte dans la ligne de commande, vous pouvez utiliser la variable `$hostnames` définie par Agent Builder. Voir "Utilisation de la variable `$hostnames` basée sur le korn shell de Agent Builder" à la page 173.

Ne placez pas la commande entre guillemets ("").

- **Délai d'attente** : valeur de délai d'attente, en secondes, pour chaque commande. Vous pouvez spécifier une nouvelle valeur, ou accepter la valeur par défaut fournie par Agent Builder. La valeur par défaut est de 300 secondes pour le démarrage et l'arrêt et de 30 secondes pour la commande de la sonde.

Utilisation de la variable `$hostnames` basée sur le korn shell de Agent Builder

Pour de nombreuses applications, en particulier celles qui reconnaissent le réseau, le nom d'hôte sur lequel l'application écoute et les requêtes des clients de services doivent être transmis à l'application sur la ligne de commande. Par conséquent, le nom d'hôte est généralement un paramètre que vous devez indiquer sur l'écran Configurer pour les commandes de démarrage, d'arrêt et de sonde du type de ressource cible. Le nom d'hôte d'écoute de l'application est propre au cluster. Le nom d'hôte est déterminé lorsque la ressource est exécutée sur un cluster. Il ne peut pas être déterminé lorsque Agent Builder génère votre code de type de ressource.

Pour résoudre ce problème, Agent Builder fournit la variable `$hostnames`, que vous pouvez spécifier sur la ligne de commande pour les commandes de démarrage, d'arrêt et de sonde.

Remarque – La variable `$hostnames` n'est prise en charge que lorsqu'elle est utilisée avec les services korn shell. Elle ne peut pas être utilisée avec les services C et GDS.

Vous spécifiez la variable `$hostnames` exactement comme vous indiqueriez un nom d'hôte réel. Exemple :

```
% /opt/network_aware/echo_server -p port-no -l $hostnames
```

Lorsqu'une ressource du type de ressource cible est exécutée sur un cluster, le nom d'hôte `LogicalHostname` ou `SharedAddress` qui est configuré pour cette ressource est remplacé par la valeur de la variable `$hostnames`. Le nom d'hôte est configuré pour cette ressource dans la propriété `Network_resources_used` de la ressource.

Si vous configurez la propriété `Network_resources_used` avec plusieurs noms d'hôte, la variable `$hostnames` contient tous les noms d'hôte, séparés par une virgule.

Utilisation des variables de propriété

Vous pouvez également récupérer les valeurs des propriétés d'un type de ressource, d'une ressource et d'un groupe de ressources Sun Cluster, sélectionnées à partir de la structure RGM à l'aide de variables de propriété. Agent Builder recherche les variables de propriété dans vos chaînes de commande de démarrage, d'arrêt ou de sonde et les remplace par leurs valeurs avant que Agent Builder n'exécute la commande.

Remarque – Les variables de propriété ne peuvent pas être utilisées avec des services korn shell.

Liste des variables de propriété

Cette section répertorie les variables de propriété que vous pouvez utiliser. Les propriétés du type de ressource, de la ressource et du groupe de ressources Sun Cluster sont décrites à la section Annexe A.

Variables des propriétés de ressource

- `HOSTNAMES`
- `RS_CHEAP_PROBE_INTERVAL`
- `RS_MONITOR_START_TIMEOUT`
- `RS_MONITOR_STOP_TIMEOUT`
- `RS_NAME`
- `RS_NUM_RESTARTS`
- `RS_RESOURCE_DEPENDENCIES`
- `RS_RESOURCE_DEPENDENCIES_WEAK`
- `RS_RETRY_COUNT`
- `RS_RETRY_INTERVAL`
- `RS_SCALABLE`
- `RS_START_TIMEOUT`
- `RS_STOP_TIMEOUT`
- `RS_THOROUGH_PROBE_INTERVAL`
- `SCHA_STATUS`

Variables de propriétés du type de ressource

- `RT_API_VERSION`
- `RT_BASEDIR`
- `RT_FAILOVER`
- `RT_INSTALLED_NODES`
- `RT_NAME`
- `RT_RT_VERSION`

- RT_SINGLE_INSTANCE

Variables de propriété du groupe de ressources

- RG_DESIRED_PRIMARIES
- RG_GLOBAL_RESOURCES_USED
- RG_IMPLICIT_NETWORK_DEPENDENCIES
- RG_MAXIMUM_PRIMARIES
- RG_NAME
- RG_NODELIST
- RG_NUM_RESTARTS
- RG_PATHPREFIX
- RG_PINGPONG_INTERVAL
- RG_RESOURCE_LIST

Syntaxe des variables de propriété

Vous devez ajouter le symbole du pourcentage (%) avant un nom de propriété pour indiquer la présence d'une variable de propriété, comme dans l'exemple suivant :

```
/opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

Si l'on prend cet exemple, Agent Builder pourrait interpréter ces variables de propriété et lancer le script `echo_server` avec les valeurs suivantes :

```
/opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

Interprétation et remplacement des variables de propriété par Agent Builder

Agent Builder interprète les types de variable de propriété de la manière suivante :

- Un nombre entier est remplacé par sa valeur réelle (`300`, par exemple).
- Une valeur booléenne est remplacée par la chaîne `TRUE` ou `FALSE`.
- Une chaîne est remplacée par la chaîne réelle (`phys-node-1`, par exemple).
- Une liste de chaînes est remplacée par tous les éléments de la liste, chaque chaîne étant séparée par une virgule (`phys-node-1,phys-node-2,phys-node-3`, par exemple).
- Une liste de nombres entiers est remplacée par tous les éléments de la liste, chaque nombre entier étant séparé par une virgule (`1,2,3`, par exemple).
- Un type énuméré est remplacé par sa valeur, sous forme de chaîne.

Réutilisation du code créé avec Agent Builder

Agent Builder vous permet de réutiliser un travail terminé de plusieurs façons :

- Vous pouvez cloner un type de ressource existant que vous avez créé avec Agent Builder.
- Vous pouvez modifier le code source généré par Agent Builder, puis recompiler le code pour créer un nouveau package.

▼ Clonage d'un type de ressources existant

Appliquez la procédure suivante pour cloner un type de ressource existant généré par Agent Builder.

Étapes

1. **Chargez un type de ressources existant dans Agent Builder, de l'une des manières suivantes :**
   - Lancez Agent Builder à partir du répertoire de travail pour un type de ressource existant créé avec Agent Builder. Vérifiez que ce répertoire contient le fichier `rtconfig`. Agent Builder charge les valeurs du type de ressources sur les écrans Créer et Configurer.
   - Utilisez l'option de chargement du type de ressource du menu déroulant Fichier.
2. **Modifiez le répertoire de travail dans l'écran Créer.**

   Vous devez utiliser l'option Parcourir pour sélectionner un répertoire. La saisie d'un nouveau nom de répertoire est insuffisante. Une fois le répertoire sélectionné, Agent Builder active de nouveau le bouton Créer.
3. **Apportez les changements souhaités au type de ressource existant.**

   Vous pouvez avoir à modifier le type de code généré pour le type de ressource. Par exemple, si vous créez une version korn shell d'un type de ressource et si vous vous apercevez ensuite que vous avez besoin d'une version C, procédez comme suit :
   - Chargez le type de ressource existant en version korn shell.
   - Paramétrez le langage de sortie sur C.
   - Cliquez sur Créer pour que Agent Builder crée une version C du type de ressource.
4. **Créez le clone du type de ressources.**
   a. **Cliquez sur Créer pour créer le type de ressources.**
   b. **Cliquez sur Suivant pour afficher l'écran Configurer.**
   c. **Cliquez sur Configurer pour configurer le type de ressource, puis sur Annuler pour terminer l'opération.**

Édition du code source généré

Pour simplifier le processus de création d'un type de ressource, Agent Builder limite la quantité d'informations que vous pouvez spécifier, ce qui limite obligatoirement l'étendue du type de ressource généré. Par conséquent, pour ajouter des fonctionnalités plus sophistiquées, vous devez modifier le code source généré ou le fichier RTR. Exemple de fonctionnalité supplémentaire : le code qui permet d'ajouter des marques de validation pour les propriétés supplémentaires ou qui permet de régler les paramètres non affichés par Agent Builder.

Les fichiers source se trouvent dans le répertoire *install-directory*/*rt-name*/`src`. Agent Builder intègre les commentaires dans le code source pour que vous puissiez y ajouter le code. Exemple (pour un code C) :

```
/* User added code -- BEGIN vvvvvvvvvvvvvvv */
/* User added code -- END   ^^^^^^^^^^^^^^^ */
```

Remarque – Ces commentaires sont identiques dans le code source korn shell, si ce n'est que la marque de commentaire (#) indique le début d'un commentaire.

Par exemple, *rt-name*`.h` indique toutes les fonctions d'utilitaire utilisées par les différents programmes. Des commentaires sont placés à la fin de cette liste pour vous permettre d'indiquer les autres fonctions que vous pouvez avoir ajoutées à votre code.

Agent Builder génère également le fichier makefile dans le répertoire *install-directory*/*rt-name*/`src` avec les cibles appropriées. Utilisez la commande `make` pour recompiler le code source. Utilisez la commande `make pkg` pour regénérer le package du type de ressource.

Le fichier RTR se trouve dans le répertoire *install-directory*/*rt-name*/`etc`. Vous pouvez le modifier à l'aide d'un éditeur de texte standard. Pour plus d'informations sur le fichier RTR, voir "Paramétrage des propriétés de ressources et de types de ressources" à la page 35. Pour en savoir plus sur les propriétés, voir Annexe A.

▼ Utilisation de la version de ligne de commande d'Agent Builder

La version de ligne de commande de Agent Builder suit la même procédure de base que l'interface utilisateur graphique. Toutefois, au lieu de saisir des informations dans l'interface utilisateur graphique, vous pouvez transmettre les arguments aux commandes `scdscreate` et `scdsconfig`. Pour en savoir plus, reportez-vous aux pages de manuel `scdscreate`(1HA) et `scdsconfig`(1HA).

Procédez aux étapes suivantes pour utiliser la version de ligne de commande de Agent Builder.

Étapes

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`.**

   Vous pouvez indiquer des variables de propriété, Les variables de propriété sont décrites à la section "Utilisation des variables de propriété" à la page 174.

3. **Remplacez les répertoires par le sous-répertoire `pkg` dans le répertoire de travail.**

4. **Utilisez la commande `pkgadd` pour installer les packages créés avec `scdscreate`.**

   - **Dans un environnement à zones sous Solaris 10, entrez la commande suivante en tant qu'administrateur global dans la *zone globale* :**

     ```
     # pkgadd -G -d . package-name
     ```

     Le package que vous avez spécifié est ajouté à la zone globale, à condition que le contenu du package n'affecte aucune des parties de la zone globale partagées avec une zone non globale.

   - **Pour toute autre version du système d'exploitation Solaris ou pour Solaris 10 dans un environnement sans zones, tapez la commande suivante :**

     ```
     # pkgadd -d . package-name
     ```

5. **(Facultatif) Modifiez le code source généré.**

6. **Exécutez le script de démarrage.**

Structure de répertoire créée par Agent Builder

Agent Builder crée une structure de répertoire pour stocker tous les fichiers qu'il crée pour le type de ressource cible. Le répertoire de travail est défini à partir de 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 ressource. Par exemple, si vous spécifiez le nom de fournisseur `SUNW` et créez un type de ressource appelé `ftp`, Agent Builder crée un répertoire nommé `SUNWftp` dans le répertoire de travail.

Dans ce sous-répertoire, Agent Builder crée et alimente les répertoires répertoriés dans le tableau suivant.

Nom du répertoire	Contenu
`bin`	Pour la sortie C, contient les fichiers binaires compilés à partir des fichiers source. Spécifique à une sortie korn shell. 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 (.), afin de constituer le nom du fichier RTR. Par exemple, si le nom du fournisseur est `SUNW` et celui du type de ressource, `ftp`, le nom du fichier RTR sera `SUNW.ftp`.
`man`	Contient les pages de manuel personnalisées pour les scripts d'utilitaire `start`, `stop` et `remove`, par exemple `startftp`(1M), `stopftp`(1M) et `removeftp`(1M). Pour pouvoir afficher ces pages de manuel, indiquez-en le chemin à l'aide de l'option `man -M`. Exemple : `% man -M` *install-directory*`/SUNWftp/man removeftp`
`pkg`	Contient le dernier package Solaris qui inclut le service de données créé.
`src`	Contient les fichiers sources générés par Agent Builder.
`util`	Contient les scripts d'utilitaire `start`, `stop` et `remove`, générés par Agent Builder. Voir "Scripts d'utilitaire et pages de manuel créées par Sun Agent Builder" à la page 181. Agent Builder ajoute le nom de l'application à chacun de ces noms de script, par exemple `startftp`, `stopftp` et `removeftp`.

Sortie d'Agent Builder

Fichiers sources et binaires

Le gestionnaire (RGM) gère les groupes de ressources et, en définitive, les ressources sur un cluster. Il utilise un modèle de rappel. Lorsque des événements spécifiques se produisent, tels qu'un échec de noeud, le RGM appelle les méthodes du type de chacune des ressources fonctionnant sur le noeud affecté. Par exemple, le RGM appelle la méthode `Stop` pour arrêter une ressource fonctionnant sur le noeud affecté et appelle la méthode `Start` de cette ressource pour la lancer sur un autre noeud. Pour en savoir plus sur ce modèle, reportez-vous aux pages de manuel "Modèle du gestionnaire de groupes de ressources" à la page 21, "Méthodes de rappel" à la page 24 et `rt_callbacks`(1HA).

Pour prendre en charge ce modèle, Agent Builder génère huit programmes C exécutables ou scripts korn shell dans le répertoire *install-directory*`/`*rt-name*`/bin`. Ces programmes ou scripts de shell servent de méthodes de rappel.

Remarque – Le programme *rt-name*_probe, qui implémente un détecteur de pannes, n'est pas un programme de rappel à proprement parler. Le RGM n'appelle pas directement *rt-name*_probe : il appelle *rt-name*_monitor_start et *rt-name*_monitor_stop. Ces méthodes démarrent et arrêtent le détecteur de pannes en appelant *rt-name* _probe.

Les huit méthodes générées par Agent Builder sont les suivantes :

- *rt-name*_monitor_check
- *rt-name*_monitor_start
- *rt-name*_monitor_stop
- *rt-name*_probe
- *rt-name*_svc_start
- *rt-name*_svc_stop
- *rt-name*_update
- *rt-name*_validate

Pour plus d'informations sur ces méthodes, reportez-vous à la page de manuel rt_callbacks(1HA).

Dans le répertoire *install-directory*/*rt-name*/src (sortie C), Agent Builder génère les fichiers suivants :

- Un fichier d'en-tête (*rt-name*.h)
- Un fichier source (*rt-name*.c) qui contient le code commun à toutes les méthodes
- Un fichier d'objets (*rt-name*.o) pour le code commun
- Des fichiers source (*.c) pour chacune des méfhodes
- Des fichiers d'objets (*.o) pour chacune des méfhodes

Agent Builder lie le fichier *rt-name*.o à chacun des fichiers de méthode .o pour créer les fichiers exécutables dans le répertoire *install-directory*/*rt-name*/bin.

Pour la sortie de korn shell, les répertoires *install-directory*/*rt-name*/bin et *install-directory*/*rt-name*/src sont identiques. Chacun d'eux comporte huit scripts d'exécution correspondant aux sept méthodes de rappel et à la méthode Probe.

Remarque – La sortie de korn shell comprend deux utilitaires compilés, gettime et gethostnames. Les méthodes de rappel particulières utilisent ces méthodes pour obtenir du temps et pour procéder à l'analyse.

Vous pouvez modifier le code source, exécuter la commande `make` pour recompiler le code et, lorsque vous avez terminé, exécuter la commande `make pkg` pour générer un nouveau package. Pour faciliter la modification du code source, Agent Builder ajoute des commentaires aux endroits où vous devez ajouter du code dans le code source. Voir "Édition du code source généré" à la page 177.

Scripts d'utilitaire et pages de manuel créées par Sun Agent Builder

Une fois que vous avez généré un type de ressource et installé son package sur un cluster, vous devez encore exécuter une instance (ressource) du type de ressource fonctionnant sur un cluster. Normalement, pour exécuter une instance, vous devez utiliser les commandes administratives ou SunPlex Manager. Toutefois, pour faciliter les choses, Agent Builder génère un script d'utilitaire personnalisé à cette fin, ainsi que des scripts permettant d'arrêter et de supprimer une ressource du type de ressource cible. Ces trois scripts, qui se trouvent dans le répertoire *install-directory*`/`*rt-name*`/util`, effectuent les opérations suivantes :

- **Script de démarrage** : enregistre le type de ressources et crée les ressources et les groupes de ressources nécessaires. Ce script crée également la ressource d'adresse réseau (`LogicalHostname` ou `SharedAddress`) qui permet à l'application de communiquer avec les clients sur le réseau.
- **Script d'arrêt** : arrête la ressource.
- **Script de suppression** : annule les actions du script de démarrage. Il permet d'arrêter les ressources, les groupes de ressources et le type de ressources cible et de les supprimer du système.

Remarque – Vous ne pouvez utiliser que le script de suppression avec une ressource qui a été démarrée par le script de démarrage correspondant, 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 de l'application est `ftp`, les scripts sont appelés `startftp`, `stopftp` et `removeftp`.

Agent Builder fournit des pages de manuel dans le répertoire *install-directory*`/`*rt-name*`/man/man1m` pour chaque script d'utilitaire. Nous vous conseillons de les lire avant de démarrer ces scripts, car elles donnent des explications sur les arguments à transférer au script.

Pour pouvoir afficher ces pages, indiquez le chemin d'accès au répertoire `man` en utilisant l'option `-M` avec la commande `man`. Par exemple, si `SUNW` est le fournisseur et `ftp`, le nom de l'application, entrez la commande suivante pour afficher la page de manuel `startftp`(1M) :

```
% man -M install-directory/SUNWftp/man startftp
```

Les scripts d'utilitaire des pages man 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 portant sur les scripts d'utilitaire sont placées dans le répertoire `/opt/`*rt-name*`/man`. Par exemple, entrez la commande suivante pour afficher la page de manuel `startftp`(1M) :

```
% man -M /opt/SUNWftp/man startftp
```

Fichiers de prise en charge créés par Agent Builder

Agent Builder place les fichiers de prise en charge, tels que `pkginfo`, `postinstall`, `postremove` et `preremove`, dans le répertoire *install-directory*`/`*rt-name*`/etc`. Ce répertoire contient également le fichier d'enregistrement du type de ressource (RTR). Le fichier RTR déclare les propriétés de ressource et de type de ressource qui sont disponibles pour le type de ressource cible et initialise les valeurs de propriété au moment où une ressource est enregistrée avec un cluster. Pour en savoir plus, voir "Paramétrage des propriétés de ressources et de types de ressources" à la page 35. Le nom du fichier RTR suit le modèle *vendor-name* . *resource-type-name*, 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. En revanche, vous devez reconstruire le package à l'aide de la commande `make pkg`.

Répertoire de package créé par Agent Builder

Le répertoire *install-directory*`/`*rt-name*`/pkg` contient un package Solaris. Le nom du package est une concaténation du nom du fournisseur et du nom de l'application, par exemple `SUNWftp`. Le fichier makefile qui se trouve dans le répertoire *install-directory*`/`*rt-name*`/src` prend en charge la création d'un nouveau package. Par exemple, si vous modifiez les fichiers source et recompilez le code, ou si vous modifiez les scripts d'utilitaire du package, utilisez la commande `make pkg` pour créer un nouveau package.

Lorsque vous supprimez un package d'un cluster, la commande `pkgrm` peut échouer si vous tentez de l'exécuter en même temps à partir de plusieurs noeuds. Vous pouvez pallier ce problème de l'une des deux façons suivantes :

- Exécutez le script `remove` *rt-name* à partir d'un des noeuds du cluster avant d'exécuter la commande `pkgrm` à partir d'un autre noeud.
- Exécutez la commande `pkgrm` à partir d'un des noeuds du cluster : cela permet d'effectuer toutes les opérations de nettoyage nécessaires. Exécutez ensuite la commande `pkgrm` à partir des autres noeuds, en même temps si nécessaire.

Si `pkgrm` échoue car vous tentez de l'exécuter en même temps à partir de plusieurs noeuds, exécutez la commande à nouveau à partir d'un seul noeud. Exécutez ensuite la commande à partir des autres noeuds.

Fichier `rtconfig`

Si vous générez du code source en shell C ou korn shell dans le répertoire de travail, Agent Builder génère un fichier de configuration appelé `rtconfig`. Ce fichier contient les informations que vous avez spécifiées sur les écrans Créer et Configurer. Si vous lancez Agent Builder à partir du répertoire de travail pour un type de ressource existant, Agent Builder lit le fichier `rtconfig`. Agent Builder se sert des informations que vous avez fournies pour le type de ressource existant pour compléter les écrans Créer et Configurer. Agent Builder fonctionne de la même manière si vous chargez un type de ressource existant en sélectionnant l'option correspondante du menu déroulant Fichier. Cette fonction est particulièrement utile lorsque vous souhaitez cloner un type de ressources existant. Voir "Réutilisation du code créé avec Agent Builder " à la page 175.

Module Cluster Agent pour Agent Builder

Le module Cluster Agent pour Agent Builder est un module NetBeans™. Ce module fournit une interface utilisateur graphique qui vous permet de créer des types de ressource pour le logiciel Sun Cluster en utilisant Sun Java Studio (anciennement Sun ONE Studio).

Remarque – La documentation de Sun Java Studio explique comment installer, configurer et utiliser Sun Java Studio. Elle est disponible à l'adresse `http://wwws.sun.com/software/sundev/jde/documentation/index.html`.

▼ Installation et configuration du module Cluster Agent

Le module Cluster Agent s'installe en même temps que le logiciel Sun Cluster. L'outil d'installation de Sun Cluster place le fichier du module Cluster Agent `scdsbuilder.jar` dans `/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, Sun Java Studio et Java 1.4 doivent être installés et accessibles sur le système sur lequel vous envisagez d'exécuter le module Cluster Agent.

Étapes

1. **Configurez un 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 avec un rôle 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 avez installé le logiciel Sun Java Studio dans un autre répertoire que `/opt/s1studio/ee`, remplacez ce chemin d'accès par le chemin que vous avez utilisé.

   - Pour être le seul à pouvoir accéder au module, créez le lien symbolique dans votre sous-répertoire `modules`.

     ```
     % cd ~your-home-dir/ffjuser40ee/modules
     % ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

2. **Fermez et relancez le logiciel Sun Java Studio.**

▼ Démarrage du module Cluster Agent

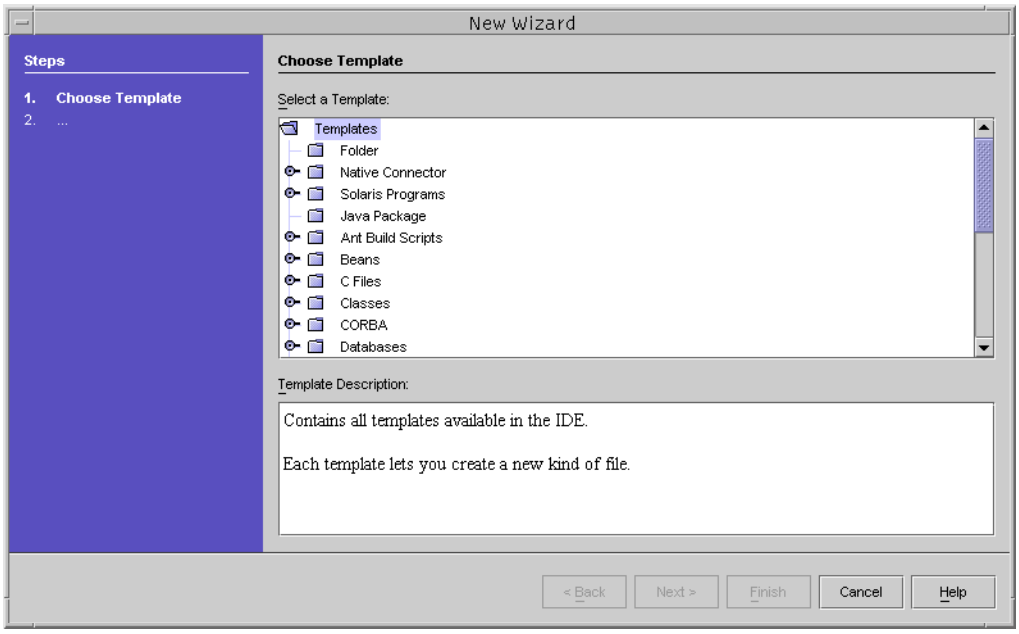
La procédure suivante décrit comment démarrer le module Cluster Agent à partir du logiciel Sun Java Studio.

Remarque – La documentation de Sun Java Studio explique comment installer, configurer et utiliser Sun Java Studio. Elle est disponible à l'adresse `http://wwws.sun.com/software/sundev/jde/documentation/index.html`.

Étapes

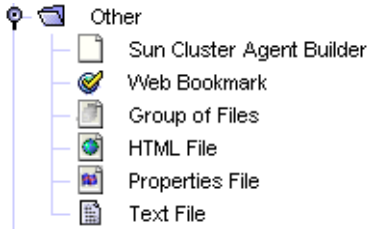
1. **À partir du menu Fichier de Sun Java Studio, sélectionnez Nouveau, ou dans la barre d'outils, cliquez sur l'icône illustrée ci-dessous :**

L'écran Assistant Nouveau apparaît.



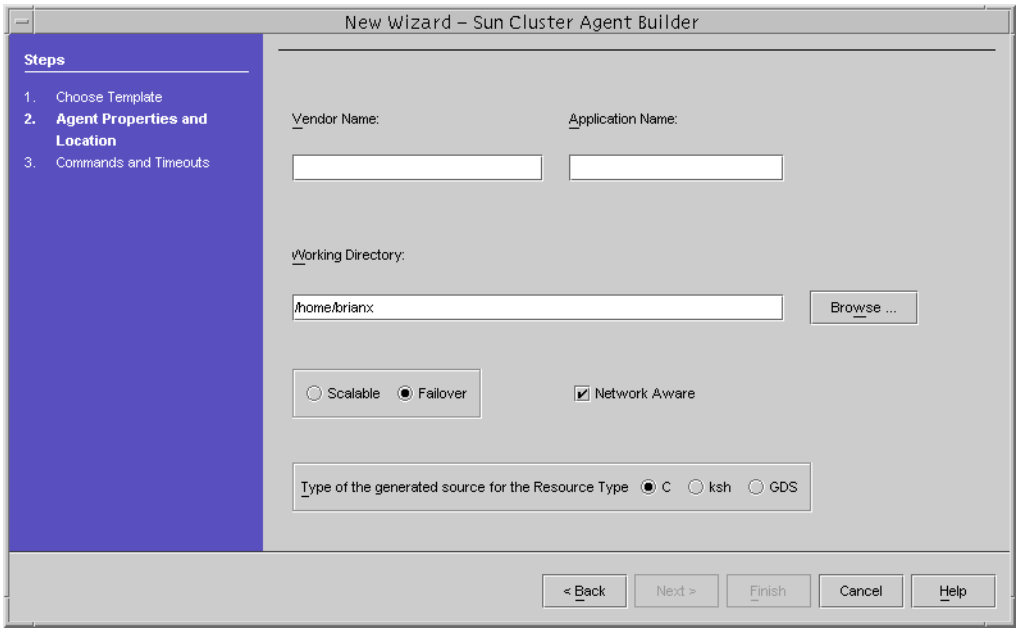
2. **Faites défiler la liste (si nécessaire) dans la fenêtre de sélection du modèle, puis cliquez sur la clé en regard du dossier Autre.**

Le dossier Autre s'ouvre.



3. **Dans le dossier 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.

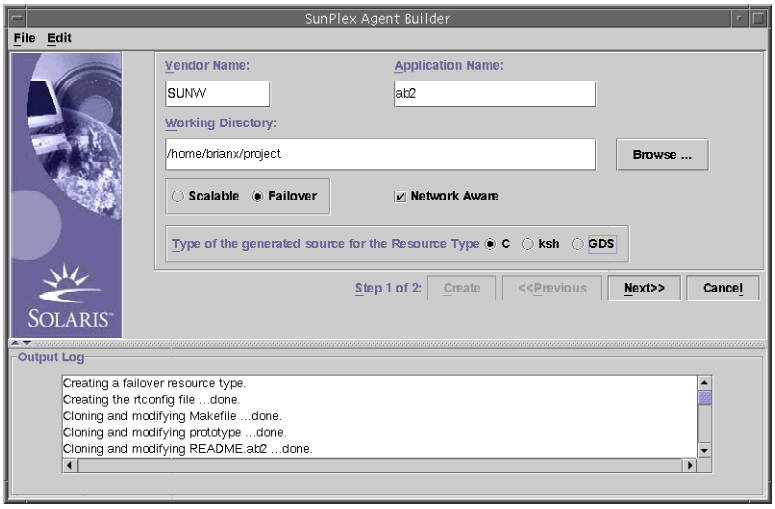


FIGURE 9–4 Écran Créer du logiciel Agent Builder

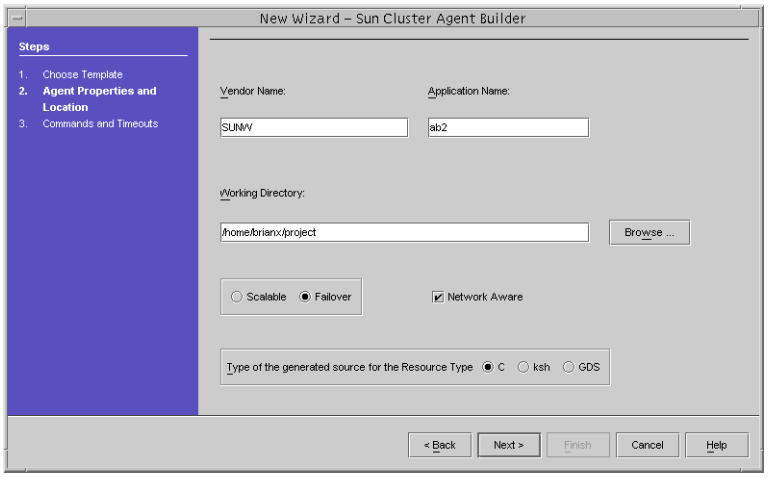


FIGURE 9–5 Écran Assistant Nouveau - Sun Cluster Agent Builder du module Cluster Agent

Différences entre le module Cluster Agent et Agent Builder

Il existe des différences mineures entre le module Cluster Agent et Agent Builder :

- 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 ressource est immédiatement créé lorsque vous cliquez sur l'option Créer de l'écran correspondant. De plus, le type de ressource est immédiatement configuré lorsque vous cliquez sur l'option Configurer de l'écran correspondant.
- Les données figurant dans la zone Journal de sortie de Agent Builder s'affichent dans une fenêtre séparée de Sun Java Studio.

CHAPITRE 10

Services de données génériques

Ce chapitre fournit des informations sur le service de données générique (Generic Data Service, GDS) et des instructions sur la création d'un service utilisant le service GDS. Vous pouvez créer ce service à l'aide des commandes d'administration SunPlex Agent Builder ou Sun Cluster.

Ce chapitre contient les rubriques suivantes :

Concepts de services de données génériques

Le GDS est un mécanisme permettant de rendre hautement disponibles ou évolutives des applications simples qui reconnaissent le réseau ou non, en les connectant à la structure du gestionnaire RGM de Sun Cluster. Ce mécanisme ne nécessite pas que vous codiez un service de données, ce qui est habituellement nécessaire 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`), et le fichier d'enregistrement du type de ressource (`rt_reg`).

Elle aborde les thèmes suivants:

- Type de ressource précompilée

- Avantages et inconvénients de l'utilisation du GDS
- Différents moyens de créer un service qui utilise le GDS
- Méthode de consignation des événements du GDS
- Propriétés de GDS requises
- Propriétés facultatives du GDS

Type de ressources précompilé

Le type de ressource `SUNW.gds` du service de données génériques est inclus dans le package `SUNWscgds`. L'utilitaire `scinstall` installe ce package lors de l'installation du cluster. Voir la page de manuel `scinstall`(1M). 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
```

Avantages et inconvénients de l'utilisation du module GDS

Le fait d'utiliser le GDS présente les avantages suivants par rapport à l'utilisation du code source généré par Agent Builder (voir la page de manuel `scdscreate`(1HA)) ou des commandes d'administration de Sun Cluster :

- Il est simple à utiliser.
- Le module et ses méthodes sont précompilés et ne peuvent donc pas être modifiés.
- Vous pouvez utiliser Agent Builder pour générer des scripts pour votre application. Ces scripts sont regroupés dans un package Solaris qui peut être réutilisé sur plusieurs clusters.

Bien que l'utilisation du GDS présente de nombreux avantages, il *ne* convient *pas* de l'utiliser dans les cas suivants :

- Lorsque vous avez besoin d'un plus grand contrôle que celui offert par le type de ressource précompilé, notamment lorsque vous devez ajouter des propriétés d'extension ou que vous devez modifier des valeurs par défaut

- Lorsque le code source doit être modifié pour pouvoir ajouter des fonctions spéciales

Création d'un service utilisant le module GDS

Il existe deux moyens de créer un service utilisant le GDS :

- Agent Builder
- Les commandes d'administration de Sun Cluster

Le GDS et Agent Builder

À l'aide de Agent Builder, sélectionnez un GDS comme type de code source généré. La saisie de l'utilisateur permet de générer un ensemble de scripts qui configurent les ressources de l'application donnée.

Le GDS et les commandes d'administration de Sun Cluster

Cette méthode utilise le code du service de données précompilé de `SUNWscgds`. Toutefois, l'administrateur du cluster doit utiliser les commandes d'administration de Sun Cluster pour créer et configurer la ressource. Voir les pages de manuel `scrgadm`(1M) et `scswitch`(1M).

Sélection d'une méthode de création d'un service basé sur GDS

De nombreuses données sont à saisir pour pouvoir exécuter les commandes `scrgadm` et `scswitch` appropriées. Vous trouverez des exemples aux sections "Utilisation des commandes d'administration de Sun Cluster afin de créer un service à haut niveau de disponibilité utilisant le module GDS" à la page 203 et "Utilisation des commandes d'administration de Sun Cluster afin de créer un service évolutif utilisant le module GDS" à la page 203.

L'utilisation du GDS avec Agent Builder simplifie le processus car il génère les scripts qui exécutent les commandes `scrgadm` et `scswitch` à votre place.

Consignation d'événements avec le module GDS

Le GDS vous permet de consigner les informations importantes qui sont transmises du GDS aux scripts exécutés par le GDS. Ces informations incluent l'état des méthodes de démarrage, de détection et d'arrêt, ainsi que les variables de propriété. Vous pouvez utiliser ces informations pour diagnostiquer des problèmes ou des erreurs dans vos scripts, ou à d'autres fins.

Vous pouvez utiliser la propriété `Log_level` décrite à la section "Propriété `Log_level`" à la page 196 pour indiquer le niveau, ou le type, des messages que le GDS doit consigner. Vous pouvez spécifier `NONE`, `INFO` ou `ERR`.

Fichiers journaux de GDS

Les deux fichiers journaux de GDS suivants sont placés dans le répertoire `/var/cluster/logs/DS/`*resource-group-name*`/`*resource-name* :

- `start_stop_log.txt`, qui contient les messages générés par les méthodes de démarrage et d'arrêt de ressources
- `probe_log.txt`, qui contient les messages générés par le détecteur de ressources

L'exemple suivant montre les types d'informations que contient `start_stop_log.txt` :

```
10/20/2005 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
10/20/2005 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

L'exemple suivant montre les types d'informations que contient `probe_log.txt` :

```
10/20/2005 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
10/20/2005 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

Propriétés requises par le module GDS

Si votre application reconnaît le réseau, vous devez fournir à la fois la propriété d'extension `Start_command` et la propriété `Port_list`. Dans le cas contraire, vous ne devez fournir que la propriété d'extension `Start_command`.

Propriété d'extension `Commande_démarrage`

La commande de démarrage, que vous spécifiez dans la propriété d'extension `Start_command`, démarre l'application. Cette commande doit être une commande UNIX et ses arguments doivent pouvoir être transmis directement à un shell pour démarrer l'application.

Propriété `Liste_ports`

La propriété `Port_list` identifie la liste des ports sur lesquels l'application écoute. La propriété `Port_list` doit figurer dans le script de démarrage créé par Agent Builder ou dans la commande `scrgadm` si vous utilisez les commandes d'administration de Sun Cluster.

Propriétés facultatives du module GDS

Les propriétés facultatives du GDS incluent à la fois les *propriétés définies par le système* et les *propriétés d'extension*. Les propriétés définies par le système sont un ensemble de propriétés standard fournies par Sun Cluster. Les propriétés qui sont définies dans le fichier RTR sont appelées propriétés d'extension. Les propriétés facultatives du GDS sont les suivantes :

- La propriété `Network_resources_used`
- La propriété d'extension `Stop_command`
- La propriété d'extension `Probe_command`
- La propriété `Start_timeout`
- La propriété `Stop_timeout`
- La propriété d'extension `Probe_timeout`
- La propriété d'extension `Child_mon_level` (utilisée uniquement avec les commandes d'administration)
- La propriété d'extension `Failover_enabled`
- La propriété d'extension `Stop_signal`
- La propriété d'extension `Log_level`

Propriété `Network_resources_used`

La valeur par défaut de cette propriété est nulle. 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 paramétrée sur `Null`, l'application est supposée écouter sur toutes les adresses.

Avant de créer la ressource GDS, une ressource `LogicalHostname` ou `SharedAddress` doit déjà avoir été configurée. Reportez-vous à la section *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* pour en savoir plus sur la configuration d'une ressource `LogicalHostname` ou `SharedAddress`.

Pour définir une valeur, indiquez un ou plusieurs noms de ressource, chaque nom pouvant contenir une ou plusieurs ressources `LogicalHostname` ou `SharedAddress`. Reportez-vous à la page de manuel `r_properties`(5) pour plus de détails.

Propriété `Stop_command`

La commande d'arrêt doit arrêter l'application et ne renvoyer une valeur qu'une fois que l'application est complètement arrêtée. Cette commande doit exécuter une commande UNIX pouvant être transmise directement à un shell pour arrêter l'application.

Si la propriété d'extension `Stop_command` est spécifiée, la méthode d'arrêt du GDS exécute la commande d'arrêt avec 80 pour cent du délai imparti à l'arrêt. Quel que soit le résultat de l'exécution de la commande d'arrêt, la méthode d'arrêt du GDS envoie la commande `SIGKILL` avec 15 pour cent du délai imparti à l'arrêt. Les 5% restants sont réservés au temps système de gestion interne.

En l'absence de la commande d'arrêt, le GDS essaie d'arrêter l'application à l'aide du signal spécifié dans `Stop_signal`.

Propriété `Commande_sonde`

La commande de détection vérifie régulièrement l'état de l'application. Cette commande doit être une commande UNIX et ses arguments doivent pouvoir être transmis directement à un shell pour sonder l'application. La commande de la sonde renvoie un état de sortie de `0` si l'application fonctionne correctement.

Cet état est utilisé pour déterminer le niveau de gravité de l'échec de l'application. L'état de sortie, également appelé *statut de détection*, doit être un nombre entier compris entre `0` (réussite) et `100` (échec total). Le statut de sonde peut également être une valeur spéciale de `201`, ce qui entraîne un basculement immédiat de l'application, sauf si `Failover_enabled` est paramétré sur `FALSE`. L'algorithme de détection du GDS utilise le statut de détection pour déterminer s'il faut redémarrer l'application en local ou procéder à son basculement. Reportez-vous à la page de manuel `scds_fm_action`(3HA) pour en savoir plus. Si l'état de sortie est `201`, l'application est immédiatement basculée.

En l'absence de la commande de détection, le GDS fournit sa propre sonde simple. Cette sonde se connecte à l'application sur le jeu d'adresses IP dérivé de la propriété `Network_resources_used` ou de la sortie de la fonction `scds_get_netaddr_list()`. Reportez-vous à la page de manuel `scds_get_netaddr_list`(3HA) pour en savoir plus. Si la connexion fonctionne, la déconnexion a lieu immédiatement. Si la connexion et la déconnexion réussissent, on considère que l'application fonctionne correctement.

Remarque – La sonde fournie avec le GDS a uniquement pour but de remplacer la sonde complète spécifique à l'application.

Propriété `Délai_démarrage`

Cette propriété spécifie le délai de démarrage de la commande de démarrage. Reportez-vous à la section "Propriété d'extension `Commande_démarrage`" à la page 192 pour obtenir des informations complémentaires. La valeur par défaut de `Start_timeout` est de 300 secondes.

Propriété Stop_timeout

Cette propriété spécifie le délai d'arrêt de la commande d'arrêt. Reportez-vous à la section "Propriété Stop_command" à la page 193 pour obtenir des informations complémentaires. La valeur par défaut de Stop_timeout est de 300 secondes.

Propriété Probe_timeout

Cette propriété spécifie la valeur de délai d'attente de la commande de détection. Reportez-vous à la section "Propriété Commande_sonde" à la page 194 pour obtenir des informations complémentaires. La valeur par défaut de Probe_timeout est de 30 secondes.

Propriété Niveau_cont_fils

Remarque – Si vous utilisez les commandes d'administration de Sun Cluster, vous pouvez utiliser la propriété Child_mon_level. Si vous utilisez Agent Builder, vous ne pouvez pas utiliser cette propriété.

Cette propriété permet de contrôler les processus gérés par le PMF. Cette propriété indique le niveau maximal de surveillance des processus fils. Son fonctionnement est identique à celui de l'argument -C sur la commande pmfadm. Reportez-vous à la page de manuel 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 processus enfants et leurs descendants sont surveillés.

Propriété Basculement_activé

Cette propriété d'extension booléenne contrôle le comportement de basculement de la ressource. Si cette propriété d'extension est définie sur TRUE, l'application bascule lorsque le nombre de redémarrages dépasse le Retry_count au cours du nombre de secondes de Retry_interval.

Si elle est définie sur FALSE, l'application ne redémarre pas ou bascule sur un autre noeud lorsque le nombre de redémarrages dépasse le Retry_count au cours du nombre de secondes de Retry_interval.

Cette propriété peut être utilisée pour empêcher la ressource de l'application de basculer le groupe de ressources. La valeur par défaut de cette propriété est TRUE.

Propriété `Stop_signal`

Le GDS utilise la valeur de la propriété d'extension de ce nombre entier pour déterminer le signal utilisé pour arrêter l'application au moyen du PMF. Reportez-vous à la page de manuel `signal`(3HEAD) pour connaître la liste des valeurs de nombre entier que vous pouvez spécifier. La valeur par défaut est `15` (`SIGTERM`).

Propriété `Log_level`

Cette propriété indique le niveau, ou le type, de messages de diagnostic qui sont consignés par le GDS. Vous pouvez spécifier `NONE`, `INFO` ou `ERR` pour cette propriété. Lorsque vous choisissez `NONE`, les messages de diagnostic ne sont pas consignés par le module GDS. Lorsque vous spécifiez `INFO`, seuls les messages à titre d'information sont consignés. Lorsque vous spécifiez `ERR`, seuls les messages d'erreur sont consignés. Par défaut, le module GDS ne consigne pas les messages de diagnostic (`NONE`).

Utilisation de Agent Builder pour créer un service utilisant le GDS

Vous pouvez utiliser Agent Builder pour créer le service qui utilise le GDS. Agent Builder est décrit plus en détail à la section Chapitre 9.

Création et configuration des scripts GDS

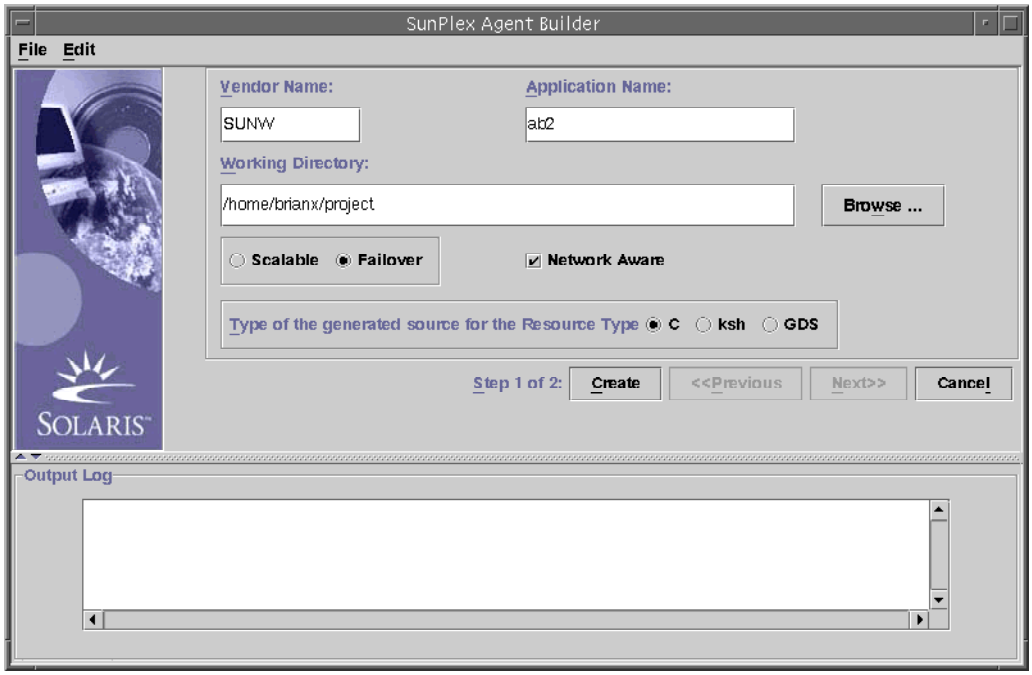
▼ Démarrage de Agent Builder et création des scripts

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Démarrage de Agent Builder.**

   ```
   # /usr/cluster/bin/scdsbuilder
   ```

3. **L'écran Créer de Agent Builder s'affiche.**



4. **Tapez le nom du fournisseur.**

5. **Tapez le nom de l'application.**

Remarque – Sous le système d'exploitation Solaris 9, l'association du nom du fournisseur et du nom de l'application peut dépasser neuf caractères. Toutefois, si vous utilisez une version précédente du système d'exploitation Solaris, cette association ne doit pas dépasser neuf caractères. Cette association sert de nom de package pour les scripts.

---

6. **Accédez au répertoire de travail.**

   Vous pouvez utiliser le menu déroulant Parcourir pour sélectionner le répertoire plutôt que de taper le chemin d'accès.

7. **Indiquez si le service de données est évolutif ou de basculement.**

   Vous n'avez pas besoin de sélectionner l'option Compatible réseau, car il s'agit du paramètre par défaut lorsque vous créez le GDS.

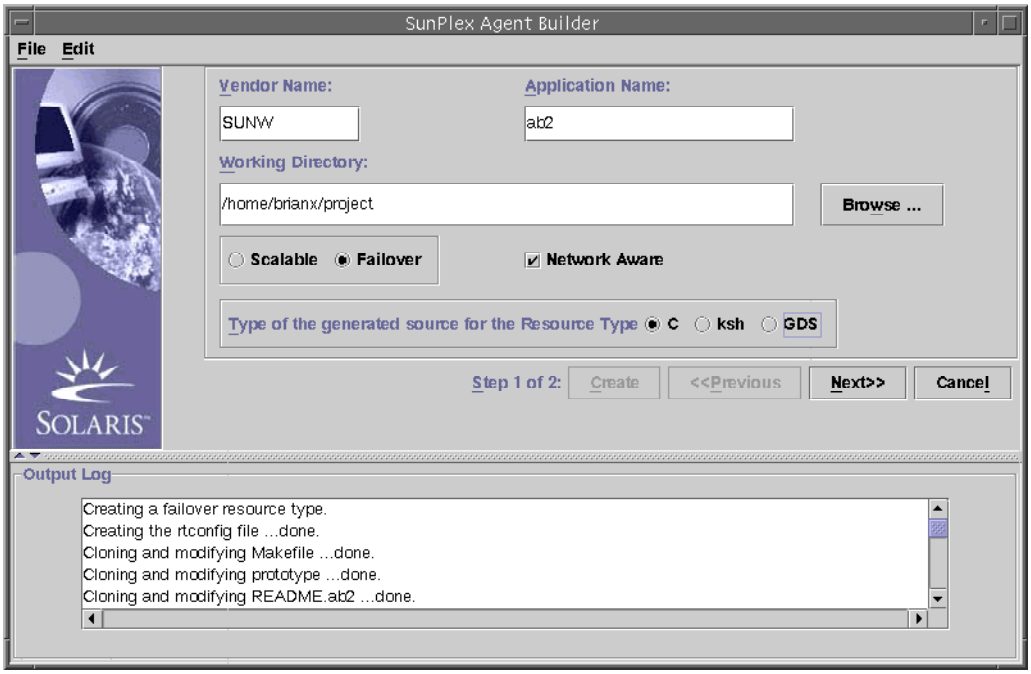
8. **Sélectionnez GDS.**

9. **(Facultatif) Changez la version de TR de la valeur par défaut indiquée.**

Remarque – Dans ce champ, vous ne pouvez utiliser aucun des caractères suivants : espace, tabulation, barre oblique (/), barre oblique inverse (\), astérisque (*), point d'interrogation (?), virgule (,), point virgule (;), crochet gauche ([) ou crochet droit (]).

10. **Cliquez sur Créer.**

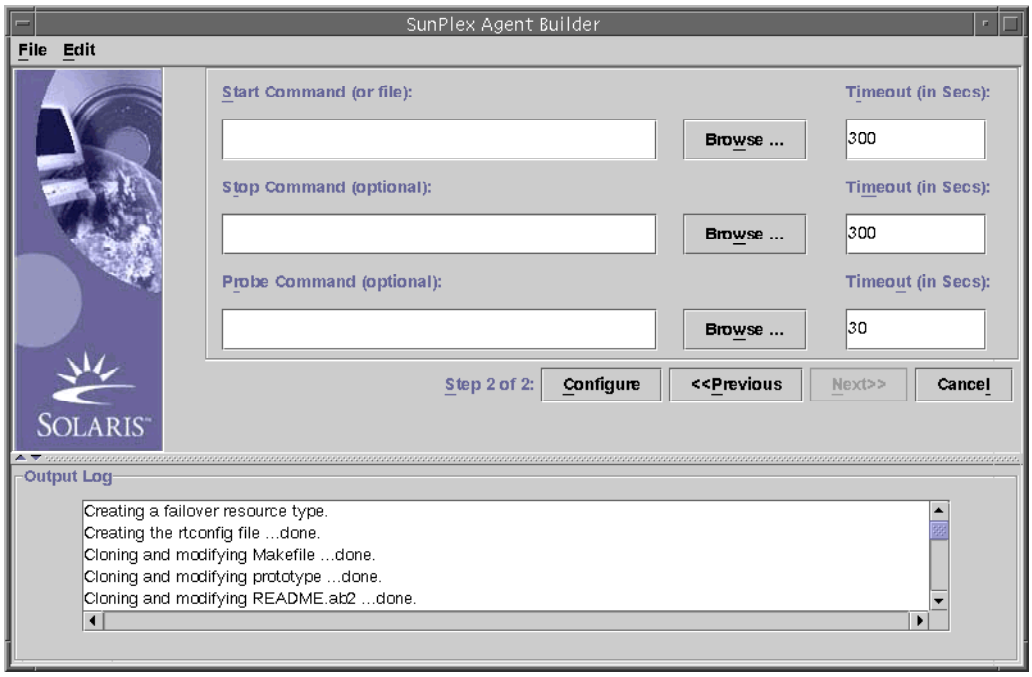
    Agent Builder crée les scripts. Les résultats sont affichés dans la zone Journal de sortie.



Notez que le bouton Créer est grisé. Vous pouvez à présent configurer les scripts.

11. **Cliquez sur Next.**

    L'écran Configurer s'affiche.



▼ Configuration des scripts

Après avoir créé les scripts, vous devez configurer le nouveau service.

Étapes

1. **Tapez l'emplacement de la commande de démarrage, ou cliquez sur Parcourir pour rechercher la commande de démarrage.**

   Vous pouvez indiquer des variables de propriété, Les variables de propriété sont décrites à la section "Utilisation des variables de propriété" à la page 174.

2. **(Facultatif) Tapez l'emplacement de la commande d'arrêt, ou cliquez sur Parcourir pour localiser la commande d'arrêt.**

   Vous pouvez indiquer des variables de propriété, Les variables de propriété sont décrites à la section "Utilisation des variables de propriété" à la page 174.

3. **(Facultatif) Tapez l'emplacement de la commande de détection, ou cliquez sur Parcourir pour localiser la commande de détection.**

   Vous pouvez indiquer des variables de propriété, Les variables de propriété sont décrites à la section "Utilisation des variables de propriété" à la page 174.

4. **(Facultatif) Spécifiez de nouvelles valeurs de délai d'attente pour les commandes de démarrage, d'arrêt et de détection.**

5. **Cliquez sur l'option Configurer.**
   Agent Builder configure les scripts.

   Remarque – Agent Builder concatène le nom du fournisseur et le nom de l'application pour créer le nom du package.

   Le package de scripts est créé et placé sous le répertoire suivant :

   working-dir/*vendor-name-application*/`pkg`

   Par exemple, `/export/wdir/NETapp/pkg`.

6. **Sur chaque noeud du cluster, devenez superutilisateur ou assumez une fonction équivalente.**

7. **Sur chaque noeud du cluster, installez le package complet.**

   - **Sous le système d'exploitation Solaris 10 dans un environnement à zones, en tant qu'administrateur global de la** *zone globale***, tapez les commandes suivantes :**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -G -d . NETapp
     ```

     Le package que vous avez spécifié est ajouté à la zone globale, à condition que son contenu n'affecte aucune partie de la zone globale qui est partagée avec une zone locale.

     Les fichiers suivants sont installés par `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
     ```

   - **Pour toute autre version du système d'exploitation Solaris ou pour le SE Solaris 10 dans un environnement sans zones, tapez les commandes suivantes :**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -d . NETapp
     ```

     Les fichiers ci-dessous sont installés par la commande `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 de script correspondent au nom de l'application que vous avez précédemment tapé sur l'écran Créer, précédé du nom de script (par exemple `startapp`).

---

8. **Sur l'un des noeuds du cluster, configurez les ressources et démarrez l'application.**

   # **/opt/NETapp/util/startapp -h** *logicalhostname* **-p** *port-and-protocol-list*

   Les arguments du script `startapp` varient en fonction du type de ressources : de basculement ou évolutives.

Remarque – Pour savoir quelle ligne de commande vous devez taper, consultez la page de manuel personnalisée ou exécutez le script `startapp` sans arguments pour afficher une instruction d'utilisation.

Pour afficher les pages de manuel, vous devez spécifier leur chemin d'accès. Par exemple, pour afficher la page de manuel `startapp`(1M), tapez :

```
# man -M /opt/NETapp/man startapp
```

Pour afficher une instruction d'utilisation, tapez :

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be
 specified. For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup-adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

Sortie de Agent Builder

Agent Builder génère trois scripts et un fichier de configuration en fonction des données que vous avez fournies lorsque vous avez créé le package. Le fichier de configuration spécifie les noms du groupe de ressources et du type de ressource.

Les scripts sont les suivants :

- **Script de démarrage** : Configure les ressources et démarre l'application qui est sous le contrôle du gestionnaire RGM.
- **Script d'arrêt** : Arrête l'application et consigne les ressources et les groupes de ressources.
- **Script de suppression** : Supprime les ressources et les groupes de ressources créés par le script de démarrage.

Ces scripts ont la même interface et le même comportement que les scripts d'utilitaire qui sont générés par Agent Builder pour les services de données non GDS. Les scripts sont regroupés dans un package Solaris que vous pouvez réutiliser sur plusieurs clusters.

Vous pouvez personnaliser le fichier de configuration pour donner vos propres noms aux groupes de ressources ou aux autres arguments qui correspondent normalement à des arguments de la commande `scrgadm`. Si vous ne personnalisez pas les scripts, Agent Builder fournit des valeurs par défaut aux arguments `scrgadm`.

Utilisation des commandes d'administration de Sun Cluster pour créer un service utilisant le GDS

Cette section explique comment ajouter des arguments au GDS. Vous devez utiliser les commandes existantes de Sun Cluster, telles que `scrgadm` et `scswitch`, pour gérer et administrer le GDS.

Si les scripts fournissent les fonctionnalités adéquates, vous n'avez pas besoin d'utiliser les commandes d'administration de niveau inférieur décrites dans cette section. Toutefois, vous pouvez les utiliser si vous souhaitez avoir un meilleur contrôle de la ressource GDS. Ces commandes sont exécutées par les scripts.

▼ Utilisation des commandes d'administration de Sun Cluster afin de créer un service à haut niveau de disponibilité utilisant le module GDS

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Enregistrez le type de ressource `SUNW.gds`.**

   ```
   # scrgadm -a -t SUNW.gds
   ```

3. **Créez le groupe de ressources contenant la ressource `LogicalHostname` et le service de basculement.**

   ```
   # scrgadm -a -g haapp_rg
   ```

4. **Créez la ressource de la ressource `LogicalHostname` .**

   ```
   # scrgadm -a -L -g haapp_rs -l hhead
   ```

5. **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
   ```

6. **Mettez en ligne le groupe de ressources `haapp_rg`.**

   ```
   # scswitch -Z -g haapp_rg
   ```

▼ Utilisation des commandes d'administration de Sun Cluster afin de créer un service évolutif utilisant le module GDS

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Enregistrez le type de ressource `SUNW.gds`.**

   ```
   # scrgadm -a -t SUNW.gds
   ```

3. **Créez le groupe de ressources de la ressource `SharedAddress` .**

   ```
   # scrgadm -a -g sa_rg
   ```

4. **Créez la ressource `SharedAddress` sur `sa_rg`.**

```
# scrgadm -a -S -g sa_rg -l hhead
```

5. **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
```

6. **Créez la ressource du 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
```

7. **Connectez le groupe de ressources contenant les ressources réseau.**

```
 # scswitch -Z -g sa_rg
```

8. **Mettez en ligne le groupe de ressources `app_rg`.**

```
# scswitch -Z -g app_rg
```

Interface de ligne de commande de Agent Builder

Agent Builder est doté d'une interface de ligne de commande dont les fonctionnalités sont identiques à celles de l'interface utilisateur graphique. Cette interface est constituée des commandes `scdscreate` et `scdsconfig`. Reportez-vous aux pages de manuel `scdscreate`(1HA) et `scdsconfig`(1HA).

▼ Utilisation de la version de ligne de commande de Agent Builder pour créer un service utilisant le GDS

Cette section explique comment utiliser l'interface de ligne de commande pour procéder aux mêmes étapes que celles décrites à la section "Utilisation de Agent Builder pour créer un service utilisant le GDS " à la page 196.

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Créez le service.**
   - Pour un service de basculement, entrez :

     # **scdscreate -g -V NET -T app -d /export/***wdir*
   - Pour un service évolutif, entrez :

     # **scdscreate -g -s -V NET -T app -d /export/***wdir*

   Remarque – L'argument `-d` est facultatif. Si vous ne le spécifiez pas, le répertoire dans lequel vous vous trouvez devient le répertoire de travail.

   ---

3. **Configurez le service.**

   # **scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" **
   -m "/export/app/bin/probe" -d /export/*réptravail*

   Vous pouvez indiquer des variables de propriété, Les variables de propriété sont décrites à la section "Utilisation des variables de propriété" à la page 174.

   Remarque – Seule la commande `start` est requise. Tous les autres arguments et options sont facultatifs.

   ---

4. **Sur chaque noeud du cluster, installez le package complet.**
   - **Sous l'environnement d'exploitation Solaris 10 dans un environnement à zones, en tant qu'administrateur global de la** *zone globale***, tapez les commandes suivantes :**

     # **cd /export/wdir/NETapp/pkg**
     # **pkgadd -G -d . NETapp**

     Le package que vous avez spécifié est ajouté à la zone globale, à condition que son contenu n'affecte aucune partie de la zone globale qui est partagée avec une zone locale.

     Les fichiers suivants sont installés par `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
```

- **Pour toute autre version du système d'exploitation Solaris ou pour le SE Solaris 10 dans un environnement sans zones, tapez les commandes suivantes :**

  ```
  # cd /export/wdir/NETapp/pkg
  # pkgadd -d . NETapp
  ```

  Les fichiers ci-dessous sont installés par la commande `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 de scripts correspondent au nom de l'application que vous avez saisi précédemment sur l'écran Créer, précédé du nom de script (par exemple `startapp`).

---

5. **Sur l'un des noeuds du cluster, configurez les ressources et démarrez l'application.**

   # **`/opt/NETapp/util/startapp -h`** *logicalhostname* **`-p`** *port-and-protocol-list*

   Les arguments du script `startapp` varient selon le type de ressource : de basculement ou évolutives.

Remarque – Pour savoir quelle ligne de commande vous devez taper, consultez la page de manuel personnalisée ou exécutez le script `startapp` sans arguments pour afficher une instruction d'utilisation.

Pour afficher les pages de manuel, vous devez spécifier leur chemin d'accès. Par exemple, pour afficher la page de manuel `startapp`(1M), tapez :

```
# man -M /opt/NETapp/man startapp
```

Pour afficher une instruction d'utilisation, tapez :

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be specified.
 For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup/adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

CHAPITRE 11

Fonctions de l'API de la bibliothèque DSDL

Ce chapitre répertorie et décrit brièvement les fonctions API de la bibliothèque de développement de service de données (DSDL). Reportez-vous aux pages man 3HA pour obtenir la description complète de chaque fonction DSDL. La bibliothèque DSDL fournit uniquement une interface C. Aucune interface de bibliothèque DSDL basée sur des scripts n'est disponible.

Ce chapitre contient les rubriques suivantes :

Fonctions polyvalentes

Les fonctions de cette section fournissent un large éventail de possibilités. Elles vous permettent d'effectuer les opérations suivantes :

- initialiser l'environnement DSDL ;
- récupérer les noms des ressources, des types et des groupes de ressources, ainsi que les valeurs des propriétés 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.

Fonctions d'initialisation

Les fonctions ci-dessous permettent d'initialiser la méthode d'appel :

- `scds_initialize`(3HA): affecte des ressources et initialise l'environnement DSDL.
- `scds_close`(3HA) : libère les ressources affectées par la fonction `scds_initialize()`.

Fonctions de récupération

Les fonctions suivantes récupèrent les informations concernant les types de ressources, les ressources, les groupes de ressources et les propriétés d'extension :

- `scds_get_resource_type_name`(3HA) : récupère le nom du type de ressource pour le programme d'appel.
- `scds_get_resource_name`(3HA) : récupère le nom de la ressource pour le programme d'appel.
- `scds_get_resource_group_name`(3HA) : récupère le nom du groupe de ressources pour le programme d'appel.
- `scds_get_ext_property`(3HA) : récupère la valeur de la propriété d'extension spécifiée.
- `scds_free_ext_property`(3HA) : libère la mémoire allouée par la fonction `scds_get_ext_property()`.

La fonction suivante permet de récupérer les informations relatives à l'état des ressources `SUNW.HAStoragePlus` utilisées par une ressource :

`scds_hasp_check`(3HA) : 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 système `Resource_dependencies` ou `Resource_dependencies_weak` définies pour cette ressource. Pour plus d'informations, consultez les pages de manuel `SUNW.HAStoragePlus`(5).

Fonctions de basculement et de redémarrage

Les fonctions suivantes permettent de basculer ou de redémarrer une ressource ou un groupe de ressources :

- `scds_failover_rg`(3HA) : bascule un groupe de ressources.
- `scds_restart_rg`(3HA) : redémarre un groupe de ressources.
- `scds_restart_resource`(3HA) : redémarre une ressource.

Fonctions d'exécution

Les deux fonctions suivantes exécutent une commande dans un délai imparti et convertissent un code d'erreur en message d'erreur :

- `scds_timerun`(3HA) : exécute une commande dans un délai d'attente imparti.
- `scds_error_string`(3HA) : convertit un 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 du type de ressource, de la ressource et du groupe de ressources appropriés, y compris à quelques propriétés d'extension couramment utilisées. La bibliothèque DSDL fournit la fonction `scds_initialize()`, qui permet d'analyser les arguments de la ligne de commande. Elle *met en cache* les diverses propriétés du type de ressource, de la ressource et du groupe de ressources appropriés.

La page de manuel `scds_property_functions`(3HA) décrit ces fonctions, parmi lesquelles :

- `scds_get_rt_`*property-name*
- `scds_get_rs_`*property-name*
- `scds_get_rg_`*property-name*
- `scds_get_ext_`*property-name*

Fonctions d'accès aux ressources réseau

Les fonctions répertoriées dans cette section permettent de récupérer, d'imprimer et de libérer les ressources réseau utilisées par les ressources et les groupes de ressources. Les fonctions `scds_get_` de cette section permettent de récupérer facilement des ressources réseau sans avoir à utiliser les fonctions APIGR pour interroger des propriétés spécifiques, telles que `Network_resources_used` et `Port_list`. Les fonctions `scds_print_`*name*`()` impriment des valeurs à partir des structures de données qui sont renvoyées par les fonctions `scds_get_`*name* `()`. Les fonctions `scds_free_`*name* `()` libèrent la mémoire allouée par les fonctions `scds_get_`*name*`()`.

Fonctions relatives aux noms d'hôtes

Les fonctions suivantes gèrent les noms d'hôtes :

- `scds_get_rs_hostnames`(3HA) : récupère une liste de noms d'hôtes utilisés par la ressource.
- `scds_get_rg_hostnames`(3HA) : récupère une liste de noms d'hôtes utilisés par les ressources réseau dans un groupe de ressources.
- `scds_print_net_list`(3HA) : imprime le contenu de la liste de noms d'hôtes renvoyée par `scds_get_rs_hostnames ()` ou `scds_get_rg_hostnames()`.
- `scds_free_net_list`(3HA) : libère la mémoire allouée par `scds_get_rs_hostnames()` ou `scds_get_rg_hostnames()`.

Fonctions relatives aux listes de ports

Les fonctions suivantes gèrent les listes de ports :

- `scds_get_port_list`(3HA) : récupère une liste des paires port/protocole utilisées par une ressource.
- `scds_print_port_list`(3HA) : imprime le contenu de la liste de paires port/protocole renvoyée par la fonction `scds_get_port_list()`.
- `scds_free_port_list`(3HA) : libère la mémoire allouée par la fonction `scds_get_port_list()`.

Fonctions relatives aux adresses réseau

Les fonctions suivantes gèrent les adresses réseau :

- `scds_get_netaddr_list`(3HA) : récupère une liste d'adresses réseau utilisées par une ressource.
- `scds_print_netaddr_list`(3HA) : imprime le contenu de la liste d'adresses réseau renvoyée par la fonction `scds_get_netaddr_list()`.
- `scds_free_netaddr_list`(3HA) : libère la mémoire allouée par la fonction `scds_get_netaddr_list()`.

Détection des pannes à l'aide de connexions TCP

Les fonctions de cette section permettent un contrôle TCP. En règle générale, un détecteur de pannes utilise ces fonctions pour établir une connexion de prise unique à un service, pour lire et pour écrire des données relatives au service afin de déterminer son état, puis se déconnecter du service.

Ces fonctions incluent :

- `scds_fm_tcp_connect`(3HA) : établit une connexion TCP à un processus qui utilise uniquement l'adressage IPv4.
- `scds_fm_net_connect`(3HA) : établit une connexion TCP à un processus qui utilise l'adressage IPv4 ou IPv6.
- `scds_fm_tcp_read`(3HA) : utilise une connexion TCP pour lire des données à partir du processus contrôlé.
- `scds_fm_tcp_write`(3HA) : utilise une connexion TCP pour écrire des données dans un processus contrôlé.
- `scds_simple_probe`(3HA) : détecte un processus en établissant et en mettant fin à une connexion TCP sur ce processus. Cette fonction gère uniquement les adresses IPv4.
- `scds_simple_net_probe`(3HA) : détecte un processus en établissant et en mettant fin à une connexion TCP sur ce processus. Cette fonction gère les adresses IPv4 et IPv6.
- `scds_fm_tcp_disconnect`(3HA) : met fin à la connexion sur un processus contrôlé. Cette fonction gère uniquement les adresses IPv4.
- `scds_fm_net_disconnect`(3HA) : met fin à la connexion sur un processus contrôlé. Cette fonction gère les adresses IPv4 et IPv6.

Fonctions PMF

Ces fonctions encapsulent la fonctionnalité PMF. Le modèle DSDL de contrôle utilisant la fonction PMF crée et utilise des valeurs implicites *tag* pour `pmfadm`. Reportez-vous à la rubrique `pmfadm`(1M) pour obtenir de plus amples informations.

La fonction PMF utilise également des valeurs implicites pour `Restart_interval` , `Retry_count` et `action_script` (les options `-t`, `-n` et `-a` de `pmfadm`). Ce qu'il faut retenir, c'est que la bibliothèque DSDL lie l'historique des pannes de processus (détectées par la fonction PMF) à celui des pannes de l'application (repérées par le détecteur de pannes) avant de choisir entre le redémarrage et le basculement.

Cette section présente les fonctions suivantes :

- `scds_pmf_get_status`(3HA) : détermine si l'instance spécifiée est contrôlée par la fonction PMF.
- `scds_pmf_restart_fm`(3HA) : utilise la fonction PMF pour redémarrer le détecteur de pannes.
- `scds_pmf_signal`(3HA) : envoie le signal spécifié à une arborescence de processus contrôlée par la fonction PMF.
- `scds_pmf_start`(3HA) : exécute un programme spécifié (y compris un détecteur de pannes) sous le contrôle de la fonction PMF.

- `scds_pmf_stop`(3HA) : met fin à un processus fonctionnant sous le contrôle de la fonction PMF.
- `scds_pmf_stop_monitoring`(3HA) : arrête la fonction de surveillance d'un processus contrôlé par la fonction PMF.

Fonctions du détecteur de pannes

Les fonctions présentées dans cette section proposent un modèle prédéterminé de détection des pannes en conservant l'historique des erreurs et en les évaluant avec les propriétés `Retry_count` et `Retry_interval`.

Cette section présente les fonctions suivantes :

- `scds_fm_sleep`(3HA) : attend un message du socket de contrôle du détecteur de pannes.
- `scds_fm_action`(3HA) : effectue une action une fois la détection terminée.
- `scds_fm_print_probes`(3HA) : complète le journal système avec les informations sur l'état de la détection.

Fonctions de l'utilitaire

Les fonctions suivantes vous permettent d'écrire des messages et des messages de débogage dans le journal système :

- `scds_syslog`(3HA) : écrit des messages dans le journal système.
- `scds_syslog_debug`(3HA) : écrit des messages de débogage dans le journal système.

CHAPITRE 12

Protocole de notification des reconfigurations de clusters

Ce chapitre présente le protocole CRNP (Cluster Reconfiguration Notification Protocol, protocole de notification des reconfigurations de clusters). Ce protocole permet aux applications évolutives et de basculement de "reconnaître les clusters." Plus précisément, il propose un mécanisme qui permet aux applications de s'enregistrer et par la suite, d'être informées de manière 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'appartenance à un cluster change et que l'état d'un groupe de ressources ou d'une ressource est modifié.

Remarque – L'implémentation du type de ressources `SUNW.Event` fournit des services CRNP à haute disponibilité sur Sun Cluster. Cette implémentation est décrite plus en détail à la page de manuel `SUNW.Event`(5).

Ce chapitre contient les rubriques suivantes :

Concepts du protocole CRNP

Le protocole CRNP définit les couches application, présentation et session de la pile de protocoles 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.

Fonctionnement du protocole CRNP

Le protocole CRNP propose des mécanismes et des démons qui génèrent des événements de reconfiguration de cluster, les acheminent via le cluster et les transmettent aux clients intéressés.

Le démon `cl_apid` interagit avec les clients. Le gestionnaire de groupe de ressources (RGM, Resource Group Manager) de Sun Cluster génère des événements de reconfiguration de cluster. Ce démon utilise `sysevent`d pour transmettre des événements à chaque noeud local. Le démon `cl_apid` communique avec les clients intéressés, via le protocole TCP/IP, en langage XML (Extensible Markup Language).

Le schéma suivant met en évidence 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.

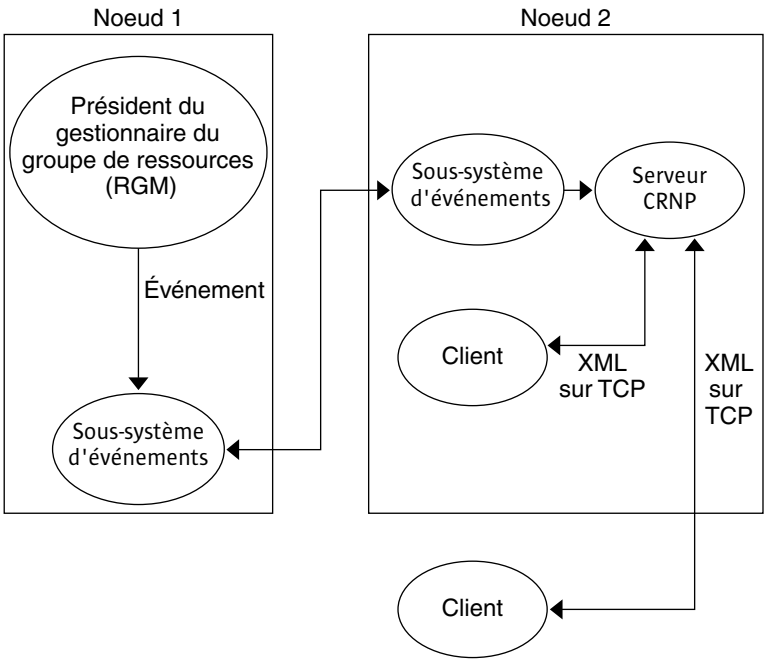


FIGURE 12–1 Flux d'événements entre les composants CRNP

Sémantique du protocole CRNP

Les clients débutent des communications en envoyant un message d'enregistrement (`SC_CALLBACK_RG`) au serveur. Ce message indique 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 à 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 client en mémoire et en conserve la trace même après la réinitialisation du cluster.

Les clients annulent leur enregistrement en envoyant un message d'enregistrement (`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 schéma suivant met en évidence le flux de communications entre un client et un serveur.

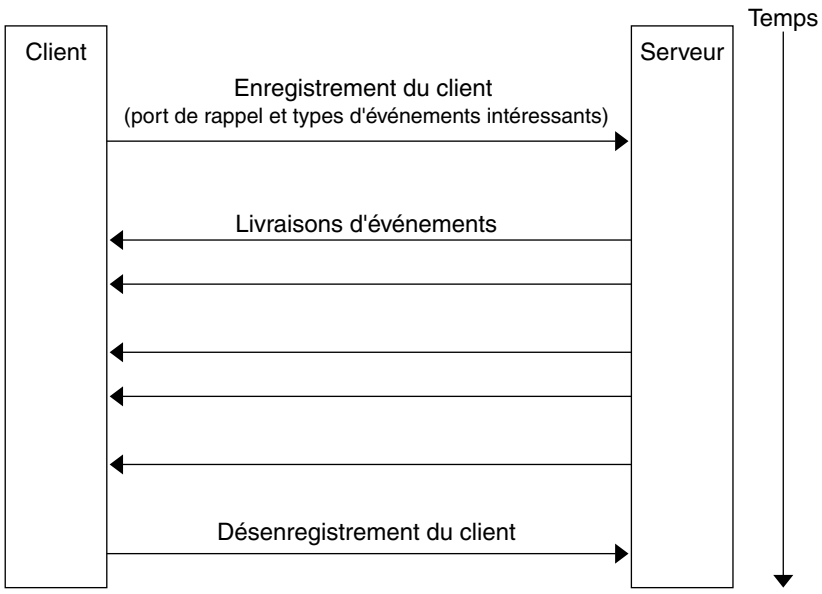


FIGURE 12–2 Flux de communications entre un client et un serveur

Types de messages utilisés par le protocole CRNP

Le protocole CRNP utilise trois types de messages XML. Leur utilisation est décrite dans le tableau suivant. Ces trois types de messages sont également décrits plus en détail dans ce chapitre.

Type de message utilisé par le protocole CRNP	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). ADD_CLIENT, ADD_EVENTS et REMOVE_EVENTS contiennent également une liste non bornée de types d'événements, comprenant chacun les informations suivantes : ■ classe de l'événement ; ■ sous-classe de l'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 du type de document) qui permet de générer les classes de SC_CALLBACK_REG est SC_CALLBACK_REG. Elle est décrite plus en détail à l'Annexe F.
SC_REPLY	Ce message contient les informations suivantes : ■ version du protocole ; ■ code d'erreur ; ■ message d'erreur. La DTD qui permet de générer les classes de SC_REPLY est SC_REPLY. Elle est décrite plus en détail à l'Annexe F.

SC_EVENT	Ce message contient les informations suivantes :
	■ version du protocole ;
	■ classe de l'événement ;
	■ sous-classe de l'événement ;
	■ fournisseur ;
	■ éditeur ;
	■ liste des paires nom/valeurs (0, 1 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 qui permet de générer les classes de SC_EVENT est SC_EVENT. Elle est décrite plus en détail à l'Annexe F.

Processus d'enregistrement d'un client auprès du serveur

Cette section indique comment un administrateur de cluster configure le serveur, comment les clients sont identifiés, comment les informations sont envoyées via les couches application et session, ainsi que les conditions d'erreur.

Hypothèses sur la configuration du serveur par les administrateurs

L'administrateur du cluster doit configurer le serveur à l'aide d'une adresse IP à haute disponibilité (une adresse IP qui n'est pas liée à une machine en particulier du cluster) et d'un numéro de port. L'administrateur du cluster doit transmettre cette adresse réseau aux clients potentiels. Le protocole CRNP ne définit pas comment ce nom de serveur est rendu accessible aux clients. L'administrateur du cluster utilise un service d'attribution de noms, ce qui permet aux clients de trouver l'adresse réseau du serveur de manière dynamique, ou ajoute le nom du réseau à un fichier de configuration que le client doit lire. Le serveur fonctionne au sein du cluster comme un type de ressource de basculement.

Processus d'identification d'un client par le serveur

Chaque client est identifié de manière unique par 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 considère que les messages `SC_CALLBACK_REG` suivants ayant la même adresse de rappel proviennent du même client, même si le port source de transmission des messages est différent.

Processus de transmission des messages `SC_CALLBACK_REG` entre un client et le serveur

Pour procéder à l'enregistrement, le client commence par ouvrir une connexion TCP sur l'adresse IP et le numéro de port du serveur. Une fois la connexion TCP établie et prête pour l'écriture, le client doit envoyer son message d'enregistrement. Il doit s'agir d'un message `SC_CALLBACK_REG` correctement formaté, qui ne doit pas être suivi ni précédé d'octets supplémentaires.

Une fois tous les octets émis, le client doit rester connecté pour recevoir une réponse du serveur. S'il ne formate pas le message correctement, le serveur n'enregistre pas le client et lui transmet un message d'erreur. Toutefois, si le client ferme la connexion (au niveau du socket) avant que le serveur n'ait envoyé sa réponse, ce dernier enregistre le client en suivant la procédure habituelle.

Les clients peuvent 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, non fonctionnel ou non valide, il renvoie un message d'erreur au client.

Pour transmettre un message `ADD_EVENTS`, `REMOVE_EVENTS` ou `REMOVE_CLIENT`, un client doit d'abord envoyer un message `ADD_CLIENT`. De même, avant de transmettre un message `REMOVE_CLIENT`, il doit envoyer un message `ADD_CLIENT`.

Si un client envoie un message `ADD_CLIENT` alors qu'il est déjà enregistré, le serveur peut tolérer ce message. Il remplace l'enregistrement client antérieur par le nouvel enregistrement spécifié dans le second message `ADD_CLIENT`.

Mais en règle générale, un client s'enregistre une seule fois auprès du serveur par l'envoi d'un message `ADD_CLIENT`. Un client annule une seule fois son enregistrement en envoyant un message `REMOVE_CLIENT` au serveur. 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`, `REMOVE_CLIENT`, `ADD_EVENTS` et `REMOVE_EVENTS` contient une liste d'événements. Le tableau ci-après présente les types d'événements que le protocole CRNP accepte, avec les paires nom/valeurs requises.

Si un client effectue l'une des actions suivantes, le serveur ignore ces messages sans vous en avertir :

- Il envoie un message `REMOVE_EVENTS` qui spécifie un ou plusieurs types d'événements pour lesquels il ne s'est pas enregistré précédemment.
- Il s'enregistre deux fois pour le même type d'événement.

Classe et sous-classe	Paire nom/valeurs	Description
`EC_Cluster` `ESC_cluster_membership`	Requise : aucune En option : aucune	Enregistrement pour toutes les notifications d'événement de modification de l'appartenance au cluster (suppression ou ajout d'un noeud)

Classe et sous-classe	Paire nom/valeurs	Description
`EC_Cluster` `ESC_cluster_rg_state`	Une seule requise comme suit : `rg_`*name* Type de valeurs : chaîne de caractères En option : aucune	Enregistrement pour toutes les notifications d'événements de modification d'état du groupe de ressources *name*
`EC_Cluster` `ESC_cluster_r_state`	Une seule requise comme suit : `r_`*name* Type de valeurs : chaîne de caractères En option : aucune	Enregistrement pour toutes les notifications d'événement de modification d'état de la ressource *nom*
`EC_Cluster` Aucun	Requise : aucune En option : aucune	Enregistrement pour toutes les notifications d'événement de Sun Cluster

Processus de réponse du client au serveur

Une fois l'enregistrement traité, le serveur qui a reçu la demande d'enregistrement envoie le message `SC_REPLY` sur la connexion TCP ouverte par le client. Le serveur ferme la connexion. Le client doit laisser 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 un message `SC_REPLY` du serveur.
5. Reçoit un message `SC_REPLY` du serveur.
6. Reçoit un message stipulant que le serveur a fermé la connexion (lecture de 0 octet du socket).
7. Ferme la connexion.

Un peu plus tard, 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 REMOVE_CLIENT).
4. Attend un message SC_REPLY du serveur.
5. Reçoit un message SC_REPLY du serveur.
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, il envoie, sur la même connexion ouverte, un message SC_REPLY indiquant si l'opération a réussi ou échoué. "DTD XML SC_REPLY " à la page 346 contient la DTD XML d'un message SC_REPLY, ainsi que les éventuels messages d'erreur que ce message peut contenir.

Contenu d'un message SC_REPLY

Un message SC_REPLY indique si une opération a réussi ou a échoué. Il contient la version du message de protocole CRNP, un code d'état et un message d'état décrivant le code d'état plus en détail. 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 temporaire. Le client doit faire une nouvelle tentative de connexion avec d'autres arguments.
LOW_RESOURCE	Les ressources du cluster sont faibles, le client peut seulement réessayer ultérieurement. L'administrateur du cluster peut également en augmenter les ressources.
SYSTEM_ERROR	Un incident grave s'est produit. Contactez l'administrateur du cluster.
FAIL	Le refus d'une autorisation ou tout autre incident a provoqué l'échec de la connexion.
MALFORMED	La requête XML est malformée et n'a pas pu être analysée.
INVALID	La requête XML n'est pas valide, c'est-à-dire qu'elle n'est pas conforme aux spécifications XML.
VERSION_TOO_HIGH	La version du message est trop élevée pour que le message soit traité avec succès.

Code d'état	Description
VERSION_TOO_LOW	La version du message est trop basse pour que le message soit traité avec succès.

Processus de gestion des conditions d'erreur par un client

Dans des conditions normales, un client qui envoie un message SC_CALLBACK_REG reçoit une réponse précisant si l'enregistrement a réussi ou a échoué.

Toutefois, le serveur peut être confronté à une condition d'erreur pendant l'enregistrement, ce qui l'empêche de transmettre un message SC_REPLY au client. L'enregistrement a pu réussir avant l'erreur, échouer ou ne pas avoir été traité.

Étant donné que le serveur doit fonctionner en tant que serveur de basculement ou de haute disponibilité dans le cluster, cette condition d'erreur ne signifie pas que le service est interrompu. De fait, le serveur peut rapidement commencer à envoyer des événements au client nouvellement enregistré.

Pour pallier ces conditions, votre client doit effectuer les opérations suivantes :

- Imposer un délai d'expiration au niveau de l'application pour les connexions d'enregistrement en attente d'un message SC_REPLY, délai après lequel il devra réessayer de s'enregistrer.
- Commencer à écouter son adresse IP de rappel et son numéro de port en vue de recevoir des notifications d'événement avant d'être enregistré pour la notification de rappel d'événement. Il doit attendre un message de confirmation d'enregistrement et des notifications d'événements en parallèle. S'il commence à recevoir des notifications d'événements avant de recevoir un message de confirmation, il doit fermer la connexion d'enregistrement.

Processus de notification des événements au client par le serveur

À mesure que les événements sont générés dans le cluster, le serveur CRNP les notifie à tous les clients qui ont demandé à être avertis de ce type d'événements. 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.

Aussitôt un client enregistré pour un type d'événement (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 peut déterminer 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 un message SC_EVENT du serveur.
4. Lit un message SC_EVENT du serveur.
5. Reçoit un message stipulant que le serveur a fermé la connexion (lecture de 0 octet du socket).
6. Ferme la connexion.

Une fois enregistrés, tous les clients doivent écouter leur adresse de rappel (adresse IP et numéro de port) en permanence afin de recevoir les éventuelles connexions entrantes 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. Pour recevoir d'autres notifications d'événements, le client doit se ré-enregistrer en envoyant un autre message SC_CALLBACK_REG contenant un message ADD_CLIENT.

Mécanisme garantissant la notification des événements

Au sein du cluster, l'avertissement des clients suit l'ordre de génération des événements. En d'autres termes, si un événement A est généré au sein du cluster avant un événement B, le client X reçoit l'événement A avant l'événement B. Cependant, l'ordre d'envoi des notifications d'événements à *tous* les clients *n'*est *pas* conservé. Cela signifie que le client Y peut recevoir les événements A et B avant que le client X ne soit notifié 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 qui suivent 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 », c'est-à-dire que le serveur peut envoyer la même notification d'événement à un client plus d'une fois. Cette autorisation est nécessaire lorsque le serveur s'arrête momentanément et qu'il est incapable de déterminer si le client a reçu les toutes dernières informations lorsqu'il se remet à fonctionner.

Contenu d'un message SC_EVENT

Le message SC_EVENT contient le message réel qui est généré au sein du cluster, retranscrit au format de message XML SC_EVENT. Le tableau ci-dessous présente les types d'événements que le protocole CRNP envoie, avec les paires nom/valeurs requises, l'éditeur et le fournisseur.

Remarque – Les positions des éléments du tableau 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.

Les autres noms commençant par ev_ et les valeurs connexes peuvent être présents, mais ne sont pas destinés à être utilisés par les clients.

Classe et sous-classe	Éditeur et fournisseur	Paire nom/valeurs
EC_Cluster	Éditeur : rgm	Nom : node_list
ESC_cluster_membership	Fournisseur : SUNW	Type de valeurs : tableau de chaînes de caractères
		Nom : state_list
		Le tableau state_list contient uniquement les numéros au format ASCII. Chaque numéro représente le numéro actuel du nœud 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 nœud est un membre du cluster.
		Type de valeurs : tableau de chaînes de caractères

Classe et sous-classe	Éditeur et fournisseur	Paire nom/valeurs
EC_Cluster	Éditeur : rgm	Nom : rg_name
ESC_cluster_rg_state	Fournisseur : SUNW	Type de valeurs : chaîne de caractères
		Nom : node_list
		Type de valeurs : tableau de chaînes de caractères
		Nom : state_list
		Le tableau state_list contient les représentations (sous la forme de chaînes de caractères) de l'état du groupe de ressources. Les valeurs valides sont celles que vous pouvez extraire à l'aide des commandes scha_cmds(1HA).
		Type de valeurs : tableau de chaînes de caractères
EC_Cluster	Éditeur : rgm	Nom : r_name
ESC_cluster_r_state	Fournisseur : SUNW	Type de valeurs : chaîne de caractères
		Nom : node_list
		Type de valeurs : tableau de chaînes de caractères
		Nom : state_list
		Le tableau state_list contient les représentations (en chaînes de caractères) de l'état de la ressource. Les valeurs valides sont celles que vous pouvez extraire à l'aide des commandes scha_cmds(1HA).
		Type de valeurs : tableau de chaînes de caractères

Processus d'authentification des clients et du serveur par le protocole CRNP

Le serveur authentifie un client à l'aide d'une forme de wrappers TCP. L'adresse IP source du message d'enregistrement, qui sert également d'adresse IP de rappel à laquelle les notifications d'événements sont envoyées, doit figurer dans la liste des clients autorisés sur le serveur. En outre, cette adresse et ce message ne doivent pas figurer dans la liste des clients refusés. Si ces informations ne figurent pas dans la liste des clients autorisés, le serveur refuse la requête et renvoie un message d'erreur au client.

Lorsque le serveur reçoit un message `SC_CALLBACK_REG ADD_CLIENT` , les messages `SC_CALLBACK_REG` suivants envoyés par ce client doivent utiliser la même adresse IP source que le premier message. Si le serveur CRNP reçoit un message `SC_CALLBACK_REG` non conforme à cette spécification, il effectue l'une des opérations suivantes :

- Il ignore la requête et envoie un message d'erreur au client.
- Il considère que la requête provient d'un nouveau client, selon le contenu du message `SC_CALLBACK_REG`.

Ce mécanisme de sécurité permet de prévenir les attaques de refus de service, où une personne tente d'annuler l'enregistrement d'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 d'enregistrement et au numéro de port qu'ils ont eux-mêmes utilisés.

Étant donné que les clients du service CRNP sont supposés se trouver à l'intérieur d'un pare-feu qui protège le cluster, le protocole CRNP ne comprend pas de mécanismes de sécurité supplémentaires

Exemple de création d'une application Java utilisant le protocole CRNP

L'exemple suivant montre comment développer une application Java simple nommée `CrnpClient` et qui utilise le protocole CRNP. Cette application enregistre le client, en vue de la réception de rappels d'événements, auprès du serveur CRNP sur le cluster ; elle écoute les rappels, puis traite les événements en imprimant leur contenu. Pour terminer, elle supprime les rappels d'événements.

Gardez ces informations à l'esprit lorsque vous étudiez cet exemple :

- Cet exemple d'application génère et analyse du code XML à l'aide de l'API Java pour le traitement XML (JAXP). Il ne vous explique pas comment utiliser JAXP. JAXP est décrit plus en détail à l'adresse `http://java.sun.com/xml/jaxp/index.html`.
- Cet exemple ne présente que des parties de l'application, que vous trouverez dans son intégralité à l'Annexe G. Pour illustrer des concepts en particulier de manière plus efficace, l'exemple de ce chapitre est légèrement différent de l'application complète qui est présentée à l'Annexe G.
- Pour plus de concision, les commentaires ont été supprimés de l'échantillon de code dans ce chapitre. L'application complète disponible à l'Annexe G inclut les commentaires.

- 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

Étapes

1. **Téléchargez puis installez JAXP ainsi que la version appropriée du compilateur Java et de la machine virtuelle.**

   Vous trouverez des instructions à l'adresse `http://java.sun.com/xml/jaxp/index.html`.

   Remarque – Cet exemple requiert au minimum la version Java 1.3.1.

2. **Placez-vous dans le répertoire qui contient le fichier source et tapez les lignes suivantes :**

   ```
   % javac -classpath jaxp-root/dom.jar:jaxp-rootjaxp-api. \
   jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
   .jar:jaxp-root/xsltc.jar -sourcepath . source-filename.java
   ```

   où *jaxp-root* est le chemin d'accès, absolu ou relatif, au répertoire contenant les fichiers jar de JAXP et *source-filename* est le nom de votre fichier source Java.

   Le fait de spécifier un `classpath` dans votre ligne de commande de compilation garantit que le programme de compilation trouvera les fichiers de classe JAXP.

3. **Lors de l'exécution de l'application, spécifiez le chemin `classpath` de manière à ce que l'application puisse charger les bons fichiers de classe JAXP (le premier chemin d'accès spécifié dans `classpath` correspond au répertoire actuel) :**

   ```
   % java -cp .:jaxp-root/dom.jar:jaxp-rootjaxp-api. \
   jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
   .jar:jaxp-root/xsltc.jar source-filename arguments
   ```

   L'environnement est à présent configuré. Vous pouvez développer l'application.

▼ Premières étapes du développement de l'application

Dans cette partie de l'exemple, vous allez créer une classe de base appelée `CrnpClient` , avec une méthode principale qui analyse les arguments de 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 déconnecte.

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.

Étape

- **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 expliquées plus loin dans ce chapitre. Le code qui permet d'importer 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 ultérieurement.

▼ Analyse des arguments de ligne de commande

Étape

- **Pour savoir comment analyser les arguments de ligne de commande, reportez-vous au code à 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 abordée plus loin dans ce chapitre.

Étapes

1. **Dans votre code, définissez une sous-classe `Thread` appelée `EventReceptionThread` qui crée une classe `ServerSocket` et attend que les événements arrivent sur le socket.**

   Dans cette partie du code, les événements ne sont ni lus ni traités. La lecture et le traitement des événements sont abordés plus loin dans ce chapitre. La classe `EventReceptionThread` crée une classe `ServerSocket` sur une adresse générique de protocole inter-réseau. `EventReceptionThread` conserve également l'objet `CrnpClient` en référence afin de pouvoir lui transmettre des événements à traiter.

```
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();
                                // Construct event from the sock stream and process it
                                sock.close();
                        }
                        // UNREACHABLE

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* private member variables */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. **Créez un objet `createEvtRecepThr`.**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
```

```
}
```

▼ Enregistrement et annulation de l'enregistrement aux rappels

L'enregistrement implique les opérations suivantes :

- ouverture d'un socket TCP de base sur le port et le protocole inter-réseau 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.

Étapes

1. **Créez le code Java mettant en oeuvre la logique précédente.**

   L'exemple de code 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 plus en détail 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 œuvre 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 plus loin dans ce chapitre.

```
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 que vous avez rédigé l'intégralité du code de mise en réseau, vous allez rédiger le code qui permet de générer et d'analyser le langage XML. Commencez par rédiger le code qui permet de générer 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 `http://java.sun.com/xml/jaxp/index.html` décrit le code de cette méthode plus en détail.

La mise en oeuvre de la classe `Event` est présentée dans l'exemple de code suivant. La classe `CallbackReg` utilise une classe `Event` qui mémorise un événement et peut le convertir en `Element` XML.

Étapes **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) {
                // Parser with specified options can't be built
                pce.printStackTrace();
                System.exit(1);
        }

        // Create the root element
        Element root = (Element) document.createElement("SC_CALLBACK_REG");

        // Add the attributes
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Add the events
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(document));
        }
        document.appendChild(root);

        // Convert the whole thing to a string
        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`.**

   La classe `CallbackReg` utilise une classe `Event`, qui elle-même utilise 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 d'enregistrement et d'annulation de l'enregistrement

Maintenant que vous avez créé les classes d'assistant qui permettent de générer les messages XML, vous pouvez écrire la mise en oeuvre de la méthode `createRegistrationString`. Cette méthode est appelée par la méthode `registerCallbacks`, décrite à la section "Enregistrement et annulation de l'enregistrement aux rappels" à la page 233.

`createRegistrationString` crée un objet `CallbackReg` et définit son type et son port de connexion. `createRegistrationString` crée ensuite plusieurs 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`.

La variable membre `regs` enregistre les arguments de 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 s'enregistrer. Le quatrième argument spécifie le type d'enregistrement, mais il est ignoré dans cet exemple. L'intégralité du code, que vous trouverez à l'Annexe G explique comment utiliser ce quatrième argument.

Étapes **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);

        // add the events
        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 d'annulation de l'enregistrement.**

   Il est plus facile de créer une chaîne de caractères d'annulation d'enregistrement qu'une chaîne de caractères d'enregistrement, car vous n'avez pas à prendre en compte 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

Vous avez créé le code de génération XML et de gestion de réseaux de l'application. Le constructeur `CrnpClient` appelle une méthode `setupXmlProcessing` . Celle-ci crée un objet `DocumentBuilderFactory` et définit plusieurs propriétés d'analyse sur cet objet. Cette méthode est décrite plus en détail dans la documentation JAXP. Voir `http://java.sun.com/xml/jaxp/index.html`.

Étape

- **Créez le code Java mettant en oeuvre la logique précédente.**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        // We don't need to bother validating
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // We want to ignore comments and whitespace
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Coalesce CDATA sections into TEXT nodes.
        dbf.setCoalescing(true);
}
```

▼ Analyse de la réponse d'enregistrement

Pour pouvoir analyser le message XML `SC_REPLY` que le serveur CRNP envoie en réponse à un message d'enregistrement ou d'annulation d'un enregistrement, 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 la méthode d'analyse de ce document. Cette méthode est décrite plus en détail dans la documentation JAXP disponible à l'adresse `http://java.sun.com/xml/jaxp/index.html`.

Étapes

1. **Créez le code Java mettant en oeuvre la logique précédente.**

   La méthode `readRegistrationReply` utilise la nouvelle classe `RegReply`.

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();
```

```
                db.setErrorHandler(new DefaultHandler());

                //parse the input file
                Document doc = db.parse(stream);

                RegReply reply = new RegReply(doc);
                reply.print(System.out);
        }
```

2. **Mettez en oeuvre la classe `RegReply`.**

   La méthode `retrieveValues` parcourt l'arborescence DOM dans le document XML et extrait le code et le message d'état. Pour en savoir plus, consultez la documentation JAXP à l'adresse `http://java.sun.com/xml/jaxp/index.html`.

```
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;

                // Find the SC_REPLY element.
                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);

                // Retrieve the value of the statusCode attribute
```

```
            statusCode = ((Element)n).getAttribute("STATUS_CODE");

            // Find the SC_STATUS_MSG element
            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;
            }
            // Get the TEXT section, if there is one.
            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;
            }

            // Retrieve the value
            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 section "Génération du langage XML" à la page 234, afin qu'elle puisse créer un `Event` à partir d'un document XML et un `Element` XML. Pour effectuer cette modification, vous aurez besoin d'un constructeur supplémentaire (exécutant un document XML), une méthode `retrieveValues`, de l'ajout de deux variables membres (`vendor` et `publisher`), de méthodes de mécanisme d'accès à tous les champs et d'une méthode d'impression.

Étapes **1. Créez le code Java mettant en oeuvre la logique précédente.**

Ce code est identique à celui de la classe `RegReply` décrit à la section "Analyse de la réponse d'enregistrement" à la page 240.

```
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;

        // Find the SC_EVENT element.
        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);

        //
        // Retrieve the values of the CLASS, SUBCLASS,
        // VENDOR and PUBLISHER attributes.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Retrieve all the nv pairs
        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` (cf Étape 1) doivent être identiques à celles 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;

                // Find the NAME element
                nl = elem.getElementsByTagName("NAME");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "NAME node.");
                   return;
                }
                // Get the TEXT section
                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;
                }

                // Retrieve the value
                name = n.getNodeValue();

                // Now get the value element
                nl = elem.getElementsByTagName("VALUE");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
```

```
                        + "VALUE node.");
                   return;
                }
                // Get the TEXT section
                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;
                }

                // Retrieve the value
                value = n.getNodeValue();
                }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3. **Mettez en oeuvre la boucle `while` dans la classe `EventReceptionThread` qui attend les rappels d'événements.**

   La classe `EventReceptionThread` est présentée à la section "Définition d'un thread de réception d'événements" à la page 231.

```
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

Étapes

1. **Connectez-vous en tant que superutilisateur ou prenez un rôle équivalent.**
2. **Exécutez votre application.**

   # **`java CrnpClient`** *crnpHost crnpPort localPort* **`...`**

   L'intégralité du code de l'application `CrnpClient` est disponible à l'Annexe G.

ANNEXE **A**

Propriétés standard

Cette annexe décrit les propriétés standard des types de ressources, des ressources et des groupes de ressources, ainsi que les attributs des propriétés de ressource, disponibles pour modifier les propriétés définies par le système et créer des propriétés d'extension.

Remarque – Les noms de propriété des groupes de ressources, des ressources et des types de ressources ne sont pas sensibles à la casse. Vous pouvez associer les majuscules et les minuscules dans les noms des propriétés.

Cette annexe aborde les sujets suivants :

Propriétés des types de ressources

Les informations suivantes présentent les propriétés des types de ressources définis par Sun Cluster. Les valeurs des propriétés sont classées comme suit :

- **Requise**. La propriété doit avoir une valeur explicite dans le fichier RTR. Dans le cas contraire, l'objet auquel appartient la propriété ne peut pas être créé. Les espaces ou les chaînes de caractères vides sont proscrits.
- **Conditionnelle**. Pour exister, la propriété doit être déclarée dans le fichier RTR. À défaut, le RGM ne la crée pas et elle n'est pas accessible aux utilitaires d'administration. Les espaces ou les chaînes de caractères vides sont autorisés. 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**. Pour exister, la propriété doit être déclarée dans le fichier RTR et posséder une valeur explicite. À défaut, le RGM ne la crée pas et elle n'est pas accessible aux utilitaires d'administration. Les espaces ou les chaînes de caractères vides sont proscrits.
- **Facultative**. La propriété peut être déclarée dans le fichier RTR. Si elle n'est pas déclarée, le RGM la crée et fournit une valeur par défaut. Si elle est déclarée dans le fichier RTR sans qu'une valeur soit spécifiée, le RGM lui attribue la valeur par défaut qu'il lui aurait donnée si elle n'avait pas été déclarée.
- **Interrogation uniquement** : ne peut être directement définie par un outil d'administration.

Les propriétés des types de ressources ne peuvent être mises à jour via les utilitaires d'administration, hormis `Installed_nodes` et `RT_system`, qui ne peuvent pas être déclarées dans le fichier RTR et doivent être définies par l'administrateur du cluster.

Les propriétés ci-dessous sont d'abord présentées par leur nom, suivi d'une description.

Remarque – les noms des propriétés de types de ressources, tels `API_version` et `Boot`, *ne sont pas* sensibles à la casse. Vous pouvez associer les majuscules et les minuscules dans le nom des propriétés.

`API_version` (`integer`)
Version minimum de l'API de gestion des ressources nécessaire à la prise en charge de l'implémentation de ce type de ressource.

Ci-dessous, vous trouverez une liste des versions de Sun Cluster avec en regard l'`API_version` maximum prise en charge.

Version 3.1 et antérieures	`2`
3.1 10/03	`3`
3.1 4/04	`4`
3.1 9/04	`5`
3.1 8/05	`6`

Si vous définissez la propriété `API_version` sur une valeur supérieure à `2` dans le fichier RTR, le type de ressources ne sera pas installé sur les versions de Sun Cluster qui prennent en charge une `API_version` maximum inférieure. Par exemple, si vous déclarez `API_version=5` pour un type de ressources, ce type de ressources ne peut être installé sur aucune des versions de Sun Cluster antérieures à la version 3.1 9/04.

Remarque – Si vous ne déclarez pas cette propriété ou si vous lui affectez sa valeur par défaut (2), le service de données peut être installé sur n'importe quelle version de Sun Cluster (ultérieure à Sun Cluster 3.0).

Catégorie : Facultatif

Valeur par défaut : 2

Réglable : NONE

`Boot` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute sur un nœud se connectant ou se reconnectant au cluster lorsqu'une ressource de ce type est déjà gérée. Cette méthode initialise les ressources de ce type de la même manière que la méthode `Init` .

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Failover` (`boolean`)
`TRUE` indique qu'il est impossible de configurer les ressources de ce type dans un groupe pouvant être connecté à plusieurs nœuds à la fois.

Le tableau suivant illustre l'utilisation de cette propriété de type de ressources avec la propriété de ressources `Scalable`.

Valeur de la propriété de types de ressources `Failover`	Valeur de la propriété de ressources `Scalable`	Description
TRUE	TRUE	Ne spécifiez pas cette combinaison, qui n'est pas logique.
TRUE	FALSE	Spécifiez cette combinaison pour un service de basculement.
FALSE	TRUE	Spécifiez cette combinaison pour un service évolutif utilisant une ressource `SharedAddress` pour l'équilibrage de charge du réseau. Pour obtenir plus d'informations sur la ressource `SharedAddress`, reportez-vous au manuel *Guide des notions fondamentales de Sun Cluster pour SE Solaris*

Valeur de la propriété de types de ressources `Failover`	Valeur de la propriété de ressources `Scalable`	Description
`FALSE`	`FALSE`	Bien qu'elle soit inhabituelle, cette combinaison peut permettre de sélectionner un service multimaître n'utilisant pas de système d'équilibrage de charge de réseau.

Pour obtenir plus d'informations, lisez la description de `Scalable` à la page de manuel `r_properties`(5) et au Chapitre 3, "Notions-clés destinées aux administrateurs système et aux développeurs d'applications" du *Guide des notions fondamentales de Sun Cluster pour SE Solaris*.

Catégorie : Facultative

Valeur par défaut : `FALSE`

Réglable : `NONE`

`Fini` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute lorsqu'une ressource du type défini bascule en ressource non gérée.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : `NONE`

`Init` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute lorsqu'une ressource du type défini bascule en ressource gérée.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : `NONE`

`Init_nodes` (`enum`)
Cette propriété désigne les nœuds sur lesquels le RGM doit appeler les méthodes `Init`, `Fini`, `Boot` et `Validate`. Elle peut prendre les valeurs `RG_PRIMARIES` (seulement les nœuds qui peuvent maîtriser la ressource) ou `RT_INSTALLED_NODES` (tous les nœuds sur lesquels le type de ressource est installé).

Catégorie : Facultative

Valeur par défaut : `RG_PRIMARIES`

Réglable : `NONE`

`Installed_nodes` (`string_array`)
Liste des noms des nœuds de cluster sur lesquels le type de ressources peut être exécuté. 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.

Catégorie : Configuration possible par l'administrateur du cluster

Valeur par défaut : Tous les nœuds du cluster

Réglable : `ANYTIME`

`Is_logical_hostname` (`boolean`
La valeur `TRUE` indique que ce type de ressources correspond à une version du type de ressources `LogicalHostname` gérant les adresses IP de basculement.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`Is_shared_address` (`boolean`)
La valeur `TRUE` indique que ce type de ressources correspond à une version du type de ressources `SharedAddress` gérant les adresses IP de basculement.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`Monitor_check` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute avant d'effectuer, sur demande du détecteur, le basculement d'une ressource du type défini.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : `NONE`

`Monitor_start` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute pour démarrer un détecteur de pannes pour une ressource du type défini.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : `NONE`

`Monitor_stop` (`string`)
Méthode de rappel requise si `Démarrage_détecteur` est défini : chemin d'accès au programme que le RGM exécute pour arrêter un détecteur de pannes qui surveille une ressource du type défini.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Pkglist` (`string_array`)
Liste facultative des packages installés lors de l'installation du type de ressources.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Postnet_stop` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute après avoir appelé la méthode `Stop` d'une ressource d'adresse réseau dont dépend une ressource du type défini. Une fois les interfaces réseau désactivées, cette méthode doit effectuer des actions `Stop`.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Prenet_start` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le gestionnaire RGM exécute avant d'appeler la méthode `Start` d'une ressource d'adresse réseau dont dépend une ressource du type défini. Cette méthode exécute les actions `Start` à effectuer avant la configuration des interfaces réseau.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Resource_list` (`string_array`)
Liste de l'ensemble des ressources du type. L'administrateur de cluster ne peut pas définir cette propriété directement. Le RGM la met à jour lorsque l'administrateur de cluster ajoute ou supprime une ressource du type défini dans un groupe de ressources, quel qu'il soit.

Catégorie : Interrogation uniquement

Valeur par défaut : Liste vide

Réglable : NONE

`Resource_type` (`string`)
Nom du type de ressources. Pour afficher les noms des types de ressources actuellement en ligne, utilisez la commande suivante :

`scrgadm` `-p`

Sous Sun Cluster 3.1 et les versions ultérieures, le nom d'un type de ressources doit obligatoirement comporter le numéro de version :

vendor-id **.** *resource-type* **:** *rt-version*

Les trois composantes d'un nom de type de ressources correspondent aux propriétés *vendor-id*, *resource-type* et *rt-version* définies dans le fichier RTR. La commande `scrgadm` ajoute le point (.) et le deux-points (:) . Le suffixe `rt-version` du nom du type de ressources correspond à la valeur de la propriété `RT_version`. Pour éviter tout risque d'ambiguité de la propriété *vendor-id*, il est recommandé d'utiliser le symbole boursier de la société lors de la création du type de ressources. Les noms des type de ressources créés sous les versions antérieures à Sun Cluster 3.1 conservent la syntaxe suivante :

vendor-id **.** *resource-type*

Catégorie : Requise

Valeur par défaut : Chaîne vide

Réglable : `NONE`

`RT_basedir` (`string`)
Chemin d'accès au répertoire permettant de compléter les chemins d'accès relatifs des méthodes de rappel. La valeur doit être le chemin du répertoire dans lequel sont installés les packages du type de ressources. Ce chemin doit être complet, c'est-à-dire qu'il doit commencer par une barre oblique (/).

Catégorie : Requise, à moins que les noms de chemin d'accès de toutes les méthodes ne soient absolus.

Valeur par défaut : Aucune

Réglable : `NONE`

`RT_description` (`string`)
Brève description du type de ressources.

Catégorie : Conditionnelle

Valeur par défaut : Chaîne vide

Réglable : `NONE`

`RT_system` (`boolean`)
Si la propriété `RT_system` d'un type de ressources est définie sur `TRUE`, ce type de ressources ne peut pas être supprimé (`scrgadm -r -t` *resource-type-name*). Cette propriété prévient la suppression accidentelle d'un type de ressources essentiel à la prise en charge de l'infrastructure du cluster, comme `LogicalHostname`, par exemple. Vous pouvez l'appliquer `RT_system` à n'importe quel type de ressources.

Pour supprimer un type de ressources dont la propriété `RT_system` et définie sur `TRUE`, vous devez d'abord définir cette propriété sur `FALSE`. Soyez prudent lorsque vous supprimez un type de ressources prenant en charge les services du cluster.

Catégorie : Facultative

Valeur par défaut : `FALSE`

Réglable : `ANYTIME`

`RT_version` (`string`)
Chaîne indiquant la version de l'implémentation du type de ressources ; elle est obligatoire à partir de Sun Cluster 3.1. `RT_version` représente le suffixe d'un nom complet de type de ressources. La propriété `RT_version`, facultative dans Sun Cluster 3.0, est obligatoire depuis Sun Cluster 3.1.

Catégorie : Conditionnelle/explicite ou requise

Valeur par défaut : Aucune

Réglable : `NONE`

`Single_instance` (`boolean`)
Si cette propriété est paramétrée sur `TRUE`, une seule ressource du type défini peut exister dans le cluster. Le gestionnaire RGM ne permet d'exécuter qu'une seule ressource de ce type à la fois à l'échelle du cluster.

Catégorie : Facultative

Valeur par défaut : `FALSE`

Réglable : `NONE`

`Start` (`string`)
Méthode de rappel : chemin d'accès au programme que le RGM exécute pour démarrer une ressource du type défini.

Catégorie : Requise à moins qu'une méthode `Prenet_start` ne soit déclarée dans le fichier RTR

Valeur par défaut : Aucune

Réglable : `NONE`

`Stop` (`string`)
Méthode de rappel : chemin d'accès au programme que le RGM exécute pour arrêter une ressource du type défini.

Catégorie : Requise à moins que le fichier RTR ne déclare une méthode `Postnet_stop`

Valeur par défaut : Aucune

Réglable : `NONE`

`Update` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute si les propriétés d'une ressource du type défini sont modifiées tandis que cette ressource est en cours d'exécution.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Validate` (`string`)
Méthode de rappel facultative : chemin d'accès au programme que le RGM exécute pour vérifier les valeurs des propriétés des ressources de ce type.

Catégorie : Conditionnelle/explicite

Valeur par défaut : Aucune

Réglable : NONE

`Vendor_ID` (`string`)
Reportez-vous à la description de la propriété `Resource_type`.

Catégorie : Conditionnelle

Valeur par défaut : Aucune

Réglable : NONE

Propriétés des ressources

Cette rubrique décrit les propriétés de ressources définies par Sun Cluster. Les valeurs des propriétés sont classées comme suit :

- **Requise**. L'administrateur du cluster doit spécifier une valeur lorsqu'il crée une ressource à l'aide d'un utilitaire d'administration.
- **Facultative**. Si l'administrateur ne propose pas de valeur lorsqu'il crée le groupe de ressources, le système fournit une valeur par défaut.
- **Conditionnelle**. Le RGM crée la propriété uniquement si celle-ci est déclarée dans le fichier RTR. Dans le cas contraire, la propriété n'existe pas et les administrateurs du cluster ne peuvent pas y accéder. Une propriété conditionnelle déclarée dans le fichier RTR est soit facultative soit requise, selon qu'une valeur par défaut est indiquée ou non dans le fichier RTR. Pour plus d'informations, reportez-vous à la description de toutes les propriétés conditionnelles.
- **Interrogation uniquement**. Ne peut pas être directement définie à l'aide d'un outil d'administration.

L'attribut `Tunable` (décrit dans la rubrique "Attributs des propriétés de ressources" à la page 281) permet d'indiquer si les propriétés de ressource peuvent être définies et quand elles peuvent l'être :

`FALSE` ou `NONE`	Jamais
`TRUE` ou `ANYTIME`	À tout moment

`AT_CREATION` Lorsque la ressource est ajoutée à un cluster

`>WHEN_DISABLED` Lorsque la ressource est désactivée

Les propriétés ci-dessous sont d'abord présentées par leur nom, suivi d'une description.

`Affinity_timeout` (`integer`)
Durée en secondes pendant laquelle les connexions d'une adresse IP cliente à l'un des services de la ressource sont envoyées vers le même nœud de serveur.

Cette propriété n'est utile que lorsque la propriété `Load_balancing_policy` est paramétrée sur `Lb_sticky` ou sur `Lb_sticky_wild`. De plus, `Weak_affinity` doit être défini sur `FALSE`.

Cette propriété n'est utilisée qu'avec les services évolutifs.

Catégorie : Facultative

Valeur par défaut : Aucune

Réglable : `ANYTIME`

`Boot_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de cette méthode a échoué. Pour un type de ressource donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` (une heure) si la méthode est déclarée dans le fichier RTR

Réglable : `ANYTIME`

`Cheap_probe_interval` (`integer`)
Durée en secondes entre les appels de détection rapide des pannes de la ressource. Cette propriété est créée par le RGM. L'administrateur du cluster 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 `Tunable` n'est pas défini dans le fichier RTR, la valeur `Tunable` de la propriété sera `WHEN_DISABLED`.

Catégorie : Conditionnelle

Valeur par défaut : Aucune

Réglable : `WHEN_DISABLED`

propriétés d'extension
Propriétés d'extension telles qu'elles sont déclarées dans le fichier RTR du type de ressources. La mise en œuvre du type de ressources définit ces propriétés. Pour obtenir plus d'informations sur les attributs des propriétés d'extension que vous pouvez définir, consultez la rubrique "Attributs des propriétés de ressources" à la page 281.

Catégorie : Conditionnelle

Valeur par défaut : Aucune

Réglable : Dépend de chaque propriété

`Failover_mode` (`enum`)
Modifie les actions de récupération entreprises par le RGM lorsqu'une ressource ne démarre ou ne s'arrête pas correctement, ou lorsqu'un détecteur de ressource demande le redémarrage ou le basculement d'une ressource défaillante.

`NONE`, `SOFT` ou `HARD` (échecs d'une méthode)

Ces paramètres n'affectent le basculement qu'en cas d'échec d'une méthode d'arrêt ou de démarrage (`Prenet_start`, `Start`, `Monitor_stop`, `Stop` ou `Postnet_stop`). Lorsque la ressource démarre normalement, `NONE`, `SOFT` et `HARD` n'ont aucun effet sur son comportement de redémarrage ou de transfert (le détecteur de la ressource initie le démarrage ou le transfert via la commande `scha_control` ou la fonction `scha_control()`). Pour obtenir plus d'informations, reportez-vous aux pages de manuel `scha_control`(1HA) et `scha_control`(3HA). `NONE` indique que le RGM n'effectuera aucune action de récupération en cas d'échec de l'une des méthodes d'arrêt ou de démarrage citées précédemment. Si la valeur `SOFT` ou `HARD` est spécifiée, le RGM relocalisera le groupe de la ressource sur un autre nœud si la méthode `Start` ou `Prenet_start` échoue. Dans les situations d'échec de `Start` ou de `Prenet_start`, `SOFT` et `HARD` ont le même effet.

Dans les cas d'échec d'une méthode d'arrêt (`Monitor_stop`, `Stop` ou `Postnet_stop`), `SOFT` est identique à `NONE`. Si `Failover_mode` est défini sur `HARD` lorsque l'une de ces méthodes échoue, le RGM redémarre le nœud pour forcer le groupe de ressources à se déconnecter. Il tente ensuite de démarrer le groupe sur un autre nœud.

`RESTART_ONLY` ou `LOG_ONLY`

Contrairement à `NONE`, `SOFT` et `HARD`, qui n'affectent le comportement de basculement que lors de l'échec d'une méthode de démarrage ou d'arrêt, `RESTART_ONLY` et `LOG_ONLY` ont une influence sur l'ensemble du comportement de basculement. Celui-ci inclut le redémarrage des ressources et des groupes de ressources, initié par le détecteur (`scha_control`) ainsi que tout transfert effectué par le détecteur de ressource (`scha_control`). `RESTART_ONLY` indique que le détecteur peut exécuter `scha_control` pour redémarrer une ressource ou un groupe de ressources. La propriété `Retry_count` détermine le nombre de redémarrages autorisés par le RGM dans un laps de temps lui-même défini dans la propriété `Retry_interval` . Si la limite de `Retry_count` est dépassée, aucun autre redémarrage n'est autorisé. Si le paramètre `Failover_mode` est défini sur `LOG_ONLY`, aucun redémarrage ni transfert de ressource n'est autorisé. Définir `Failover_mode` sur `LOG_ONLY` revient à définir `Failover_mode` sur `RESTART_ONLY` en donnant à `Retry_count` une valeur de zéro.

`RESTART_ONLY` ou `LOG_ONLY` (échecs d'une méthode)

Si une méthode du type Prenet_start, Start, Monitor_stop , Stop ou Postnet_stop échoue, RESTART_ONLY et LOG_ONLY équivalent à NONE : le nœud ne sera ni basculé, ni réinitialisé.

Effet des paramètres de Failover_mode sur un service de données

L'effet de chaque paramètre de Failover_mode sur un service de données diffère selon que le service est contrôlé ou non et, le cas échéant, selon son type (basé sur la bibliothèque DSDL ou non).

- Le service de données est dit contrôlé s'il implémente une méthode Monitor_start et que le contrôle de la ressource est activé. Le RGM démarre le détecteur de la ressource en exécutant la méthode Monitor_start après avoir démarré la ressource elle-même. Le détecteur de ressource envoie une sonde pour contrôler le bon fonctionnement de la ressource. Si la sonde échoue, le détecteur de ressource peut demander un redémarrage ou un basculement en appelant la fonction scha_control(). Pour les ressources basées sur la bibliothèque DSDL, une vérification peut faire apparaître un échec partiel (dégradation) ou total du service de données. Les échecs partiels répétés s'accumulent pour former un échec total.
- Le service de données n'est pas contrôlé s'il ne fournit aucune méthode Monitor_start ou si le contrôle de la ressource a été désactivé.
- Les services de données basés sur la bibliothèque DSDL peuvent être développés à l'aide d'Agent Builder, du GDS, ou en utilisant directement la DSDL. Certains services de données (comme HA Oracle) ont été développés sans la DSDL.

NONE, SOFT et HARD (échecs d'une sonde)

Si vous définissez Failover_mode sur NONE, SOFT ou HARD ; si le service de données est contrôlé et basé sur la DSDL et si la sonde échoue, le détecteur appelle la fonction scha_control () pour demander un redémarrage de la ressource. Si les sondes échouent encore, la ressource est redémarrée jusqu'à Retry_count fois, dans l'intervalle Retry_interval. Si les sondes échouent toujours après Retry_count tentatives, le détecteur demande un basculement du groupe de ressources sur un autre noeud.

Si vous définissez Failover_mode sur NONE, SOFT ou HARD et si le service de données est un service non contrôlé basé sur la DSDL, le seul échec détecté est la mort de l'arborescence de processus de la ressource. Si cette arborescence meurt, la ressource est redémarrée.

Si le service de données n'est pas basé sur DSDL, le comportement de redémarrage ou de basculement dépend du code du détecteur de la ressource. Par exemple, le détecteur de ressources Oracle récupère en redémarrant la ressource ou le groupe de ressources ou en basculant ce dernier.

RESTART_ONLY (échecs de sonde)

Si vous définissez `Failover_mode` sur `RESTART_ONLY`, si le service de données est contrôlé et basé sur la DSDL et si la sonde échoue complètement, alors la ressource est redémarrée `Retry_count` fois, pendant `Retry_interval` . Cependant, si la limite de `Retry_count` est dépassée, le détecteur de ressource se termine, définit l'état de la ressource sur `FAULTED` et génère un message indiquant que l'application a rencontré un problème, mais n'a pas été redémarrée et que la sonde a été arrêtée. Dans ce cas, même si le contrôle est toujours activé, la ressource n'est pas contrôlée tant qu'elle n'a pas été réparée et redémarrée par l'administrateur du cluster.

Si vous définissez `Failover_mode` sur `RESTART_ONLY` ; si le service de données est un service non contrôlé basé sur la DSDL et si l'arborescence de processus meurt, la ressource *n'est pas* redémarrée.

Si un service de données contrôlé n'est pas basé sur la DSDL, le comportement de récupération dépend du code du détecteur de la ressource. Si vous définissez `Failover_mode` sur `RESTART_ONLY`, la ressource ou le groupe de ressources peuvent être redémarrés via la fonction `scha_control()` jusqu'à `Retry_count` fois dans l'intervalle `Retry_interval`. Si le détecteur de la ressource dépasse le nombre défini par `Retry_count`, la tentative de redémarrage échoue. De même, si le détecteur appelle la fonction `scha_control()` pour demander un basculement, sa demande échoue.

`LOG_ONLY` (échecs de sonde)

Si vous définissez `Failover_mode` sur `LOG_ONLY` pour un service de données quelconque, toutes les requêtes `scha_control()` (que ce soit pour le redémarrage d'une ressource ou d'un groupe de ressources ou pour le basculement d'un groupe de ressources) sont exclues. Si le service de données est basé sur la DSDL, un message est consigné en cas d'échec de la sonde mais la ressource n'est pas redémarrée. Si la sonde échoue plus de `Retry_count` fois dans l'intervalle défini par `Retry_interval`, le détecteur de la ressource se termine, définit l'état de la ressource sur `FAULTED` et génère un message signalant que l'application a rencontré un problème mais n'a pas été redémarrée et que la sonde a été arrêtée. Dans ce cas, même si le contrôle est toujours activé, la ressource n'est pas contrôlée tant qu'elle n'a pas été réparée et redémarrée par l'administrateur du cluster.

Si vous définissez `Failover_mode` sur `LOG_ONLY` ; si le service de données est un service non contrôlé basé sur la DSDL et si l'arborescence de processus meurt, un message est consigné mais la ressource n'est pas redémarrée.

Si un service de données contrôlé n'est pas basé sur la DSDL, le comportement de récupération dépend du code du détecteur de ressource. Si vous définissez `Failover_mode` sur `LOG_ONLY`, toutes les requêtes `scha_control()` (demandes de redémarrage d'une ressource ou d'un groupe de ressources et demandes de basculement d'un groupe de ressources) échouent.

Catégorie : Facultative

Valeur par défaut : `NONE`

Réglable : ANYTIME

`Fini_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressources donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : ANYTIME

`Init_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressources donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : ANYTIME

`Load_balancing_policy` (`string`)
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. La propriété `Load_balancing_policy` peut prendre les valeurs suivantes :

`Équilibrage_charge_pondéré` (par défaut). La charge est répartie entre plusieurs nœuds en fonction des poids définis dans la propriété `Load_balancing_weights`.

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

`Lb_sticky_wild`. L'adresse IP d'un client donné connecté à l'adresse IP d'un service de rétention générique du client (wildcard sticky) est toujours envoyée vers le même nœud de cluster, quel que soit le numéro du port de destination de l'adresse IP.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `Lb_weighted`

Réglable : AT_CREATION

`Load_balancing_weights` (`string_array`)
Pour les ressources évolutives uniquement. Le logiciel RGM crée automatiquement cette propriété si la propriété `Évolutivité` est déclarée dans le fichier RTR. Format : *weight@node,weight@node*, *weight* étant un nombre entier qui indique la part

relative de la charge distribuée au *node* spécifié. Cette part correspond au poids du nœud 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 nœuds auxquels aucun poids explicite n'est attribué reçoivent un poids par défaut de 1.

Si l'attribut `Tunable` n'est pas spécifié dans le fichier RTR, la valeur `Tunable` de la propriété sera `ANYTIME`. En changeant cette propriété, vous modifiez uniquement la répartition des nouvelles connexions.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : Chaîne vide ("")

Réglable : `ANYTIME`

`Monitor_check_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressources donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Monitor_start_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressources donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Monitor_stop_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressources donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Monitored_switch` (`enum`)
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 la propriété est définie sur `Disabled`, le contrôle de la ressource est arrêté, même si la ressource elle-même reste en ligne. La méthode `Monitor_start` n'est appelée que lorsque le contrôle est de nouveau activé. Si la ressource n'a pas de méthode de rappel du détecteur, cette propriété n'existe pas.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`Network_resources_used` (`string_array`)
Liste des ressources (noms d'hôte logique ou adresses réseau 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 RGM crée automatiquement cette propriété si la propriété `Scalable` est déclarée dans le fichier RTR. Si elle n'est pas déclarée dans le fichier RTR, `Network_resources_used` n'est pas disponible à moins qu'elle ne soit explicitement déclarée dans le fichier RTR.

Si l'attribut `Tunable` n'est pas spécifié dans le fichier RTR, la valeur `Tunable` de la propriété sera `AT_CREATION`.

Remarque – La page de manuel `SUNW.Event`(5) explique en détail comment configurer cette propriété pour le CRNP.

Catégorie : Conditionnelle ou requise

Valeur par défaut : Aucune

Réglable : `AT_CREATION`

`Num_resource_restarts` sur chaque nœud de cluster (`integer`)
Vous ne pouvez pas définir cette propriété directement. C'est le gestionnaire RGM qui la définit sur le nombre d'appels `scha_control`, `Resource_restart` ou `Resource_is_restarted` effectués pour la ressource sur le nœud au cours des *n* dernières secondes. *n* correspond à la valeur de la propriété `Retry_interval` de la ressource. Le RGM réinitialise le compteur de redémarrage à zéro (`0`) chaque fois que cette ressource exécute un transfert `scha_control` (que la tentative de transfert réussisse ou non).

Si un type de ressource ne déclare pas la propriété `Retry_interval`, la propriété `Num_resource_restarts` n'est pas disponible pour les ressources de ce type.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`Num_rg_restarts` sur chaque nœud de cluster (`integer`)
Vous ne pouvez pas définir cette propriété directement. Le gestionnaire RGM la définit sur le nombre d'appels `scha_control Restart` que la ressource a effectués sur le nœud, pour le groupe auquel elle appartient, au cours des *n* dernières secondes. *n* correspond à la valeur de la propriété `Retry_interval` de la ressource. Si un type de ressource ne déclare pas la propriété `Retry_interval`, la propriété `Num_rg_restarts` n'est pas disponible pour les ressources de ce type.

Catégorie : Voir la description

Valeur par défaut : Aucune

Réglable : `NONE`

`On_off_switch` (`enum`)
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. En cas de désactivation, la ressource est déconnectée et aucun rappel n'est effectué tant qu'elle n'est pas réactivée.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`Port_list` (`string_array`)
Liste des numéros de port de réception du serveur. Chaque numéro de port est suivi d'une barre oblique (`/`), puis du protocole utilisé par le port, par exemple : `Port_list=80/tcp` ou `Port_list=80/tcp6,40/udp6`. Vous pouvez indiquer les valeurs de protocole suivantes :

- `tcp`, pour TCP IPv4 ;
- `tcp6`, pour TCP IPv6 ;
- `udp`, pour UDP IPv4 ;
- `udp6`, pour UDP IPv6.

Si la propriété `Scalable` est déclarée dans le fichier RTR, le RGM crée automatiquement la propriété `Port_list`. 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 décrit dans le manuel *Sun Cluster Data Service for Apache Guide for Solaris OS*.

Catégorie : Conditionnelle ou requise

Valeur par défaut : Aucune

Réglable : `ANYTIME`

`Postnet_stop_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressource donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Prenet_start_timeout` pour chaque méthode de rappel du type (`integer`)
Délai en secondes après lequel le gestionnaire RGM conclut que l'exécution de la méthode a échoué. Pour un type de ressource donné, seules les méthodes déclarées dans le fichier RTR possèdent des propriétés de délai d'attente définies.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`R_description` (`string`)
Brève description de la ressource.

Catégorie : Facultatif

Valeur par défaut : Chaîne vide

Réglable : `ANYTIME`

`Resource_dependencies` (`string_array`)
Liste de ressources du même groupe ou d'un groupe différent, dont la ressource `Resource_dependencies` dépend fortement. La ressource ne peut être démarrée que si toutes les ressources de la liste sont activées. Si cette ressource et l'une des ressources de la liste démarrent en même temps, le gestionnaire RGM attend pour démarrer cette ressource que la ressource de la liste démarre. Si la ressource dominante ne démarre pas, la ressource dépendante reste déconnectée. Il se peut que la ressource dominante ne démarre pas parce que son groupe est déconnecté ou se trouve à l'état `START_FAILED`. Si la ressource dépendante reste déconnectée parce qu'elle dépend d'une ressource d'un autre groupe, qui ne démarre pas, le groupe de la ressource dépendante passe à l'état `PENDING_ONLINE_BLOCKED` .

Si la ressource dépendante est déconnectée en même temps que ses ressources dominantes, elle est arrêtée en premier. Cependant, si elle reste connectée ou ne parvient pas à s'arrêter, ses ressources dominantes appartenant à d'autres groupes peuvent néanmoins s'arrêter. Les ressources de la liste ne peuvent pas être désactivées, à moins que la ressource dépendante ne soit désactivée en premier.

Par défaut, dans les groupes de ressources, les ressources d'application ont une forte relation de dépendance implicite avec les ressources d'adresse réseau. Pour obtenir plus d'informations, reportez-vous au paragraphe `Implicit_network_dependencies` de la rubrique "Propriétés du groupe de ressources" à la page 271.

Au sein d'un groupe de ressources, les méthodes `Prenet_start` sont exécutées par ordre de dépendance avant les méthodes `Start`. Les méthodes `Postnet_stop` sont exécutées après les méthodes `Stop`, selon l'ordre de leurs dépendances. Si les ressources se trouvent dans des groupes différents, la ressource dépendante attend que la ressource dont elle dépend termine `Prenet_start` et `Start` avant d'exécuter `Prenet_start` . Réciproquement, la ressource dont elle dépend attend qu'elle termine `Stop` et `Postnet_stop` avant d'exécuter `Stop`.

Catégorie : Facultative

Valeur par défaut : Liste vide

Réglable : `ANYTIME`

`Resource_dependencies_restart` (`string_array`)
Liste de ressources du même groupe ou d'un groupe différent, dont la ressource `Resource_dependencies_restart` dépend pour le redémarrage.

Cette propriété fonctionne de la même manière que `Resource_dependencies`, à une exception près. Si l'une des ressources de la liste des dépendances de redémarrage est redémarrée, cette ressource l'est également. Le RGM redémarre la ressource dépendante une fois la ressource dominante reconnectée.

Catégorie : Facultative

Valeur par défaut : Liste vide

Réglable : `ANYTIME`

`Resource_dependencies_weak` (`string_array`)
Liste de ressources du même groupe ou de groupes différents, par rapport auxquelles la ressource `Resource_dependencies_weak` a une dépendance faible. Une dépendance faible détermine l'ordre des appels de méthode. Le RGM appelle les méthodes `Start` des ressources de cette liste, avant d'appeler la méthode `Start` de la ressource dépendante. Le RGM appelle les méthodes `Stop` de la ressource dépendante avant d'appeler les méthodes `Stop` de celles de la liste. Si les ressources de la liste ne parviennent pas à démarrer ou restent déconnectées, la ressource dépendante peut néanmoins démarrer.

Si la ressource dépendante démarre en même temps qu'une des ressources de sa liste `Resource_dependencies_weak`, le RGM attend le démarrage de la ressource dominante avant de démarrer la ressource dépendante. Si la ressource dominante ne démarre pas – si, par exemple, le groupe de ressources auquel elle appartient reste déconnecté ou si elle se trouve à l'état `START_FAILED`, la ressource dépendante démarre. Le groupe de ressources de la ressource dépendante peut passer temporairement à l'état `PENDING_ONLINE_BLOCKED` au démarrage des ressources de sa liste `Resource_dependencies_weak`. Lorsque toutes les ressources de la liste ont démarré, ou refusé de démarrer, la ressource dépendante démarre et son groupe repasse à l'état `PENDING_ONLINE`.

Si la ressource est déconnectée en même temps que celles de la liste, elle est arrêtée avant celles de la liste. Si la ressource reste connectée ou ne parvient pas à s'arrêter, la ressource de la liste peut tout de même s'arrêter. Vous ne pouvez pas désactiver les ressources de la liste, à moins d'avoir au préalable désactivé l'autre ressource.

Au sein d'un groupe de ressources, les méthodes `Prenet_start` sont exécutées par ordre de dépendance avant les méthodes `Start`. Les méthodes `Postnet_stop` sont exécutées après les méthodes `Stop`, selon l'ordre de leurs dépendances. Si les ressources se trouvent dans des groupes différents, la ressource dépendante attend que la ressource dont elle dépend effectue `Prenet_start` et `Start` avant d'exécuter `Prenet_start` . Réciproquement, la ressource dont elle dépend attend que la ressource qui dépend d'elle termine `Stop` et `Postnet_stop` avant d'exécuter `Stop`.

Catégorie : Facultative

Valeur par défaut : Liste vide

Réglable : `ANYTIME`

`Resource_name` (`string`)
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.

Catégorie : Requis

Valeur par défaut : Aucune

Réglable : `NONE`

`Resource_project_name` (`string`)
Nom de projet Solaris associé à la ressource. Utilisez cette propriété pour appliquer aux services de données du cluster les fonctions de gestion de ressources Solaris, telles que les partages de processeurs et les pools de ressources. Lorsque le gestionnaire RGM connecte les ressources, il démarre les processus connexes sous ce nom de projet. Si cette propriété n'est pas spécifiée, le nom du projet est tiré de la propriété `RG_project_name` du groupe de ressources comprenant la ressource (pour plus d'informations, consultez la page de manuel `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 apparaître dans la base de données des projets (pour plus d'informations, consultez la page de manuel `projects`(1) et le document *System Administration Guide: Solaris Containers-Resource Management and Solaris Zones*).

Cette propriété n'est prise en charge que si elle est exécutée sous Solaris 9.

Remarque – Les modifications apportées à la propriété prennent effet lors du démarrage de la ressource.

Catégorie : Facultatif

Valeur par défaut : Null

Réglable : ANYTIME

`Resource_state` sur chaque noeud de cluster (`enum`)
État de la ressource déterminé par le RGM sur chaque nœud du cluster. Les états possibles sont les suivants : `ONLINE`, `OFFLINE`, `START_FAILED` , `STOP_FAILED`, `MONITOR_FAILED`, `ONLINE_NOT_MONITORED`, `STARTING` et `STOPPING` .

Vous ne pouvez pas configurer cette propriété.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : NONE

`Retry_count` (`integer`)
Nombre de fois qu'un détecteur tente de redémarrer une ressource si celle-ci échoue. Si le nombre défini par `Retry_count` est dépassé, le détecteur effectue l'une des actions suivantes (en fonction du service de données spécifique et du paramétrage de la propriété `Failover_mode`) :

- Il peut autoriser le groupe de ressources à rester sur le noeud principal actuel, même si la ressource est dans un état défectueux
- Il peut demander un basculement du groupe de ressources sur un autre noeud

Cette propriété est créée par le RGM. L'administrateur du cluster 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 `Tunable` n'est pas spécifié dans le fichier RTR, la valeur `Tunable` de la propriété sera `WHEN_DISABLED`.

Remarque – Si vous spécifiez une valeur négative pour cette propriété, le détecteur tente de redémarrer la ressource un nombre illimité de fois.

Catégorie : Conditionnelle

Valeur par défaut : voir ci-dessus

Réglable : WHEN_DISABLED

`Retry_interval` (`integer`)
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 est créée par le RGM. L'administrateur du cluster 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 `Tunable` n'est pas spécifié dans le fichier RTR, la valeur `Tunable` de la propriété sera `WHEN_DISABLED`.

Catégorie : Conditionnelle

Valeur par défaut : pas de valeur par défaut (voir ci-dessus)

Réglable : `WHEN_DISABLED`

`Scalable` (`boolean`)
Indique si la ressource est évolutive - c'est-à-dire si elle utilise les fonctions d'équilibrage de charge réseau de Sun Cluster.

Si cette propriété est déclarée dans le fichier RTR, le RGM crée automatiquement les propriétés de service évolutif pour les ressources de ce type : `Affinity_timeout` , `Load_balancing_policy`, `Load_balancing_weights` , `Network_resources_used`, `Port_list`, `UDP_affinity` et `Weak_affinity`. 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 `Scalable`, lorsqu'elle est déclarée dans le fichier RTR, est `TRUE`.

Si cette propriété est déclarée dans le fichier RTR, on ne peut lui affecter un attribut `Tunable` différent de `AT_CREATION`.

Si elle n'est pas déclarée dans le fichier RTR, la ressource n'est pas évolutive, vous ne pouvez pas la régler et aucune propriété de service évolutif n'est définie par le RGM. Cependant, vous pouvez déclarer les propriétés `Network_resources_used` et `Port_list` de manière explicite dans le fichier RTR. Ces propriétés peuvent être aussi utiles dans un service non évolutif que dans un service évolutif.

L'utilisation de cette propriété de ressource avec la propriété de type de ressource `Failover` est détaillée plus avant dans la page de manuel `r_properties`(5).

Catégorie : Facultatif

Valeur par défaut : Aucune

Réglable : `AT_CREATION`

`Start_timeout` pour chaque méthode de rappel de type (`integer`)
Intervalle de temps en secondes après lequel le gestionnaire RGM conclut que l'exécution de cette méthode a échoué. Pour un type de ressource donné, les propriétés de l'intervalle d'attente sont uniquement définies pour les méthodes déclarées dans le fichier RTR.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Status` sur chaque noeud de cluster (`enum`)
Défini par le détecteur de ressource via la commande `scha_resource_setstatus` ou la fonction `scha_resource_setstatus()`. Les valeurs possibles sont les suivantes : `OK`, `degraded`, `faulted`, `unknown` et

offline. Lorsqu'une ressource est connectée ou déconnectée, le RGM définit automatiquement la valeur Status si cette dernière n'est pas définie par le détecteur ou les méthodes de la ressource.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : NONE

Status_msg sur chaque noeud de cluster (string)
Cette propriété est définie par le détecteur de ressources en même temps que la propriété Statut. Lorsqu'une ressource est activée ou désactivée, le RGM réinitialise automatiquement cette propriété avec une chaîne vide si elle n'est pas définie par les méthodes de la ressource.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : NONE

Stop_timeout pour chaque méthode de rappel de type (integer)
Intervalle de temps en secondes après lequel le gestionnaire RGM conclut que l'exécution de cette méthode a échoué. Pour un type de ressource donné, les propriétés de l'intervalle d'attente sont uniquement définies pour les méthodes déclarées dans le fichier RTR.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : 3600 secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : ANYTIME

Thorough_probe_interval (integer)
Durée en secondes entre les appels de détection des pannes nécessitant un temps système important de la ressource. Cette propriété est créée par le RGM. L'administrateur du cluster 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 Tunable n'est pas spécifié dans le fichier RTR, la valeur Tunable de la propriété sera WHEN_DISABLED.

Catégorie : Conditionnelle

Valeur par défaut : Aucune

Réglable : WHEN_DISABLED

Type (string)
Type de ressource dont cette ressource est une instance.

Catégorie : Requis

Valeur par défaut : Aucune

Réglable : NONE

`Type_version` (`string`)
Indique la version de type de ressource actuellement associée à cette ressource. Le gestionnaire RGM crée automatiquement cette propriété qui ne peut être déclarée dans le fichier RTR. Sa valeur est identique à celle de la propriété `RT_version` du type de la ressource. Lors de la création d'une ressource, la propriété `Type_version` n'est pas explicitement indiquée, mais elle peut apparaître en tant que suffixe du nom du type de ressource. Lorsqu'une ressource est éditée, la valeur de la propriété `Type_version` peut être modifiée.

Les possibilités de réglage de la propriété dépendent des éléments suivants :

- la version actuelle du type de ressource ;
- L'instruction `#$upgrade_from` figurant dans le fichier RTR.

Catégorie : Voir la description

Valeur par défaut : Aucune

Réglable : Voir la description

`UDP_affinity` (`boolean`)
Si cette propriété est définie sur `TRUE`, elle envoie tout le trafic UDP depuis un client donné vers le noeud de serveur qui gère actuellement tout le trafic TCP du client.

Cette propriété n'est utile que lorsque la propriété `Load_balancing_policy` est paramétrée sur `Lb_sticky` ou sur `Lb_sticky_wild`. De plus, `Weak_affinity` doit être défini sur `FALSE`.

Cette propriété n'est utilisée qu'avec les services évolutifs.

Catégorie : Facultatif

Valeur par défaut : Aucune

Réglable : `WHEN_DISABLED`

`Update_timeout` pour chaque méthode de rappel de type (`integer`)
Intervalle de temps en secondes après lequel le gestionnaire RGM conclut que l'exécution de cette méthode a échoué. Pour un type de ressource donné, les propriétés de l'intervalle d'attente sont uniquement définies pour les méthodes déclarées dans le fichier RTR.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Validate_timeout` pour chaque méthode de rappel de type (`integer`)
Intervalle de temps en secondes après lequel le gestionnaire RGM conclut que l'exécution de cette méthode a échoué. Pour un type de ressource donné, les propriétés de l'intervalle d'attente sont uniquement définies pour les méthodes déclarées dans le fichier RTR.

Catégorie : Conditionnelle ou facultative

Valeur par défaut : `3600` secondes (une heure) si la méthode est déclarée dans le fichier RTR.

Réglable : `ANYTIME`

`Weak_affinity` (`boolean`)
Si cette propriété est définie sur `TRUE`, elle active la forme faible d'affinité du client. Avec cette affinité, les connexions d'un client donné sont envoyées sur le même nœud de serveur, sauf dans les cas suivants :

- Un listener de serveur démarre en réponse à divers événements - comme par exemple le redémarrage d'un détecteur de pannes, le basculement d'une ressource ou l'admission d'un noeud dans un cluster après un échec.
- La valeur de `Load_balancing_weights` change pour la ressource évolutive car l'administrateur du cluster a effectué une action d'administration.

L'affinité faible offre une alternative de faible déperdition par rapport à la configuration par défaut, tant en termes d'utilisation de la mémoire que de cycles du processeur.

Cette propriété n'est utile que lorsque la propriété `Load_balancing_policy` est paramétrée sur `Lb_sticky` ou sur `Lb_sticky_wild`.

Cette propriété n'est utilisée qu'avec les services évolutifs.

Catégorie : Facultatif

Valeur par défaut : Aucune

Réglable : `WHEN_DISABLED`

Propriétés du groupe de ressources

Les informations suivantes décrivent les propriétés des groupes de ressources définis par Sun Cluster. Les valeurs des propriétés sont classées comme suit :

- **Requise**. L'administrateur du cluster doit spécifier une valeur au moment de créer un groupe de ressources à l'aide d'un utilitaire d'administration.
- **Facultatif**. Si l'administrateur ne propose pas de valeur lorsqu'il crée le groupe de ressources, le système fournit une valeur par défaut.

- **Interrogation seulement**. Ne peut être directement défini par un outil administratif.

Les propriétés ci-dessous sont d'abord présentées par leur nom, suivi d'une description.

`Auto_start_on_new_cluster` (`boolean`)
Cette propriété vérifie que le RGM démarre automatiquement le groupe de ressources lors de la formation d'un nouveau cluster. La valeur par défaut est `TRUE`.

Si elle est définie sur `TRUE`, le RGM essaie de lancer le groupe de ressources automatiquement afin d'obtenir les `Desired_primaries` lors de la réinitialisation simultanée de l'ensemble des noeuds du cluster.

En cas de définition sur `FALSE`, le groupe de ressources ne démarre pas automatiquement lors de la réinitialisation du cluster. Le groupe de ressources reste déconnecté jusqu'à ce qu'il soit manuellement connecté, via la commande `scswitch` ou via une instruction équivalente de l'interface graphique utilisateur. Après cela, le groupe de ressources reprend un comportement de basculement normal.

Catégorie : Facultatif

Valeur par défaut : `TRUE`

Réglable : `ANYTIME`

`Desired_primaries` (`integer`)
Nombre total des noeuds sur lesquels le groupe peut s'exécuter simultanément.

La valeur par défaut est 1. Si la propriété `RG_mode` est définie sur `Failover`, sa valeur ne doit pas être supérieure à 1. Si la propriété `RG_mode` est définie sur `Scalable`, vous pouvez lui octroyer une valeur supérieure à 1.

Catégorie : Facultatif

Valeur par défaut : `1`

Réglable : `ANYTIME`

`Failback` (`boolean`)
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 qui ne sont pas des noeuds de prédilection et à les connecter aux noeuds de prédilection.

Catégorie : Facultatif

Valeur par défaut : `FALSE`

Réglable : `ANYTIME`

`Global_resources_used` (`string_array`)
Indique si les systèmes de fichiers du cluster sont utilisés par une ressource de ce groupe. L'administrateur de cluster peut utiliser deux valeurs : l'astérisque (*) pour indiquer toutes les ressources globales et la chaîne de caractères vide ("") pour indiquer aucune ressource globale.

Catégorie : Facultatif

Valeur par défaut : Toutes les ressources globales

Réglable : `ANYTIME`

`Implicit_network_dependencies` (`boolean`)
Valeur booléenne indiquant, lorsque `TRUE` est défini, que le RGM doit appliquer les dépendances implicites fortes des ressources d'adresse non réseau aux ressources d'adresses réseau au sein du groupe. Par conséquent, le RGM démarre toutes les ressources d'adresses réseau avant toutes les autres et arrête ces ressources après toutes les autres dans le 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.

Catégorie : Facultatif

Valeur par défaut : `TRUE`

Réglable : `ANYTIME`

`Maximum_primaries` (`integer`)
Nombre maximum de noeuds auxquels le groupe peut être connecté simultanément.

Si la propriété `RG_mode` est définie sur `Failover`, sa valeur ne doit pas être supérieure à 1. Si la propriété `RG_mode` est définie sur `Scalable`, vous pouvez lui octroyer une valeur supérieure à 1.

Catégorie : Facultatif

Valeur par défaut : `1`

Réglable : `ANYTIME`

`Nodelist` (`string_array`)
Liste des noeuds de cluster sur lequels le groupe peut être mis en ligne, par ordre de préférence. Ces noeuds correspondent aux noeuds principaux potentiels ou maîtres du groupe de ressources.

Catégorie : Facultatif

Valeur par défaut : liste de tous les noeuds du cluster dans un ordre arbitraire

Réglable : `ANYTIME`

`Pathprefix` (`string`)
Répertoire dans le système de fichiers de cluster dans lequel les ressources du groupe peuvent enregistrer des fichiers d'administration stratégiques. Certaines ressources demandent cette propriété. Faites en sorte que `Préfixe_chemin_accès` soit unique pour chaque groupe de ressources.

Catégorie : Facultatif

Valeur par défaut : Chaîne vide

Réglable : `ANYTIME`

`Pingpong_interval` (`integer`)
Valeur d'entier non négatif (en secondes) que le RGM utilise pour déterminer sur quels noeuds le groupe de ressources doit être connecté dans les cas suivants :

- Reconfiguration
- Après l'exécution d'une commande/fonction `scha_control GIVEOVER`

Dans le cas d'une reconfiguration, la tentative de connexion du groupe de ressources peut échouer plus d'une fois sur un noeud spécifique durant l'intervalle (en secondes) défini par `Pingpong_interval`. Cet échec se produit lorsque les méthodes `Start` ou `Prenet_start` d'une ressource se sont terminées avec un état différent de zéro ou ont dépassé le délai d'attente autorisé. Par conséquent, le RGM considère que le noeud n'est pas à même d'héberger le groupe de ressources et cherche un autre maître.

Si une commande `scha_control` ou `scha_control GIVEOVER` est exécutée par une ressource sur un noeud donné (ce qui amène son groupe de ressources à basculer sur un autre noeud), le premier noeud (sur lequel s'exécutait `scha_control`) ne peut servir de destination au résultat d'une autre commande `scha_control GIVEOVER` initiée par cette même ressource tant que `Pingpong_interval` secondes ne se sont pas écoulées.

Catégorie : Facultatif

Valeur par défaut : `3600` (une heure)

Réglable : `ANYTIME`

`Resource_list` (`string_array`)
Liste des ressources contenues dans le groupe. L'administrateur du cluster ne peut pas définir directement cette propriété. Le RGM met à jour cette propriété lorsque l'administrateur du cluster ajoute ou supprime des ressources dans le groupe de ressources.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : `NONE`

`RG_affinities` (`string`)
Le RGM s'efforce de localiser un groupe de ressources sur un noeud actuellement maître d'un groupe de ressources donné (affinité positive) ou qui n'est pas le maître d'un groupe de ressources donné (affinité négative).

Vous pouvez paramétrer la propriété `RG_affinities` sur les valeurs suivantes :

- ++ (affinité positive forte) ;
- + (affinité positive faible) ;
- - (affinité négative faible) ;
- -- (affinité négative forte) ;
- +++ (affinité positive forte avec délégation de basculement).

Par exemple, `RG_affinities=+RG2,--RG3` indique que le groupe de ressources dispose d'une affinité positive faible avec RG2 et d'une affinité négative forte avec RG3.

L'utilisation de la propriété `RG_affinities` est détaillée plus avant dans le Chapitre 2, "Administering Data Service Resources" du *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

Catégorie : Facultatif

Valeur par défaut : Chaîne vide

Réglable : `ANYTIME`

`RG_dependencies` (`string_array`)
Liste (facultative) des groupes de ressources. Elle indique l'ordre choisi pour la connexion ou la déconnexion d'autres groupes sur un même noeud. Le graphique de l'ensemble des `RG_affinities` fortes (positives et négatives), comprenant également les `RG_dependencies`, ne doit pas contenir de cycles.

Supposons, par exemple, que le groupe de ressources RG2 soit répertorié dans la liste des `RG_dependencies` du groupe de ressources RG1 (de ce fait, RG1 a une dépendance de groupe de ressources par rapport à RG2). La liste suivante résume les effets de cette dépendance :

- Lorsqu'un nœud joint le cluster, les méthodes `Boot` sur ce nœud ne sont pas exécutées sur les ressources du groupe RG1 tant que l'exécution de toutes les méthodes `Boot` de ce nœud n'est pas terminée sur les ressources du groupe RG2.
- Si RG1 et RG2 sont tous deux à l'état `PENDING_ONLINE` sur un même noeud, au même moment, les méthodes de démarrage (`Prenet_start` ou `Start`) ne sont pas exécutées sur les ressources de RG1 tant que l'ensemble des ressources de RG2 n'ont pas exécuté leurs propres méthodes de démarrage.
- Si RG1 et RG2 sont tous deux à l'état `PENDING_OFFLINE` sur un même noeud, au même moment, les méthodes d'arrêt (`Postnet_stop` ou `Stop`) ne sont pas exécutées sur les ressources de RG2 tant que l'ensemble des ressources de RG1 n'ont pas exécuté leurs propres méthodes d'arrêt.

- La tentative de basculement des éléments principaux de RG1 ou de RG2 échoue si, du fait de cette opération, le groupe RG1 reste en ligne sur un nœud et que RG2 est désactivé sur tous les nœuds. Pour plus d'informations à ce sujet, consultez les pages de manuel `scswitch`(1M) et `scsetup`(1M).
- Si la propriété `Desired_primaries` est définie sur zéro sur RG2, sa valeur ne peut être supérieure à zéro sur RG1.
- Vous ne pouvez pas paramétrer la propriété `Auto_start_on_new_cluster` sur `TRUE` pour RG1, si cette même propriété est paramétrée sur `FALSE` pour RG2.

Catégorie : Facultatif

Valeur par défaut : Liste vide

Réglable : `ANYTIME`

`RG_description` (`string`)
Brève description du groupe de ressources.

Catégorie : Facultatif

Valeur par défaut : Chaîne vide

Réglable : `ANYTIME`

`RG_is_frozen` (`boolean`)
Valeur booléenne indiquant si un périphérique global dont dépend un groupe de ressources est en cours de basculement. Si cette propriété est définie sur `TRUE`, le périphérique global est basculé. Si elle est paramétrée sur `FALSE`, aucun périphérique global n'est en cours de basculement. La propriété `Global_resources_used` du groupe de ressources indique s'il dépend de périphériques globaux.

Vous ne pouvez pas définir directement la propriété `RG_is_frozen`. Le RGM met à jour la propriété `RG_is_frozen` lors de la modification de l'état des périphériques globaux.

Catégorie : Facultatif

Valeur par défaut : Aucune

Réglable : `NONE`

`RG_mode` (`enum`)
Indique si le groupe de ressources est un groupe de basculement ou évolutif. Si elle est définie sur `Failover`, le RGM définit la propriété `Maximum_primaries` du groupe sur 1. Par conséquent, le groupe de ressources ne peut être géré que par un seul noeud.

Si elle est définie sur `Scalable`, le RGM permet à la propriété `Maximum_primaries` d'être configurée sur une valeur supérieure à 1. Le cas échéant, le groupe peut être géré par plusieurs noeuds en même temps. Le RGM n'autorise pas l'ajout d'une ressource dont la propriété `Failover` est définie sur `TRUE` dans un groupe de ressources dont la propriété `RG_mode` est définie sur `Scalable`.

Si `Maximum_primaries` est défini sur 1, la valeur par défaut est `Failover`. Si la valeur de `Maximum_primaries` est supérieure à 1, la valeur par défaut est `Scalable`.

Catégorie : Facultatif

Valeur par défaut : Dépend de la valeur de la propriété `Maximum_primaries`

Réglable : `NONE`

`RG_name` (`string`)
Nom du groupe de ressources. Cette propriété est nécessaire et doit être unique au sein du cluster.

Catégorie : Requis

Valeur par défaut : Aucune

Réglable : `NONE`

`RG_project_name` (`string`)
Nom du projet Solaris (consultez la page de manuel `projects`(1)) associé au groupe de ressources. Utilisez cette propriété pour appliquer aux services de données du cluster les fonctions de gestion de ressources Solaris, telles que les partages de processeurs et les pools de ressources. Lorsque le RGM connecte des groupes de ressources, il démarre les processus associés sous le nom du projet pour les ressources dont la propriété `Resource_project_name` n'est pas définie (consultez la page de manuel `r_properties`(5) pour obtenir plus d'informations). Le nom de projet spécifié doit apparaître dans la base de données des projets (pour plus d'informations, consultez la page de manuel `projects`(1) et le document *System Administration Guide: Solaris Containers-Resource Management and Solaris Zones*).

Cette propriété n'est prise en charge que si elle est exécutée sous Solaris 9.

Remarque – Les modifications apportées à la propriété prennent effet lors du démarrage de la ressource.

Catégorie : Facultatif

Valeur par défaut : La chaîne de texte "`default`"

Réglable : `ANYTIME`

`RG_state` sur chaque noeud du cluster (`enum`)
Définie par le RGM sur `UNMANAGED`, `ONLINE`, `OFFLINE`, `PENDING_ONLINE`, `PENDING_OFFLINE`, `ERROR_STOP_FAILED`, `ONLINE_FAULTED` ou `PENDING_ONLINE_BLOCKED` afin de décrire l'état du groupe sur chaque noeud du cluster.

Vous ne pouvez pas configurer cette propriété. Toutefois, vous pouvez définir cette propriété de manière indirecte via la commande `scswitch` ou la commande `scsetup` équivalente - ou encore via les commandes de SunPlex Manager. Un groupe peut exister à l'état `UNMANAGED` s'il n'est pas géré par le RGM.

Les descriptions qui suivent résument chaque état.

Remarque – Ces états ne s'appliquent qu'aux noeuds individuels - sauf `UNMANAGED`, qui s'applique à l'ensemble des noeuds. Par exemple, un groupe de ressources peut être `OFFLINE` sur le noeud A et `PENDING_ONLINE` sur le noeud B.

`UNMANAGED`	État initial d'un groupe de ressources qui vient d'être créé ou qui a été géré précédemment. Soit les méthodes `Init` n'ont pas encore été exécutées sur les ressources du groupe, soit les méthodes `Fini` ont été exécutées sur ces ressources. Le groupe n'est pas géré par le RGM.
`ONLINE`	Le groupe de ressources a été démarré sur le noeud. En d'autres termes, les méthodes de démarrage `Prenet_start`, `Start` et `Monitor_start` (applicables à chaque ressource) se sont exécutées sur toutes les ressources activées du groupe.
`OFFLINE`	Le groupe de ressources a été arrêté sur le noeud. En d'autres termes, les méthodes d'arrêt `Monitor_stop`, `Stop` et `Postnet_stop` (applicables à chaque ressource) se sont exécutées sur toutes les ressources activées du groupe. Cet état s'applique également avant le premier démarrage d'un groupe de ressources sur le noeud.

PENDING_ONLINE	Le groupe de ressources est en cours de démarrage sur le noeud. Les méthodes de démarrage `Prenet_start`, `Start` et `Monitor_start` (applicables à chaque ressource) se sont exécutées sur toutes les ressources activées du groupe.
PENDING_OFFLINE	Le groupe de ressources est en cours d'arrêt sur le noeud. Les méthodes d'arrêt `Monitor_stop`, `Stop` et `Postnet_stop` (applicables à chaque ressource) se sont exécutées sur toutes les ressources activées du groupe.
ERROR_STOP_FAILED	Une ou plusieurs des ressources du groupe ne s'est pas arrêtée correctement et se trouve à l'état `Stop_failed`. Les autres ressources du groupe peuvent rester connectées ou déconnectées. Ce groupe de ressources ne peut démarrer sur aucun noeud tant que l'état `ERROR_STOP_FAILED` n'est pas corrigé. Vous devez utiliser une commande administrative (telle que `scswitch -c`) pour arrêter manuellement la ressource `Stop_failed` et redéfinir son état sur `OFFLINE`.
ONLINE_FAULTED	Le groupe de ressources avait l'état `PENDING_ONLINE` et a terminé son démarrage sur ce noeud. Cependant, une ou plusieurs ressources sont passées à l'état `Start_failed` ou `Faulted`.
PENDING_ONLINE_BLOCKED	Le groupe de ressources n'a pas pu démarrer correctement car une ou plusieurs ressources au sein du groupe n'ont pas satisfait une dépendance forte par rapport à une autre ressource, dans un autre groupe. Ces ressources restent à l'état `OFFLINE`. Lorsque les dépendances sont satisfaites, le groupe de ressources repasse automatiquement à l'état `PENDING_ONLINE`.

Catégorie : Interrogation uniquement

Valeur par défaut : Aucune

Réglable : NONE

RG_system (boolean)
Si la propriété RG_system est définie sur TRUE pour un groupe de ressources donné, certaines opérations ne sont plus accessibles au groupe de ressources et aux ressources qu'il contient. Cette restriction est destinée à éviter les modifications ou suppressions accidentelles de groupes ou ressources importantes. Seules les commandes scrgadm et scswitch sont affectées par cette propriété. Les opérations de scha_control(1HA) et scha_control(3HA) ne sont pas concernées.

Avant d'effectuer une opération dont l'accès est limité sur un groupe de ressources (ou sur une ressource du groupe), vous devez au préalable paramétrer la propriété RG_system correspondante sur FALSE. Soyez très vigilant lors de la modification ou de la suppression d'un groupe de ressources (ou une ressource de ce groupe) prenant en charge des services de cluster.

Opération	Exemple
Suppression d'un groupe de ressources	**scrgadm -r -g RG1**
Modification d'une propriété d'un groupe de ressources (sauf RG_system)	**scrgadm -c -t RG1 -y nodelist=...**
Ajout d'une ressource dans un groupe de ressources	**scrgadm -a -j R1 -g RG1**
Suppression d'une ressource dans un groupe de ressources	**scrgadm -r -j R1 -g RG1**
Modification d'une propriété d'une ressource appartenant à un groupe de ressources	**scrgadm -c -j R1**
Basculement et déconnexion d'un groupe de ressources	**scswitch -F -g RG1**
Gestion d'un groupe de ressources	**scswitch -o -g RG1**
Basculement d'un groupe de ressources à l'état non géré	**scswitch -u -g RG1**
Activation d'une ressource	**scswitch -e -j R1**
Activation du contrôle d'une ressource	**scswitch -e -M -j R1**
Désactivation d'une ressource	**scswitch -n -j R1**
Désactivation du contrôle d'une ressource	**scswitch -n -M -j R1**

Si la propriété RG_system est définie sur TRUE pour un groupe de ressources donné, la seule propriété du groupe de ressources que vous pouvez modifier est RG_system elle-même. En d'autres termes, il est toujours possible de modifier la propriété RG_system.

Catégorie :	Facultatif
Valeur par défaut :	`FALSE`
Réglable :	`ANYTIME`

Attributs des propriétés de ressources

La rubrique suivante présente les attributs des propriétés de ressource 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 `boolean`, `enum` ou `int` sur les valeurs par défaut suivantes : `Null` ou une chaîne de caractères vide (“”) .

Les propriétés ci-dessous sont d'abord présentées par leur nom, suivi d'une description.

`Array_maxsize`
: Nombre maximum d'éléments de tableau autorisé pour le type `stringarray`.

`Array_minsize`
: Nombre minimum d'éléments de tableau autorisé pour le type `stringarray`.

`Default`
: Indique une valeur par défaut pour la propriété.

`Description`
: Chaîne de caractères présentant une brève description de la propriété. L'attribut `Description` ne peut pas être défini dans le fichier RTR pour les propriétés définies par le système.

`Enumlist`
: Pour un type `énum`, ensemble de valeurs de chaînes de caractères permises pour la propriété.

`Extension`
: S'il est utilisé, cet attribut indique que l'entrée du fichier RTR déclare une propriété d'extension définie par l'implémentation du type de ressource. Si cet attribut n'apparaît pas, l'entrée correspond à une propriété définie par le système.

`Max`
: Valeur maximale autorisée pour la propriété du type `int`.

`Maxlength`
Pour les types `string` et `stringarray`, longueur maximale permise pour une chaîne.

`Min`
Valeur minimale autorisée pour la propriété du type `int`.

`Minlength`
Pour les types `string` et `stringarray`, longueur minimale permise pour une chaîne.

`Property`
Nom de la propriété de ressource.

`Tunable`
Indique si l'administrateur du cluster peut lui-même définir la valeur de cette propriété dans une ressource. Définissez-la sur `NONE` ou `FALSE` pour empêcher l'administrateur du cluster de définir la propriété. Les valeurs permettant à un administrateur du cluster de modifier une propriété sont les suivantes : `TRUE` ou `ANYTIME` (à n'importe quel moment), `AT_CREATION` (uniquement lorsque la ressource est créée) ou `WHEN_DISABLED` (lorsque la ressource est désactivée). Pour établir d'autres conditions ("lorsque le contrôle est désactivé" ou "en cas de déconnexion"), définissez cet attribut sur `ANYTIME` et validez l'état de la ressource dans la méthode `Validate`.

La valeur par défaut est différente pour chaque propriété de ressource standard, comme le montre l'entrée suivante. La valeur par défaut permettant de modifier une propriété d'extension, sauf indication contraire dans le fichier RTR, est `TRUE` (`ANYTIME`).

Type de la propriété
Les types autorisés sont `string`, `boolean`, `integer`, `enum` et `stringarray`. 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.

ANNEXE **B**

Liste des exemples de codes de service de données

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 ressource (RTR).

Cette annexe aborde les points suivants :

Liste de code du fichier RTR

Le fichier RTR contient les déclarations de ressource et de propriétés du type de ressource qui définissent la configuration initiale du service de données au moment où l'administrateur du cluster l'enregistre.

EXEMPLE B–1 Fichier RTR `SUNW.Sample`

```
#
# Copyright (c) 1998-2005 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

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

Resource_type = "sample";
Vendor_id = SUNW;
RT_description = "Domain Name Service on Sun Cluster";

RT_version ="1.0";
API_version = 2;
Failover = TRUE;

RT_basedir=/opt/SUNWsample/bin;
Pkglist = SUNWsample;

Start              = dns_svc_start;
Stop               = dns_svc_stop;

Validate           = dns_validate;
Update             = dns_update;

Monitor_start      = dns_monitor_start;
Monitor_stop       = dns_monitor_stop;
Monitor_check      = dns_monitor_check;

# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.
#
# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.

# The MIN value for all method timeouts is set to 60 seconds. This
# prevents administrators from setting shorter timeouts, which do not
# improve switchover/failover performance, and can lead to undesired
# RGM actions (false failovers, node reboot, or moving the resource group
# to ERROR_STOP_FAILED state, requiring operator intervention). Setting
# too-short method timeouts leads to a *decrease* in overall availability
# of the data service.
{
        PROPERTY = Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
        DEFAULT=300;
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

```
}
{
        PROPERTY = Update_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Monitor_Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Thorough_Probe_Interval;
        MIN=1;
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Set Retry_interval as a multiple of 60 since it is converted from seconds
# to minutes, rounding up. For example, a value of 50 (seconds)
# is converted to 1 minute. Use this property to time the number of
# retries (Retry_count).
{
        PROPERTY = Retry_interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

{
        PROPERTY = Network_resources_used;
        TUNABLE = AT_CREATION;
        DEFAULT = "";
}

#
```

EXEMPLE B–1 Fichier RTR `SUNW.Sample` *(Suite)*

```
# Extension Properties
#

# The cluster administrator must set the value of this property to point to the
# directory that contains the configuration files used by the application.
# For this application, DNS, specify the path of the DNS configuration file on
# PXFS (typically named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "The Configuration Directory Path";
}

# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 30;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Listing de code de la méthode `Start`

Le gestionnaire RGM exécute la méthode `Start` sur un noeud de cluster lorsque le groupe de ressources qui contient la ressource du service de données est mis en ligne sur ce noeud ou lorsque la ressource est activée. Dans l'application échantillon, la méthode `Start` active le démon `in.named` (DNS) sur ce noeud.

EXEMPLE B–2 Méthode `dns_svc_start`

```
#!/bin/ksh
#
# Start Method for HA-DNS.
#
# This method starts the data service under the control of PMF. Before starting
# the in.named process for DNS, it performs some sanity checks. The PMF tag for
# the data service is $RESOURCE_NAME.named. PMF tries to start the service a
# specified number of times (Retry_count) and if the number of attempts exceeds
# this value within a specified interval (Retry_interval) PMF reports a failure
# to start the service. Retry_count and Retry_interval are both properties of the
# resource set in the RTR file.
```

EXEMPLE B–2 Méthode dns_svc_start *(Suite)*

```
#pragma ident   "@(#)dns_svc_start   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"
```

EXEMPLE B–2 Méthode `dns_svc_start` *(Suite)*

```
PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Get the value of the Confdir property of the resource in order to start
# DNS. Using the resource name and the resource group entered, find the value of
# Confdir value set by the cluster administrator when adding theresource.
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get returns the "type" as well as the "value" for the extension
# properties. Get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Check if $CONFIG_DIR is 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

# Change to the $CONFIG_DIR directory in case there are relative
# path names in the data files.
cd $CONFIG_DIR

# Check that the named.conf file is present in the $CONFIG_DIR directory.
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

# Get the value for Retry_count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Get the value for Retry_interval from the RTR file. Convert this value, which is in
# seconds, to minutes for passing to pmfadm. Note that this is a conversion with
# round-up, for example, 50 seconds rounds up to one minute.
((RETRY_INTRVAL = `scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ 60))

# Start the in.named daemon under the control of PMF. Let it crash and restart
# up to $RETRY_COUNT times in a period of $RETRY_INTERVAL; if it crashes
# more often than that, PMF will cease trying to restart it. If there is a
# process already registered under the tag <$PMF_TAG>, then, PMF sends out
# an alert message that the process is already running.
echo "Retry interval is "$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Log a message indicating that HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
```

EXEMPLE B–2 Méthode dns_svc_start *(Suite)*

```
            "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Listing de code de la méthode Stop

Le gestionnaire RGM exécute la méthode Stop sur un noeud de cluster lorsque le groupe de ressources qui contient la ressource HA-DNS est mis hors ligne sur ce noeud ou que la ressource est désactivée. Cette méthode arrête le démon in.named (DNS) sur ce nœud.

EXEMPLE B–3 Méthode dns_svc_stop

```
#!/bin/ksh
#
# Stop method for HA-DNS
#
# Stop the data service using PMF. If the service is not running the
# method exits with status 0 as returning any other value puts the resource
# in STOP_FAILED state.


#pragma ident   "@(#)dns_svc_stop   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
```

EXEMPLE B–3 Méthode `dns_svc_stop` *(Suite)*

```
                *)
                    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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# Obtain the Stop_timeout value from the RTR file.
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAMÈ

# Attempt to stop the data service in an orderly manner using a SIGTERM
# signal through PMF. Wait for up to 80% of the Stop_timeout value to
# see if SIGTERM is successful in stopping the data service. If not, send SIGKILL
# to stop the data service. Use up to 15% of the Stop_timeout value to see
# if SIGKILL is successful. If not, there is a failure and the method exits with
# non-zero status. The remaining 5% of the Stop_timeout is for other uses.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG.named; then
   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   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"

      # Since the data service did not stop with a SIGTERM signal, use
      # SIGKILL now and wait for another 15% of the total timeout value.
```

EXEMPLE B–3 Méthode `dns_svc_stop` *(Suite)*

```
      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 UNSUCCESSFUL"

          exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
           "HA-DNS is not started"

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service in STOP_FAILED State.
   exit 0
fi

# Successfully stopped DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

Listing de code de l'utilitaire `gettime`

L'utilitaire `gettime` est un programme C utilisé par la commande `PROBE` pour relever le temps écoulé entre les redémarrages de la sonde. Vous devez compiler ce programme et le placer dans le même répertoire que les méthodes de rappel, à savoir le répertoire indiqué par la propriété `RT_basedir`.

EXEMPLE B–4 Programme utilitaire `gettime.c`

```
# This utility program, used by the probe method of the data service, tracks
# the elapsed time in seconds from a known reference point (epoch point). It
# must be compiled and placed in the same directory as the data service callback
# methods (RT_basedir).

#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));
```

EXEMPLE B–4 Programme utilitaire `gettime.c` *(Suite)*

```
    exit(0);
}
```

Listing de code du programme PROBE

Le programme `PROBE` vérifie la disponibilité du service de données à l'aide des commandes `nslookup` (reportez-vous à la page de manuel `nslookup`(1M)). La méthode de rappel `Monitor_start` lance ce programme, et la méthode de rappel `Monitor_stop` l'arrête.

EXEMPLE B–5 Programme `dns_probe`

```
#!/bin/ksh
#pragma ident   "@(#)dns_probe   1.1   00/04/19 SMI"
#
# Probe method for HA-DNS.
#
# This program checks the availability of the data service using nslookup, which
# queries the DNS server to look for the DNS server itself. If the server
# does not respond or if the query is replied to by some other server,
# then the probe concludes that there is some problem with the data service
# and fails the service over to another node in the cluster. Probing is done
# at a specific interval set by THOROUGH_PROBE_INTERVAL in the RTR file.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
```

EXEMPLE B–5 Programme dns_probe *(Suite)*

```
                ;;
            *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                "ERROR: Option $OPTARG unknown"
                 exit 1
                 ;;
            esac
        done
}

#############################################################################
# restart_service ()
#
# This function tries to restart the data service by calling the Stop method
# followed by the Start method of the dataservice. If the dataservice has
# already died and no tag is registered for the dataservice under PMF,
# then this function fails the service over to another node in the cluster.
#
function restart_service
{
        # To restart the dataservice, first, verify that the
        # dataservice itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the dataservice is still registered under
                # PMF, first stop the dataservice and start it back up again.
                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`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

                # Obtain the Start method name and the START_TIMEOUT value for
                # this resource.
                START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
                START_METHOD=`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
```

EXEMPLE B–5 Programme dns_probe *(Suite)*

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

        else
                # The absence of the TAG for the dataservice
                # implies that the dataservice has already
                # exceeded the maximum retries allowed under PMF.
                # Therefore, do not attempt to restart the
                # dataservice again, but try to failover
                # to another node in the cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}

###############################################################################
# decide_restart_or_failover ()
#
# This function decides the action to be taken upon the failure of a probe:
# restart the data service locally or fail over to another node in the cluster.
#
function decide_restart_or_failover
{

   # Check if this is the first restart attempt.
   if [ $retries -eq 0 ]; then
         # This is the first failure. Note the time of
         # this first attempt.
         start_time=`$RT_BASEDIR/gettimè
         retries=`expr $retries + 1`
         # Because this is the first failure, attempt to restart
         # the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart data service."
            exit 1
         fi
   else
      # This is not the first failure
      current_time=`$RT_BASEDIR/gettimè
      time_diff=`expr $current_time - $start_timè
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # This failure happened after the time window
         # elapsed, so reset the retries counter,
         # slide the window, and do a retry.
         retries=1
         start_time=$current_time
```

EXEMPLE B–5 Programme dns_probe *(Suite)*

```
         # Because the previous failure occurred more than
         # Retry_interval ago, attempt to restart the data service.
         restart_service
         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
         # Still within the time window,
         # and the retry counter expired, so fail over.
         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
         # Still within the time window,
         # and the retry counter has not expired,
         # so do another retry.
         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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# The interval at which probing is to be done is set in the system defined
# property THOROUGH_PROBE_INTERVAL. Obtain the value of this property with
# scha_resource_get
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL \
```

EXEMPLE B–5 Programme `dns_probe` *(Suite)*

```
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtain the timeout value allowed for the probe, which is set in the
# PROBE_TIMEOUT extension property in the RTR file. The default timeout for
# nslookup is 1.5 minutes.
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Probe_timeout`
PROBE_TIMEOUT=`echo $probe_timeout_info | awk '{print $2}'`

# Identify the server on which DNS is serving by obtaining the value
# of the NETWORK_RESOURCES_USED property of the resource.
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Get the retry count value from the system defined property Retry_count
RETRY_COUNT =`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Get the retry interval value from the system defined property
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Obtain the full path for the gettime utility from the
# RT_basedir property of the resource type.
RT_BASEDIR=scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# The probe runs in an infinite loop, trying nslookup commands.
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep for a
   # duration of <THOROUGH_PROBE_INTERVAL>
   sleep $PROBE_INTERVAL

   # Run the probe, which queries the IP address on
   # which DNS is serving.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
           > $DNSPROBEFILE 2>&1

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

   # Make sure that the reply to nslookup command comes from the HA-DNS
   # server and not from another name server listed in the
```

EXEMPLE B–5 Programme dns_probe *(Suite)*

```
   # /etc/resolv.conf file.
   if [ $probefail -eq 0 ]; then
      # Get the name of the server that replied to the nslookup query.
                  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

   # If the probefail variable is not set to 0, either the nslookup command
   # timed out or the reply to the query was came from another server
   # (specified in the /etc/resolv.conf file). In either case, the DNS server is
   # not responding and the method calls decide_restart_or_failover,
   # which evaluates whether to restart the data service or to fail it over
   # to another node.

   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
```

Listing de code de la méthode Monitor_start

Cette méthode lance le programme PROBE du service de données.

EXEMPLE B–6 Méthode dns_monitor_start

```
#!/bin/ksh
#
# Monitor start Method for HA-DNS.
#
# This method starts the monitor (probe) for the data service under the
# control of PMF. The monitor is a process that probes the data service
# at periodic intervals and if there is a problem restarts it on the same node
# or fails it over to another node in the cluster. The PMF tag for the
# monitor is $RESOURCE_NAME.monitor.
```

EXEMPLE B–6 Méthode `dns_monitor_start` *(Suite)*

```
#pragma ident   "@(#)dns_monitor_start   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

EXEMPLE B–6 Méthode dns_monitor_start *(Suite)*

```
# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the data service.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Start the probe for the data service under PMF. Use the infinite retries
# option to start the probe. Pass the resource name, group, and type to the
# probe method.
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
    -T $RESOURCETYPE_NAME

# Log a message indicating that the monitor for HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} Monitor for HA-DNS successfully started"
fi
exit 0
```

Listing de code de la méthode Monitor_stop

Cette méthode interrompt le programme PROBE du service de données.

EXEMPLE B–7 Méthode dns_monitor_stop

```
#!/bin/ksh
# Monitor stop method for HA-DNS
# Stops the monitor that is running using PMF.

#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"

###############################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
```

EXEMPLE B–7 Méthode `dns_monitor_stop` *(Suite)*

```
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# See if the monitor is running, and if so, kill it.
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 " \
          $RESOURCE_NAME
           exit 1
   else
      # Could successfully stop the monitor. Log a message.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
          "${ARGV0} Monitor for resource " $RESOURCE_NAME \
          " successfully stopped"
   fi
fi
exit 0
```

Listing de code de la méthode Monitor_check

Cette méthode vérifie l'existence du répertoire indiqué par la propriété Confdir. Le gestionnaire RGM appelle Monitor_check chaque fois que la méthode PROBE bascule le service de données sur un nouveau noeud, et pour contrôler les noeuds maîtres potentiels.

EXEMPLE B–8 Méthode dns_monitor_check

```
#!/bin/ksh#
# Monitor check  Method for DNS.
#
# The RGM calls this method whenever the fault monitor fails the data service
# over to a new node. Monitor_check calls the Validate method to verify
# that the configuration directory and files are available on the new node.

#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
   typeset opt

   while getopts 'R:G:T:' opt
   do
      case "$opt" in

      R)
      # Name of the DNS resource.
      RESOURCE_NAME=$OPTARG
      ;;

      G)
      # Name of the resource group in which the resource is
      # configured.
      RESOURCEGROUP_NAME=$OPTARG
      ;;

      T)
      # Name of the resource type.
      RESOURCETYPE_NAME=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
      "ERROR: Option $OPTARG unknown"
      exit 1
```

EXEMPLE B–8 Méthode `dns_monitor_check` *(Suite)*

```
      ;;
      esac
   done

}

###############################################################################
# MAIN
###############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method.
parse_args "$@"

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

# Obtain the full path for the Validate method from
# the RT_basedir property of the resource type.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Obtain the name of the Validate method for this resource.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# Obtain the value of the Confdir property in order to start the
# data service. Use the resource name and the resource group entered to
# obtain the Confdir value set at the time of adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the type as well as the value for extension
# properties. Use awk to get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Call the validate method so that the dataservice can be failed over
# successfully to the new node.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
-T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

# Log a message indicating that monitor check was successful.
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."
```

EXEMPLE B–8 Méthode `dns_monitor_check` *(Suite)*

```
   exit 1
fi
```

Listing de code de la méthode `Validate`

Cette méthode vérifie l'existence du répertoire indiqué par la propriété `Confdir`. Le gestionnaire RGM appelle cette méthode à la création du service de données et lorsque l'administrateur du cluster met à jour les propriétés du service de données. `Monitor_check` appelle cette méthode chaque fois que le détecteur de pannes bascule le service de données sur un nouveau noeud.

EXEMPLE B–9 Méthode `dns_validate`

```
#!/bin/ksh
# Validate method for HA-DNS.
# This method validates the Confdir property of the resource. The Validate
# method gets called in two scenarios. When the resource is being created and
# when a resource property is getting updated. When the resource is being
# created, this method gets called with the -c flag and all the system-defined
# and extension properties are passed as command-line arguments. When a resource
# property is being updated, the Validate method gets called with the -u flag,
# and only the property/value pair of the property being updated is passed as a
# command-line argument.
#
# ex: When the resource is being created command args will be
#
# dns_validate -c -R <..> -G <...> -T <..> -r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# when the resource property is being updated
#
# dns_validate -u -R <..> -G <...> -T <..> -r <sys-prop_being_updated=value>
#   OR
# dns_validate -u -R <..> -G <...> -T <..> -x <extn-prop_being_updated=value>

#pragma ident   "@(#)dns_validate   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
   typeset opt
```

EXEMPLE B–9 Méthode `dns_validate` *(Suite)*

```
while getopts 'cur:x:g:R:T:G:' opt
do
            case "$opt" in
            R)
                    # Name of the DNS resource.
                    RESOURCE_NAME=$OPTARG
                    ;;
            G)
                    # Name of the resource group in which the resource is
                    # configured.
                    RESOURCEGROUP_NAME=$OPTARG
                    ;;
            T)
                    # Name of the resource type.
                    RESOURCETYPE_NAME=$OPTARG
                    ;;
            r)
                    #The method is not accessing any system defined
                    #properties, so this is a no-op.
                    ;;
            g)
                    # The method is not accessing any resource group
                    # properties, so this is a no-op.
                    ;;
            c)
                    # Indicates the Validate method is being called while
                    # creating the resource, so this flag is a no-op.
                    ;;
            u)
                    # Indicates the updating of a property when the
                    # resource already exists. If the update is to the
                    # Confdir property then Confdir should appear in the
                    # command-line arguments. If it does not, the method must
                    # look for it specifically using scha_resource_get.
                    UPDATE_PROPERTY=1
                    ;;
            x)
                    # Extension property list. Separate the property and
                    # value pairs using "=" as the separator.
                    PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                    VAL=echo $OPTARG | awk -F= '{print $2}'`

                    # If the Confdir extension property is found on the
                    # command line, note its value.
                    if [ $PROPERTY == "Confdir" ];
                    then
                    CONFDIR=$VAL
                    CONFDIR_FOUND=1
                    fi
                    ;;
            *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$SYSLOG_TAG] \
```

EXEMPLE B–9 Méthode dns_validate *(Suite)*

```
                        "ERROR: Option $OPTARG unknown"
                        exit 1
                        ;;
                esac
   done
}

###############################################################################
# MAIN
#
###############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Set the Value of CONFDIR to null. Later, this method retrieves the value
# of the Confdir property from the command line or using scha_resource_get.
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0

# Parse the arguments that have been passed to this method.
parse_args "$@"

# If the validate method is being called due to the updating of properties
# try to retrieve the value of the Confdir extension property from the command
# line. Otherwise, obtain the value of Confdir using 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

# Verify that the Confdir property has a value. If not there is a failure
# and exit with status 1.
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
       "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
   exit 1
fi

# Now validate the actual Confdir property value.

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

# Check that the named.conf file is present in the Confdir directory.
```

EXEMPLE B–9 Méthode `dns_validate` *(Suite)*

```
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

# Log a message indicating that the Validate method was successful.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

Listing de code de la méthode `Update`

Le gestionnaire RGM appelle la méthode `Update` pour signaler à une ressource en cours d'exécution que ses propriétés ont été modifiées.

EXEMPLE B–10 Méthode `dns_update`

```
#!/bin/ksh
# Update method for HA-DNS.
# The actual updates to properties are done by the RGM. Updates affect only
# the fault monitor so this method must restart the fault monitor.

#pragma ident   "@(#)dns_update   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
```

EXEMPLE B–10 Méthode `dns_update` *(Suite)*

```
                        # Name of the resource type.
                        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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the resource.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ

# When the Update method is called, the RGM updates the value of the property
# being updated. This method must check if the fault monitor (probe)
# is running, and if so, kill it and then restart it.
if pmfadm -q $PMF_TAG.monitor; then

# Kill the monitor that is running already
        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
                # Could successfully stop DNS. Log a message.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully stopped"
        fi
   # Restart the monitor.
   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 "
```

EXEMPLE B–10 Méthode `dns_update` *(Suite)*

```
      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 de l'exemple de type de ressource de la bibliothèque DSDL

Cette annexe fournit le code complet de chaque méthode du type de ressource `SUNW.xfnts`. Elle comprend la liste de `xfnts.c`, qui contient le code des sous-routines appelées par les méthodes de rappel. Le Chapitre 8 décrit l'exemple de type de ressource `SUNW.xfnts` plus en détail.

Cette annexe aborde les points suivants :

Liste des fichiers `xfnts.c`

Ce fichier met en oeuvre les sous-routines appelées par les méthodes `SUNW.xfnts`.

EXEMPLE C–1 `xfnts.c`

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 * to probe the health of the data service. The probe just returns either
 * success or failure. Action is taken based on this returned value in the
 * method found in the file xfnts_probe.c
 *
 */

#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"

/*
 * The initial timeout allowed for the HAXFS data service to
 * be fully up and running. We will wait for 3 % (SVC_WAIT_PCT)
 * of the start_timeout time before probing the service.
 */
#define   SVC_WAIT_PCT      3

/*
 * We need to use 95% of probe_timeout to connect to the port and the
 * remaining time is used to disconnect from port in the svc_probe function.
 */
#define   SVC_CONNECT_TIMEOUT_PCT      95

/*
 * SVC_WAIT_TIME is used only during starting in svc_wait().
 * In svc_wait() we need to be sure that the service is up
 * before returning, thus we need to call svc_probe() to
 * monitor the service. SVC_WAIT_TIME is the time between
 * such probes.
 */
#define   SVC_WAIT_TIME      5

/*
 * This value will be used as disconnect timeout, if there is no
 * time left from the probe_timeout.
 */
#define   SVC_DISCONNECT_TIMEOUT_SECONDS      2


/*
 * svc_validate():
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 *
 * Do HA-XFS specific validation of the resource configuration.
 *
 * svc_validate will check for the following
 * 1. Confdir_list extension property
 * 2. fontserver.cfg file
 * 3. xfs binary
 * 4. port_list property
 * 5. network resources
 * 6. other extension properties
 *
 * If any of the above validation fails then, Return > 0 otherwise return 0 for
 * success
 */

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;

   /*
    * Get the configuration directory for the XFS dataservice from the
    * confdir_list extension property.
    */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   /* Return an error if there is no confdir_list extension property */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Property Confdir_list is not set properly.");
      return (1); /* Validation failure */
   }

   /*
    * Construct the path to the configuration file from the extension
    * property confdir_list. Since HA-XFS has only one configuration
    * we will need to use the first entry of the confdir_list property.
    */
   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /*
    * Check to see if the HA-XFS configuration file is in the right place.
    * Try to access the HA-XFS configuration file and make sure the
    * permissions are set properly
    */
   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
        * suppress lint error because errno.h prototype
        * is missing void arg
        */
       scds_syslog(LOG_ERR,
           "Failed to access file <%s> : <%s>",
           xfnts_conf, strerror(errno));   /*lint !e746 */
       return (1);
   }

   /*
    * Make sure that xfs binary exists and that the permissions
    * are correct. The XFS binary are assumed to be on the local
    * File system and not on the Global File System
    */
   if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
       scds_syslog(LOG_ERR,
           "Cannot access XFS binary : <%s> ", strerror(errno));
       return (1);
   }

   /* HA-XFS will have only 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); /* Validation Failure */
   }

#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); /* Validation Failure */
   }
#endif

   /*
    * Return an error if there is an error when trying to get the
    * available network address resources for this resource
    */
   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); /* Validation Failure */
   }

   /* Return an error if there are no network address resources */
   if (snrlp == NULL || snrlp->num_netresources == 0) {
       scds_syslog(LOG_ERR,
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
            "No network address resource in resource group.");
      rc = 1;
      goto finished;
   }

   /* Check to make sure other important extension props are set */
   if (scds_get_ext_monitor_retry_count(scds_handle) <= 0)
   {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_count is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }
   if (scds_get_ext_monitor_retry_interval(scds_handle) <= 0) {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_interval is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }

   /* All validation checks were successful */
   scds_syslog(LOG_INFO, "Successful validation.");
   rc = 0;

finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* return result of validation */
}

/*
 * svc_start():
 *
 * Start up the X font server
 * Return 0 on success, > 0 on failures.
 *
 * The XFS service will be started by running the command
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS will be started under PMF. XFS will be started as a single instance
 * service. The PMF tag for the data service will be of the form
 * <resourcegroupname,resourcename,instance_number.svc>. In case of XFS, since
 * there will be only one instance the instance_number in the tag will be 0.
 */

int
svc_start(scds_handle_t scds_handle)
{
   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;
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
   /* get the configuration directory from the confdir_list property */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /* obtain the port to be used by XFS from the Port_list property */
   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);
   }

   /*
    * Construct the command to start HA-XFS.
    * NOTE: XFS daemon prints the following message while stopping the XFS
    * "/usr/openwin/bin/xfs notice: terminating"
    * In order to suppress the daemon message,
    * the output is redirected to /dev/null.
    */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);

   /*
    * Start HA-XFS under PMF. Note that HA-XFS is started as a single
    * instance service. The last argument to the scds_pmf_start function
    * denotes the level of children to be monitored. A value of -1 for
    * this parameter means that all the children along with the original
    * process are to be monitored.
    */
   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); /* return Success/failure status */
}


/*
 * svc_stop():
 *
 * Stop the XFS server
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 * Return 0 on success, > 0 on failures.
 *
 * svc_stop will stop the server by calling the toolkit function:
 * scds_pmf_stop.
 */
int
svc_stop(scds_handle_t scds_handle)
{
   scha_err_t   err;

   /*
    * The timeout value for the stop method to succeed is set in the
    * Stop_Timeout (system defined) property
    */
   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); /* Successfully stopped */
}

/*
 * svc_wait():
 *
 * wait for the data service to start up fully and make sure it is running
 * healthy
 */

int
svc_wait(scds_handle_t scds_handle)
{
   int rc, svc_start_timeout, probe_timeout;
   scds_netaddr_list_t   *netaddr;

   /* obtain the network resource to use for probing */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resources found in resource group.");
      return (1);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
            "No network address resource in resource group.");
      return (1);
   }

   /*
    * Get the Start method timeout, port number on which to probe,
    * the Probe timeout value
    */
   svc_start_timeout = scds_get_rs_start_timeout(scds_handle);
   probe_timeout = scds_get_ext_probe_timeout(scds_handle);

   /*
    * sleep for SVC_WAIT_PCT percentage of start_timeout time
    * before actually probing the dataservice. This is to allow
    * the dataservice to be fully up in order to reply to the
    * probe. NOTE: the value for SVC_WAIT_PCT could be different
    * for different data services.
    * Instead of calling sleep(),
    * call scds_svc_wait() so that if service fails too
    * many times, we give up and return early.
    */
   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 {
      /*
       * probe the data service on the IP address of the
       * network resource and the portname
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Success. Free up resources and return */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

      /*
       * Dataservice is still trying to come up. Sleep for a while
       * before probing again. Instead of calling sleep(),
       * call scds_svc_wait() so that if service fails too
       * many times, we give up and return early.
       */
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
   /* We rely on RGM to timeout and terminate the program */
   } while (1);

}

/*
 * This function starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as <RG-name,RS-name,instance_number.mon>. The restart option
 * of PMF is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

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));

   /*
    * The probe xfnts_probe is assumed to be available in the same
    * subdirectory where the other callback methods for the RT are
    * installed. The last parameter to scds_pmf_start denotes the
    * child monitor level. Since we are starting the probe under PMF
    * we need to monitor the probe process only and hence we are using
    * a value of 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); /* Successfully started Monitor */
}


/*
 * This function stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on <RG-name_RS-name,instance_number.mon>.
 */
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
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); /* Successfully stopped monitor */
}

/*
 * svc_probe(): Do data service specific probing. Return a float value
 * between 0 (success) and 100(complete failure).
 *
 * The probe does a simple socket connection to the XFS server on the specified
 * port which is configured as the resource extension property (Port_list) and
 * pings the dataservice. If the probe fails to connect to the port, we return
 * a value of 100 indicating that there is a total failure. If the connection
 * goes through and the disconnect to the port fails, then a value of 50 is
 * returned indicating a partial failure.
 */
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;


   /*
    * probe the dataservice by doing a socket connection to the port
    * specified in the port_list property to the host that is
    * serving the XFS dataservice. If the XFS service which is configured
    * to listen on the specified port, replies to the connection, then
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
 * the probe is successful. Else we will wait for a time period set
 * in probe_timeout property before concluding that the probe failed.
 */

/*
 * Use the SVC_CONNECT_TIMEOUT_PCT percentage of timeout
 * to connect to the port
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
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));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

t2 = (hrtime_t)(gethrtime()/1E9);

/*
 * Compute the actual time it took to connect. This should be less than
 * or equal to connect_timeout, the time allocated to connect.
 * If the connect uses all the time that is allocated for it,
 * then the remaining value from the probe_timeout that is passed to
 * this function will be used as disconnect timeout. Otherwise, the
 * the remaining time from the connect call will also be added to
 * the disconnect timeout.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Use the remaining time(timeout - time_took_to_connect) to disconnect
 */

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
            "%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;
   }

   /*
    * Return partial failure in case of disconnection failure.
    * Reason: The connect call is successful, which means
    * the application is alive. A disconnection failure
    * could happen due to a hung application or heavy load.
    * If it is the later case, don't declare the application
    * as dead by returning complete failure. Instead, declare
    * it as partial failure. If this situation persists, the
    * disconnect call will fail again and the application will be
    * restarted.
    */
   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));
      /* this is a partial failure */
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }

   t2 = (hrtime_t)(gethrtime()/1E9);
   time_used = (int)(t2 - t1);
   time_remaining = timeout - time_used;

   /*
    * If there is no time left, don't do the full test with
    * fsinfo. Return SCDS_PROBE_COMPLETE_FAILURE/2
    * instead. This will make sure that if this timeout
    * persists, server will be restarted.
    */
   if (time_remaining <= 0) {
      scds_syslog(LOG_ERR, "Probe timed out.");
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }

   /*
    * The connection and disconnection to port is successful,
    * Run the fsinfo command to perform a full check of
    * server health.
    * Redirect stdout, otherwise the output from fsinfo
    * ends up on the console.
    */
   (void) sprintf(testcmd,
```

EXEMPLE C–1 xfnts.c *(Suite)*

```
        "/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);
}
```

Liste des codes de la méthode xfnts_monitor_check

Cette méthode vérifie que la configuration de base du type de ressources est valide.

EXEMPLE C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_check.c - Monitor Check method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * just make a simple validate check on the service
 */

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

   /* Process the arguments passed by RGM and initialize syslog */
```

EXEMPLE C–2 xfnts_monitor_check.c *(Suite)*

```
   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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method run as part of monitor check */
   return (rc);
}
```

Liste des codes de la méthode xfnts_monitor_start

Cette méthode exécute la méthode xfnts_probe.

EXEMPLE C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_start.c - Monitor Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as RG-name,RS-name.mon. The restart option of PMF
 * is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */
```

EXEMPLE C–3 xfnts_monitor_start.c *(Suite)*

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

   /* Process arguments passed by RGM and initialize 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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of monitor_start method */
   return (rc);
}
```

Liste des codes de la méthode xfnts_monitor_stop

Cette méthode arrête la méthode de sonde_xfnts.

EXEMPLE C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_stop.c - Monitor Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on RG-name_RS-name.mon.
 */
```

EXEMPLE C–4 xfnts_monitor_stop.c *(Suite)*

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

   scds_handle_t   scds_handle;
   int    rc;

   /* Process arguments passed by RGM and initialize 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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of monitor stop method */
   return (rc);
}
```

Liste des codes de la méthode xfnts_probe

La méthode xfnts_probe vérifie la disponibilité de l'application et détermine s'il faut basculer ou redémarrer le service de données. La méthode de rappel xfnts_monitor_start démarre ce programme, tandis que la méthode de rappel xfnts_monitor_stop l'arrête.

EXEMPLE C–5 xfnts_probe.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_probe.c - Probe for HA-XFS
 */

#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
```

EXEMPLE C–5 xfnts_probe.c *(Suite)*

```
#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():
 * Just an infinite loop which sleep()s for sometime, waiting for
 * the PMF action script to interrupt the sleep(). When interrupted
 * It calls the start method for HA-XFS to restart it.
 *
 */

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);
   }

   /* Get the ip addresses available for this resource */
   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);
   }

   /* Return an error if there are no network resources */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      return (1);
   }

   /*
    * Set the timeout from the X props. This means that each probe
```

EXEMPLE C–5 xfnts_probe.c *(Suite)*

```
     * iteration will get a full timeout on each network resource
     * without chopping up the timeout between all of the network
     * resources configured for this resource.
     */
    timeout = scds_get_ext_probe_timeout(scds_handle);

    for (;;) {

       /*
        * sleep for a duration of thorough_probe_interval between
        *  successive probes.
        */
       (void) scds_fm_sleep(scds_handle,
           scds_get_rs_thorough_probe_interval(scds_handle));

       /*
        * Now probe all ipaddress we use. Loop over
        * 1. All net resources we use.
        * 2. All ipaddresses in a given resource.
        * For each of the ipaddress that is probed,
        * compute the failure history.
        */
       probe_result = 0;
       /*
        * Iterate through the all resources to get each
        * IP address to use for calling svc_probe()
        */
       for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
          /*
           * Grab the hostname and port on which the
           * health has to be monitored.
           */
          hostname = netaddr->netaddrs[ip].hostname;
          port = netaddr->netaddrs[ip].port_proto.port;
          /*
           * HA-XFS supports only one port and
           * hence obtain the port value from the
           * first entry in the array of ports.
           */
          ht1 = gethrtime(); /* Latch probe start time */
          scds_syslog(LOG_INFO, "Probing the service on "
              "port: %d.", port);

          probe_result =
          svc_probe(scds_handle, hostname, port, timeout);

          /*
           * Update service probe history,
           * take action if necessary.
           * Latch probe end time.
           */
          ht2 = gethrtime();
```

EXEMPLE C–5 xfnts_probe.c *(Suite)*

```
         /* Convert to milliseconds */
         dt = (ulong_t)((ht2 - ht1) / 1e6);

         /*
          * Compute failure history and take
          * action if needed
          */
         (void) scds_fm_action(scds_handle,
             probe_result, (long)dt);
      }   /* Each net resource */
   }    /* Keep probing forever */
}
```

Liste des codes de la méthode xfnts_start

Le gestionnaire RGM exécute la méthode `Start` sur un noeud de cluster lorsque le groupe de ressources qui contient la ressource du service de données est mis en ligne sur ce noeud ou lorsque la ressource est activée. La méthode de`démarrage_xfnts` active le démon `xfs` sur ce noeud.

EXEMPLE C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_start.c - Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * The start method for HA-XFS. Does some sanity checks on
 * the resource settings then starts the HA-XFS under PMF with
 * an action script.
 */

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

EXEMPLE C–6 xfnts_start.c *(Suite)*

```
   int rc;

   /*
    * Process all the arguments that have been passed to us from RGM
    * and do some initialization for syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Validate the configuration and if there is an error return back */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }

   /* Start the data service, if it fails return with an error */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

   /* Wait for the service to start up fully */
   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:
   /* Free up the Environment resources that were allocated */
   scds_close(&scds_handle);

   return (rc);
}
```

Liste des codes de la méthode `xfnts_stop`

Le RGM exécute la méthode `Stop` sur un noeud de cluster lorsque le groupe de ressources qui contient la ressource HA-XFS est déconnecté sur 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) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_stop.c - Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Stops the HA-XFS process using PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

   /* Process the arguments passed by RGM and initialize 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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of svc_stop method */
   return (rc);
}
```

Liste des codes de la méthode `xfnts_update`

Le RGM appelle la méthode `Update` pour notifier à une ressource en cours d'exécution que ses propriétés ont été modifiées. Le RGM exécute la méthode `Update` lorsqu'une action administrative réussit à définir les 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) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_update.c - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Some of the resource properties might have been updated. All such
 * updatable properties are related to the fault monitor. Hence, just
 * restarting the monitor should be enough.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /*
    * check if the Fault monitor is already running and if so stop and
    * restart it. The second parameter to scds_pmf_restart_fm() uniquely
    * identifies the instance of the fault monitor that needs to be
    * restarted.
    */
```

EXEMPLE C–8 xfnts_update.c *(Suite)*

```
   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.");
      /* Free up all the memory allocated by scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Completed successfully.");

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   return (0);
}
```

Liste des codes de la méthode xfnts_validate

Cette méthode vérifie l'existence du répertoire indiqué par la propriété Confdir_list. Le gestionnaire RGM appelle cette méthode lorsque l'administrateur du cluster crée le service de données et met à jour ses propriétés. La méthode Monitor_check appelle cette méthode chaque fois que le détecteur de pannes bascule le service de données sur un autre noeud.

EXEMPLE C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_validate.c - validate method for HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Check to make sure that the properties have been set properly.
 */
```

EXEMPLE C–9 xfnts_validate.c *(Suite)*

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

   /* Process arguments passed by RGM and initialize 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);

   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);

   /* Return the result of validate method */
   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.

Cette annexe aborde les points suivants :

Noms légaux RGM

Les noms RGM se divisent en plusieurs catégories :

- noms de groupes de ressources ;
- noms de types de ressources ;
- noms de ressources ;
- noms de propriétés ;
- noms de libellés d'énumération.

Règles s'appliquant aux noms, à l'exception des noms de types de ressources

Tous les noms (sauf ceux des types de ressources) doivent respecter les règles suivantes :

- Ils doivent être au format ASCII.
- Ils doivent commencer par une lettre.
- Ils peuvent comprendre des majuscules, des minucules, des chiffres, des tirets (-) et des caractères de soulignement (_) sont autorisés.

- Ils ne doivent pas comprendre plus de 255 caractères.

Format des noms de types de ressources

Le format du nom complet d'un type de ressources dépend du type de la ressource :

- Si le fichier RTR du type de ressources contient la directive `#$upgrade`, le format est le suivant :

  vendor-id . *base-rt-name* : *rt-version*

- Si le fichier RTR du type de ressources *ne* contient *pas* la directive `#$upgrade`, le format est le suivant :

  vendor-id . *base-rt-name*

Un point sépare *vendor-id* et *base-rt-name*. Un deux-points sépare *base-rt-name* et *rt-version*.

Les variables de ce format sont les suivantes :

id-fournisseur	Spécifie l'ID fournisseur utilisé comme préfixe, c'est-à-dire la valeur de la propriété de type de ressources `Vendor_id` dans le fichier RTR. Si vous développez un type de ressources, choisissez un ID fournisseur unique, par exemple, le symbole boursier de votre société. L'ID fournisseur des types de ressources développés par Sun Microsystems, Inc. est `SUNW`.
nom-rt-base	Spécifie le nom court du type de ressources, c'est-à-dire la valeur de la propriété de type de ressources `Resource_type` dans le fichier RTR.
rt-version	Spécifie le numéro de version utilisé comme suffixe, c'est-à-dire la valeur de la propriété de type de ressources `RT_version` dans le fichier RTR. Le suffixe de version fait partie du nom complet du type de ressources *uniquement* si le fichier RTR contient la directive `#$upgrade`. La directive `#$upgrade` a été introduite dans la version 3.1 du logiciel Sun Cluster.

Remarque – Si une seule version d'un nom court de type de ressources est enregistrée, il n'est pas nécessaire d'utiliser le nom complet dans les commandes `scrgadm`. Vous pouvez ne pas mentionner le préfixe d'ID fournisseur et/ou le suffixe du numéro de version.

Pour obtenir plus d'informations, reportez-vous à la rubrique "Propriétés des types de ressources" à la page 247.

EXEMPLE D–1 Nom complet de type de ressources avec la directive `#$upgrade`

Cet exemple présente le nom complet d'un type de ressources dont les propriétés sont définies comme suit dans le fichier RTR :

- `Vendor_id=SUNW`
- `Resource_type=sample`
- `RT_version=2.0`

Le nom complet du type de ressources défini par le fichier RTR est le suivant :

```
SUNW.sample:2.0
```

EXEMPLE D–2 Nom complet d'un type de ressources sans directive `#$upgrade`

Cet exemple présente le nom complet d'un type de ressources dont les propriétés sont définies comme suit dans le fichier RTR :

- `Vendor_id=SUNW`
- `Resource_type=nfs`

Le nom complet du type de ressources défini par le fichier RTR est le suivant :

```
SUNW.nfs
```

Valeurs RGM

Les valeurs utilisées par RGM se classent en deux catégories : les valeurs de propriété et les valeurs de description. Ces deux catégories sont soumises 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 pas contenir les caractères suivants :
  - Null
  - Nouvelle ligne
  - Virgule (,)
  - 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 répondre à des exigences particulières pour pouvoir prétendre devenir une application à haut niveau de disponibilité (HA). Ces exigences sont répertoriées dans la rubrique "Analyse du caractère approprié de l'application" à la page 29. Cette annexe fournit de plus amples détails sur certains éléments de cette liste.

Pour assurer la haute disponibilité d'une application, l'on configure ses ressources en groupes de ressources. L'on place ensuite ses données dans un système de fichiers de cluster à haut niveau de disponibilité : ainsi, en cas de panne d'un serveur, elles seront accessibles aux serveurs opérationnels. Pour plus d'informations sur les systèmes de fichiers de cluster, voir *Guide des notions fondamentales de Sun Cluster pour SE Solaris*.

Pour permettre aux clients d'accéder au réseau, on configure une adresse IP réseau logique 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.

Cette annexe aborde les points suivants :

Données multi-hôtes

Les ensembles de disques des systèmes de fichiers de clusters à haut niveau de disponibilité sont multi-hôtes. Ainsi, en cas de panne d'un hôte physique, l'un des hôtes opérationnels peut accéder au disque. Pour qu'une application présente un haut niveau de disponibilité, ses données doivent être hautement disponibles. Elles doivent donc se trouver dans des systèmes de fichiers accessibles depuis un grand nombre de nœuds du cluster. Sun Cluster prend en charge divers systèmes de fichiers à haut niveau de disponibilité, notamment les systèmes de fichiers HA globaux, le système FFS (Failover File System, système de fichiers de basculement) et, dans les environnements qui utilisent Oracle Real Application Clusters, le système de fichiers partagé QFS.

Le système de fichiers du cluster est monté sur des groupes de périphériques créés en tant qu'entités indépendantes. Vous pouvez utiliser certains de ces groupes de périphériques en tant que systèmes de fichiers de cluster montés et d'autres, comme périphériques de disques bruts à utiliser avec un service de données tel que HA Oracle.

Une application peut comporter des options de ligne de commande ou des fichiers de configuration pointant vers l'emplacement des fichiers de données. Si elle utilise des chemins codés en dur, vous pouvez les remplacer par des liens symboliques pointant vers un système de fichiers de cluster, sans modifier le code de l'application. Pour plus d'informations sur l'utilisation des liens symboliques, voir "Utilisation de liens symboliques pour déterminer l'emplacement des données multi-hôtes " à la page 338.

Dans l'hypothèse la plus pessimiste, vous devrez modifier le code source de l'application afin d'y ajouter un mécanisme qui pointe vers l'emplacement réel des données et que vous implémenterez en créant des arguments de ligne de commande supplémentaires, par exemple.

Sun Cluster prend en charge les systèmes de fichiers UFS UNIX et les périphériques de disques bruts HA configurés dans un gestionnaire de volumes. Lors de l'installation et de la configuration du logiciel, l'administrateur du cluster doit définir les ressources de disque à utiliser pour les systèmes de fichiers UFS et pour les périphériques de disques bruts. En règle générale, les périphériques de disques bruts 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 que les chemins des fichiers de données d'une application soient codés en dur, sans aucun mécanisme permettant de les ignorer. Pour ne pas avoir à modifier le code de l'application, vous pouvez utiliser des liens symboliques.

Supposons que l'application nomme son fichier de données en utilisant le nom de chemin codé en dur `/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, un lien symbolique vers `/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. Supposons que l'application mette à jour un fichier en créant un nouveau fichier temporaire `/etc/mydatafile.new`, puis en lui donnant le nom du fichier initial, à l'aide de l'appel système `rename()` (ou de la commande `mv`). La démarche adoptée par le service de données (créer un fichier temporaire, puis le renommer) vise à garantir que le contenu de son fichier de données sera toujours bien formé.

Malheureusement, l'action `rename()` détruit le lien symbolique. Le nom `/etc/mydatafile` désigne alors un fichier standard qui réside sur le même système de fichiers que le répertoire `/etc`, et non dans le système de fichiers 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 est le suivant : l'application ne « reconnaît » pas le lien symbolique – elle n'a d'ailleurs pas été écrite pour gérer ce type de lien. 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 œuvre de l'application doit adopter un comportement qui ne supprime pas les liens symboliques. Les liens symboliques ne constituent donc pas la solution parfaite au problème du stockage des données dans les systèmes de fichiers 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. Si c'est le cas, il vous faudra peut-être le modifier pour qu'il cesse d'utiliser le nom d'hôte physique et se serve d'un nom d'hôte logique, c'est-à-dire d'un nom d'hôte configuré dans une ressource de noms d'hôtes logiques située dans le même groupe de ressources que la ressource d'application.

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

On appelle *hôte multihome* un hôte situé sur plusieurs réseaux publics à la fois. Les hôtes de ce type ont plusieurs noms d'hôte et adresses IP : une paire nom d'hôte–adresse IP pour chaque 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 même qu'un nom d'hôte physique pourra correspondre à plusieurs paires nom d'hôte–adresse IP, chaque groupe de ressources pourra avoir plusieurs paires nom d'hôte–adresse IP (une pour chaque réseau public). Quand Sun Cluster déplace un groupe de ressources d'un hôte physique à l'autre, il transfère toutes les paires nom d'hôte–adresse IP du groupe de ressources.

Celles-ci sont configurées comme des ressources de nom d'hôte logique contenues dans le groupe de ressources. Elles sont définies par l'administrateur du cluster au moment de la création et de la configuration du groupe de ressources. L'API du service de données Sun Cluster propose des utilitaires permettant de rechercher ces paires nom d'hôte–adresse IP.

La plupart des démons de service de données disponibles dans le commerce et dédiés à l'origine à l'environnement 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 lie efficacement à toutes les adresses IP actuellement configurées sur la machine. Ainsi, il ne sera généralement pas nécessaire de modifier un démon de service de données qui utilise `INADDR_ANY` pour la gestion des adresses réseau logiques Sun Cluster.

Liaison à `INADDR_ANY` et liaison à des 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. La machine a une adresse IP pour son propre hôte physique et des adresses IP supplémentaires pour les adresses réseau (noms d'hôtes logiques) qu'elle gère. 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 dans un environnement Sun Cluster s'ils se lient 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 `>Network_resources_used` permet à l'utilisateur final de configurer un ensemble spécifique de ressources d'adresse réseau auquel la ressource de l'application peut être liée. La propriété `Network_resources_used` doit être déclarée dans le fichier RTR des types de ressources qui ont besoin de cette option.

Lorsque le RGM met le groupe de ressources en ligne ou hors ligne, il respecte un ordre spécifique pour connecter, déconnecter et configurer les adresses réseau comme actives ou inactives, suivant le moment où il appelle les méthodes des ressources du service de données. Voir "Choix des méthodes `Start` et `Stop` à utiliser" à la page 46.

Lorsque la méthode `Stop` du service de données est retournée, celui-ci doit s'être arrêté, à l'aide des adresses réseau du groupe de ressources. De même, lorsque la méthode `Start` 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 ses méthodes d'arrêt et de démarrage arrêtent et redémarrent ses démons, le service de données s'arrête et démarre en utilisant les adresses réseau au bon moment.

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. S'ils 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 notifient l'utilisateur qu'un long processus de relance est en cours et lui permettent de choisir s'il souhaite continuer.

ANNEXE **F**

Définitions de types de documents pour le protocole CRNP

Cette annexe comprend les définitions de types de documents suivants (DTD) pour le protocole CRNP (Cluster Reconfiguration Notification Protocol) :

DTD XML SC_CALLBACK_REG

Remarque – La structure de données NVPAIR utilisée à la fois par SC_CALLBACK_REG et par SC_EVENT n'est définie qu'une seule fois.

```
<!— SC_CALLBACK_REG XML format specification
        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:

A client of the Cluster Reconfiguration Notification Protocol should use this xml format
to register initially with the service, to subsequently register for more events, to
subsequently remove registration of some events, or to remove itself from the service
entirely.

A client is uniquely identified by its callback IP and port.  The port is defined in the
SC_CALLBACK_REG element, and the IP is taken as the source IP of the registration
connection.  The final attribute of the root SC_CALLBACK_REG element is either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS, depending on which form of the
message the client is using.
```

```
The SC_CALLBACK_REG contains 0 or more SC_EVENT_REG sub-elements.

One SC_EVENT_REG is the specification for one event type.  A client may specify only the
CLASS (an attribute of the SC_EVENT_REG element), or may specify a SUBCLASS (an optional
attribute) for further granularity.  Also, the SC_EVENT_REG has as subelements 0 or more
NVPAIRs, which can be used to further specify the event.

Thus, the client can specify events to whatever granularity it wants.  Note that a client
cannot both register for and unregister for events in the same message.  However a client
can subscribe to the service and sign up for events in the same message.

Note on versioning: the VERSION attribute of each root element is marked "fixed", which
means that all message adhering to these DTDs must have the version value specified.  If a
new version of the protocol is created, the revised DTDs will have a new value for this
fixed" VERSION attribute, such that all message adhering to the new version must have the
new version number.
—>
<!— SC_CALLBACK_REG definition

The root element of the XML document is a registration message.  A registration message
consists of the callback port and the protocol version as attributes, and either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS attribute, specifying the
registration type.  The ADD_CLIENT, ADD_EVENTS, and REMOVE_EVENTS types should have one or
more SC_EVENT_REG subelements.  The REMOVE_CLIENT should not specify an SC_EVENT_REG
subelement.
        ATTRIBUTES:
               VERSION        The CRNP protocol version of the message.
               PORT           The callback port.
               REG_TYPE       The type of registration.  One of:
                       ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENTS:
SUBELEMENTS: SC_EVENT_REG (0 or more)
—>
<!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

>
<!— SC_EVENT_REG definition

The SC_EVENT_REG defines an event for which the client is either registering or
unregistering interest in receiving event notifications.  The registration can be for any
level of granularity, from only event class down to specific name/value pairs that must be
present.  Thus, the only required attribute is the CLASS.  The SUBCLASS attribute, and the
NVPAIRS sub-elements are optional, for higher granularity.

Registrations that specify name/value pairs are registering interest in notification of
messages from the class/subclass specified with ALL name/value pairs present.
Unregistrations that specify name/value pairs are unregistering interest in notifications
that have EXACTLY those name/value pairs in granularity previously specified.
Unregistrations that do not specify name/value pairs unregister interest in ALL event
notifications of the specified class/subclass.
```

```
        ATTRIBUTES:
               CLASS:        The event class for which this element is registering
                             or unregistering interest.
               SUBCLASS:     The subclass of the event (optional).

        CONTENTS:
               SUBELEMENTS: 0 or more NVPAIRs.
—>
<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA               #REQUIRED
       SUBCLASS       CDATA               #IMPLIED
>
```

DTD XML NVPAIR

```
<!— NVPAIR XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:
              An nvpair element is meant to be used in an SC_EVENT or SC_CALLBACK_REG
              element.
—>
<!— NVPAIR definition

       The NVPAIR is a name/value pair to represent arbitrary name/value combinations.
       It is intended to be a direct, generic, translation of the Solaris nvpair_t
       structure used by the sysevent framework.  However, there is no type information
       associated with the name or the value (they are both arbitrary text) in this xml
       element.

       The NVPAIR consists simply of one NAME element and one or more VALUE elements.
       One VALUE element represents a scalar value, while multiple represent an array
       VALUE.

       ATTRIBUTES:

       CONTENTS:
              SUBELEMENTS: NAME(1), VALUE(1 or more)
—>
<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— NAME definition

        The NAME is simply an arbitrary length string.

        ATTRIBUTES:
```

```
        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT NAME (#PCDATA)>

<!— VALUE definition
        The VALUE is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT VALUE (#PCDATA)>
```

DTD XML SC_REPLY

```
<!— SC_REPLY XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.
—>
<!— SC_REPLY definition

        The root element of the XML document represents a reply to a message.  The reply
        contains a status code and a status message.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                STATUS_CODE:    The return code for the message.  One of the
                                following: OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
                                MALFORMED, INVALID_XML, VERSION_TOO_HIGH, or
                                VERSION_TOO_LOW.

                CONTENTS:
                        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
>
<!— SC_STATUS_MSG definition
        The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
        code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.
```

```
        ATTRIBUTES:

        CONTENTS:
                Arbitrary string.
—>
<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

DTD XML SC_EVENT

Remarque – La structure de données NVPAIR utilisée à la fois par SC_CALLBACK_REG et par SC_EVENT n'est définie qu'une seule fois.

```
<!— SC_EVENT XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        The root element of the XML document is intended to be a direct, generic,
        translation of the Solaris syseventd message format.  It has attributes to
        represent the class, subclass, vendor, and publisher, and contains any number of
        NVPAIR elements.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                CLASS:          The sysevent class of the event
                SUBCLASS:       The subclass of the event
                VENDOR:         The vendor associated with the event
                PUBLISHER:      The publisher of the event
        CONTENTS:
                SUBELEMENTS: NVPAIR (0 or more)
—>
<!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 présente l'application complète `CrnpClient.java`, qui est décrite plus en détail au Chapitre 12.

Contenu de `CrnpClient.java`

```
/*
 * CrnpClient.java
 * ================
 *
 * Note regarding XML parsing:
 *
 * This program uses the Sun Java Architecture for XML Processing (JAXP) API.
 * See http://java.sun.com/xml/jaxp/index.html for API documentation and
 * availability information.
 *
 * This program was written for Java 1.3.1 or higher.
 *
 * Program overview:
 *
 * The main thread of the program creates a CrnpClient object, waits for the
 * user to terminate the demo, then calls shutdown on the CrnpClient object
 * and exits the program.
 *
 * The CrnpClient constructor creates an EventReceptionThread object,
 * opens a connection to the CRNP server (using the host and port specified
 * on the command line), constructs a registration message (based on the
 * command-line specifications), sends the registartion message, and reads
 * and parses the reply.
 *
 * The EventReceptionThread creates a listening socket bound to
 * the hostname of the machine on which this program runs, and the port
 * specified on the command line. It waits for an incoming event callback,
```

```
 * at which point it constructs an XML Document from the incoming socket
 * stream, which is then passed back to the CrnpClient object to process.
 *
 * The shutdown method in the CrnpClient just sends an unregistration
 * (REMOVE_CLIENT) SC_CALLBACK_REG message to the crnp server.
 *
 * Note regarding error handling: for the sake of brevity, this program just
 * exits on most errors.  Obviously, a real application would attempt to handle
 * some errors in various ways, such as retrying when appropriate.
 */

// JAXP packages
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.*;

// standard packages
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * class CrnpClient
 * ----------------
 * See file header comments above.
 */
class CrnpClient
{
    /*
     * main
     * ----
     * The entry point of the execution, main simply verifies the
     * number of command-line arguments, and constructs an instance
     * of a CrnpClient to do all the work.
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

        /* Verify the number of command-line arguments */
        if (args.length < 4) {
            System.out.println(
                "Usage: java CrnpClient crnpHost crnpPort "
                + "localPort (-ac | -ae | -re) "
                + "[(M | A | RG=name | R=name) [...]]");
            System.exit(1);
        }


        /*
```

```
     * We expect the command line to contain the ip/port of the
     * crnp server, the local port on which we should listen, and
     * arguments specifying the type of registration.
     */
    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);
    }

    // Create the CrnpClient
    CrnpClient client = new CrnpClient(regIp, regPort, localPort,
        args);

    // Now wait until the user wants to end the program
    System.out.println("Hit return to terminate demo...");

    // read will block until the user enters something
    try {
        System.in.read();
    } catch (IOException e) {
        System.out.println(e.toString());
    }

    // shutdown the client
    client.shutdown();
    System.exit(0);
}

/*
 * ======================
 * public methods
 * ======================
 */

/*
 * CrnpClient constructor
 * ----------------------
 * Parses the command line arguments so we know how to contact
 * the crnp server, creates the event reception thread, and starts it
 * running, creates the XML DocumentBuilderFactory obect, and, finally,
 * registers for callbacks with the crnp server.
 */
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
        localPort = localPortIn;
        regs = clArgs;
```

```
        /*
         * Setup the document builder factory for
         * xml processing.
         */
        setupXmlProcessing();

        /*
         * Create the EventReceptionThread, which creates a
         * ServerSocket and binds it to a local ip and port.
         */
        createEvtRecepThr();

        /*
         * Register with the crnp server.
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * ---------------
 * Callback into the CrnpClient, used by the EventReceptionThread
 * when it receives event callbacks.
 */
public void processEvent(Event event)
{
    /*
     * For demonstration purposes, simply print the event
     * to System.out.  A real application would obviously make
     * use of the event in some way.
     */
    event.print(System.out);
}

/*
 * shutdown
 * -------------
 * Unregister from the CRNP server.
 */
public void shutdown()
{
    try {
        /* send an unregistration message to the server */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}
```

```
/*
 * ======================
 * private helper methods
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Create the document builder factory for
 * parsing the xml replies and events.
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // We don't need to bother validating
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // We want to ignore comments and whitespace
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // Coalesce CDATA sections into TEXT nodes.
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -------------------
 * Creates a new EventReceptionThread object, saves the ip
 * and port to which its listening socket is bound, and
 * starts the thread running.
 */
private void createEvtRecepThr() throws Exception
{
    /* create the thread object */
    evtThr = new EventReceptionThread(this);

    /*
     * Now start the thread running to begin listening
     * for event delivery callbacks.
     */
    evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
 * Creates a socket connection to the crnp server and sends
 * an event registration message.
```

```
 */
private void registerCallbacks() throws Exception
{
    System.out.println("About to register");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the registration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}

/*
 * unregister
 * ----------
 * As in registerCallbacks, we create a socket connection to
 * the crnp server, send the unregistration message, wait for
 * the reply from the server, then close the socket.
 */
private void unregister() throws Exception
{
    System.out.println("About to unregister");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the unregistration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}
```

```
/*
 * createRegistrationString
 * ------------------
 * Constructs a CallbackReg object based on the command line arguments
 * to this program, then retrieves the XML string from the CallbackReg
 * object.
 */
private String createRegistrationString() throws Exception
{
    /*
     * create the actual CallbackReg class and set the port.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    // set the registration type
    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);
    }

    // add the events
    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
 * ----------------
 * Creates an XML registartion event with class EC_Cluster, and no
 * subclass.
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
```

```
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
 * ---------------------
 * Creates an XML registration event with class EC_Cluster, subclass
 * ESC_cluster_memberhip.
 */
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
 * subclass ESC_cluster_rg_state, and one "rg_name" nvpair (based
 * on input parameter).
 */
private Event createRgEvent(String rgname)
{
    /*
     * Create a Resource Group state change event for the
     * rgname Resource Group.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    /*
     * Construct the event object and set the class and subclass.
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * Create the nvpair object and add it to the Event.
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
}

/*
 * createREvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
```

```
 * subclass ESC_cluster_r_state, and one "r_name" nvpair (based
 * on input parameter).
 */
private Event createREvent(String rname)
{
    /*
     * Create a Resource state change event for the
     * rgname Resource.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    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
 * ------------------
 * Constructs a REMOVE_CLIENT CallbackReg object, then retrieves
 * the XML string from the CallbackReg object.
 */
private String createUnregistrationString() throws Exception
{
    /*
     * Crate the CallbackReg object.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
     * we marshall the registration to the OutputStream
     */
    String xmlStr = cbReg.convertToXml();

    // Print the string for debugging purposes
    System.out.println(xmlStr);
    return (xmlStr);
}


/*
 * readRegistrationReply
 * ------------------------
 * Parse the xml into a Document, construct a RegReply object
 * from the document, and print the RegReply object.  Note that
 * a real application would take action based on the status_code
```

```
     * of the RegReply object.
     */
    private void readRegistrationReply(InputStream stream)
        throws Exception
    {
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
    }

    /* private member variables */
    private InetAddress regIp;
    private int regPort;
    private EventReceptionThread evtThr;
    private String regs[];

    /* public member variables */
    public int localPort;
    public DocumentBuilderFactory dbf;
}

/*
 * class EventReceptionThread
 * --------------------------
 * See file header comments above.
 */
class EventReceptionThread extends Thread
{
    /*
     * EventReceptionThread constructor
     * --------------------------------
     * Creates a new ServerSocket, bound to the local hostname and
     * a wildcard port.
     */
    public EventReceptionThread(CrnpClient clientIn) throws IOException
    {
        /*
         * keep a reference to the client so we can call it back
         * when we get an event.
         */
        client = clientIn;

        /*
         * Specify the IP to which we should bind.  It's
```

```
     * simply the local host ip.  If there is more
     * than one public interface configured on this
     * machine, we'll go with whichever one
     * InetAddress.getLocalHost comes up with.
     *
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
        System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Called by the Thread.Start method.
 *
 * Loops forever, waiting for incoming connections on the ServerSocket.
 *
 * As each incoming connection is accepted, an Event object
 * is created from the xml stream, which is then passed back to
 * the CrnpClient object for processing.
 */
public void run()
{
    /*
     * Loop forever.
     */
    try {
        //
        // Create the document builder using the document
        // builder factory in the CrnpClient.
        //
        DocumentBuilder db = client.dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        while(true) {
            /* wait for a callback from the server */
            Socket sock = listeningSock.accept();

            // parse the input file
            Document doc = db.parse(sock.getInputStream());

            Event event = new Event(doc);
            client.processEvent(event);

            /* close the socket */
            sock.close();
        }
        // UNREACHABLE
```

```
        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }

    /* private member variables */
    private ServerSocket listeningSock;
    private CrnpClient client;
}

/*
 * class NVPair
 * ------------
 * This class stores a name/value pair (both Strings).  It knows how to
 * construct an NVPAIR XML message from its members, and how to parse
 * an NVPAIR XML Element into its members.
 *
 * Note that the formal specification of an NVPAIR allows for multiple values.
 * We make the simplifying assumption of only one value.
 */
class NVPair
{
    /*
     * Two constructors: the first creates an empty NVPair, the second
     * creates an NVPair from an NVPAIR XML Element.
     */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
     * Public setters.
     */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }

    /*
     * Prints the name and value on a single line.
     */
    public void print(PrintStream out)
    {
```

```
    out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an NVPAIR XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.
 */
public Element createXmlElement(Document doc)
{
    // Create the element.
    Element nvpair = (Element)
        doc.createElement("NVPAIR");
    //
    // Add the name.  Note that the actual name is
    // a separate CDATA section.
    //
    Element eName = doc.createElement("NAME");
    Node nameData = doc.createCDATASection(name);
    eName.appendChild(nameData);
    nvpair.appendChild(eName);
    //
    // Add the value.  Note that the actual value is
    // a separate CDATA section.
    //
    Element eValue = doc.createElement("VALUE");
    Node valueData = doc.createCDATASection(value);
    eValue.appendChild(valueData);
    nvpair.appendChild(eValue);

    return (nvpair);
}

/*
 * retrieveValues
 * ----------------
 * Parse the XML Element to retrieve the name and value.
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // Find the NAME element
    //
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }
```

```
        //
        // Get the TEXT section
        //
        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;
        }

        // Retrieve the value
        name = n.getNodeValue();

        //
        // Now get the value element
        //
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "VALUE node.");
            return;
        }

        //
        // Get the TEXT section
        //
        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;
        }

        // Retrieve the value
        value = n.getNodeValue();
    }


    /*
     * Public accessors
     */
    public String getName()
    {
        return (name);
    }

    public String getValue()
    {
        return (value);
    }

    // Private member vars
    private String name, value;
}
```

```
/*
 * class Event
 * -----------
 * This class stores an event, which consists of a class, subclass, vendor,
 * publisher, and list of name/value pairs.  It knows how to
 * construct an SC_EVENT_REG XML Element from its members, and how to parse
 * an SC_EVENT XML Element into its members.  Note that there is an assymetry
 * here: we parse SC_EVENT elements, but construct SC_EVENT_REG elements.
 * That is because SC_EVENT_REG elements are used in registration messages
 * (which we must construct), while SC_EVENT elements are used in event
 * deliveries (which we must parse).  The only difference is that SC_EVENT_REG
 * elements don't have a vendor or publisher.
 */
class Event
{

    /*
     * Two constructors: the first creates an empty Event; the second
     * creates an Event from an SC_EVENT XML Document.
     */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // Convert the document to a string to print for debugging
        // purposes.
        //
        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());

        // Do the actual parsing.
        retrieveValues(doc);
    }

    /*
     * Public setters.
```

```
 */
public void setClass(String classIn)
{
    regClass = classIn;
}

public void setSubclass(String subclassIn)
{
    regSubclass = subclassIn;
}

public void addNvpair(NVPair nvpair)
{
    nvpairs.add(nvpair);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an SC_EVENT_REG XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.  Relies on the NVPair createXmlElement ability.
 */
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);
}

/*
 * Prints the member vars on multiple lines.
 */
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
 * ----------------
 * Parse the XML Document to retrieve the class, subclass, vendor,
 * publisher, and nvpairs.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_EVENT element.
    //
    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);

    //
    // Retrieve the values of the CLASS, SUBCLASS,
    // VENDOR and PUBLISHER attributes.
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // Retrieve all the nv pairs
    //
    for (Node child = n.getFirstChild(); child != null;
         child = child.getNextSibling())
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 * Public accessor methods.
 */
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 member vars.
    private String regClass, regSubclass;
    private Vector nvpairs;
    private String vendor, publisher;
}


/*
 * class CallbackReg
 * -----------
 * This class stores a port and regType (both Strings), and a list of Events.
 * It knows how to construct an SC_CALLBACK_REG XML message from its members.
 *
 * Note that this class does not need to be able to parse SC_CALLBACK_REG
 * messages, because only the CRNP server must parse SC_CALLBACK_REG
 * messages.
 */
class CallbackReg
{
    // Useful defines for the setRegType method
    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 setters.
     */
    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
 * ------------------
 * Constructs an SC_CALLBACK_REG XML Document from the member
 * variables.  Relies on the Event createXmlElement ability.
 */
public String convertToXml()
{
    Document document = null;
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.newDocument();
    } catch (ParserConfigurationException pce) {
        // Parser with specified options can't be built
        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);

        //
        // Now convert the document to a string.
        //
        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 member vars
    private String port;
    private String regType;
    private Vector regEvents;
}

/*
 * class RegReply
 * ------------
 * This class stores a status_code and status_msg (both Strings).
 * It knows how to parse an SC_REPLY XML Element into its members.
 */
class RegReply
{

    /*
     * The only constructor takes an XML Document and parses it.
     */
    public RegReply(Document doc)
    {
        //
        // Now convert the document to a string.
        //
        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);
}

/*
 * Public accessors
 */
public String getStatusCode()
{
    return (statusCode);
}

public String getStatusMsg()
{
    return (statusMsg);
}

/*
 * Prints the info on a single line.
 */
public void print(PrintStream out)
{
    out.println(statusCode + ": " +
        (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the statusCode and statusMsg.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_REPLY element.
    //
    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);

    // Retrieve the value of the STATUS_CODE attribute
    statusCode = ((Element)n).getAttribute("STATUS_CODE");

    //
    // Find the SC_STATUS_MSG element
```

```
        //
        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;
        }

        //
        // Get the TEXT section, if there is one.
        //
        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;
        }

        // Retrieve the value
        statusMsg = n.getNodeValue();
    }

    // private member vars
    private String statusCode;
    private String statusMsg;
}
```

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X