Sun Cluster Entwicklerhandbuch Datendienste für Solaris OS

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

Teilenr.: 819–0182
September 2004, Revision A

041105@10082

Inhalt

Vorwort

Das *Sun Cluster Entwicklerhandbuch Datendienste für Solaris OS* enthält Informationen zur Verwendung der Ressourcenverwaltungs-API für die Entwicklung von Sun™ Cluster-Datendiensten sowohl unter SPARC® als auch für x86-basierte Systeme.

Hinweis – In diesem Dokument bezieht sich der Begriff “x86” auf die Intel 32-Bit-Familie von Mikroprozessorchips sowie auf kompatible, von AMD hergestellte Mikroprozessorchips.

Hinweis – Sun Cluster-Software läuft auf zwei Plattformen, SPARC und x86. Die Informationen in diesem Dokument beziehen sich auf beide Plattformen, wenn nicht in einem eigenen Kapitel, Abschnitt, Anmerkung, Unterpunkt, Abbildung, Tabelle oder Beispiel anderweitige Angaben erfolgen.

Zielgruppe dieses Handbuchs

Dieses Dokument richtet sich an Entwickler mit weitreichender Erfahrung im Umgang mit Software und Hardware von Sun. Die Informationen in diesem Buch setzen Kenntnisse des Solaris™-Betriebssystems voraus.

Aufbau dieses Buches

Das *Sun Cluster Entwicklerhandbuch Datendienste für Solaris OS* enthält folgende Kapitel und Anhänge:

- Kapitel 1 bietet einen Überblick über die erforderlichen Konzepte zum Entwickeln eines Datendienstes.
- Kapitel 2 enthält detaillierte Informationen zum Entwickeln eines Datendienstes.
- Kapitel 3 behandelt die Themen, die zum Aufrüsten eines Ressourcentyps und Migrieren einer Ressource bekannt sein müssen.
- Kapitel 4 beschreibt die Zugriffsfunktionen und Rückmeldemethoden, aus denen sich die Ressourcenverwaltungs-API (Resource Management API, RMAPI) zusammensetzt.
- Kapitel 5 enthält einen Sun Cluster-Beispieldatendienst für die Anwendung `in.named()`.
- Kapitel 6 bietet einen Überblick über die Anwendungsprogrammierschnittstellen, aus denen sich die DSDL (Data Services Development Library, Datendienst-Entwicklungsbibliothek) zusammensetzt.
- Kapitel 7 erläutert, wie die DSDL in der Regel beim Entwurf und der Implementierung von Ressourcentypen eingesetzt wird.
- Kapitel 8 beschreibt einen mit DSDL implementierten Beispielressourcentyp.
- Kapitel 9 beschreibt SunPlex™ Agent Builder.
- Kapitel 10 beschreibt das Erstellen eines generischen Datendienstes.
- Kapitel 11 beschreibt die DSDL-API-Funktionen.
- Kapitel 12 enthält Informationen zum CRNP (Cluster Reconfiguration Notification Protocol). Das CRNP ermöglicht die “Cluster-Unterstützung” von Failover- und Scalable-Anwendungen.
- Anhang A beschreibt die Standardressourcentyp-, Ressourcengruppen- und Ressourceneigenschaften.
- Anhang B enthält den vollständigen Code für jede Methode im Beispieldatendienst.
- Anhang C listet den vollständgen Code für jede Methode im `SUNW.xfnts()` -Ressourcentyp auf.
- Anhang D listet die Anforderungen für zulässige Zeichen in Namen und Werten von Ressourcengruppen-Manager (RGM) auf.
- Anhang E listet die Anforderungen an gewöhnliche Anwendungen ohne Cluster-Unterstützung auf, die für den Einsatz als hoch verfügbare Anwendung (HA-Anwendung) erfüllt sein müssen.
- Anhang F listet die Dokumenttypdefinitionen für CRNP auf.

- Anhang G zeigt die vollständige `CrnpClient.java`-Anwendung, die in Kapitel 12 besprochen wird.

Verwandte Dokumentation

Informationen zu verwandten Sun Cluster-Themen finden Sie in der Dokumentation, die in der folgenden Tabelle genannt ist. Sämtliche Sun Cluster-Dokumentationen stehen unter `http://docs.sun.com` zur Verfügung.

Thema	Dokumentation
Überblick	*Sun Cluster Überblick für Solaris OS*
Konzepte	*Sun Cluster Konzepthandbuch für Solaris OS*
Hardware-Installation und -Verwaltung	*Sun Cluster 3.x Hardware Administration Manual for Solaris OS*
	Einzelne Hardwareverwaltungshandbücher
Softwareinstallation	*Sun Cluster Handbuch Softwareinstallation für Solaris OS*
Datendienstinstallation und -verwaltung	*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*
	Einzelne Datendiensthandbücher
Datendienstentwicklung	*Sun Cluster Entwicklerhandbuch Datendienste für Solaris OS*
Systemverwaltung	*Sun Cluster Handbuch Systemverwaltung für Solaris OS*
Fehlermeldungen	*Sun Cluster Error Messages Guide for Solaris OS*
Befehle und Funktionen	*Sun Cluster Reference Manual for Solaris OS*

Eine vollständige Liste der Sun Cluster-Dokumentation finden Sie in den Versionshinweisen zu Ihrer Sun Cluster-Version unter `http://docs.sun.com`.

Eine vollständige Liste der Sun Cluster-Dokumentationen ist in den Versionshinweisen für Ihre Sun Cluster-Version auf `http://docs.sun.com` enthalten.

Hilfe anfordern

Wenden Sie sich im Falle von Problemen bei der Installation oder Verwendung von Sun Cluster an Ihren Kundendienst, und geben Sie folgende Informationen an:

- Ihren Namen und E-Mail-Adresse (ggf.)
- Firmennamen, Adresse, Telefonnummer
- Modell- und Seriennummern Ihrer Systeme
- Versionsnummer des Betriebssystems (zum Beispiel Solaris 10)
- Versionsnummer von Sun Cluster (z. B. Sun Cluster 3.1)

Sammeln Sie für Ihren Kundendienst mithilfe folgender Befehle Systeminformationen.

Befehl	Funktion
`prtconf -v`	Zeigt die Größe des Systemspeichers an und gibt Informationen zu Peripheriegeräten zurück.
`psrinfo -v`	Zeigt Informationen zu Prozessoren an.
`showrev -p`	Gibt die installierten Korrekturversionen zurück.
SPARC: `prtdiag -v`	Zeigt Informationen zu Systemdiagnosen an.
`/usr/cluster/bin/scinstall - pv`	Zeigt die Sun Cluster-Version und Paketversion an.

Halten Sie zudem den Inhalt der Datei `/var/adm/messages` bereit.

Zugriff auf die Online-Dokumentation von Sun

Über die Website docs.sun.com^{SM} erhalten Sie Zugriff auf die technische Online-Dokumentation von Sun. Sie können das Archiv unter docs.sun.com durchsuchen oder nach einem bestimmten Buchtitel oder Thema suchen. Die URL lautet: `http://docs.sun.com`.

Bestellen von Sun-Dokumentation

Ausgewählte Produktdokumentationen bietet Sun Microsystems auch in gedruckter Form an. Eine Liste dieser Dokumente und Hinweise zum Bezug finden Sie unter „Buy printed documentation" auf der Website http://docs.sun.com.

Typografische Konventionen

Die folgende Tabelle beschreibt die in diesem Buch verwendeten typographischen Kennzeichnungen.

TABELLE P–1 Typografische Konventionen

Schriftart oder Symbol	Bedeutung	Beispiel
`AaBbCc123`	Die Namen von Befehlen, Dateien, Verzeichnissen; Bildschirmausgabe.	Bearbeiten Sie Ihre `.login`-Datei. Verwenden Sie `ls -a` , um eine Liste aller Dateien zu erhalten. `Rechnername% Sie haben eine neue Nachricht.`
`AaBbCc123`	Die Eingaben des Benutzers, im Gegensatz zu den Bildschirmausgaben des Computers	`Rechnername%` **`su`** `Passwort:`
AaBbCc123	Befehlszeilen-Variable: durch einen realen Namen oder Wert ersetzen	Um eine Datei zu löschen, geben Sie Folgendes ein: **`rm`** *Dateiname*.
AaBbCc123	Buchtitel, neue Wörter oder Begriffe bzw. hervorzuhebende Wörter.	Lesen Sie dazu auch Kapitel 6 im *Benutzerhandbuch*. Diese werden *class*-Optionen genannt. Sie *müssen* als root angemeldet sein, um dies zu tun.

Beispiele für Shell-Eingabeaufforderungen in Befehlen

Die folgende Tabelle zeigt die Standard-Systemeingabeaufforderung und die Superbenutzer-Eingabeaufforderung für die C-Shell, die Bourne-Shell und die Korn-Shell.

TABELLE P–2 Shell-Eingabeaufforderungen

Shell	Eingabeaufforderung
C Shell-Eingabeaufforderung	`Rechnername%`
C Shell-Superbenutzer-Eingabeaufforderung	`Rechnername#`
Bourne Shell- und Korn Shell-Eingabeaufforderung	`$`
Bourne Shell- und Korn Shell-Superbenutzer-Eingabeaufforderung	`#`

KAPITEL 1

Überblick über die Ressourcenverwaltung

Dieses Buch enthält Richtlinien zum Erstellen von Ressourcentypen für Softwareanwendungen wie Oracle®, Sun Java™ System Web Server (früher Sun ™ ONE Web Server), DNS usw. Insofern ist es für Ressourcentypentwickler konzipiert.

Dieses Kapitel bietet einen Überblick über die Konzepte, die Sie für die Entwicklung eines Datendienstes beherrschen sollten, und enthält folgende Informationen:

Hinweis – In diesem Buch werden die Begriffe *Ressourcentyp* und *Datendienst* synonym verwendet. Der Begriff *Agent* kommt in dem Handbuch kaum vor, entspricht aber *Ressourcentyp* und *Datendienst*.

Sun Cluster-Anwendungsumgebung

Mithilfe des Sun Cluster-Systems können Anwendungen als hoch verfügbare und Scalable-Ressourcen ausgeführt und verwaltet werden. Das Cluster-Programm mit der Bezeichnung Ressourcengruppen-Manager bzw. RGM stellt den Mechanismus für hohe Verfügbarkeit und Skalierbarkeit bereit. Die Programmierschnittstelle für dieses Programm setzt sich aus folgenden Elementen zusammen.

- Einem Satz Rückmeldemethoden, die Sie schreiben und mit deren Hilfe RGM eine Anwendung auf dem Cluster steuern kann.

- Die Ressourcenverwaltungs-API (RMAPI), ein Satz Routinen und Funktionen auf niedriger Ebene, die zum Schreiben von Rückmeldemethoden eingesetzt werden können. Diese APIs sind in der `libscha.so`-Bibliothek implementiert.
- Prozessverwaltungsprogramme (Process Management Facilities, PMF) für das Überwachen und Neustarten von Prozessen auf dem Cluster.
- DSDL (Data Service Development Library, Datendienst-Entwicklungsbibliothek), eine Reihe von Bibliotheksfunktionen, welche die API niedriger Ebene und Prozessverwaltungsfunktionen auf einer höheren Ebene einkapseln. Sie bieten weitere Funktionalität und erleichtern damit das Schreiben von Rückmeldemethoden. Diese Funktionen sind in der `libdsdev.so` -Bibliothek implementiert.

Die folgende Abbildung verdeutlicht die Beziehungen zwischen den aufgeführten Elementen.

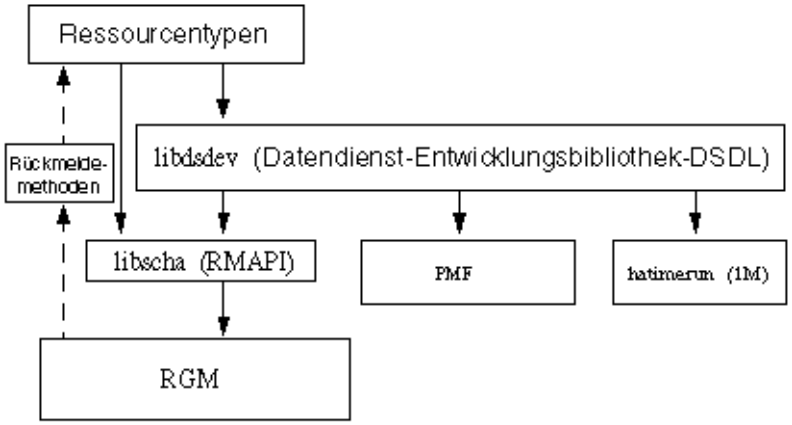


ABBILDUNG 1–1 Programmierarchitektur

Im Sun Cluster-Paket ist SunPlex™ Agent Builder enthalten, ein Tool zur Automatisierung des Datendienst-Erstellungsprozesses (siehe Kapitel 9). Agent Builder generiert Datendienstcode entweder in C unter Verwendung von DSDL-Funktionen zum Schreiben der Rückmeldemethoden oder in Korn-Shell (`ksh`) unter Verwendung von API-Befehlen auf niedriger Ebene zum Schreiben der Rückmeldemethoden.

RGM wird als Dämon auf jedem Cluster-Knoten ausgeführt und startet und stoppt die Ressourcen auf ausgewählten Knoten automatisch, entsprechend den vorkonfigurierten Richtlinien. RGM macht eine Ressource hoch verfügbar, wenn ein Knoten versagt oder neu startet, indem die Ressource auf dem betroffenen Knoten

gestoppt und auf einem anderen Knoten neu gestartet wird. RGM sorgt auch für das automatische Starten und Stoppen der ressourcenspezifischen Monitore, die Ressourcenfehler feststellen und fehlerhafte Ressourcen auf einen anderen Knoten verschieben sowie andere Aspekte der Ressourcenleistung überwachen können.

RGM unterstützt sowohl Failover-Ressourcen, die jeweils nur auf einem Knoten online sein können, als auch Scalable-Ressourcen, die auf mehreren Knoten gleichzeitig online sein können.

RGM-Modell

Dieser Abschnitt stellt einige grundlegende Begriffe vor und geht auf Einzelheiten von RGM und der dazugehörigen Schnittstellen ein.

RGM unterstützt drei große Gruppen von miteinander verbundenen Objekten: Ressourcentypen, Ressourcen und Ressourcengruppen. Anhand des folgenden Beispiels sollen diese Objekte vorgestellt werden.

Ein Entwickler implementiert einen Ressourcentyp, `ha-oracle`, der eine vorhandene Oracle DBMS-Anwendung hoch verfügbar macht. Ein Endbenutzer definiert jeweils eine Datenbank für Marketing, IT und Finanzen. Alle Datenbanken sind Ressourcen des Typs `ha-oracle`. Der Cluster-Administrator legt diese Ressourcen in unterschiedlichen Ressourcengruppen ab, so dass sie auf verschiedenen Knoten laufen und unabhängig voneinander Failover ausführen können. Ein Entwickler erstellt einen zweiten Ressourcentyp, ha-calender, um einen hoch verfügbaren Kalenderserver zu implementieren, der eine Oracle-Datenbank benötigt. Der Cluster-Administrator legt die Ressource für den Finanzkalender in derselben Ressourcengruppe wie die Finanzdatenbankressource ab, so dass beide Ressourcen auf demselben Knoten laufen und gemeinsam Failover ausführen.

Ressourcentypen

Ein Ressourcentyp besteht aus einer Softwareanwendung, die auf dem Cluster ausgeführt wird, Steuerprogrammen, die von RGM als Rückmeldemethoden zum Verwalten der Anwendung als Cluster-Ressource verwendet werden, sowie einem Satz Eigenschaften, die Bestandteil der statischen Cluster-Konfiguration sind. RGM verwendet Ressourcentypeigenschaften für die Verwaltung von Ressourcen eines bestimmten Typs.

Hinweis – Neben einer Softwareanwendung kann ein Ressourcentyp weitere Systemressourcen wie Netzwerkadressen darstellen.

Der Ressourcentypentwickler gibt die Eigenschaften für den Ressourcentyp an und stellt deren Werte in einer Ressourcentyp-Registrierungsdatei (RTR-Datei) ein. Die RTR-Datei hat ein klar definiertes Format, das in „Einstellen der Ressourcen- und Ressourcentypeigenschaften" auf Seite 35 und in der Online-Dokumentation unter `rt_reg`(4) beschrieben wird. Die Beschreibung einer RTR-Beispieldatei finden Sie in „Definieren der Ressourcentyp-Registrierungsdatei" auf Seite 92.

„Ressourcentypeigenschaften" auf Seite 253 enthält eine Liste der Ressourcentypeigenschaften.

Der Cluster-Administrator installiert und registriert die Ressourcentypimplementierung und die zugrunde liegende Anwendung auf einem Cluster. Das Registrierungsverfahren gibt die Informationen aus der Ressourcentyp-Registrierungsdatei in die Cluster-Konfiguration ein. Das Verfahren für das Registrieren eines Datendienstes wird im *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* beschrieben.

Ressourcen

Eine Ressource erbt die Eigenschaften und Werte ihres Ressourcentyps. Zusätzlich kann ein Entwickler Ressourceneigenschaften in der Ressourcentyp-Registrierungsdatei deklarieren. „Ressourceneigenschaften" auf Seite 260 enthält eine Liste der Ressourceneigenschaften.

Der Cluster-Administrator kann die Werte bestimmter Eigenschaften ändern, abhängig davon, wie sie in der Ressourcentyp-Registrierungsdatei (RTR-Datei) angegeben wurden. Eigenschaftsdefinitionen können zum Beispiel einen Bereich zulässiger Werte angeben und bestimmen, wann die Eigenschaft einstellbar ist. Beispiel: Bei Erstellung, Jederzeit, Nie. Innerhalb dieser Spezifikationen kann der Cluster-Administrator mithilfe von Verwaltungsbefehlen Änderungen an den Eigenschaften vornehmen.

Der Cluster-Administrator kann viele Ressourcen desselben Typs erstellen. Dabei hat jede Ressource ihren eigenen Namen und einen eigenen Satz Eigenschaftswerte, so dass mehr als eine Instanz der zugrunde liegenden Anwendung auf dem Cluster laufen kann. Für jede Instanz ist ein einmaliger Name innerhalb des Clusters erforderlich.

Ressourcengruppen

Jede Ressource muss in einer Ressourcengruppe konfiguriert werden. RGM bringt alle Ressourcen in einer Gruppe gemeinsam auf demselben Knoten online bzw. offline. Wenn RGM eine Ressourcengruppe online oder offline bringt, ruft das Programm Rückmeldemethoden für die einzelnen Ressourcen in der Gruppe auf.

Die Knoten, auf denen eine Ressourcengruppe zurzeit online ist, werden als *primär* bzw. *Primärknoten* bezeichnet. Eine Ressourcengruppe wird von jedem ihrer Primärknoten *unterstützt*. Jeder Ressourcengruppe ist eine `Nodelist`-Eigenschaft zugeordnet, die vom Cluster-Administrator eingestellt wird und die alle *potenziellen Primärknoten* bzw. Master der Ressourcengruppe identifiziert.

Eine Ressourcengruppe verfügt zudem über einen Satz Eigenschaften. Diese Eigenschaften umfassen Konfigurationseigenschaften, die vom Cluster-Administrator eingestellt werden, sowie dynamische Eigenschaften, die RGM einstellt und die den aktiven Zustand der Ressourcengruppe wiedergeben.

RGM definiert zwei Typen von Ressourcengruppen, Failover und Scalable. Eine Failover-Ressourcengruppe kann nur jeweils auf einem Knoten online sein, während eine Scalable-Ressourcengruppe auf mehreren Knoten gleichzeitig online sein kann. RGM stellt einen Satz Eigenschaften bereit, um die Erstellung der einzelnen Ressourcengruppentypen zu unterstützen. Weitere Einzelheiten zu diesen Eigenschaften finden Sie unter „Übertragen eines Datendienstes auf einen Cluster" auf Seite 34 und „Implementieren von Rückmeldemethoden" auf Seite 44.

„Ressourcengruppeneigenschaften" auf Seite 272 enthält eine Liste der Ressourcengruppeneigenschaften.

Ressourcengruppen-Manager

Ressourcengruppen-Manager (RGM) wird als Dämon, `rgmd`, implementiert, der auf jedem Mitgliedsknoten des Clusters läuft. Alle `rgmd`-Prozesse kommunizieren miteinander und arbeiten als eine Cluster-weite Funktion zusammen.

RGM unterstützt folgende Funktionen:

- RGM versucht bei jedem Knotenstart oder -absturz, alle verwalteten Ressourcen verfügbar zu halten, indem sie automatisch auf den entsprechenden Mastern online gebracht werden.
- Wenn eine bestimmte Ressource fehlschlägt, kann ihr Überwachungsprogramm anfordern, dass die Ressourcengruppe auf demselben Master neu gestartet wird oder dass sie zu einem neuen Master wechselt.
- Der Cluster-Administrator kann einen Verwaltungsbefehl ausgeben, um eine der folgenden Aktionen anzufordern:

- Ändern des Masters für eine Ressourcengruppe,
- Aktivieren oder Deaktivieren einer bestimmten Ressource innerhalb einer Ressourcengruppe,
- Erstellen, Löschen oder Ändern einer Ressource, einer Ressourcengruppe oder eines Ressourcentyps.

Wenn RGM Konfigurationsänderungen aktiviert, koordiniert das Programm seine Aktionen auf allen Mitgliedsknoten des Clusters. Diese Aktivität wird als Rekonfiguration bezeichnet. Um eine Zustandsänderung bei einer einzelnen Ressource vorzunehmen, ruft RGM eine für den Ressourcentyp spezifische Rückmeldemethode auf.

Rückmeldemethoden

Das Sun Cluster Framework verwendet einen Rückmeldemechanimus für die Kommunikation zwischen einem Datendienst und RGM. Das Framework definiert eine Reihe von Rückmeldemethoden, einschließlich deren Argumente und Rückgabewerte sowie der Umstände, unter denen RGM jede Methode aufruft.

Ein Datendienst wird erstellt, indem der Entwickler eine Reihe von einzelnen Rückmeldemethoden codiert und jede Methode als ein von RGM aufrufbares Steuerprogramm implementiert. Das bedeutet, dass der Datendienst nicht aus einer einzigen ausführbaren Datei besteht, sondern aus einer Reihe ausführbarer Skripts (`ksh`) oder Binärdateien (C), die jeweils direkt von RGM aufgerufen werden können.

Rückmeldemethoden werden bei RGM über die Ressourcentyp-Registrierungsdatei (RTR-Datei) registriert. In der RTR-Datei wird das Programm für jede Methode identifiziert, die Sie für den Datendienst implementiert haben. Wenn ein Systemadministrator den Datendienst auf einem Cluster registriert, liest RGM die RTR-Datei, die neben anderen Informationen die Identität der Rückmeldeprogramme enthält.

Die einzigen erforderlichen Rückmeldemethoden für einen Ressourcentyp sind eine Start-Methode (`Start` oder `Prenet_start`) und eine Stopp-Methode (`Stop` oder `Postnet_stop`).

Die Rückmeldemethoden lassen sich in folgende Kategorien zusammenfassen:

- Steuerungs- und Initialisierungsmethoden
  - `Start` und `Stop` starten und stoppen Ressourcen in einer Gruppe, die online bzw. offline gebracht wird.
  - `Init`, `Fini` und `Boot` führen Initialisierungs- und Beendigungscode für Ressourcen aus.

- Verwaltungsunterstützungsmethoden
  - `Validate` überprüft von einer Verwaltungsaktion eingestellte Eigenschaften.
  - `Update` aktualisiert die Eigenschaftseinstellungen einer Online-Ressource.
- Netzbezogene Methoden
  - `Prenet_start` und `Postnet_stop` führen spezielle Aktionen zum Hoch- bzw. Herunterfahren aus, bevor Netzwerkadressen in derselben Ressourcengruppe als aktiv bzw. inaktiv konfiguriert werden.
- Monitor-Steuerungsmethoden
  - `Monitor_start` und `Monitor_stop` starten bzw. stoppen den Monitor für eine Ressource.
  - `Monitor_check` beurteilt die Zuverlässigkeit eines Knotens, bevor eine Ressourcengruppe auf den Knoten verschoben wird.

Weitere Informationen zu den Rückmeldemethoden finden Sie in Kapitel 4 und in der Online-Dokumentation unter `rt_callbacks`(1HA). Rückmeldemethoden in Beispieldatendiensten finden Sie in Kapitel 5 und Kapitel 8.

Programmierschnittstellen

Für das Schreiben von Datendienstcode stellt die Ressourcenverwaltungsarchitektur Folgendes bereit. eine API auf niedriger Ebene bzw. Basis-API, eine Bibliothek auf höherer Ebene, die auf der Basis-API aufbaut, sowie das Tool SunPlex Agent Builder. Letzteres generiert einen Datendienst anhand einiger grundlegender Benutzereingaben automatisch.

RMAPI

Die RMAPI (Resource Management API, Ressourcenverwaltungs-API) stellt eine Reihe von Routinen auf niedriger Ebene bereit, mit denen ein Datendienst auf Informationen zu Ressourcen, Ressourcentypen und Ressourcengruppen im System zugreifen, einen lokalen Neustart oder Failover anfordern und den Ressourcenstatus einstellen kann. Der Zugriff auf diese Funktionen erfolgt über die `libscha.so`-Bibliothek. Die RMAPI stellt diese Rückmeldemethoden sowohl in Form von Shell-Befehlen als auch in Form von C-Funktionen bereit. Weitere Informationen zu den RMAPI-Routinen finden Sie unter `scha_calls`(3HA) und in Kapitel 4. Beispiele für die Verwendung der Routinen in Rückmeldemethoden für Beispieldatendienste finden Sie in Kapitel 5.

DSDL

Auf der RMAPI setzt die DSDL auf, die ein integriertes Framework auf höherer Ebene bereitstellt, jedoch das zugrunde liegende Methoden-Rückmeldemodell von RGM beibehält. Die DSDL stellt mehrere Funktionen für die Datendienstentwicklung zusammen, zu denen u. a. folgende gehören:

- `libscha.so` — die Ressourcenverwaltungs-APIs auf niedriger Ebene
- PMF — die Prozessverwaltungsfunktion, die das Überwachen von Prozessen und untergeordneten Prozessen sowie bei Versagen deren Neustart ermöglicht (siehe `pmfadm`(1M) und `rpc.pmfd`(1M)).
- `hatimerun` — eine Funktion für das Ausführen von Programmen mit Zeitüberschreitungen (siehe `hatimerun`(1M)).

Für die meisten Anwendungen bietet die DSDL den größten Teil der Funktionen, die für das Erstellen eines Datendienstes erforderlich sind. Beachten Sie jedoch, dass die DSDL die API auf niedriger Ebene nicht ersetzt, sondern einkapselt und erweitert. Viele DSDL-Funktionen rufen die `libscha.so`-Funktionen auf. Sie können `libscha.so`-Funktionen auch direkt aufrufen, während Sie die DSDL zum Codieren eines Großteils des Datendienstes verwenden. Die `libdsdev.so`-Bibliothek enthält die DSDL-Funktionen.

Weitere Informationen zur DSDL finden Sie in Kapitel 6 und in der Online-Dokumentation unter `scha_calls`(3HA).

SunPlex Agent Builder

Agent Builder ist ein Tool, das die Erstellung eines Datendienstes automatisiert. Der Benutzer gibt grundlegende Informationen über die Zielanwendung und den zu erstellenden Datendienst ein.Agent Builder generiert einen Datendienst mit Quell- und ausführbarem Code (C- oder Korn-Shell), einer angepassten RTR-Datei und einem Solaris™-Paket.

Für die meisten Anwendungen können Sie Agent Builder zum Generieren eines vollständigen Datendienstes einsetzen. Anschließend sind nur noch kleinere manuelle Änderungen erforderlich. Anwendungen mit komplizierteren Anforderungen, wie zum Beispiel Validierungsprüfungen für zusätzliche Eigenschaften, stellen möglicherweise Ansprüche, denen Agent Builder nicht gerecht werden kann. Auch in diesen Fällen können Sie jedoch Agent Builder für das Generieren eines großen Teils des Codes einsetzen und den restlichen Code manuell erstellen. Zumindest kann Agent Builder für das Generieren des Solaris-Pakets verwendet werden.

Verwaltungsschnittstelle von Ressourcengruppen-Manager

Sun Cluster stellt sowohl eine grafische Benutzeroberfläche als auch eine Reihe von Befehlen für die Verwaltung eines Clusters bereit.

SunPlex-Manager

SunPlex-Manager ist ein webbasiertes Tool, mit dem Sie folgende Aufgaben ausführen können:

- Installieren eines Clusters,
- Verwalten eines Clusters,
- Erstellen und Konfigurieren von Ressourcen und Ressourcengruppen,
- Konfigurieren von Datendiensten mit der Sun Cluster-Software.

Anweisungen zum Installieren von SunPlex-Manager und zur Verwendung von SunPlex-Manager für die Installation der Cluster-Software finden Sie im *Sun Cluster Software Installation Guide for Solaris OS*. SunPlex-Manager stellt für die meisten einmaligen Verwaltungsaufgaben Online-Hilfe bereit.

Verwaltungsbefehle

Die Sun Cluster-Befehle für die Verwaltung von RGM-Objekten sind `scrgadm`(1M), `scswitch`(1M) und `scstat`(1M) - `g`.

Der `scrgadm`-Befehl ermöglicht das Anzeigen, Erstellen, Konfigurieren und Löschen der von RGM verwendeten Ressourcentypen, Ressourcengruppen und Ressourcenobjekte. Der Befehl ist Bestandteil der Verwaltungsschnittstelle für den Cluster und darf nicht in demselben Programmierkontext wie die im Rest dieses Kapitels beschriebene Anwendungsschnittstelle verwendet werden. `scrgadm` ist jedoch das Tool für den Aufbau der Cluster-Konfiguration, in der die API arbeitet. Ein Verständnis der Verwaltungsschnittstelle stellt den Kontext für das Verstehen der Anwendungsschnittstelle bereit. In der Online-Dokumentation `scrgadm`(1M) finden Sie Einzelheiten über die Verwaltungsaufgaben, die mit dem Befehl ausgeführt werden können.

Der `scswitch`-Befehl schaltet die Ressourcengruppen auf angegebenen Knoten zwischen online und offline um und aktiviert bzw. deaktiviert eine Ressource oder deren Überwachung. Einzelheiten zu den Verwaltungsaufgaben, die mit diesem Befehl ausgeführt werden können, finden Sie in der Online-Dokumentation unter `scswitch`(1M).

Der Befehl `scstat -g` zeigt den aktuellen dynamischen Zustand aller Ressourcengruppen und Ressourcen an.

KAPITEL 2

Entwickeln eines Datendienstes

Dieses Kapitel enthält detaillierte Informationen zum Entwickeln eines Datendienstes.

Dieses Kapitel behandelt die folgenden Themen:

Analysieren der Eignung einer Anwendung

Als erster Schritt beim Erstellen eines Datendienstes muss überprüft werden, ob die Zielanwendung alle Anforderungen für eine hohe Verfügbarkeit bzw. Skalierbarkeit erfüllt. Wenn die Anwendung nicht allen Anforderungen entspricht, können Sie möglicherweise deren Quellcode ändern, um die Anforderungen zu erfüllen.

Die folgende Liste fasst die Anforderungen für eine Anwendung, die hoch verfügbar oder skalierbar gemacht werden soll, zusammen. Weitere Einzelheiten und Hilfe beim Ändern des Anwendungsquellcodes finden Sie in Anhang B.

Hinweis – Ein Scalable-Dienst muss alle folgenden Bedingungen für hohe Verfügbarkeit sowie einige zusätzliche Kriterien erfüllen.

- Sowohl Anwendungen mit Netzwerkunterstützung (Client/Server-Modell) als auch Anwendungen ohne Netzwerkunterstützung (ohne Clients) können in der Sun Cluster-Umgebung hoch verfügbar oder skalierbar gemacht werden. Sun Cluster kann jedoch keine verbesserte Verfügbarkeit in Time-Sharing-Umgebungen bereitstellen, in denen Anwendungen auf einem Server mit Zugriff über `telnet` oder `rlogin` ausgeführt werden.
- Die Anwendung muss Abstürze tolerieren. Das bedeutet, dass sie bei Bedarf Plattendaten wiederherstellen muss, wenn sie nach einem unerwarteten Knotenausfall neu gestartet wird. Zudem muss die Wiederherstellungszeit nach einem Absturz begrenzt sein. Absturztoleranz ist eine Voraussetzung dafür, dass eine Anwendung hoch verfügbar gemacht wird, da die Fähigkeit zum Wiederherstellen der Platte und Neustarten der Anwendung die Datenintegrität sicherstellt. Es ist nicht erforderlich, dass der Datendienst Verbindungen wiederherstellen kann.
- Die Anwendung darf nicht von dem realen Hostnamen des Knotens, auf dem sie ausgeführt wird, abhängen. Weitere Informationen finden Sie unter „Hostnamen" auf Seite 337.
- Die Anwendung muss korrekt in Umgebungen laufen, in denen mehrere IP-Adressen als aktiv konfiguriert sind. Zum Beispiel sind dies Umgebungen mit Multihomed-Hosts, in denen sich der Knoten in mehr als einem öffentlichen Netzwerk befindet, oder Umgebungen mit Knoten, auf denen mehrere logische Schnittstellen auf einer Hardware-Schnittstelle als aktiv konfiguriert sind.
- Um hoch verfügbar zu sein, müssen sich die Anwendungsdaten in den Cluster-Dateisystemen befinden — siehe „Multihost-Daten" auf Seite 335.

  Wenn die Anwendung einen fest verdrahteten Pfadnamen als Datenspeicherort verwendet, können Sie diesen Pfad in eine symbolische Verknüpfung ändern, die zu einem Speicherort im Cluster-Dateisystem zeigt, ohne dabei den Anwendungsquellcode zu ändern. Weitere Informationen finden Sie unter „Verwenden von symbolischen Verknüpfungen für Multihost-Datenablage" auf Seite 336.
- Anwendungsbinärdateien und Bibliotheken können lokal auf jedem Knoten des Cluster-Dateisystems residieren. Der Vorteil hierbei besteht darin, dass eine einzige Installation ausreicht. Der Nachteil ist, dass laufende Aufrüstungen zu einem Problem werden, da die Binärdateien verwendet werden, solange die Anwendung unter Steuerung durch RGM ausgeführt wird.
- Der Client sollte in der Lage sein, eine Abfrage automatisch zu wiederholen, wenn das Zeitlimit für den ersten Versuch abgelaufen ist. Wenn die Anwendung und das Protokoll bereits den Fall eines einzelnen Serverabsturzes und -neustarts

unterstützen, sind sie auch für das Failover bzw. Switchover der darin enthaltenen Ressourcengruppe geeignet. Weitere Informationen finden Sie unter „Client-Wiederholversuch" auf Seite 339.

- Die Anwendung darf keine UNIX®-Domain-Sockets oder benannte Datenaustauschkanäle im Cluster-Dateisystem haben.

Zudem müssen Scalable-Dienste die folgenden Anforderungen erfüllen:

- Die Anwendung muss in der Lage sein, mehrere Instanzen auszuführen, die alle mit denselben Anwendungsdaten im Cluster-Dateisystem arbeiten.
- Die Anwendung muss Datenkonsistenz für den simultanen Zugriff von mehreren Knoten aus bereitstellen.
- Die Anwendung muss eine ausreichende Sperre mit einem global sichtbaren Mechanismus implementieren, wie zum Beispiel das Cluster-Dateisystem.

Bei einem Scalable-Dienst legen die Anwendungseigenschaften auch das Lastausgleichsverfahren fest. Zum Beispiel funktioniert das Lastausgleichsverfahren `LB_WEIGHTED`, das jeder Instanz die Antwort auf Client-Anforderungen ermöglicht, nicht für eine Anwendung, die einen Cache-Speicher auf dem Server für Client-Verbindungen verwendet. In diesem Fall muss ein Lastausgleichsverfahren angegeben werden, das den Datenverkehr eines bestimmten Clients auf eine Instanz der Anwendung beschränkt. Die Lastausgleichsverfahren `LB_STICKY` und `LB_STICKY_WILD` senden wiederholt alle Anforderungen von einem Client an dieselbe Anwendungsinstanz — wo sie einen Cache-Speicher verwenden können. Beachten Sie, dass RGM mehrere Client-Anforderungen von unterschiedlichen Clients unter den Instanzen des Dienstes verteilt. Weitere Informationen zum Einstellen der Lastausgleichsverfahren für Scalable-Datendienste finden Sie unter „Implementieren einer Failover-Ressource" auf Seite 53.

Festlegen der zu verwendenden Schnittstelle

Das Sun Cluster-Entwickler-Unterstützungspaket (`SUNWscdev`) stellt zwei Schnittstellen für das Codieren von Datendienstmethoden bereit:

- Die Ressourcenverwaltungs-API (RMAPI), ein Satz Routinen auf niedriger Ebene (in der `libscha.so`-Bibliothek),
- Die DSDL (Data Services Development Library), ein Satz Funktionen auf höherer Ebene in der `libdsdev.so`-Bibliothek, welche die Funktionen der RMAPI einkapseln und zusätzliche Funktionen bereitstellen.

Im Sun Cluster-Entwickler-Unterstützungspaket ist auch SunPlex Agent Builder enthalten, ein Tool, das die Erstellung eines Datendienstes automatisiert.

Folgende Vorgehensweise wird beim Entwickeln eines Datendienstes empfohlen:

1. Entscheiden Sie, ob in C oder der Korn-Shell codiert werden soll. Wenn Sie sich für die Verwendung der Korn-Shell entscheiden, können Sie die DSDL nicht verwenden, da diese nur eine C-Schnittstelle enthält.
2. Führen Sie Agent Builder aus, geben Sie die erforderlichen Angaben ein, und generieren Sie einen Datendienst mit Quell- und ausführbarem Code, einer RTR-Datei und einem Paket.
3. Wenn der generierte Datendienst angepasst werden muss, können Sie den generierten Quelldateien DSDL-Code hinzufügen. Agent Builder gibt mithilfe von Kommentaren bestimmte Stellen in den Quelldateien an, an denen eigener Code eingefügt werden kann.
4. Wenn der Code weiter angepasst werden muss, um die Zielanwendung zu unterstützen, können Sie dem vorhandenen Quellcode RMAPI-Funktionen hinzufügen.

In der Praxis gibt es viele unterschiedliche Möglichkeiten zum Erstellen eines Datendienstes. Statt an bestimmten Stellen des von Agent Builder generierten Codes eigenen Code einzufügen, können Sie zum Beispiel eine der generierten Methoden oder das generierte Überwachungsprogramm komplett durch ein Programm ersetzen, das Sie mit DSDL- oder RMAPI-Funktionen völlig neu schreiben. Unabhängig von der Vorgehensweise ist es jedoch fast immer sinnvoll, mit Agent Builder zu beginnen, und zwar aus folgenden Gründen:

- Der von Agent Builder generierte Code ist zwar seiner Art nach generisch, wurde aber in zahlreichen Datendiensten getestet.
- Agent Builder generiert eine RTR-Datei, ein Makefile, ein Paket für die Ressource und weitere Unterstützungsdateien für den Datendienst. Auch wenn Sie keinen Datendienstcode verwenden, kann Ihnen der Einsatz dieser anderen Dateien viel Arbeit ersparen.
- Sie können den generierten Code ändern.

Hinweis – Im Unterschied zur RMAPI, die mehrere C-Funktionen und eine Reihe von Befehlen für die Verwendung in Skripts bereitstellt, verfügt die DSDL nur über eine C-Funktionsschnittstelle. Wenn Sie also in Agent Builder Korn-Shell (`ksh`)-Ausgabe festlegen, ruft der generierte Quellcode die RMAPI auf, weil keine DSDL-`ksh`-Befehle vorhanden sind.

Konfigurieren der Entwicklungsumgebung für das Schreiben eines Datendienstes

Bevor Sie mit der Datendienstentwicklung beginnen, müssen Sie das Sun Cluster-Entwicklungspaket (`SUNWscdev`) installieren, um Zugriff auf die Sun Cluster-Header- und Bibliotheksdateien zu haben. Obwohl dieses Paket bereits auf allen Cluster-Knoten installiert ist, erfolgt die Entwicklung in der Regel auf einem eigenen, nicht auf einem Cluster-Knoten vorhandenen Entwicklungsrechner. In diesem typischen Fall müssen Sie den Befehl `pkgadd` verwenden, um das `SUNWscdev`-Paket auf Ihrem Entwicklungsrechner zu installieren.

Beim Kompilieren und Verknüpfen des Codes müssen Sie besondere Optionen für die Identifizierung der Header- und Bibliotheksdateien einstellen.

Hinweis – Im Solaris-Betriebssystem und in den Sun Cluster-Produkten kann im Kompatibilitätsmodus kompilierter C++-Code nicht mit im Standardmodus kompiliertem C++-Code gemischt werden. Wenn Sie beabsichtigen, einen C++-basierten Datendienst für die Verwendung in Sun Cluster zu erstellen, müssen Sie diesen Datendienst daher wie folgt kompilieren:

- Verwenden Sie den Kompatibilitätsmodus für Sun Cluster 3.0 und frühere Versionen.
- Verwenden Sie ab Sun Cluster 3.1 den Standardmodus.

Nach Beenden der Entwicklung auf einem Nicht-Cluster-Knoten können Sie den fertigen Datendienst auf einen Cluster übertragen und dort ausführen und testen.

Hinweis – Verwenden Sie unbedingt die Softwaregruppe "Developer" oder "Entire Distribution" von Solaris 5.8 oder einer höheren Version des Solaris-Betriebssystems.

Verwenden Sie die Verfahren in diesem Abschnitt zu folgenden Zwecken:

- Installieren des Sun Cluster-Entwicklungspakets (`SUNWscdev`) und Festlegen der geeigneten Compiler- und Verknüpfer-Optionen.
- Übertragen des Datendienstes auf einen Cluster.

▼ Konfigurieren der Entwicklungsumgebung

Dieses Verfahren beschreibt die Installation des `SUNWscdev`-Pakets und das Einstellen der Compiler- und Verknüpfer-Optionen für die Datendienstentwicklung.

1. **Melden Sie sich als Superbenutzer oder in einer äquivalenten Rolle an, und ändern Sie das Verzeichnis in das gewünschte CD-ROM-Verzeichnis.**

   # **cd** *CD-ROM_Verzeichnis*

2. **Installieren Sie das SUNWscdev-Paket im aktuellen Verzeichnis.**

   # **pkgadd -d . SUNWscdev**

3. **Geben Sie im `Makefile` die Compiler- und Verknüpfer-Optionen an, mit denen die Include- und Bibliotheksdateien für den Datendienstcode identifiziert werden.**

   Geben Sie die Option `-I` zur Identifikation der Sun Cluster-Header-Dateien an, die Option `-L` zum Angeben des Kompilierungszeit-Bibliotheksuchpfads im Entwicklungssystem, und die Option `-R` zum Angeben des Bibliotheksuchpfads zum Laufzeitverknüpfer auf dem Cluster.

   ```
   # Makefile für Beispieldatendienst

   -I /usr/cluster/include

   -L /usr/cluster/lib

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

Übertragen eines Datendienstes auf einen Cluster

Nach Beendigung der Entwicklung eines Datendienstes auf dem Entwicklungsrechner müssen Sie den Datendienst auf einen Cluster übertragen, um ihn zu testen. Um die Fehlermöglichkeiten einzuschränken, erstellen Sie am besten ein Paket aus dem Datendienstcode und der RTR-Datei, und installieren Sie das Paket auf allen Knoten des Clusters.

Hinweis – Unabhängig davon, ob Sie den Befehl `pkgadd` oder eine andere Methode zum Installieren des Datendienstes verwenden, müssen Sie den Datendienst auf allen Cluster-Knoten ablegen. Agent Builder erstellt automatisch ein Paket aus der RTR-Datei und dem Datendienstcode.

Einstellen der Ressourcen- und Ressourcentypeigenschaften

Sun Cluster stellt einen Satz Ressourcentypeigenschaften und Ressourceneigenschaften bereit, die zum Definieren der statischen Konfiguration eines Datendienstes verwendet werden. Ressourcentypeigenschaften geben den Typ der Ressource, deren Version, die API-Version, usw., sowie die Pfade zu jeder der Rückmeldemethoden an. „Ressourcentypeigenschaften" auf Seite 253 listet alle Ressourcentypeigenschaften auf.

Ressourceneigenschaften wie `Failover_mode`, `Thorough_probe_interval` und Methodenzeitüberschreitungen definieren ebenfalls die statische Konfiguration der Ressource. Dynamische Ressourceneigenschaften wie `Resource_state` und `Status` geben den Zustand einer verwalteten Ressource wieder. „Ressourceneigenschaften" auf Seite 260 beschreibt die Ressourceneigenschaften.

Der Ressourcentyp und die Ressourceneigenschaften werden in der Ressourcentyp-Registrierungsdatei (RTR-Datei) deklariert. Diese Datei ist eine grundlegende Komponente eines Datendienstes. Die RTR-Datei definiert die anfängliche Konfiguration des Datendienstes zu dem Zeitpunkt, zu dem der Cluster-Administrator den Datendienst bei Sun Cluster registriert.

Sie sollten Agent Builder zum Generieren der RTR-Datei für Ihren Datendienst verwenden, da Agent Builder den Satz Eigenschaften deklariert, der für jeden Datendienst nützlich und erforderlich ist. Es müssen zum Beispiel bestimmte Eigenschaften wie `Resource_type` in der RTR-Datei deklariert werden; andernfalls schlägt die Registrierung des Datendienstes fehl. Andere Eigenschaften sind zwar nicht erforderlich, stehen aber dem Systemadministrator nur dann zur Verfügung, wenn Sie sie in der RTR-Datei deklarieren. Einige Eigenschaften sind immer verfügbar, unabhängig davon, ob Sie sie deklarieren oder nicht, da sie von RGM definiert und mit einem Standardwert versehen werden. Um diese komplexen Fragen zu umgehen, können Sie einfach Agent Builder einsetzen, um die Generierung einer geeigneten RTR-Datei sicherzustellen. Später können Sie die RTR-Datei bearbeiten und bei Bedarf bestimmte Werte ändern.

Im letzten Teil dieses Abschnitts werden Sie durch eine von Agent Builder erstellte Beispiel-RTR-Datei geführt.

Deklarieren von Ressourcentypeigenschaften

Der Cluster-Administrator kann die von Ihnen in der RTR-Datei deklarierten Ressourcentypeigenschaften nicht konfigurieren. Sie werden Bestandteil der permanenten Ressourcentypkonfiguration.

Hinweis – Eine Ressourcentypeigenschaft, `Installed_nodes`, kann vom Systemadministrator konfiguriert werden. Sie kann sogar nur von einem Systemadministrator konfiguriert und nicht in der RTR-Datei deklariert werden.

Die Syntax für Ressourcentypdeklarationen lautet:

Eigenschaftsname = *Wert*;

Hinweis – RGM unterscheidet bei Eigenschaftsnamen nicht zwischen Groß- und Kleinschreibung. Die Konvention für Eigenschaften in von Sun gelieferten RTR-Dateien, mit Ausnahme der Methodennamen, ist Großschreibung des ersten Buchstabens und Kleinschreibung der restlichen Buchstaben des Namens. Methodennamen werden — ebenso wie Eigenschaftsattribute — ganz in Großbuchstaben geschrieben.

Es folgen die Ressourcentypdeklarationen in der RTR-Datei für einen Beispieldatendienst (`smpl`):

```
# Sun Cluster Data Services Builder template version 1.0
# Registration information and resources for smpl
#
#NOTE: Keywords are case insensitive, i.e., you can use
#any capitalization style you prefer.
#
Resource_type = "smpl";
Vendor_id = SUNW;
RT_description = "Sample Service on 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;
```

Tipp – Als ersten Eintrag in der RTR-Datei müssen Sie die `Resource_type`-Eigenschaft deklarieren. Andernfalls schlägt die Registrierung des Ressourcentyps fehl.

Der erste Satz Ressourcentypdeklarationen liefert grundlegende Informationen über den Ressourcentyp, wie folgt:

`Resource_type` und `Vendor_id`

Geben dem Ressourcentyp einen Namen. Sie können den Ressourcentypnamen entweder durch die `Resource_type`-Eigenschaft alleine angeben (`smpl`) oder `Vendor_id` als Präfix mit "." zur Abtrennung vom Ressourcentyp verwenden (`SUNW.smpl`), wie im Beispiel gezeigt. Wenn Sie `Vendor_id` verwenden, nehmen Sie das Börsensymbol für das Unternehmen, das den Ressourcentyp definiert. Der Ressourcentypname muss im Cluster einmalig sein.

Hinweis – Als Konvention wird der Ressourcentypname (*Resource_typeVendor_id*) als Paketname verwendet. Paketnamen sind auf neun Zeichen beschränkt, so dass Sie die Gesamtzahl der Zeichen in diesen beiden Eigenschaften auf neun oder weniger Zeichen beschränken sollten, auch wenn RGM diese Beschränkung nicht erzwingt. Agent Builder generiert jedoch den Paketnamen explizit anhand des Ressourcentypnamens, so dass er die Beschränkung auf neun Zeichen erzwingt.

`RT_version`

Identifiziert die Version des Beispieldatendienstes.

`API_version`

Identifiziert die API-Version. So gibt `API_version = 2` zum Beispiel an, dass der Datendienst unter Sun Cluster, Version 3.0, ausgeführt wird.

`Failover = TRUE`

Gibt an, dass der Datendienst nicht in einer Ressourcengruppe laufen kann, die auf mehreren Knoten gleichzeitig online sein kann. Damit wird also ein Failover-Datendienst

	angegeben. Weitere Informationen finden Sie unter „Übertragen eines Datendienstes auf einen Cluster" auf Seite 34.
`Start`, `Stop`, `Validate`, usw.	Geben die Pfade zu den entsprechenden Rückmeldemethodenprogrammen an, die von RGM aufgerufen werden. Diese Pfade sind relativ zu dem Verzeichnis, das durch `RT_basedir` angegeben wird.

Die restlichen Ressourcentypdeklarationen geben folgende Konfigurationsinformationen an:

`Init_nodes = RG_PRIMARIES`	Gibt an, dass RGM die Methoden `Init`, `Boot`, `Fini` und `Validate` nur auf Knoten aufruft, die als Master des Datendienstes eingesetzt werden können. Die von `RG_PRIMARIES` angegebenen Knoten sind eine Untermenge aller Knoten, auf denen der Datendienst installiert ist. Setzen Sie den Wert auf `RT_INSTALLED_NODES`, um anzugeben, dass RGM diese Methoden auf allen Knoten aufruft, auf denen der Datendienst installiert ist.
`RT_basedir`	Zeigt auf `/opt/SUNWsample/bin` als Verzeichnispfad zu vollständigen relativen Pfaden, wie den Rückmeldemethodepfaden.
`Start`, `Stop`, `Validate`, usw.	Geben die Pfade zu den entsprechenden Rückmeldemethodenprogrammen an, die von RGM aufgerufen werden. Diese Pfade sind relativ zu dem Verzeichnis, das durch `RT_basedir` angegeben wird.

Deklarieren von Ressourceneigenschaften

Genau wie die Ressourcentypeigenschaften werden auch die Ressourceneigenschaften in der RTR-Datei deklariert. Als Konvention folgen die Ressourceneigenschaftsdeklarationen in der RTR-Datei auf die Ressourcentypdeklarationen. Die Syntax für Ressourcendeklarationen ist ein Satz von Attributwertepaaren, die zwischen geschweiften Klammern stehen:

```
{
    Attribut = Wert;
    Attribut = Wert;
                .
                .
                .
    Attribut = Wert;
}
```

Für von Sun Cluster bereitgestellte Ressourceneigenschaften (so genannte *systemdefinierte* Eigenschaften) können Sie bestimmte Attribute in der RTR-Datei ändern. So stellt Sun Cluster zum Beispiel Methoden-Zeitüberschreitungseigenschaften für jede Rückmeldemethode bereit und gibt Standardwerte an. In der RTR-Datei können Sie andere Standardwerte festlegen.

Sie können in der RTR-Datei auch neue Ressourceneigenschaften definieren, so genannte *Erweiterungseigenschaften*, indem Sie einen Satz der von Sun Cluster bereitgestellten Eigenschaftsattribute verwenden. „Ressourceneigenschaftsattribute" auf Seite 278 listet die Attribute für das Ändern und Definieren von Ressourceneigenschaften auf. Erweiterungseigenschaftsdeklarationen folgen in der RTR-Datei auf die Deklarationen der systemdefinierten Eigenschaften.

Der erste Satz systemdefinierter Ressourceneigenschaften gibt die Zeitüberschreitungswerte für die Rückmeldemethoden an:

```
...

# 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;
}
```

Der Name der Eigenschaft (`PROPERTY` = *Wert*) muss das erste Attribut für jede Ressourceneigenschaftsdeklaration sein. Sie können Ressourceneigenschaften innerhalb der von den Eigenschaftsattributen definierten Grenzwerten in der RTR-Datei konfigurieren. So beträgt zum Beispiel der Standardwert für jede Methoden-Zeitüberschreitung im Beispiel 300 Sekunden. Der Verwalter kann diesen Wert ändern. Der im `MIN`-Attribut angegebene zulässige Mindestwert ist jedoch 60 Sekunden. „Ressourceneigenschaftsattribute" auf Seite 278 enthält eine Liste der Ressourceneigenschaftsattribute.

Der nächste Satz Ressourceneigenschaften definiert Eigenschaften, die im Datendienst bestimmten Zwecken dienen.

```
{
        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 = AT_CREATION;
        DEFAULT = ;
}
```

Diese Ressourceneigenschaftsdeklarationen fügen das `TUNABLE`-Attribut hinzu, das die Zeitpunkte einschränkt, zu denen der Systemadministrator die Werte ändern kann. `AT_CREATION` bedeutet, dass der Administrator den Wert nur bei Erstellung der Ressource angeben, ihn aber später nicht mehr ändern kann.

Für die meisten dieser Eigenschaften können Sie die von Agent Builder generierten Standardwerte akzeptieren, wenn kein besonderer Grund für eine Änderung vorliegt. Informationen zu diesen Eigenschaften finden Sie im Folgenden. Weiterführende Informationen sind unter „Ressourceneigenschaften" auf Seite 260 und der Online-Dokumentation unter `r_properties`(5) enthalten.

`Failover_mode`
: Gibt an, ob RGM die Ressourcengruppe verschieben oder den Knoten abbrechen soll, wenn eine `Start`- oder `Stop`-Methode fehlschlägt.

`Thorough_probe_interval`, `Retry_count`, `Retry_interval`
: Wird im Fehler-Monitor verwendet. `Tunable` entspricht `ANYTIME`, so dass ein Systemverwalter sie anpassen kann, wenn der Fehler-Monitor nicht optimal funktioniert.

`Network_resources_used`
: Eine Liste logischer Hostnamen- bzw. gemeinsam genutzter Adressressourcen, die vom Datendienst verwendet werden. Agent Builder deklariert diese Eigenschaft, so dass ein Systemverwalter beim Konfigurieren des Datendienstes eine Liste der Ressourcen (falls vorhanden) angeben kann.

`Scalable`
Wird auf `FALSE` eingestellt, um anzugeben, dass diese Ressource nicht die Cluster-Netzwerkfunktion (gemeinsam genutzte Adresse) verwendet. Diese Einstellung stimmt mit derjenigen der `Failover`-Ressourcentypeigenschaft überein, die auf `TRUE` eingestellt ist, um einen Failover-Dienst anzugeben. Weitere Informationen zur Verwendung dieser Eigenschaft finden Sie unter „Übertragen eines Datendienstes auf einen Cluster" auf Seite 34 und „Implementieren von Rückmeldemethoden" auf Seite 44.

`Load_balancing_policy`, `Load_balancing_weights`
Erklärt diese Eigenschaften automatisch. Sie haben jedoch in einem Failover-Ressourcentyp keinen Verwendungszweck.

`Port_list`
Eine Liste mit Portnummern, die der Server abhört. Agent Builder deklariert diese Eigenschaft, so dass ein Systemverwalter beim Konfigurieren des Datendienstes eine Port-Liste angeben kann.

Deklarieren von Erweiterungseigenschaften

Am Ende der RTR-Beispieldatei befinden sich Erweiterungseigenschaften, wie in der folgenden Auflistung gezeigt.

```
# 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 erstellt einige Erweiterungseigenschaften, die für die meisten Datendienste nützlich sind:

`Confdir_list`
: Gibt den Pfad zum Anwendungskonfigurationsverzeichnis an. Diese Informationen sind für viele Anwendungen nützlich. Der Systemverwalter kann beim Konfigurieren des Datendienstes den Pfad zu diesem Verzeichnis bereitstellen.

`Monitor_retry_count`, `Monitor_retry_interval`, `Probe_timeout`
: Steuert die Neustarts des Fehler-Monitors selbst, nicht diejenigen des Server-Dämons.

`Child_mon_level`
: Stellt die Ebene der von PMF ausgeführten Überwachung ein. Weitere Informationen finden Sie unter `pmfadm`(1M).

Sie können in dem Bereich, der durch die Kommentare `Vom Benutzer hinzugefügter Code` eingegrenzt ist, weitere Erweiterungseigenschaften erstellen.

Implementieren von Rückmeldemethoden

Dieser Abschnitt enthält Informationen, die sich auf das Implementieren von Rückmeldemethoden allgemein beziehen.

Zugreifen auf Informationen über Ressourcen- und Ressourcengruppeneigenschaften

Im Allgemeinen benötigen Rückmeldemethoden Zugriff auf die Eigenschaften der Ressource. Die RMAPI stellt sowohl Shell-Befehle als auch C-Funktionen bereit, die Sie in Rückmeldemethoden für den Zugriff auf systemdefinierte und Erweiterungseigenschaften von Ressourcen verwenden können. Weitere Informationen finden Sie in der Online-Dokumentation unter `scha_resource_get`(1HA) und `scha_resource_get`(3HA)..

Die DSDL stellt zum Zugriff auf systemdefinierte Eigenschaften einen Satz C-Funktionen bereit (eine pro Eigenschaft) sowie eine Funktion zum Zugriff auf Erweiterungseigenschaften. Weitere Informationen finden Sie in der Online-Dokumentation unter `scds_property_functions`(3HA) und `scds_get_ext_property`(3HA).

Sie können den Eigenschaftsmechanismus nicht für das Speichern von dynamischen Zustandsinformationen für einen Datendienst verwenden, da keine API-Funktionen für das Einstellen von Ressourceneigenschaften vorhanden sind, mit Ausnahme derjenigen zum Einstellen von `Status` und `Status_msg`. Stattdessen müssen Sie dynamische Zustandsinformationen in globalen Dateien speichern.

Hinweis – Der Cluster-Verwalter kann bestimmte Ressourceneigenschaften mithilfe des Befehls `scrgadm` oder über einen vorhandenen grafischen Verwaltungsbefehl bzw. eine vorhandene grafische Verwaltungs-Benutzeroberfläche einstellen. `scrgadm` darf jedoch nicht von einer Rückmeldemethode aus aufgerufen werden, da `scrgadm` während einer Cluster-Rekonfiguration fehlschlägt, also wenn RGM die Methode aufruft.

Idempotenz für Methoden

Im Allgemeinen ruft RGM keine Methode mehr als einmal nacheinander für die gleiche Ressource mit den gleichen Argumenten auf. Wenn jedoch eine `Start`-Methode fehlschlägt, könnte RGM eine `Stop`-Methode für eine Ressource

aufrufen, obwohl diese Ressource gar nicht gestartet wurde. Ebenso könnte ein Ressourcen-Dämon von selbst ausfallen und RGM dennoch die `Stop`-Methode aufrufen. Die gleichen Möglichkeiten gelten für die Methoden `Monitor_start` und `Monitor_stop`.

Aus diesen Gründen müssen Sie Idempotenz in Ihre `Stop`- und `Monitor_stop`-Methoden einbauen. Wiederholte Aufrufe von `Stop` oder `Monitor_stop` für die gleiche Ressource mit den gleichen Parametern erzielen dann das gleiche Ergebnis wie ein einziger Aufruf.

Eine Auswirkung der Idempotenz ist, dass `Stop` und `Monitor_stop` 0 (Erfolg) zurückgeben müssen, auch wenn die Ressource bzw. der Monitor bereits gestoppt sind und keine Aufgabe ausgeführt wird.

Hinweis – Die Methoden `Init`, `Fini`, `Boot` und `Update` müssen ebenfalls idempotent sein. Eine `Start`-Methode muss nicht idempotent sein.

Generischer Datendienst

Ein generischer Datendienst (Generic Data Service, GDS) ist ein Mechanismus, mit dem einfachen Anwendungen hohe Verfügbarkeit bzw. Skalierbarkeit verliehen wird, indem sie in das Framework des Ressourcengruppen-Managers von Sun Cluster eingefügt werden. Dieser Mechanismus erfordert keine Agentencodierung, wie dies sonst beim Einrichten von hoher Verfügbarkeit bzw. Skalierbarkeit üblich ist.

Das GDS-Modell beruht auf einem vorkompilierten Ressourcentyp, SUNW.gds, der die Zusammenarbeit mit dem RGM-Framework übernimmt.

Weitere Informationen hierzu finden Sie in Kapitel 10.

Steuern einer Anwendung

Mithilfe von Rückmeldemethoden kann RGM die Steuerung der zugrunde liegenden Ressource (Anwendung) übernehmen, sobald dem Cluster Knoten hinzugefügt bzw. daraus entfernt werden.

Starten und Stoppen einer Ressource

Für die Implementierung eines Ressourcentyps sind mindestens eine `Start`-Methode und eine `Stop`-Methode erforderlich. RGM ruft die Methodenprogramme eines Ressourcentyps zu geeigneten Zeiten und auf den entsprechenden Knoten auf, um Ressourcengruppen offline bzw. online zu bringen. Nach dem Absturz eines Cluster-Knotens verschiebt RGM zum Beispiel alle von diesem Knoten unterstützten Ressourcengruppen auf einen neuen Knoten. Es muss eine `Start`-Methode implementiert werden, damit RGM die Möglichkeit hat, jede Ressource auf dem noch laufenden Host-Knoten neu zu starten.

Eine `Start`-Methode darf nichts zurückgeben, bis die Ressource auf dem lokalen Knoten gestartet wurde und verfügbar ist. Vergewissern Sie sich, dass für Ressourcentypen, deren Initialisierung lange dauert, ausreichend lange Zeitüberschreitungswerte in den `Start`-Methoden eingestellt sind (stellen Sie die Standard- und Mindestwerte für die `Start_timeout`-Eigenschaft in der Ressourcentyp-Registrierungsdatei ein).

Eine `Stop`-Methode muss für Situationen implementiert werden, in denen RGM eine Ressourcengruppe offline nimmt. Angenommen, eine Ressource wird auf Knoten1 offline genommen und auf Knoten2 wieder online gebracht. Während die Ressourcengruppe offline genommen wird, ruft RGM die `Stop`-Methode für die Ressourcen in der Gruppe auf, um alle Aktivitäten auf Knoten1 zu stoppen. Nach Beenden der `Stop`-Methoden für alle Ressourcen auf Knoten1 bringt RGM die Ressourcengruppe auf Knoten2 wieder online.

Eine `Stop`-Methode darf nichts zurückgeben, bis die Ressource ihre Aktivität auf dem lokalen Knoten vollständig eingestellt hat und ganz heruntergefahren wurde. Die sicherste Implementierung einer `Stop`-Methode beendet alle Prozesse auf dem lokalen Knoten, die mit der Ressource in Beziehung stehen. Für Ressourcentypen, deren Herunterfahren lange dauert, müssen ausreichend lange Zeitüberschreitungswerte in den entsprechenden `Stop`-Methoden eingestellt werden. Stellen Sie die `Stop_timeout`-Eigenschaft in der Ressourcentyp-Registrierungsdatei ein.

Ein Fehlschlagen bzw. die Zeitüberschreitung einer `Stop`-Methode führt dazu, dass die Ressourcengruppe in einen Fehlerzustand gerät, der einen Bedienereingriff erforderlich macht. Um diesen Zustand zu vermeiden, müssen die Implementierungen der `Stop`- und `Monitor_stop`-Methoden eine Wiederherstellung unter allen möglichen Fehlerbedingungen versuchen. Idealerweise sollten diese Methoden mit dem Fehlerstatus 0 (Erfolg) beendet werden, nachdem jegliche Aktivität der Ressource und deren Monitor auf dem lokalen Knoten erfolgreich gestoppt wurde.

Bestimmen der zu verwendenden `Start`- und `Stop`-Methoden

Dieser Abschnitt enthält einige Tipps dazu, wann die `Start`- und `Stop`-Methoden bzw. die `Prenet_start`- und `Postnet_stop`-Methoden verwendet werden sollen. Voraussetzung zur Entscheidung für die richtigen Methoden sind sehr gute Kenntnisse sowohl des Clients als auch des Client/Server-Netzwerkprotokolls des Datendienstes.

Für Dienste, die Netzwerkadressressourcen verwenden, muss das Starten und Stoppen möglicherweise in einer bestimmten Reihenfolge stattfinden, die in Bezug zur logischen Hostname-Adresskonfiguration steht. Die optionalen Rückmeldemethoden `Prenet_start` und `Postnet_stop` ermöglichen es einer Ressourcentypimplementierung, besondere Aktionen beim Herauf- bzw. Herunterfahren auszuführen, bevor und nachdem Netzwerkadressen in derselben Ressourcengruppe als aktiv bzw. inaktiv konfiguriert werden.

Vor dem Aufruf der `Prenet_start`-Methode des Datendienstes ruft RGM Methoden zur Anmeldung der Netzwerkadressen auf (die sie aber nicht als aktiv konfigurieren). Nach dem Aufruf der `Postnet_stop`-Methoden des Datendienstes ruft RGM die Methoden auf, die Netzwerkadressen abmelden. RGM bringt eine Ressourcengruppe in dieser Reihenfolge online:

1. Anmelden der Netzwerkadressen.
2. Aufrufen der `Prenet_start`-Methode des Datendienstes (falls vorhanden).
3. Aktiv-Konfigurieren der Netzwerkadressen.
4. Aufrufen der `Start`-Methode des Datendienstes (falls vorhanden).

Wenn RGM eine Ressourcengruppe offline nimmt, wird in umgekehrter Reihenfolge verfahren:

1. Aufrufen der `Stop`-Methode des Datendienstes (falls vorhanden).
2. Inaktiv-Konfigurieren der Netzwerkadressen.
3. Aufrufen der `Postnet_stop`-Methode des Datendienstes (falls vorhanden).
4. Abmelden der Netzwerkadressen.

Bei der Entscheidung darüber, ob `Start`-, `Stop`-, `Prenet_start`- oder `Postnet_stop`-Methoden verwendet werden sollten, muss zunächst die Serverseite betrachtet werden. Beim Online-bringen einer Ressourcengruppe, die sowohl Datendienst-Anwendungsressourcen als auch Netzwerkadressressourcen enthält, ruft RGM Methoden auf, um die Netzwerkadressen als aktiv zu konfigurieren, bevor er die `Start`-Methoden der Datendienstressourcen aufruft. Wenn also für einen Datendienst Netzwerkadressen zum Startzeitpunkt als aktiv konfiguriert werden müssen, verwenden Sie die `Start`-Methode zum Starten des Datendienstes.

Ebenso ruft RGM beim Offline-nehmen einer Ressourcengruppe, die sowohl Datendienstressourcen als auch Netzwerkadressressourcen enthält, Methoden zum Inaktiv-Konfigurieren der Netzwerkadressen auf, nachdem die `Stop`-Methoden der Datendienstressource aufgerufen wurden. Wenn also für einen Datendienst zum Stoppzeitpunkt Netzwerkadressen als inaktiv konfiguriert werden müssen, verwenden Sie die `Stop`-Methode zum Stoppen des Datendienstes.

Wenn Sie zum Beispiel einen Datendienst starten oder stoppen möchten, müssen Sie möglicherweise die Verwaltungsdienstprogramme oder Bibliotheken des Datendienstes aufrufen. Manchmal verfügt der Datendienst über Verwaltungsdienstprogramme bzw. -bibliotheken, die eine Client/Server-Netzwerkschnittstelle für die Verwaltung verwenden. Dabei ruft ein Verwaltungsdienstprogramm den Server-Dämon auf, so dass möglicherweise die Netzwerkadresse aktiv sein muss, damit das Verwaltungsprogramm oder die Bibliothek verwendet werden können. Verwenden Sie unter diesen Umständen die `Start`- und `Stop`-Methoden.

Wenn beim Starten und Stoppen des Datendienstes die Netzwerkadressen als inaktiv konfiguriert sein müssen, verwenden Sie die `Prenet_start`- und `Postnet_stop`-Methoden zum Starten und Stoppen des Datendienstes. Überprüfen Sie, ob die Client-Software unterschiedlich antwortet, je nachdem, ob zuerst die Netzwerkadresse oder der Datendienst nach einer Cluster-Rekonfiguration online gebracht werden (entweder `scha_control()` mit dem Argument `SCHA_GIVEOVER` oder ein Switchover mit `scswitch`). Die Client-Implementierung führt möglicherweise nur sehr wenige Wiederholversuche aus und stellt diese bald ein, wenn sie feststellt, dass der Datendienst-Port nicht verfügbar ist.

Wenn der Datendienst nicht erfordert, dass die Netzwerkadresse beim Starten als aktiv konfiguriert ist, starten Sie ihn vor der Aktiv-Konfigurierung der Netzwerkschnittstelle. So wird sichergestellt, dass der Datendienst sofort auf Client-Anforderungen reagieren kann, sobald die Netzwerkadresse als aktiv konfiguriert ist. Somit ist es weniger wahrscheinlich, dass die Clients ihre Wiederholversuche einstellen. Unter diesen Umständen sollten Sie die `Prenet_start`-Methode anstelle der `Start`-Methode zum Starten des Datendienstes verwenden.

Wenn Sie die `Postnet_stop`-Methode verwenden, ist die Datendienstressource zu dem Zeitpunkt noch aktiv, an dem die Netzwerkadresse bereits als inaktiv konfiguriert wurde. Erst wenn die Netzwerkadresse als inaktiv konfiguriert wurde, wird die `Postnet_stop`-Methode aufgerufen. Daher wird das TCP bzw. der UDP-Dienst-Port oder dessen RPC-Programmnummer den Clients im Netzwerk immer als verfügbar angezeigt, außer wenn die Netzwerkadresse ebenfalls nicht antwortet.

Hinweis – Wenn Sie einen RPC-Dienst auf dem Cluster installieren, darf der Dienst folgende Programmnummern nicht verwenden: 100141, 100142 und 100248. Diese Nummern sind den Sun Cluster-Dämonen `rgmd_receptionist`, `fed` und `pmfd` vorbehalten. Wenn der von Ihnen installierte RPC-Dienst eine dieser Programmnummern verwendet, müssen Sie ihn dahingehend ändern, dass er eine andere Programmnummer verwendet.

Bei der Entscheidung darüber, ob die `Start`- und `Stop`-Methoden oder die `Prenet_start`- und `Postnet_stop`-Methoden bzw. beide zusammen verwendet werden sollten, müssen die Anforderungen und das Verhalten von Server und Client in Betracht gezogen werden.

`Init`-, `Fini`- und `Boot`-Methoden

Mithilfe von drei optionalen Methoden, `Init`, `Fini` und `Boot`, kann RGM Initialisierungs- und Beendigungscode für eine Ressource ausführen. RGM ruft die `Init`-Methode auf, um eine einmalige Initialisierung der Ressource auszuführen, wenn diese in einen verwalteten Zustand versetzt wird — entweder, weil die Ressourcengruppe aus einem nicht verwalteten in einen verwalteten Zustand versetzt wird, oder weil sie in einer bereits verwalteten Ressourcengruppe erstellt wird.

RGM ruft die `Fini`-Methode auf, um nach der Ressource zu bereinigen, wenn diese in einen unverwalteten Zustand versetzt wird — entweder, wenn die Ressourcengruppe in einen nicht verwalteten Zustand gebracht wird, oder wenn sie aus einer verwalteten Ressourcengruppe gelöscht wird. Die Bereinigung muss idempotent sein. Wenn also die Bereinigung bereits stattgefunden hat, muss `Fini` mit 0 (Erfolg) beendet werden.

RGM ruft die `Boot`-Methode auf Knoten auf, die dem Cluster neu beigetreten sind, die also gestartet bzw. neu gestartet wurden.

Die `Boot`-Methode führt in der Regel die gleiche Initialisierung wie `Init` aus. Diese Initialisierung muss idempotent sein. Wenn die Ressource also bereits auf dem lokalen Knoten initialisiert wurde, müssen `Boot` und `Init` mit 0 (Erfolg) beendet werden.

Überwachen einer Ressource

Üblicherweise werden Monitore so implementiert, dass sie in bestimmten Zeitabständen Fehlertestsignale an die Ressourcen senden, um festzustellen, ob die getesteten Ressourcen korrekt arbeiten. Wenn ein Fehlertestsignal einen Fehler ergibt, kann der Monitor versuchen, lokal neu zu starten oder ein Failover für die betroffene Ressourcengruppe durch Aufrufen der RMAPI-Funktion `scha_control()` bzw. der DSDL-Funktion `scds_fm_action()` anzufordern.

Sie können auch die Leistung einer Ressource überwachen und sie einstellen bzw. einen Leistungsbericht erstellen. Das Schreiben eines ressourcentypspezifischen Fehler-Monitors ist völlig optional. Selbst wenn Sie sich dafür entscheiden, keinen Fehler-Monitor zu schreiben, profitiert der Ressourcentyp von der Basisüberwachung des Clusters, die Sun Cluster selbst ausführt. Sun Cluster stellt Fehler der Host-Hardware, schwerwiegende Fehler des Host-Betriebssystems sowie Fehlschlagen der Host-Kommunikation auf den öffentlichen Netzwerken fest.

Obwohl RGM keinen Ressourcen-Monitor direkt aufruft, ermöglicht das Programm den automatischen Start von Monitoren für Ressourcen. Beim Offline-nehmen einer Ressource ruft RGM die `Monitor_stop`-Methode auf, um den Ressourcen-Monitor auf den lokalen Knoten zu stoppen, bevor die Ressource selbst gestoppt wird. Beim Online-bringen einer Ressource ruft RGM die `Monitor_start`-Methode auf, nachdem die Ressource selbst gestartet wurde.

Mithilfe der RMAPI-Funktion `scha_control()` und der DSDL-Funktion `scds_fm_action()` (die `scha_control()` aufruft) können die Ressourcen-Monitore das Failover einer Ressourcengruppe auf einen anderen Knoten anfordern. Als eine der Kontrollprüfungen ruft `scha_control()` den Befehl `Monitor_check` auf (falls definiert), um festzustellen, ob der angeforderte Knoten zuverlässig genug ist, um die Ressourcengruppe mit der Ressource zu unterstützen. Wenn `Monitor_check` zurückmeldet, dass der Knoten nicht zuverlässig ist, oder das Zeitlimit für die Methode überschritten wird, sucht RGM nach einem anderen Knoten, um der Failover-Anforderung nachzukommen. Wenn `Monitor_check` auf allen Knoten fehlschlägt, wird das Failover abgebrochen.

Der Ressourcen-Monitor kann die Eigenschaften `Status` und `Status_msg` einstellen, um die Monitorsicht des Ressourcenzustands wiederzugeben. Verwenden Sie die RMAPI-Funktion `scha_resource_setstatus()`, den Befehl `scha_resource_setstatus` oder die DSDL-Funktion `scds_fm_action()` zum Einstellen dieser Eigenschaften.

Hinweis – `Status` und `Status_msg` eignen sich zwar besonders für einen Ressourcen-Monitor; diese Eigenschaften können jedoch von jedem beliebigen Programm eingestellt werden.

Ein Beispiel eines mit der RMAPI implementierten Fehler-Monitors finden Sie unter „Definieren eines Fehler-Monitors" auf Seite 109. Ein Beispiel eines mit der DSDL implementierten Fehler-Monitors finden Sie unter „`SUNW.xfnts`-Fehler-Monitor" auf Seite 155. Informationen zu Fehler-Monitoren, die in von Sun gelieferte Datendienste eingebaut sind, finden Sie im *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

Hinzufügen von Meldungsprotokollierung zu einer Ressource

Wenn Sie Statusmeldungen in derselben Protokolldatei wie andere Cluster-Meldungen aufzeichnen möchten, verwenden Sie die erweiterte Funktion `scha_cluster_getlogfacility()`, um die Funktionsnummer abzurufen, die für das Protokollieren der Cluster-Meldungen verwendet wird.

Verwenden Sie diese Funktionsnummer mit der normalen Solaris-Funktion `syslog()`, um Meldungen in das Cluster-Protokoll zu schreiben. Sie können auch über die generische Schnittstelle `scha_cluster_get()` auf die Informationen zur Cluster-Protokollfunktion zugreifen.

Bereitstellen von Prozessverwaltung

Die RMAPI und die DSDL stellen Prozessverwaltungsfunktionen für die Implementierung von Ressourcen-Monitoren und Ressourcensteuerungs-Rückmeldungen bereit. Die RMAPI definiert die folgenden Funktionen. In der Online-Dokumentation finden Sie Details zu jedem dieser Befehle und Programme.

Process Monitor Facility: `pmfadm` und `rpc.pmfd`
: Die Prozessüberwachungsfunktion (Process Monitor Facility, PMF) ermöglicht das Überwachen von Prozessen und untergeordneten Prozessen sowie deren Neustart, wenn sie abgebrochen werden. Die Funktion besteht aus dem `pmfadm`-Befehl für das Starten und Steuern von überwachten Prozessen sowie dem `rpc.pmfd`-Dämon.

`halockrun`
: Ein Programm für das Ausführen eines untergeordneten Programms mit Dateisperre. Dieser Befehl lässt sich gut in Shell-Skripts verwenden.

`hatimerun`
: Ein Programm für das Ausführen eines untergeordneten Programms unter Zeitüberschreitungssteuerung. Dieser Befehl lässt sich gut in Shell-Skripts verwenden.

Die DSDL stellt die `scds_hatimerun`-Funktion für die Implementierung von `hatimerun` bereit.

Die DSDL stellt einen Funktionssatz (`scds_pmf_*`) für die Implementierung von PMF bereit. Einen Überblick über die Funktionen von DSDL-PMF und eine Auflistung der einzelnen Funktionen finden Sie unter „PMF-Funktionen" auf Seite 217.

Verwaltungsunterstützung für eine Ressource

Zu den Verwaltungsaktionen bei Ressourcen gehört das Einstellen und Ändern von Ressourceneigenschaften. Die API definiert die Rückmeldemethoden `Validate` und `Update`, so dass Sie diese Verwaltungsaktionen nutzen können.

RGM ruft die optionale `Validate`-Methode auf, wenn eine Ressource erstellt wird und wenn eine Verwaltungsaktion die Eigenschaften der Ressource bzw. der Gruppe, zu der sie gehört, aktualisiert. RGM übergibt die Eigenschaftswerte für die Ressource und deren Ressourcengruppe an die `Validate`-Methode. RGM ruft `Validate` für den Satz von Cluster-Knoten auf, der von der `Init_nodes`-Eigenschaft des Ressourcentyps angegeben wird. (Informationen zu `Init_nodes` finden Sie unter „Ressourcentypeigenschaften" auf Seite 253 bzw. in der Online-Dokumentation unter `rt_properties`(5).) RGM ruft `Validate` auf, bevor die Erstellung bzw. Aktualisierung angewendet werden kann. Ein fehlerhafter Beendigungscode von der Methode oder einem der Knoten führt zu einem Fehlschlagen der Erstellung bzw. Aktualisierung.

RGM ruft `Validate` nur dann auf, wenn Ressourcen- bzw. Gruppeneigenschaften über eine Verwaltungsaktion geändert werden, und nicht, wenn RGM Eigenschaften einstellt oder wenn ein Monitor die Ressourceneigenschaften `Status` und `Status_msg` einstellt.

RGM ruft die optionale `Update`-Methode auf, um eine laufende Ressource darüber zu benachrichtigen, dass Eigenschaften geändert wurden. RGM ruft `Update` auf, nachdem eine Verwaltungsaktion die Eigenschaften einer Ressource bzw. deren Gruppe erfolgreich eingestellt hat. RGM ruft diese Methode auf denjenigen Knoten auf, auf denen die Ressource online ist. Die Methode kann die API-Zugriffsmethoden verwenden, um Eigenschaftswerte zu lesen, die eine aktive Ressource betreffen könnten, und um die laufende Ressource entsprechend anzupassen.

Implementieren einer Failover-Ressource

Eine Failover-Ressourcengruppe enthält Netzwerkadressen wie die eingebauten Ressourcentypen logischer Hostname und gemeinsam genutzte Adresse, sowie Failover-Ressourcen wie die Datendienst-Anwendungsressourcen für einen Failover-Datendienst. Die Netzwerkadressressourcen werden zusammen mit ihren abhängigen Datendienstressourcen zwischen Cluster-Knoten verschoben, wenn Datendienste ein Failover bzw. Switchover ausführen. RGM stellt eine Reihe von Eigenschaften zur Unterstützung der Implementierung einer Failover-Ressource bereit.

Die boolesche Ressourcentypeigenschaft `Failover` muss auf `TRUE` eingestellt werden, um zu verhindern, dass die Ressource in einer Ressourcengruppe konfiguriert wird, die auf mehr als einem Knoten gleichzeitig online sein kann. Diese Eigenschaft verwendet standardmäßig `FALSE`, so dass Sie sie für eine Failover-Ressource in der RTR-Datei als `TRUE` deklarieren müssen.

Die Ressourceneigenschaft `Scalable` legt fest, ob die Ressource die Cluster-Funktion gemeinsam genutzte Adresse verwendet. Bei einer Failover-Ressource muss `Scalable` auf `FALSE` eingestellt werden, weil eine Failover-Ressource keine gemeinsam genutzten Adressen verwendet.

Die Ressourcengruppeneigenschaft `RG_mode` ermöglicht es dem Cluster-Verwalter, eine Ressourcengruppe als Failover bzw. Scalable zu identifizieren. Wenn `RG_mode` auf `FAILOVER` eingestellt ist, setzt RGM die `Maximum_primaries`-Eigenschaft der Gruppe auf 1 und beschränkt die Ressourcengruppe, so dass sie nur von einem einzigen Knoten unterstützt wird. RGM lässt nicht zu, dass eine Ressource, deren `Failover`-Eigenschaft `TRUE` ist, in einer Ressourcengruppe erstellt wird, deren `RG_mode` auf `SCALABLE` eingestellt ist.

Die Ressourcengruppeneigenschaft `Implicit_network_dependencies` gibt an, dass RGM implizite starke Abhängigkeiten der Nicht-Netzwerkadressressourcen von allen Netzwerkadressressourcen (logischer Hostname und gemeinsam genutzte Adresse) innerhalb der Gruppe erzwingt. Das bedeutet, dass für die Nicht-Netzwerkadressressourcen (Datendienste) in der Gruppe keine `Start`-Methoden aufgerufen werden, bevor die Netzwerkadressen in der Gruppe als aktiv konfiguriert werden. Die Eigenschaft `Implicit_network_dependencies` wird standardmäßig auf `TRUE` eingestellt.

Implementieren einer Scalabe-Ressource

Eine skalierbare Ressource kann auf mehr als einem Knoten gleichzeitig online sein. Skalierbare Ressourcen sind Datendienste wie Sun Cluster HA für Sun Java System Web Server (früher Sun Cluster HA für Sun ONE Web Server) und Sun Cluster HA für Apache.

RGM stellt mehrere Eigenschaften bereit, mit denen die Implementierung einer skalierbaren Ressource unterstützt wird.

Die boolesche Ressourcentypeigenschaft `Failover` muss auf `FALSE` eingestellt werden, damit die Ressource in einer Ressourcengruppe konfiguriert werden kann, die auf mehr als einem Knoten gleichzeitig online sein kann.

Die Ressourceneigenschaft `Scalable` legt fest, ob die Ressource die Cluster-Funktion gemeinsam genutzte Adresse verwendet. Dieser Wert muss auf `TRUE` eingestellt werden, da ein Scalable-Dienst eine gemeinsam genutzte Adressressource verwendet, damit mehrere Instanzen des Scalable-Dienstes dem Client als ein einziger Dienst dargestellt werden.

Mithilfe der `RG_mode`-Eigenschaft kann der Cluster-Verwalter Ressourcengruppen als Failover oder Scalable identifizieren. Wenn `RG_mode` auf `SCALABLE` eingestellt ist, lässt RGM für `Maximum_primaries` einen Wert größer als 1 zu, was bedeutet, dass die Gruppe von mehreren Knoten gleichzeitig unterstützt werden kann. RGM lässt zu, dass eine Ressource, deren `Failover`-Eigenschaft `FALSE` ist, in einer Ressourcengruppe instanziiert wird, deren `RG_mode` auf `SCALABLE` eingestellt ist.

Der Cluster-Verwalter erstellt eine skalierbare Ressourcengruppe für Scalable-Dienstressourcen und eine eigene Failover-Ressourcengruppe für die Ressourcen mit gemeinsam genutzter Adresse, von der die skalierbare-Ressource abhängt.

Der Cluster-Verwalter verwendet die Ressourcengruppeneigenschaft `RG_dependencies` zum Angeben der Reihenfolge, in der Ressourcengruppen auf einem Knoten online bzw. offline gebracht werden. Diese Reihenfolge ist für einen Scalable-Dienst wichtig, da sich die skalierbaren Ressourcen und die gemeinsam genutzten Adressressourcen, von denen sie abhängen, in unterschiedlichen Ressourcengruppen befinden. Für einen Scalable-Datendienst müssen die Netzwerkadressressourcen (gemeinsam genutzte Adressen) als aktiv konfiguriert werden, bevor der Dienst gestartet wird. Daher muss der Verwalter die Eigenschaft `RG_dependencies` der Ressourcengruppe mit dem Scalable-Dienst so einstellen, dass sie die Ressourcengruppe mit den gemeinsam genutzten Adressressourcen enthält.

Wenn die Scalable-Eigenschaft in der RTR-Datei für eine Ressource deklariert wird, erstellt RGM automatisch den folgenden Satz Scalable-Eigenschaften für die Ressource:

`Network_resources_used`
: Identifiziert die gemeinsam genutzten Adressressourcen, die von dieser Ressource verwendet werden. Diese Eigenschaft enthält standardmäßig eine leere Zeichenkette, so dass der Cluster-Verwalter beim Erstellen der Ressource die aktuelle Liste gemeinsam genutzter Adressen angeben muss, die der Scalable-Dienst verwendet. Der `scsetup`-Befehl und SunPlex-Manager stellen Funktionen bereit, mit denen die erforderlichen Ressourcen und Gruppen für Scalable-Dienste automatisch konfiguriert werden können.

`Load_balancing_policy`
: Gibt das Lastausgleichsverfahren für die Ressource an. Sie können das Verfahren in der RTR-Datei explizit einrichten oder den Standardwert LB_WEIGHTED zulassen. In beiden Fällen kann der Cluster-Verwalter beim Erstellen der Ressource diesen Wert ändern, sofern nicht in der RTR-Datei `Tunable` für `Load_balancing_policy` auf `NONE` oder `FALSE` gesetzt wird. Zulässige Werte sind:

  `LB_WEIGHTED`
  : Die Last wird auf mehrere Knoten verteilt, entsprechend den in der Eigenschaft `Load_balancing_weights` eingestellten Gewichtungen.

  `LB_STICKY`
  : Ein bestimmter Client des Scalable-Dienstes (identifiziert durch die Client-IP-Adresse) wird immer an denselben Cluster-Knoten gesendet.

  `LB_STICKY_WILD`
  : Ein bestimmter Client (identifiziert durch die Client-IP-Adresse), der eine Verbindung mit einer IP-Adresse eines Sticky-Dienstes mit Platzhalter herstellt, wird immer zu demselben Cluster-Knoten gesendet, unabhängig von der Port-Nummer, an die er gelangt.

  Für einen Scalable-Dienst mit `Load_balancing_policy`, `LB_STICKY` oder `LB_STICKY_WILD` kann das Ändern von `Load_balancing_weights`, während der Dienst online ist, zu einem Zurücksetzen von vorhandenen Client-Affinitäten führen. In diesem Fall kann es vorkommen, dass ein anderer Knoten eine anschließende Client-Anforderung bedient, auch wenn der Client vorher von einem anderen Cluster-Knoten bedient wurde.

	Auch wenn eine neue Instanz des Dienstes auf einem Cluster gestartet wird, können vorhandene Client-Affinitäten zurückgesetzt werden.
`Load_balancing_weights`	Gibt die Last an, die an jeden Knoten gesendet wird. Das Format ist *Gewichtung@Knoten, Gewichtung@Knoten*, wobei *Gewichtung* eine Ganzzahl ist, welche den relativen Anteil der Last für den angegebenen *Knoten* darstellt. Der an einen Knoten verteilte Lastanteil ist die Gewichtung dieses Knotens, geteilt durch die Summe aller Gewichtungen aktiver Instanzen. Zum Beispiel gibt `1@1,3@2` an, dass Knoten 1 1/4 der Last und Knoten 2 3/4 erhält.
`Port_list`	Identifiziert die Ports, die der Server abhört. Diese Eigenschaft enthält standardmäßig eine leere Zeichenkette. In der RTR-Datei können Sie eine Port-Liste bereitstellen. Andernfalls muss der Cluster-Verwalter die aktuelle Port-Liste bereitstellen, wenn er die Ressource erstellt.

Sie können einen Datendienst erstellen, den der Administrator dann als Scalable oder als Failover konfigurieren kann. Dafür werden sowohl die Ressourcentypeigenschaft `Failover` als auch die Ressourceneigenschaft `Scalable` in der RTR-Datei des Datendienstes als `FALSE` deklariert. Beim Erstellen muss die `Scalable`-Eigenschaft als “Tunable” angegeben werden.

Der `Failover`-Eigenschaftswert (`FALSE`) lässt zu, dass die Ressource in eine skalierbare Ressourcengruppe konfiguriert wird. Der Verwalter kann gemeinsam genutzte Adressen aktivieren, indem er den Wert beim Erstellen der Ressource von `Scalable` zu `TRUE` ändert und so einen Scalable-Dienst erstellt.

Andererseits kann der Verwalter auch dann die Ressource in eine Failover-Ressourcengruppe konfigurieren, um einen Failover-Dienst zu implementieren, wenn `Failover` auf `FALSE` eingestellt ist. Der Verwalter ändert den Wert von `Scalable`, der `FALSE` lautet, nicht. Zum Unterstützen dieser Möglichkeit sollte die `Validate`-Methode in der `Scalable`-Eigenschaft aktiviert werden. Wenn `Scalable` auf `FALSE` eingestellt ist, muss sichergestellt werden, dass die Ressource in eine Failover-Ressourcengruppe konfiguriert wurde.

Das *Sun Cluster Concepts Guide for Solaris OS* enthält weitere Informationen über skalierbare Ressourcen.

Validierungsprüfungen für Scalable-Dienste

Wenn eine Ressource erstellt oder aktualisiert wird und die Scalable-Eigenschaft auf `TRUE` eingestellt ist, validiert RGM verschiedene Ressourceneigenschaften. Wenn die Eigenschaften nicht richtig konfiguriert sind, weist RGM die versuchte Aktualisierung bzw. die Erstellung zurück. RGM führt folgende Prüfungen aus:

- Die Eigenschaft `Network_resources_used` darf nicht leer sein und muss die Namen der vorhandenen gemeinsam genutzten Adressressourcen enthalten. Jeder Knoten in der `Nodelist` der Ressourcengruppe mit der skalierbaren Ressource muss entweder in der `NetIfList`-Eigenschaft oder in der `AuxNodeList`-Eigenschaft jeder der benannten gemeinsam genutzten Adressressourcen stehen.
- Die RG_dependencies-Eigenschaft der Ressourcengruppe mit der skalierbaren Ressource muss die Ressourcengruppen aller gemeinsam genutzten Adressressourcen enthalten, die in der Eigenschaft `Network_resources_used` der skalierbaren Ressource aufgelistet sind.
- Die Port_list-Eigenschaft darf nicht leer sein und muss eine Liste der Port-Protokollpaare enthalten, damit das Protokoll entweder TCP oder UDP ist. Beispiel:

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

Schreiben und Testen von Datendiensten

Dieser Abschnitt enthält einige Informationen zum Schreiben und Testen von Datendiensten.

Verwenden von Keep-Alives

Auf der Serverseite schützt die Verwendung von TCP-Keep-Alives den Server vor der Verschwendung von Systemressourcen an einen heruntergefahrenen bzw. netzwerkpartitionierten Client. Wenn diese Ressourcen nicht bereinigt werden (auf einem Server, der lange genug aktiv bleibt), wachsen die verschwendeten Ressourcen schließlich unkontrolliert an, während die Clients abstürzen und neu starten.

Wenn die Client/Server-Kommunikation einen TCP-Strom verwendet, sollten sowohl der Client als auch der Server den TCP-Keep-Alive-Mechanismus aktivieren. Dies gilt auch für einzelne Nicht-HA-Server.

Andere verbindungsorientierte Protokolle können ebenfalls über einen Keep-Alive-Mechanismus verfügen.

Auf der Clientseite ermöglicht die Verwendung von TCP-Keep-Alives dem Client, eine Benachrichtigung zu erhalten, wenn eine Netzwerkressource ein Failover bzw. Switchover von einem realen Host zu einem anderen ausgeführt hat. Diese Übertragung der Netzwerkadressressource bricht die TCP-Verbindung ab. Wenn der Client allerdings das Keep-Alive nicht aktiviert hat, wird die Verbindungsunterbrechung möglicherweise nicht festgestellt, wenn die Verbindung zu diesem Zeitpunkt ruhte.

Angenommen, der Client wartet zum Beispiel auf die Antwort des Servers auf eine langfristige Anforderung, und die Anforderungsmeldung des Clients ist bereits beim Server angekommen und wurde auf TCP-Ebene bestätigt. In dieser Situation braucht das TCP-Modul des Clients die Anforderung nicht immer neu zu übertragen, und die Client-Anwendung ist blockiert, während sie die Antwort auf die Anforderung abwartet.

Womöglich sollte die Client-Anwendung zusätzlich zur Verwendung des TCP-Keep-Alive-Mechanismus ebenfalls in regelmäßigen Abständen ihr eigenes Keep-Alive ausführen, da der TCP-Mechanismus nicht in allen möglichen Grenzfällen perfekt funktioniert. Für die Verwendung eines Keep-Alive auf Anwendungsebene muss das Client/Server-Protokoll normalerweise einen Null-Vorgang unterstützen, oder zumindest einen effizienten schreibgeschützten Vorgang, wie einen Statusvorgang.

Testen von HA-Datendiensten

Dieser Abschnitt enthält Vorschläge zum Testen einer Datendienstimplementierung in der HA-Umgebung. Die Testfälle sind Beispiele und daher nicht umfassend. Sie benötigen Zugriff auf eine Testkonfiguration von Sun Cluster, damit sich das Testen nicht auf die Produktionsrechner auswirkt.

Testen Sie, ob sich der HA-Datendienst in allen Fällen richtig verhält, wenn eine Ressourcengruppe zwischen realen Hosts verschoben wird. Diese Fälle schließen Systemabstürze und die Verwendung des `scswitch`-Befehls mit ein. Testen Sie, ob die Client-Rechner nach diesen Ereignissen weiterhin Dienste erhalten.

Testen Sie die Idempotenz der Methoden. Ersetzen Sie zum Beispiel jede Methode vorübergehend mit einem kurzen Shell-Skript, das die Originalmethode zwei oder mehrere Male aufruft.

Koordinieren von Abhängigkeiten zwischen Ressourcen

Manchmal stellt ein Client/Server-Datendienst Anforderungen an einen anderen Client/Server-Datendienst, während er gleichzeitig einer Client-Anforderung nachkommt. Grob gesagt hängt ein Datendienst A von einem Datendienst B ab, wenn A seinen Dienst nur bereitstellen kann, sofern B seinen Dienst bereitstellt. Um dieser Anforderung zu entsprechen, lässt Sun Cluster die Konfiguration von Ressourcenabhängigkeiten innerhalb einer Ressourcengruppe zu. Die Abhängigkeiten wirken sich auf die Reihenfolge aus, in der Sun Cluster Datendienste startet und stoppt. Einzelheiten finden Sie in der Online-Dokumentation unter `scrgadm`(1M).

Wenn Ressourcen Ihres Ressourcentyps von Ressourcen eines anderen Typs abhängen, müssen Sie den Benutzer anweisen, die Ressourcen und Ressourcengruppen entsprechend zu konfigurieren bzw. Skripts oder Tools für die korrekte Konfiguration bereitstellen. Wenn die abhängige Ressource auf demselben Knoten wie diejenige Ressource, von der sie abhängt, läuft, müssen beide Ressourcen in derselben Ressourcengruppe konfiguriert werden.

Sie müssen entscheiden, ob Sie explizite Ressourcenabhängigkeiten verwenden möchten oder darauf verzichten und sich für die Verfügbarkeit von anderen Datendiensten im eigenen Code Ihres HA-Datendienstes entscheiden. Falls die abhängige Ressource und diejenige, von der sie abhängt, auf unterschiedlichen Knoten laufen können, werden sie in unterschiedlichen Ressourcengruppen konfiguriert. In diesem Fall ist ein Abruf erforderlich, da keine Ressourcenabhängigkeiten zwischen Gruppen konfiguriert werden können.

Einige Datendienste speichern keine Daten direkt, sondern hängen von einem anderen Backend-Datendienst für das Speichern aller Daten ab. Ein solcher Datendienst überträgt alle Lese- und Aktualisierungsanforderungen in Aufrufe an den Backend-Datendienst. Ein Beispiel wäre ein Client/Server-Terminkalenderdienst, der alle Daten in einer SQL-Datenbank wie Oracle aufbewahrt. Der Terminkalenderdienst verfügt über ein eigenes Client/Server-Netzwerkprotokoll. Es kann zum Beispiel ein Protokoll unter Verwendung einer RPC-Spezifikationssprache definiert haben, wie ONC RPC.

In der Sun Cluster-Umgebung können Sie HA-ORACLE verwenden, um die Backend-ORACLE-Datenbank hoch verfügbar zu machen. Dann können Sie einfache Methoden für das Starten und Stoppen des Terminkalenderdämons schreiben. Der Endbenutzer registriert den Terminkalender-Ressourcentyp bei Sun Cluster.

Wenn die Terminkalenderanwendung auf demselben Knoten wie die Oracle-Datenbank laufen muss, konfiguriert der Endbenutzer die Terminkalenderressource in derselben Ressourcengruppe wie die HA-ORACLE-Ressource und macht die Terminkalenderressource von der HA-ORACLE-Ressource abhängig. Diese Abhängigkeit wird über die Eigenschaftsmarkierung `Resource_dependencies` in `scrgadm` angegeben.

Wenn die HA-ORACLE-Ressource auf einem anderen Knoten als die Terminkalenderressource laufen kann, konfiguriert der Endbenutzer sie in zwei getrennten Ressourcengruppen. Der Endbenutzer kann eine Ressourcengruppenabhängigkeit der Terminkalendergruppe von der Oracle-Ressourcengruppe konfigurieren. Die Ressourcengruppenabhängigkeiten sind jedoch nur dann wirksam, wenn beide Ressourcengruppen gleichzeitig auf demselben Knoten gestartet bzw. gestoppt werden. Daher kann der Kalender-Datendienstdämon nach dem Starten das Warten auf die Verfügbarkeit der Oracle-Datenbank aufrufen. Die `Start`-Methode des Kalenderressourcentyps gibt in einem solchen Fall in der Regel einfach Erfolg zurück, da eine unbegrenzte Sperre der `Start`-Methode ihre Ressourcengruppe in einen belegten Zustand versetzen würde. Dann wären keine weiteren Zustandsänderungen der Gruppe wie zum Beispiel Bearbeitungen, Failover oder Switchover mehr möglich. Wenn jedoch die Zeit für die `Start`-Methode der Kalenderressource überschritten bzw. als Nicht-Null beendet wird, könnte es vorkommen, dass die Ressourcengruppe in eine Schleife zwischen zwei oder mehr Knoten gerät, während die Oracle-Datenbank nicht verfügbar bleibt.

KAPITEL 3

Aufrüsten eines Ressourcentyps

Dieses Kapitel behandelt die Themen, die zum Aufrüsten eines Ressourcentyps und Migrieren einer Ressource bekannt sein müssen.

Überblick

Systemverwalter müssen in der Lage sein, eine neue Version eines vorhandenen Ressourcentyps zu installieren und zu registrieren, die Registrierung mehrerer Versionen eines bestimmten Ressourcentyps zuzulassen und eine vorhandene Ressource zu einer neuen Version des Ressourcentyps zu migrieren, ohne dabei die Ressource löschen und neu erstellen zu müssen. Ressourcenentwickler müssen die Anforderungen für eine Ressourcentypaufrüstung und Ressourcenmigration verstehen.

Ressourcentypen, die für spätere Aufrüstungen geeignet sind, werden als *aufrüstfähig* bezeichnet.

Eine neue Version eines Ressourcentyps kann sich von einer vorherigen Version in mehreren Punkten unterscheiden:

- Die Attribute der Ressourcentypeigenschaften können sich ändern.
- Der Satz der deklarierten Ressourceneigenschaften, einschließlich Standard- und Erweiterungseigenschaften, kann sich ändern.
- Die Attribute von Ressourceneigenschaften, wie `default`, `min`, `max`, `arraymin`, `arraymax`, oder die Einstellbarkeit können sich ändern.
- Der Satz der deklarierten Methoden kann unterschiedlich sein.
- Die Implementierung von Methoden oder Monitoren kann sich ändern.

Der Ressourcentypentwickler entscheidet, wann eine vorhandene Ressource zu einer neuen Version migrieren kann, und wählt dabei unter folgenden Einstellbarkeitsoptionen. Die Optionen werden von am wenigsten zu am stärksten einschränkend aufgelistet:

- Jederzeit (`ANYTIME`)
- Wenn die Ressource nicht überwacht wird (`WHEN_UNMONITORED`)
- Wenn die Ressource offline ist (`WHEN_OFFLINE`)
- Wenn die Ressource deaktiviert ist (`WHEN_DISABLED`)
- Wenn die Ressourcengruppe nicht verwaltet ist (`WHEN_UNMANAGED`)
- Bei der Erstellung (`AT_CREATION`)

Eine Erklärung der einzelnen Optionen finden Sie unter „`Type_version`-Ressourceneigenschaft" auf Seite 64.

Hinweis – Im gesamten Kapitel wird der `scrgadm`-Befehl verwendet, wenn die Ausführung einer Aufrüstung behandelt wird. Der Verwalter kann jedoch neben dem `scrgadm`-Befehl auch die GUI oder den `scsetup`-Befehl verwenden, um die Aufrüstung auszuführen.

Ressourcentyp-Registrierungsdatei

Ressourcentypname

Die drei Komponenten des Ressourcentypnamens sind Eigenschaften, die in der RTR-Datei als *Vendor_id*, *Resource_type* und *RT_version* angegeben werden. Der `scrgadm`-Befehl fügt den Punkt und das Semikolon als Trennzeichen ein, um den Namen des Ressourcentyps zu erstellen:

```
vendor_id.resource_type:rt_version
```

Das Präfix *Vendor_id* dient zur Unterscheidung zwischen zwei Registrierungsdateien des gleichen Namens, die von verschiedenen Herstellern geliefert werden. *RT_version* unterscheidet zwischen mehreren registrierten Versionen (Aufrüstungen) des gleichen Basisressourcentyps. Um sicherzustellen, dass *Vendor_id* einmalig ist, wird empfohlen, das Börsensymbol für das Unternehmen, das den Ressourcentyp erstellt, zu verwenden.

Die Registrierung des Ressourcentyps schlägt fehl, wenn die *RT_version*-Zeichenkette eines der folgenden Zeichen enthält: Leerzeichen, Tabulator, Schrägstrich (`/`), umgekehrter Schrägstrich (`\`), Sternchen (`*`), Fragezeichen (`?`), Komma (`,`), Strichpunkt (`;`), linke eckige Klammer (`[`) oder rechte eckige Klammer (`]`).

Die `RT_Version`-Eigenschaft, die in Sun Cluster 3.0 optional war, ist in Sun Cluster 3.1 verbindlich.

Der vollständig qualifizierte Name ist der Name, der von folgendem Befehl zurückgegeben wird:

`scha_resource_get -O Type -R` *Ressourcenname* `-G` *Ressourcengruppenname*

Ressourcentypnamen, die vor Sun Cluster 3.1 erstellt wurden, verwenden weiterhin folgende Form:

```
vendor_id.resource_type
```

Anweisungen

RTR-Dateien für aufrüstfähige Ressourcentypen müssen eine `#$upgrade`-Anweisung enthalten, gefolgt von Null oder mehr Anweisungen der Form:

`#$upgrade_from` *Version Einstellbarkeit*

Die `upgrade_from`-Anweisung besteht aus der Zeichenkette `#$upgrade_from`, gefolgt von der Einstellbarkeitseinschränkung `RT_Version` für die Ressource. Wenn der aufzurüstende Ressourcentyp keine Version hat, wird `RT_Version` als leere Zeichenkette angegeben, wie im letzten Beispiel unten gezeigt.

```
#$upgrade_from   "1.1"   when_offline
#$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
```

RGM erzwingt diese Einschränkungen bei einer Ressource, wenn der Systemverwalter versucht, `Type_version` der Ressource zu ändern. Wenn die aktuelle Version des Ressourcentyps nicht in der Liste angezeigt wird, erzwingt RGM die Einstellbarkeit von `WHEN_UNMANAGED`.

Diese Anweisungen müssen zwischen dem Abschnitt der Ressourcentyp-Eigenschaftsdeklarationen in der RTR-Datei und dem Abschnitt der Ressourcendeklarationen in der RTR-Datei stehen. Siehe `rt_reg`(4).

Ändern von `RT_Version` in einer RTR-Datei

Die `RT_Version`-Zeichenkette in einer RTR-Datei ist immer dann zu ändern, wenn sich der Inhalt der RTR-Datei ändert. Der Wert dieser Eigenschaft muss deutlich machen, welche die neuere und welche die ältere Version des Ressourcentyps ist. Eine Änderung der `RT_Version`-Zeichenkette ist nicht erforderlich, solange keine Änderungen in der RTR-Datei vorgenommen werden.

Ressourcentypnamen in früheren Versionen von Sun Cluster

Ressourcentypnamen in Sun Cluster 3.0 enthielten kein Versionssuffix:

```
vendor_id.resource_name
```

Ein Ressourcentyp, der ursprünglich unter Sun Cluster 3.0 registriert wurde, behält diese Form bei, auch wenn die Cluster-Software auf Sun Cluster 3.1 oder höhere Versionen aufgerüstet wird. Ebenso erhält ein Ressourcentyp, in dessen RTR-Datei die `#$upgrade`-Anweisung fehlt, einen Namen im Sun Cluster 3.0-Format ohne Versionssuffix, wenn die RTR-Datei auf einem Cluster registriert ist, der mit Sun Cluster 3.1-Software bzw. einer höheren Version läuft.

Sie können RTR-Dateien mit der `#$upgrade`- oder `#$upgrade_from` -Anweisung in Sun Cluster 3.0 registrieren. Das Migrieren vorhandener Ressourcen zu neuen Ressourcentypen in Sun Cluster 3.0 wird jedoch nicht unterstützt.

`Type_version`-Ressourceneigenschaft

Die Standardressourceneigenschaft `Type_version` speichert die `RT_Version`-Eigenschaft eines Ressourcentyps. Diese Eigenschaft ist nicht in der RTR-Datei vorhanden. Der Systemverwalter bearbeitet diesen Eigenschaftswert mithilfe des folgenden Befehls:

`scrgadm -c -j` *Ressource* `-y Type_version=`*neue_Version*

Die Einstellbarkeit dieser Eigenschaft ergibt sich aus:

- Der aktuellen Version des Ressourcentyps,
- Den `#$upgrade_from`-Anweisungen in der RTR-Datei.

Verwenden Sie die folgenden Einstellbarkeitswerte in den `#$upgrade_from`-Anweisungen:

`ANYTIME`
: Wenn keine Einschränkungen der Aufrüstungszeitpunkte für die Ressource vorliegen. Die Ressource kann vollständig online sein.

WHEN_UNMONITORED
: Wenn bekannt ist, dass die Methoden `Update`, `Stop`, `Monitor_check` und `Postnet_stop` des neuen Ressourcentyps mit den Start-Methoden älterer Ressourcentypversionen kompatibel sind (`Prenet_stop` und `Start`) und wenn bekannt ist, dass die `Fini`-Methode der neuen Ressourcentypversion mit der `Init`-Methode der älteren Versionen kompatibel ist. Für dieses Szenario ist lediglich erforderlich, dass das Ressourcen-Monitor-Programm vor der Aufrüstung gestoppt wird.

WHEN_OFFLINE
: Wenn die Inkompatibilität der `Update`, `Stop`, `Monitor_check` bzw. `Postnet_stop`-Methoden der neuen Ressourcentypversion mit den Start-Methoden von älteren Ressourcentypversionen (`Prenet_stop` und `Start`) bekannt ist, jedoch ebenso bekannt ist, dass sie mit der `Init`-Methode älterer Versionen kompatibel sind, muss die Ressource während der Typaufrüstung offline sein.

WHEN_DISABLED
: Ähnlich wie `WHEN_OFFLINE`. Dieser Einstellbarkeitswert erzwingt jedoch die stärkere Bedingung, dass die Ressource deaktiviert sein muss.

WHEN_UNMANAGED
: Wenn die `Fini`-Methode der neuen Ressourcentypversion nicht mit der `Init`-Methode der älteren Versionen kompatibel ist. Dieser Einstellbarkeitswert erfordert, dass die vorhandene Ressourcengruppe in den nicht verwalteten Zustand versetzt wird, bevor Sie die Ressource aufrüsten können.

AT_CREATION
: Wenn Ressourcen nicht auf die neue Ressourcentypversion aufgerüstet werden können. Nur neue Ressourcen können mit der neuen Version erstellt werden.

  Die Einstellbarkeit von `AT_CREATION` bedeutet, dass der Ressourcentypentwickler die Migration einer vorhandenen Ressource zu dem neuen Typ verbieten kann. In diesem Fall muss der Systemverwalter die Ressource löschen und neu erstellen. Dies ist äquivalent mit der Deklaration, dass die Ressourcenversion nur zum Zeitpunkt der Erstellung eingestellt werden kann.

Migrieren einer Ressource zu einer anderen Version

Eine vorhandene Ressource nimmt die neue Ressourcentypversion an, wenn der Systemverwalter die `Type_version`-Eigenschaft der Ressource bearbeitet. Dabei werden die gleichen Konventionen wie bei der Bearbeitung anderer Ressourceneigenschaften beachtet, mit der Ausnahme, dass einige Informationen von der neuen Ressourcentypversion abgeleitet bzw. übernommen werden statt von der aktuellen Version.

- Ressourceneigenschaftsattribute für alle Eigenschaften wie `min`, `max`, `arraymin`, `arraymax`, default und tunability werden von der neuen Ressourcentypversion übernommen.
- Die Einstellbarkeit, die auf die `Type_version`-Eigenschaft angewendet wird, wird aus den `#$upgrade_from`-Anweisungen in der RTR-Datei und der `RT_Version`-Eigenschaft des Ressourcentyps für die vorhandene Ressource übernommen. Diese Einstellbarkeit weicht von der unter `property_attributes`(5) beschriebenen Einstellbarkeit ab.
- Die `Validate`-Methode für die neue Ressourcentypversion wird angewendet. So wird sichergestellt, dass die Eigenschaftsattribute für den neuen Ressourcentyp gültig sind. Wenn die vorhandenen Ressourceneigenschaftsattribute nicht für die Validierungsbedingungen der neuen Ressourcentypversion ausreichen, muss der Systemverwalter gültige Werte für diese Eigenschaften an der `scrgadm`-Befehlszeile eingeben. Dieser Fall kann eintreten, wenn die neuere Ressourcentypversion eine Eigenschaft verwendet, die in der früheren Version nicht deklariert war und die keinen Standardwert hat. Zudem kann er eintreten, wenn die vorhandene Ressource bereits über eine Eigenschaft verfügt, der ein Wert zugewiesen wurde, der für die neuere Ressourcentypversion ungültig ist.
- Die Deklaration von Ressourceneigenschaften, die in einer älteren Ressourcentypversion deklariert wurden, kann in der neueren Version aufgehoben werden. Wenn die Ressource zu einer neueren Version migriert, wird die Eigenschaft aus der Ressource gelöscht.

Hinweis – Die `Validate`-Methode kann die aktuelle `Type_version` der Ressource mithilfe von `scha_resource_get` abfragen, ebenso wie die neue `Type_version`, die an der `Validate`-Befehlszeile eingegeben wird. Daher kann `Validate` Aufrüstungen von nicht unterstützten Versionen ausschließen.

Auf- und Abrüsten eines Ressourcentyps

Weitere Informationen zum Aufrüsten bzw. Migrieren eines Ressourcentyps finden Sie im Abschnitt „Upgrading a Resource Type" in *Sun Cluster Data Services Planning and Administration Guide for Solaris OS*.

▼ So rüsten Sie einen Ressourcentyp auf

1. **In der Aufrüstungsdokumentation für den neuen Ressourcentyp finden Sie Informationen zu den Ressourcentypänderungen und Einschränkungen der Ressourceneinstellbarkeit.**

2. **Installieren Sie das Aufrüstungspaket für den Ressourcentyp auf allen Cluster-Knoten.**

   Die empfohlene Vorgehensweise für das Installieren von neuen Ressourcentyppaketen ist eine laufende Aufrüstung: `pkgadd` wird ausgeführt, während der Knoten im Nicht-Cluster-Modus gebootet wird.

   In einigen Fällen wäre es möglich, neue Ressourcentyppakete auf einem Knoten in Cluster-Modus zu installieren:

   - Wenn der Methodencode durch die Ressourcentyp-Paketinstallation keinen Änderungen unterliegt und nur der Monitor aktualisiert wird, muss die Überwachung aller Ressourcen dieses Typs während der Installation gestoppt werden.
   - Wenn die Ressourcentyp-Paketinstallation weder den Methoden- noch den Monitor-Code ändert, muss die Überwachung der Ressource während der Installation nicht gestoppt werden, da die Installation lediglich eine neue RTR-Datei auf der Platte ablegt.

3. **Registrieren Sie die neue Ressourcentypversion mit dem `scrgadm`-Befehl bzw. einem äquivalenten Befehl, womit auf die RTR-Datei der Aufrüstung verwiesen wird.**

   RGM erstellt einen neuen Ressourcentyp, dessen Name folgende Form hat:

   ```
   vendor_id.resource_type:version
   ```

4. **Wenn die Ressourcentypaufrüstung nur auf einer Untermenge der Knoten installiert ist, müssen Sie die `Installed_nodes`-Eigenschaft des neuen Ressourcentyps auf die Knoten einstellen, auf denen die Aufrüstung tatsächlich installiert ist.**

   Wenn eine Ressource den neuen Typ übernimmt (entweder, weil sie neu erstellt wird oder weil sie aufgerüstet wurde), ist es für RGM erforderlich, dass die Ressourcengruppen-`nodelist` eine Untermenge der `Installed_nodes`-Liste des Ressourcentyps ist.

   `scrgadm -c -t` *Ressourcentyp* `-h` *Liste_installierter_Knoten*

5. **Für jede Ressource vom nicht aufgerüsteten Typ, die zum aufgerüsteten Typ migriert werden soll, muss `scswitch` aufgerufen werden, um den Zustand der Ressource bzw. deren Ressourcengruppe in denjenigen Zustand zu ändern, der von der Aufrüstungsdokumentation vorgeschrieben wird.**

6. **Bearbeiten Sie jede Ressource des nicht aufgerüsteten Typs, die zum aufgerüsteten Typ migriert werden soll, indem Sie die `Type_version`-Eigenschaft in die neue Version ändern.**

   `scrgadm -c -j` *Ressource* `-y Type_version=`*neue_Version*

   Bei Bedarf werden mit dem gleichen Befehl weitere Eigenschaften derselben Ressource auf die richtigen Werte eingestellt.

7. **Der vorherige Zustand der Ressource bzw. der Ressourcengruppe wird durch Rückgängigmachen des in Schritt 5 aufgerufenen Befehls wiederhergestellt.**

▼ So rüsten Sie eine Ressource auf eine ältere Version des Ressourcentyps ab

Sie können eine Ressource auf eine ältere Version ihres Ressourcentyps abrüsten. Die Bedingungen, unter denen Sie eine Ressource auf eine ältere Version des Ressourcentyps abrüsten können, sind stärker einschränkend als diejenigen für die Aufrüstung auf eine neuere Version des Ressourcentyps. Zunächst müssen Sie die Ressourcengruppe in einen nicht verwalteten Zustand versetzen. Außerdem können Sie eine Ressource nur zu einer aufrüstungsfähigen Version des Ressourcentyps abrüsten. Aufrüstungsfähige Versionen werden mithilfe des `scrgadm -p`-Befehls identifiziert. In der Ausgabe enthalten aufrüstungsfähige Versionen das Suffix `:`*Version*.

1. **Bringen Sie die Ressourcengruppe mit der abzurüstenden Ressource offline.**

   `scswitch -F -g` *Ressourcengruppe*

2. **Deaktivieren Sie die Ressource, die abgerüstet werden soll, sowie alle weiteren Ressourcen in der Ressourcengruppe.**

   `scswitch -n -j` *aufzurüstende_Ressource*
   `scswitch -n -j` *Ressource1*
   `scswitch -n -j` *Ressource2*
   `scswitch -n -j` *Ressource3*
   `...`

   Hinweis – Deaktivieren Sie die Ressourcen in der Reihenfolge ihrer Abhängigkeit, wobei Sie mit den am stärksten abhängigen Ressourcen (Anwendungsressourcen) beginnen und mit den am wenigsten abhängigen Ressourcen (Netzwerkadressressourcen) enden.

   ---

3. **Beenden Sie die Verwaltung der Ressourcengruppe.**

   `scswitch -u -g` *Ressourcengruppe*

4. **Ist die alte Version des Ressourcentyps, auf den Sie abrüsten möchten, noch im Cluster registriert?**

   - Wenn ja, gehen Sie zum nächsten Schritt.
   - Wenn nicht, registrieren Sie die gewünschte alte Version erneut.

     `scrgadm -a -t` *Ressourcentypname*

5. **Rüsten Sie die Ressource ab, indem Sie die alte Version für `Type_version` angeben.**

   `scrgadm -c -j` *abzurüstende_Ressource* **`-y Type_version=`***alte_Version*

   Bei Bedarf werden mit dem gleichen Befehl weitere Eigenschaften derselben Ressource auf die richtigen Werte eingestellt.

6. **Bringen Sie die Ressourcengruppe mit der abgerüsteten Ressource in einen verwalteten Zustand, aktivieren Sie alle Ressourcen, und bringen Sie die Gruppe online.**

   `scswitch -Z -g` *Ressourcengruppe*

Standardeigenschaftswerte

RGM speichert alle Ressourcen so, dass jede Eigenschaft, die nicht explizit vom Systemverwalter eingestellt (und mit einem Standardwert versehen) wurde, nicht im Ressourceneintrag im CCR (Cluster Configuration Repository, Cluster-Konfigurations-Repository) gespeichert wird. RGM ruft den Standardwert für eine fehlende Ressourceneigenschaft aus dem Ressourcentyp ab. Wenn darin kein Wert definiert ist, wird ein systemdefinierter Standardwert verwendet, wenn eine Ressource vom CCR gelesen wird. Mittels dieser Methode der Eigenschaftsspeicherung kann ein aufgerüsteter Ressourcentyp neue Eigenschaften bzw. neue Standardwerte für vorhandene Eigenschaften definieren.

Wenn Ressourceneigenschaften bearbeitet werden, speichert RGM diejenigen Eigenschaften im CCR, die im Edit-Befehl angegeben wurden.

Wenn für eine aufgerüstete Version des Ressourcentyps ein neuer Standardwert für eine mit Standardwert versehene Eigenschaft deklariert wird, dann wird der neue Standardwert an die vorhandenen Ressourcen vererbt, auch wenn die Einstellbarkeit der Ressource nur als `AT_CREATION` oder `WHEN_DISABLED` deklariert wurde. Wenn die Anwendung des neuen Standardwerts bedeuten würde, dass eine Methode wie `Stop`, `Postnet_stop` oder `Fini` fehlschlägt, muss die Person, die den Ressourcentyp implementiert, den Zustand der Ressource bei der Aufrüstung entsprechend einschränken. Dies geschieht durch Einschränken der Einstellbarkeit der `Type_version`-Eigenschaft.

Die `Validate`-Methode der neuen Ressourcentypversion kann prüfen, ob die vorhandenen Eigenschaftsattribute geeignet sind. Wenn dies nicht der Fall ist, kann der Systemverwalter die Eigenschaften einer vorhandenen Ressource auf die geeigneten Werte einstellen, und zwar mit dem gleichen Befehl, mit dem die `Type_version`-Eigenschaft bearbeitet wird, um die Ressource auf die neue Ressourcentypversion aufzurüsten.

Hinweis – Ressourcen, die unter Sun Cluster 3.0 erstellt wurden, übernehmen nicht automatisch neue Standardeigenschaftswerte vom Ressourcentyp, wenn sie zu einer höheren Version migriert werden, da ihre Standardeigenschaften im CCR gespeichert sind.

Ressourcentyp-Entwicklerdokumentation

Der Ressourcentypentwickler muss für eine neue Ressource Dokumentation bereitstellen, die folgende Informationen enthält:

- Beschreibung aller hinzugefügten, geänderten oder gelöschten Eigenschaften,
- Beschreibung, wie die Eigenschaften den neuen Anforderungen angepasst werden können,
- Angabe der Einstellbarkeitseinschränkungen für die Ressourcen,
- Hervorheben aller neuen Standardeigenschaftsattribute,
- Hinweis an den Systemverwalter, dass sich die vorhandenen Ressourceneigenschaften mit dem gleichen Befehl auf die geeigneten Werte einstellen lassen, der für die Bearbeitung der `Type_version`-Eigenschaft verwendet wird, um die Ressource auf die neue Ressourcentypversion aufzurüsten.

Ressourcentypnamens- und Ressourcentyp-Monitor-Implementierungen

Sie können einen aufrüstungsfähigen Ressourcentyp unter Sun Cluster 3.0 registrieren. Sein Name wird jedoch im CCR ohne Versionssuffix gespeichert. Um korrekt sowohl in Sun Cluster 3.0 und Sun Cluster 3.1 bzw. höheren Versionen zu laufen, muss der Monitor für diesen Ressourcentyp mit beiden Namenskonventionen umgehen können:

```
vendor_id.resource_name:version
vendor_id.resource_name
```

Der Monitor-Code kann den richtigen zu verwendenden Namen bestimmen, indem ein dem Folgenden entsprechender Befehl ausgeführt wird:

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

Dann werden die Ausgabewerte mit `vers` verglichen. Nur einer dieser Befehle ist für einen bestimmten Wert von `vers` erfolgreich, da dieselbe Version eines Ressourcentyps nicht zweimal unter verschiedenen Namen registriert werden kann.

Anwendungsaufrüstungen

Die Aufrüstung von Anwendungscode weicht von der Aufrüstung des Agenten-Codes ab, auch wenn sich einige der Schritte gleichen. Eine Anwendungsaufrüstung kann, muss jedoch nicht, mit einer Ressourcentypaufrüstung kombiniert werden.

Beispiele für Ressourcentypaufrüstungen

Diese Beispiele zeigen mehrere unterschiedliche Szenarios für die Ressourcentypinstallation und -aufrüstung. Die Einstellbarkeits- und Paketinformationen wurden anhand der Änderungstypen ausgewählt, die an der Ressourcentypimplementierung vorgenommen werden. Die Einstellbarkeit bezieht sich auf die Migration der Ressource zum neuen Ressourcentyp.

Alle Beispiele gehen von folgenden Annahmen aus:

- Der Ressourcentyp wird in einem Solaris-Paket geliefert. Siehe `pkgadd`(1M) und `pkgrm`(1M).
- Es ist nur eine vorherige Version des Ressourcentyps vorhanden, und daher enthält die neue RTR-Datei nur eine `#$upgrade_from`-Anweisung.
- Das Installationsverfahren entfernt bzw. überschreibt keine Methoden, wenn die Möglichkeit besteht, dass RGM diese Methoden während des Entfernens von der Platte aufruft.
- Neue Methoden sind mit den alten Methoden kompatibel, sofern nicht anders angegeben.
- Ressourcen und Ressourcengruppen werden vor der Installation bzw. Migration mithilfe des Befehls `scswitch`(1M) oder eines äquivalenten Befehls in den erforderlichen Zustand versetzt. Das folgende Beispiel zeigt, wie die Ressourcengruppe in einen nicht verwalteten Zustand versetzt wird:

  ```
  scswitch -M -n -j Ressource
  scswitch -n -j Ressource
  scswitch -F -g Ressourcengruppe
  scswitch -u -g Ressourcengruppe
  ```

- Der Ressourcentyp wird mit folgendem Befehl registriert:

  ```
  scrgadm -a -t Ressourcentyp -f Pfad_zur_RTR_Datei
  ```

- Eine Ressource wird mit folgendem Befehl migriert:

  ```
  scrgadm -c -j Ressource -y Type_version=Version \
             -y Eigenschaft=Wert \
  ```

`-x` *Eigenschaft=Wert* `...`

- Ressourcen und Ressourcengruppen werden nach der Migration wieder in ihren vorherigen Zustand zurückversetzt. Dies geschieht über den Befehl `scswitch`(1M) bzw. einen äquivalenten Befehl.

  ```
  scswitch -M -e -j Ressource
  scswitch -e -j Ressource
  scswitch -o -g Ressourcengruppe
  scswitch -Z -g Ressourcengruppe
  ```

Der Ressourcentypentwickler muss möglicherweise stärker eingeschränkte Einstellbarkeitswerte angeben als die in diesen Beispielen verwendeten Werte. Die Einstellbarkeitswerte sind abhängig von den genauen Änderungen, die an der Ressourcentypimplementierung vorgenommen werden. Ebenso kann sich der Ressourcentypentwickler dafür entscheiden, ein anderes Paketschema als das in diesen Beispielen eingesetzte Solaris-Paket zu verwenden.

TABELLE 3–1 Beispiele für das Aufrüsten eines Ressourcentyps

Änderungstyp	Einstellbarkeit	Paket	Verfahren
Eigenschaftsänderungen werden nur in der RTR-Datei vorgenommen.	`ANYTIME`	Liefern Sie nur die neue RTR-Datei.	Führen Sie `pkgadd` für die neue RTR-Datei auf allen Knoten aus. Registrieren Sie den neuen Ressourcentyp. Migrieren Sie die Ressource.
Die Methoden werden aktualisiert.	`ANYTIME`	Legen Sie die aktualisierten Methoden unter einem anderen Pfad als die alten Methoden ab.	Führen Sie `pkgadd` für die aktualisierten Methoden auf allen Knoten aus. Registrieren Sie den neuen Ressourcentyp. Migrieren Sie die Ressource.
Neues Monitor-Programm.	`WHEN_UNMONITORED`	Überschreiben Sie nur die vorherige Version des Monitors.	Deaktivieren Sie die Überwachung. Führen Sie `pkgadd` für das neue Monitor-Programm auf allen Knoten aus. Registrieren Sie den neuen Ressourcentyp. Migrieren Sie die Ressource. Aktivieren Sie die Überwachung.

TABELLE 3–1 Beispiele für das Aufrüsten eines Ressourcentyps *(Fortsetzung)*

Änderungstyp	Einstellbarkeit	Paket	Verfahren
Die Methoden werden aktualisiert. Die neuen `Update`/`Stop`-Methoden sind mit den alten `Start`-Methoden inkompatibel.	`WHEN_OFFLINE`	Legen Sie die aktualisierten Methoden unter einem anderen Pfad als die alten Methoden ab.	Führen Sie `pkgadd` für die aktualisierten Methoden auf allen Knoten aus. Registrieren Sie den neuen Ressourcentyp. Nehmen Sie die Ressource offline. Migrieren Sie die Ressource. Bringen Sie die Ressource online.
Die Methoden werden aktualisiert und die neuen Eigenschaften der RTR-Datei hinzugefügt. Die neuen Methoden benötigen neue Eigenschaften. (Ziel ist es, der betroffenen Ressourcengruppe das Online-bleiben zu ermöglichen, gleichzeitig jedoch zu verhindern, dass die Ressource online gebracht wird, wenn die Ressourcengruppe auf einem Knoten von einem Offline- in einen Online-Zustand übergeht.)	`WHEN_DISABLED`	Überschreiben Sie die vorherigen Versionen der Methoden.	Deaktivieren Sie die Ressource. Für jeden Knoten: ■ Nehmen Sie den Knoten aus dem Cluster. ■ Führen Sie `pkgrm`/`pkgadd` für die zu aktualisierenden Methoden aus. ■ Fügen Sie den Knoten wieder dem Cluster hinzu. Registrieren Sie den neuen Ressourcentyp. Migrieren Sie die Ressource. Aktivieren Sie die Ressource.

TABELLE 3–1 Beispiele für das Aufrüsten eines Ressourcentyps *(Fortsetzung)*

Änderungstyp	Einstellbarkeit	Paket	Verfahren
Die Methoden werden aktualisiert und die neuen Eigenschaften der RTR-Datei hinzugefügt. Die neuen Methoden benötigen keine neue Eigenschaften.	`ANYTIME`	Überschreiben Sie die vorherigen Versionen der Methoden.	Für jeden Knoten: ■ Nehmen Sie den Knoten aus dem Cluster. ■ Führen Sie `pkgrm`/`pkgadd` für die zu aktualisierenden Methoden aus. ■ Fügen Sie den Knoten wieder dem Cluster hinzu. Während dieses Vorgangs ruft RGM die neuen Methoden auf, auch wenn die Migration, durch die die neuen Eigenschaften konfiguriert werden, noch nicht ausgeführt wurde. Die neuen Methoden müssen unbedingt in der Lage sein, ohne die neuen Eigenschaften zu funktionieren. Registrieren Sie den neuen Ressourcentyp. Migrieren Sie die Ressource.
Die Methoden werden aktualisiert. Die neue `Fini`-Methode ist inkompatibel mit der alten `Init`-Methode.	`WHEN_UNMANAGED`	Legen Sie die aktualisierten Methoden unter einem anderen Pfad als die alten Methoden ab.	Bringen Sie die betreffende Ressourcengruppe in einen unverwalteten Zustand. Führen Sie `pkgadd` für die aktualisierten Methoden auf allen Knoten aus. Registrieren Sie den Ressourcentyp. Migrieren Sie die Ressource. Versetzen Sie die betreffende Ressourcengruppe in einen verwalteten Zustand.

TABELLE 3–1 Beispiele für das Aufrüsten eines Ressourcentyps *(Fortsetzung)*

Änderungstyp	Einstellbarkeit	Paket	Verfahren
Die Methoden werden aktualisiert. An der RTR-Datei werden keine Änderungen vorgenommen.	Nicht zutreffend. An der RTR-Datei werden keine Änderungen vorgenommen.	Überschreiben Sie die vorherigen Versionen der Methoden.	Für jeden Knoten: ■ Nehmen Sie den Knoten aus dem Cluster. ■ Führen Sie `pkgadd` für die aktualisierten Methoden aus. ■ Fügen Sie den Knoten wieder dem Cluster hinzu Da an der RTR-Datei nichts geändert wurde, muss die Ressource nicht registriert oder migriert werden.

Installationsanforderungen für Ressourcentyppakete

Bezüglich der Installation neuer Ressourcentyppakete gibt es zwei Anforderungen:

- Wenn ein neuer Ressourcentyp registriert wird, muss der Zugriff auf dessen RTR-Datei auf der Platte möglich sein.
- Wenn eine Ressource des neuen Typs erstellt wird, müssen sich alle deklarierten Methodenpfadnamen und das Monitor-Programm für den neuen Typ auf der Platte befinden und ausführbar sein. Die alten Methoden- und Monitor-Programme müssen so lange an ihrem Platz bleiben, wie die Ressource in Verwendung ist.

  Bei der Entscheidung über das am besten geeignete Paket muss der Ressourcentypimplementierer folgende Punkte in Betracht ziehen:

  - Wird die RTR-Datei geändert?
  - Werden der Standardwert oder die Einstellbarkeit einer Eigenschaft geändert?
  - Wird der `min`- oder `max`-Wert einer Eigenschaft geändert?
  - Werden bei der Aufrüstung Eigenschaften hinzugefügt oder gelöscht?
  - Wird der Methodencode geändert?
  - Wird der Monitor-Code geändert?
  - Sind die neuen Methoden- bzw. Monitor-Codes mit der vorherigen Version kompatibel?

Erforderliche Informationen vor dem Ändern der RTR-Datei

Einige Ressourcentypaufrüstungen sind nicht mit neuem Methoden- oder Monitor-Code verbunden. So kann zum Beispiel bei einer Ressourcentypänderung nur der Standardwert oder die Einstellbarkeit einer Ressourceneigenschaft geändert werden. Da der Methodencode nicht geändert wird, ist die einzige Anforderung für die Installation der Aufrüstung ein gültiger Pfadname zu einer lesbaren RTR-Datei.

Wenn der alte Ressourcentyp nicht erneut registriert werden muss, kann die neue RTR-Datei die vorherige Version überschreiben. Andernfalls kann die neue RTR-Datei unter einem neuen Pfadnamen abgelegt werden.

Wenn die Aufrüstung den Standardwert oder die Einstellbarkeit einer Eigenschaft ändert, kann die `Validate`-Methode für die neue Version beim Migrieren feststellen, ob die vorhandenen Eigenschaftsattribute für den neuen Ressourcentyp gültig sind. Wenn die Aufrüstung das `min`, `max`- oder `type`-Attribut einer Eigenschaft ändert, validiert der `scrgadm`-Befehl zum Zeitpunkt der Migration automatisch diese Einschränkungen.

Die Aufrüstungsdokumentation muss alle neuen Standardeigenschaftsattribute hervorheben. Die Dokumentation muss den Systemverwalter darüber informieren, dass sich die vorhandenen Ressourceneigenschaften mit dem gleichen Befehl auf die geeigneten Werte einstellen lassen, der für die Bearbeitung der `Type_version`-Eigenschaft verwendet wird, um die Ressource auf die neue Ressourcentypversion aufzurüsten.

Wenn bei der Aufrüstung Eigenschaften hinzugefügt oder gelöscht werden, müssen wahrscheinlich auch einige Rückmeldemethoden oder Monitor-Code geändert werden.

Ändern von Monitor-Code

Wenn der Monitor-Code die einzige Änderung an dem aktualisierten Ressourcentyp ist, kann die Paketinstallation die Monitor-Binärdateien überschreiben. Die Dokumentation muss den Systemverwalter anweisen, die Überwachung vor Installation des neuen Pakets aufzuheben.

Ändern von Methodencode

Wenn der Methodencode die einzige Änderung an dem aktualisierten Ressourcentyp ist, muss festgestellt werden, ob der neue Methodencode mit der vorherigen Version kompatibel ist. Davon hängt ab, ob der neue Methodencode unter einem neuen Pfadnamen abgelegt werden muss oder ob die alten Methoden überschrieben werden können.

Wenn die neuen `Stop`-, `Postnet_stop`- und `Fini`-Methoden (falls deklariert) auf die Ressourcen angewendet werden können, die mit den alten Versionen der `Start`-, `Prenet_stop`- oder `Init`-Methoden initialisiert bzw. gestartet wurden, können die alten Methoden mit den neuen Methoden überschrieben werden.

Wenn der neue Methodencode nicht mit der vorherigen Version kompatibel ist, müssen die Ressourcen, welche die alten Methodenversionen verwenden, gestoppt bzw. dekonfiguriert werden, bevor sie zum aufgerüsteten Ressourcentyp migriert werden können. Wenn die neuen Methoden die alten Methoden überschreiben, müssen möglicherweise alle Ressourcen des Typs heruntergefahren und gegebenenfalls in einen nicht verwalteten Zustand versetzt werden, bevor die Ressourcentypaufrüstung ausgeführt wird. Wenn die neuen Methoden getrennt von den alten Methoden gespeichert werden und auf beide gleichzeitig zugegriffen werden kann, dann ist es auch ohne Abwärtskompatibilität möglich, die neue Ressourcentypversion zu installieren und die Ressourcen nacheinander aufzurüsten.

Auch wenn die neuen Methoden abwärtskompatibel sind, kann es erforderlich sein, jeweils nur eine Ressource für die Verwendung der neuen Methoden aufzurüsten, während die restlichen Ressourcen weiterhin die alten Methoden verwenden. Auch in diesem Fall müssen die neuen Methoden in einem getrennten Verzeichnis gespeichert werden, anstatt die alten Methoden zu überschreiben.

Ein Vorteil beim Speichern der Methoden pro Ressourcentyp in einem eigenen Verzeichnis ist, dass Ressourcen leicht zur älteren Ressourcentypversion zurückgewechselt werden können, wenn mit der neuen Version ein Problem auftritt.

Ein Ansatz für Pakete besteht darin, alle früheren Versionen, die noch unterstützt werden, in das Paket mit aufzunehmen. So kann die neue Paketversion die ältere Version ersetzen, ohne die alten Methodenpfade zu überschreiben oder zu löschen. Der Ressourcentypentwickler muss entscheiden, wie viele frühere Versionen unterstützt werden sollen.

Hinweis – Es wird davon abgeraten, Methoden oder `pkgrm`-/`pkgadd`-Methoden auf einem Knoten zu überschreiben, der sich aktuell im Cluster befindet. Falls RGM eine Methode aufruft, während auf die Methode auf der Platte nicht zugegriffen werden kann, kann dies zu unerwarteten Ergebnissen führen. Auch das Entfernen oder Ersetzen der Binärdatei einer laufenden Methode kann zu unerwarteten Ergebnissen führen.

KAPITEL 4

Ressourcenverwaltungs-API-Referenz

Dieses Kapitel erläutert die Zugriffsfunktionen und Rückmeldemethoden, aus denen sich die Ressourcenverwaltungs-API (RMAPI) zusammensetzt. Es listet alle Funktionen und Methoden auf und beschreibt sie kurz. Als maßgebliche Referenz für diese Funktionen und Methoden wird jedoch auf die RMAPI-Online-Dokumentation verwiesen.

In diesem Kapitel finden Sie folgende Informationen:

- „RMAPI-Zugriffsmethoden" auf Seite 80 – in Form von Shell-Skript-Befehlen und C-Funktionen
  - `scha_resource_get`(1HA), `scha_resource_close`(3HA), `scha_resource_get`(3HA), `scha_resource_open`(3HA)
  - `scha_resource_setstatus`(1HA), `scha_resource_setstatus`(3HA)
  - `scha_resourcetype_get`(1HA), `scha_resourcetype_close`(3HA), `scha_resourcetype_get`(3HA), `scha_resourcetype_open`(3HA)
  - `scha_resourcegroup_get`(1HA), `scha_resourcegroup_get`(3HA), `scha_resourcegroup_close`(3HA), `scha_resourcegroup_open`(3HA)
  - `scha_control`(1HA), `scha_control`(3HA).
  - `scha_cluster_get`(1HA), `scha_cluster_close`(3HA), `scha_cluster_get`(3HA), `scha_cluster_open`(3HA)
  - `scha_cluster_getlogfacility`(3HA)
  - `scha_cluster_getnodename`(3HA)
  - `scha_strerror`(3HA)
- „RMAPI-Rückmeldemethoden" auf Seite 85 – beschrieben in der Online-Dokumentation unter `rt_callbacks`(1HA).
  - `Start`
  - `Stop`
  - `Init`
  - `Fini`
  - `Boot`

- `Prenet_start`
- `Postnet_stop`
- `Monitor_start`
- `Monitor_stop`
- `Monitor_check`
- `Update`
- `ValidateS`

RMAPI-Zugriffsmethoden

Die API stellt Funktionen bereit, mit denen auf Ressourcen-, Ressourcentyp- und Ressourcengruppeneigenschaften sowie auf anderweitige Cluster-Informationen zugegriffen werden kann. Diese Funktionen sind sowohl in Form von Shell-Befehlen als auch als C-Funktionen vorhanden. So können die Ressourcentyphersteller Steuerprogramme als Shell-Skripts oder als C-Programme implementieren.

RMAPI-Shell-Befehle

Shell-Befehle werden in Shell-Skript-Implementierungen der Rückmeldemethoden für Ressourcentypen verwendet, die von RGM des Clusters gesteuerte Dienste darstellen. Diese Befehle können zu folgenden Zwecken eingesetzt werden:

- Zugriff auf Informationen über Ressourcen, Ressourcentypen, Ressourcengruppen und Cluster,
- Verwendung zusammen mit einem Monitor, um die `Status`- und `Status_msg`-Eigenschaften einer Ressource einzustellen,
- Anforderung des Neustarts bzw. der Verschiebung einer Ressourcengruppe,

---

Hinweis – Dieser Abschnitt enthält kurze Beschreibungen der Shell-Befehle. Als maßgebliche Referenz für die Shell-Befehle wird jedoch auf die einzelnen Seiten der 1HA-Online-Dokumentation verwiesen. Für jeden Befehl ist eine gleichnamige Seite in der Online-Dokumentation vorhanden, sofern nicht anders angegeben.

RMAPI-Ressourcenbefehle

Mit diesen Befehlen können Sie auf Informationen über eine Ressource zugreifen oder die `Status`- und `Status_msg`-Eigenschaften einer Ressource einstellen.

`scha_resource_get`
: Greift auf Informationen über eine Ressource bzw. einen Ressourcentyp unter RGM-Steuerung zu. Stellt die gleichen Informationen wie die `scha_resource_get()`-Funktion bereit.

`scha_resource_setstatus`
Stellt die `Status`- und `Status_msg`-Eigenschaften einer Ressource unter RGM-Steuerung ein. Wird vom Ressourcen-Monitor verwendet, um den vom Monitor wahrgenommenen Ressourcenzustand anzugeben. Bietet die gleiche Funktionalität wie die C-Funktion `scha_resource_setstatus()`.

Hinweis – `scha_resource_setstatus()` ist zwar für einen Ressourcen-Monitor besonders nützlich, kann jedoch von jedem beliebigen Programm aufgerufen werden.

Ressourcentypbefehl

Dieser Befehl greift auf Informationen über einen bei RGM registrierten Ressourcentyp zu.

`scha_resourcetype_get`
Dieser Befehl bietet die gleiche Funktionalität wie die C-Funktion `scha_resourcetype_get()`.

Ressourcengruppenbefehle

Mit diesen Befehlen können Sie auf Informationen über eine Ressourcengruppe zugreifen bzw. diese neu starten.

`scha_resourcegroup_get`
Greift auf Informationen über eine Ressourcengruppe unter RGM-Steuerung zu. Dieser Befehl bietet die gleiche Funktionalität wie die C-Funktion `scha_resourcetype_get()`.

`scha_control`
Fordert den Neustart einer Ressourcengruppe unter RGM-Steuerung bzw. die Verschiebung zu einem anderen Knoten an. Dieser Befehl bietet die gleiche Funktionalität wie die C-Funktion `scha_control()`.

Cluster-Befehl

Dieser Befehl greift auf Informationen über einen Cluster zu, wie Knotennamen, IDs, Zustand, Cluster-Name, Ressourcengruppen usw.

`scha_cluster_get`
Dieser Befehl stellt die gleichen Informationen wie die C-Funktion `scha_cluster_get()` bereit.

C-Funktionen

C-Funktionen werden in C-Programmimplementierungen der Rückmeldemethoden für Ressourcentypen eingesetzt, die Dienste unter RGM-Steuerung des Clusters darstellen. Sie können diese Funktionen zu folgenden Zwecken einsetzen:

- Zugriff auf Informationen über Ressourcen, Ressourcentypen, Ressourcengruppen und Cluster,
- Verwendung zusammen mit einem Monitor, um die `Status`- und `Status_msg`-Eigenschaften einer Ressource einzustellen,
- Anforderung des Neustarts bzw. der Verschiebung einer Ressourcengruppe,
- Konvertieren eines Fehlercodes in die entsprechende Fehlermeldung.

Hinweis – Dieser Abschnitt enthält zwar kurze Beschreibungen der C-Funktionen. Als maßgebliche Referenz für diese Funktionen wird jedoch auf die einzelnen Seiten der `(3HA)`-Online-Dokumentation verwiesen. Für jede Funktion ist eine gleichnamige Seite in der Online-Dokumentation vorhanden, sofern nicht anders angegeben. Informationen zu Ausgabeargumenten und Rückgabecodes der C-Funktionen finden Sie in der Online-Dokumentation unter `scha_calls`(3HA).

Ressourcenfunktionen

Diese Funktionen greifen auf Informationen über eine von RGM verwaltete Ressource zu bzw. geben den vom Monitor festgestellten Zustand der Ressource an.

`scha_resource_open()`, `scha_resource_get ()` und `scha_resource_close ()`

Zusammen greifen diese Funktionen auf Informationen über eine von RGM verwaltete Ressource zu. Die `scha_resource_open()`-Funktion initialisiert den Zugriff auf eine Ressource und gibt ein Handle für `scha_resource_get()` zurück, womit auf die Ressourceninformationen zugegriffen wird. Die `scha_resource_close()`-Funktion invalidiert das Handle und gibt den für die Rückgabewerte von `scha_resource_get()` zugewiesenen Speicherplatz frei.

Eine Ressource kann durch eine Cluster-Rekonfiguration oder einen Verwaltungsbefehl geändert werden, nachdem `scha_resource_open()` das Handle für die Ressource zurückgegeben hat. In diesem Fall können die von `scha_resource_get()` über das Handle abgerufenen Informationen falsch sein. Im Fall einer Cluster-Rekonfiguration oder einer Verwaltungsaktion an einer Ressource gibt RGM den Fehlercode `scha_err_seqid` an `scha_resource_get ()` zurück, um anzugeben, dass sich die Ressourceninformationen geändert haben könnten. Diese Meldung gibt keinen schwerwiegenden Fehler an; die Funktion gibt Erfolg zurück. Sie können die Meldung ignorieren und die zurückgegebenen Informationen akzeptieren, oder das aktuelle Handle schließen und ein neues Handle zum Zugreifen auf Ressurceninformationen öffnen.

Eine gemeinsame Online-Dokumentationsseite beschreibt diese drei Funktionen. Auf diese Seite können Sie über jede der einzelnen Funktionen, `scha_resource_open`(3HA), `scha_resource_get`(3HA) oder `scha_resource_close`(3HA) zugreifen.

`scha_resource_setstatus()`
Stellt die `Status`- und `Status_msg`-Eigenschaften einer Ressource unter RGM-Steuerung ein. Der Ressourcen-Monitor verwendet diese Funktion, um den Ressourcenzustand anzugeben.

Hinweis – `scha_resource_setstatus()` ist zwar für einen Ressourcen-Monitor besonders nützlich, kann jedoch von jedem beliebigen Programm aufgerufen werden.

Ressourcentypfunktionen

Diese Funktionen greifen gemeinsam auf Informationen über einen bei RGM registrierten Ressourcentyp zu.

`scha_resourcetype_open()`, `scha_resourcetype_get()`, `scha_resourcetype_close()`
Die Funktion `scha_resourcetype_open()` initialisiert den Zugriff auf eine Ressource und gibt ein Handle für `scha_resourcetype_get()` zurück, womit auf die Ressourcentypinformationen zugegriffen wird. Die Funktion `scha_resourcetype_close()` invalidiert das Handle und gibt den für die Rückgabewerte von `scha_resourcetype_get()` zugewiesenen Speicherplatz frei.

Ein Ressourcentyp kann durch eine Cluster-Rekonfiguration oder einen Verwaltungsbefehl geändert werden, nachdem `scha_resourcetype_open()` das Ressourcentyp-Handle zurückgegeben hat. In diesem Fall können die von `scha_resourcetype_get()` über das Handle abgerufenen Informationen falsch sein. Im Fall einer Cluster-Rekonfiguration oder einer Verwaltungsaktion an einem Ressourcentyp gibt RGM den Fehlercode `scha_err_seqid` an `scha_resourcetype_get()` zurück, um anzugeben, dass sich die Ressourcentypinformationen geändert haben könnten. Diese Meldung gibt keinen schwerwiegenden Fehler an; die Funktion gibt Erfolg zurück. Sie können die Meldung ignorieren und die zurückgegebenen Informationen akzeptieren, oder das aktuelle Handle schließen und ein neues Handle zum Zugreifen auf Ressourcentypinformationen öffnen.

Eine gemeinsame Online-Dokumentationsseite beschreibt diese drei Funktionen. Auf diese Seite können Sie über jede der einzelnen Funktionen, `scha_resourcetype_open`(3HA), `scha_resourcetype_get`(3HA) oder `scha_resourcetype_close`(3HA) zugreifen.

Ressourcengruppenfunktionen

Mit diesen Funktionen können Sie auf Informationen über eine Ressourcengruppe zugreifen bzw. sie neu starten.

`scha_resourcegroup_open`(3HA), `scha_resourcegroup_get`(3HA) und `scha_resourcegroup_close`(3HA)
: Zusammen greifen diese Funktionen auf eine von RGM verwaltete Ressourcengruppe zu. Die Funktion `scha_resourcegroup_open()` initialisiert den Zugriff auf eine Ressourcengruppe und gibt ein Handle für `scha_resourcegroup_get()` zurück, womit auf die Ressourcengruppeninformationen zugegriffen wird. Die Funktion `scha_resourcegroup_close()` invalidiert das Handle und gibt den für die Rückgabewerte von `scha_resourcegroup_get()` zugewiesenen Speicherplatz frei.

  Eine Ressourcengruppe kann durch eine Cluster-Rekonfiguration oder eine Verwaltungsaktion geändert werden, nachdem `scha_resourcegroup_open()` das Ressourcengruppen-Handle zurückgegeben hat. In diesem Fall können die von `scha_resourcegroup_get()` über das Handle abgerufenen Informationen falsch sein. Im Fall einer Cluster-Rekonfiguration oder einer Verwaltungsaktion an einer Ressourcengruppe gibt RGM den Fehlercode `scha_err_seqid` an `scha_resourcegroup_get()` zurück, um anzugeben, dass sich die Ressourcengruppeninformationen geändert haben könnten. Diese Meldung gibt keinen schwerwiegenden Fehler an; die Funktion gibt Erfolg zurück. Sie können die Meldung ignorieren und die zurückgegebenen Informationen akzeptieren, oder das aktuelle Handle schließen und ein neues Handle zum Zugreifen auf Ressourcengruppeninformationen öffnen.

`scha_control`(3HA)
: Fordert den Neustart einer Ressourcengruppe unter RGM-Steuerung bzw. die Verschiebung zu einem anderen Knoten an.

Cluster-Funktionen

Diese Funktionen greifen auf Informationen über einen Cluster zu bzw. geben diese zurück.

`scha_cluster_open`(3HA), `scha_cluster_get`(3HA), `scha_cluster_close`(3HA)
: Diese Funktionen greifen gemeinsam auf Informationen über einen Cluster zu, wie Knotennamen, IDs, Zustand, Cluster-Name, Ressourcengruppe usw.

  Ein Cluster kann — durch Rekonfiguration oder eine Verwaltungsaktion — geändert werden, nachdem `scha_cluster_open()` das Cluster-Handle zurückgegeben hat. In diesem Fall können die von `scha_cluster_get()` über das Handle abgerufenen Informationen falsch sein. Im Fall einer Rekonfiguration oder Verwaltungsaktion auf einem Cluster gibt RGM den Fehlercode `scha_err_seqid` an `scha_cluster_get()` zurück, um anzugeben, dass sich

die Cluster-Informationen geändert haben könnten. Diese Meldung gibt keinen schwerwiegenden Fehler an; die Funktion gibt Erfolg zurück. Sie können die Meldung ignorieren und die zurückgegebenen Informationen akzeptieren, oder das aktuelle Handle schließen und ein neues Handle zum Zugreifen auf Cluster-Informationen öffnen.

`scha_cluster_getlogfacility`(3HA)
Gibt die Nummer der Systemprotokollierung zurück, die als Cluster-Protokoll verwendet wird. Verwendet den von der Solaris-Funktion `syslog()` zurückgegebenen Wert zum Aufzeichnen von Ereignissen und Statusmeldungen im Cluster-Protokoll.

`scha_cluster_getnodename`(3HA)
Gibt den Namen des Cluster-Knotens zurück, auf dem die Funktion aufgerufen wird.

Dienstprogrammfunktion

Diese Funktion konvertiert einen Fehlercode in eine Fehlermeldung.

`scha_strerror`(3HA)
Überträgt einen Fehlercode — zurückgegeben von einer der `scha_`-Funktionen — in die entsprechende Fehlermeldung. Verwenden Sie diese Funktion mit `logger`, um Meldungen im Systemprotokoll (`syslog`) zu protokollieren.

RMAPI-Rückmeldemethoden

Rückmeldemethoden sind die von der API bereitgestellten Schlüsselelemente für eine Ressourcentypimplementierung. Mithilfe der Rückmeldemethoden kann RGM Ressourcen im Cluster steuern, wenn eine Änderung der Cluster-Mitgliedschaft eintritt, wie zum Beispiel ein Knotenstart oder -absturz.

Hinweis – Die Rückmeldemethoden werden von RGM mit Root-Berechtigungen ausgeführt, weil die Client-Programme HA-Dienste im Cluster-System steuern. Installieren und verwalten Sie diese Methoden mit eingeschränkter Dateieigentümerschaft und eingeschränkten Berechtigungen. Geben Sie ihnen einen eigens privilegierten Eigentümer, wie `bin` oder `root`, und erteilen Sie nur Lesezugriff.

Dieser Abschnitt beschreibt Rückmeldemethodenargumente und Beendigungscodes. Die Rückmeldemethoden werden in folgenden Kategorien aufgelistet und beschrieben:

- Steuerungs- und Initialisierungsmethoden
- Verwaltungsunterstützungsmethoden
- Netzbezogene Methoden
- Monitor-Steuerungsmethoden

Hinweis – Dieser Abschnitt enthält zwar kurze Beschreibungen der Rückmeldemethoden und beschreibt den Zeitpunkt, zu dem die Methode aufgerufen wird und welche Auswirkung dies auf die Ressource haben soll. Als maßgebliche Referenz für die Rückmeldemethoden wird jedoch auf die Online-Dokumentation `rt_callbacks`(1HA) verwiesen.

Methodenargumente

RGM ruft Rückmeldemethoden folgendermaßen auf:

Methode `-R` *Ressourcenname* `-T` *Typname* `-G` *Gruppenname*

Die Methode ist der Pfadname des Programms, das als `Start`, `Stop` oder sonstige Rückmeldung registriert ist. Die Rückmeldemethoden eines Ressourcentyps werden in dessen Registrierungsdatei deklariert.

Alle Rückmeldemethodenargumente werden als Werte mit Flags übergeben, wobei `-R` den Namen der Ressourceninstanz, `-T` den Ressourcentyp und `-G` die Gruppe angibt, in der die Ressource konfiguriert wird. Verwenden Sie die Argumente mit Zugriffsfunktionen, um Informationen über die Ressource abzurufen.

Die `Validate`-Methode wird mit zusätzlichen Argumenten aufgerufen, das heißt mit den Eigenschaftswerten der Ressource und Ressourcengruppe, in denen sie aufgerufen wird.

Weitere Informationen hierzu finden Sie unter `scha_calls`(3HA).

Beendigungscodes

Für alle Rückmeldemethoden sind die gleichen Beendigungscodes definiert, um die Auswirkung des Methodenaufrufs auf den Ressourcenzustand anzugeben. Eine Beschreibung der Beendigungscodes finden Sie in der Online-Dokumentation unter `scha_calls`(3HA). Die Beendigungscodes sind:

- 0 – Methode war erfolgreich
- Alle Werte ungleich Null – Methode ist fehlgeschlagen

RGM verwaltet auch außergewöhnliche Fehlschläge der Rückmeldemethodenausführung, wie Zeitüberschreitungen oder Speicherabbilder.

Methodenimplementierungen müssen die Fehlschlaginformationen für jeden Knoten über `syslog` ausgeben. Wenn die Ausgabe an `stdout` oder `stderr` geschrieben wird, ist nicht garantiert, dass sie dem Benutzer zugestellt wird, auch wenn sie aktuell auf der Konsole des lokalen Knotens angezeigt wird.

Steuerungs- und Initialisierungs-Rückmeldemethoden

Die primären Steuerungs- und Initialisierungs-Rückmeldemethoden starten und stoppen eine Ressource. Andere Methoden führen für eine Ressource Initialisierungs- und Beendigungscode aus.

`Start`
: Diese erforderliche Methode wird auf einem Cluster-Knoten aufgerufen, wenn die Ressourcengruppe mit der Ressource auf diesem Knoten online gebracht wird. Die Methode aktiviert die Ressource auf dem Knoten.

  Eine `Start`-Methode darf erst dann beendet werden, wenn die von ihr aktivierte Ressource gestartet wurde und auf dem lokalen Knoten verfügbar ist. Daher muss die `Start`-Methode vor Beendigung die Ressource abrufen, um festzustellen, ob sie gestartet wurde. Außerdem muss für diese Methode ein ausreichend langer Zeitüberschreitungswert eingestellt werden. Einige Ressourcen, wie zum Beispiel Datenbankdämone, brauchen mehr Zeit zum Starten. Daher benötigt die entsprechende Start-Methode einen höheren Zeitüberschreitungswert.

  Die Reaktion von RGM auf einen Fehlschlag der `Start`-Methode hängt von der Einstellung der `Failover_mode`-Eigenschaft ab.

  Die `START_TIMEOUT`-Eigenschaft in der Ressourcentyp-Registrierungsdatei stellt den Zeitüberschreitungswert für die `Start`-Methode einer Ressource ein.

`Stop`
: Diese erforderliche Methode wird auf einem Cluster-Knoten aufgerufen, wenn die Ressourcengruppe mit der Ressource auf diesem Knoten offline gebracht wird. Die Methode deaktiviert die Ressource, wenn sie aktiv ist.

  Eine `Stop`-Methode darf erst dann beendet werden, wenn die von ihr gesteuerte Anwendung vollständig gestoppt wurde, alle Aktivitäten eingestellt und alle Dateideskriptoren geschlossen hat. Andernfalls nimmt RGM an, dass die Ressource gestoppt wurde, während sie in Wirklichkeit noch läuft, was zu Datenfehlern führen kann. Der sicherste Weg zum Vermeiden von Datenfehlern besteht darin, alle Prozesse auf dem lokalen Knoten zu stoppen, die mit der Ressource in Zusammenhang stehen.

  Die `Stop`-Methode muss vor der Beendigung die Ressource abrufen, um festzustellen, ob sie gestoppt wurde. Außerdem muss für diese Methode ein ausreichend langer Zeitüberschreitungswert eingestellt werden. Einige Ressourcen, wie zum Beispiel Datenbankdämone, brauchen mehr Zeit zum Stoppen. Daher benötigt die entsprechende Stopp-Methode einen höheren Zeitüberschreitungswert.

Die Reaktion von RGM auf einen Fehlschlag der `Stop`-Methode hängt von der Einstellung der `Failover_mode`-Eigenschaft ab (siehe „Ressourceneigenschaften" auf Seite 260).

Die `STOP_TIMEOUT`-Eigenschaft in der Ressourcentyp-Registrierungsdatei stellt den Zeitüberschreitungswert für die `Stop`-Methode einer Ressource ein.

`Init`
Diese optionale Methode wird aufgerufen, um eine einmalige Initialisierung der Ressource auszuführen, wenn diese in einen verwalteten Zustand versetzt wird — entweder, weil die Ressourcengruppe, in der sie sich befindet, aus einem nicht verwalteten in einen verwalteten Zustand versetzt wird, oder weil die Ressource in einer bereits verwalteten Ressourcengruppe erstellt wird. Die Methode wird auf den in der `Init_nodes`-Ressourceneigenschaft festgelegten Knoten aufgerufen.

`Fini`
Diese optionale Methode wird aufgerufen, um nach der Ressource zu bereinigen, wenn diese in einen unverwalteten Zustand versetzt wird — entweder, weil die Ressourcengruppe, in der sie sich befindet, in einen unverwalteten Zustand versetzt wird, oder weil die Ressource aus einer verwalteten Ressourcengruppe gelöscht wird. Die Methode wird auf den in der `Init_nodes`-Ressourceneigenschaft festgelegten Knoten aufgerufen.

`Boot`
Diese Init ähnliche optionale Methode wird aufgerufen, um die Ressource auf Knoten zu initialisieren, die dem Cluster beitreten, nachdem die Ressourcengruppe mit der Ressource bereits unter RGM-Verwaltung gestellt wurde. Die Methode wird auf den in der `Init_nodes`-Ressourceneigenschaft festgelegten Knoten aufgerufen. Die `Boot`-Methode wird aufgerufen, wenn der Knoten dem Cluster beitritt bzw. wenn er als Ergebnis eines Starts oder Neustarts erneut beitritt.

Hinweis – Ein Fehlschlag der `Init`-, `Fini`- oder `Boot`-Methode bewirkt das Generieren einer Fehlermeldung durch die `syslog()`-Funktion, hat aber ansonsten keine Auswirkungen auf die RGM-Verwaltung der Ressource.

Verwaltungsunterstützungsmethoden

Verwaltungsaktionen an Ressourcen umfassen das Einstellen und Ändern von Ressourceneigenschaften. Die `Validate`- und `Update`-Rückmeldemethoden ermöglichen es einer Ressourcentypimplementierung, diese Verwaltungsaktionen zu nutzen.

`Validate`
Diese optionale Methode wird aufgerufen, wenn eine Ressource erstellt wird und wenn eine Verwaltungsaktion die Eigenschaften der Ressource bzw. ihrer Ressourcengruppe aktualisiert. Die Methode wird auf denjenigen Clustern aufgerufen, die von der Init_nodes-Eigenschaft des Ressourcentyps angegeben

werden. `Validate` wird aufgerufen, bevor die Erstellung bzw. Aktualisierung angewendet wird. Ein Fehlerbeendigungscode der Methode auf einem beliebigen Knoten hat den Abbruch der Erstellung bzw. Aktualisierung zur Folge.

`Validate` wird nur dann aufgerufen, wenn Ressourcen- bzw. Ressourcengruppeneigenschaften über eine Verwaltungsaktion geändert werden, und nicht, wenn RGM Eigenschaften einstellt oder wenn ein Monitor die Ressourceneigenschaften `Status` und `Status_msg` einstellt.

`Update`
: Diese optionale Methode wird aufgerufen, um eine laufende Ressource über die Änderung ihrer Eigenschaften zu benachrichtigen. `Update` wird aufgerufen, nachdem eine Verwaltungsaktion erfolgreich die Eigenschaften einer Ressource bzw. deren Gruppe eingestellt hat. Die Methode wird auf denjenigen Knoten aufgerufen, auf denen die Ressource online ist. Die Methode verwendet die API-Zugriffsfunktionen zum Lesen der Eigenschaftswerte, die eine aktive Ressource betreffen könnten, und zum dementsprechenden Anpassen der laufenden Ressource.

  Ein Fehlschlag der `Update`-Methode bewirkt das Generieren einer Fehlermeldung durch die `syslog()`-Funktion, hat aber ansonsten keine Auswirkungen auf die RGM-Verwaltung der Ressource.

Netzwerkbezogene Rückmeldemethoden

Für Dienste, die Netzwerkadressressourcen verwenden, muss das Starten und Stoppen möglicherweise in einer bestimmten Reihenfolge stattfinden, die in Bezug zur Netzwerkadresskonfiguration steht. Die folgenden optionalen Rückmeldemethoden, `Prenet_start` und `Postnet_stop`, ermöglichen es einer Ressourcentypimplementierung, besondere Start- und Schließfunktionen auszuführen, bevor und nachdem eine entsprechende Netzwerkadresse konfiguriert bzw. dekonfiguriert wird.

`Prenet_start`
: Diese optionale Methode wird aufgerufen, um besondere Startaktionen auszuführen, bevor die Netzwerkadressen in derselben Ressourcengruppe konfiguriert werden.

`Postnet_stop`
: Diese optionale Methode wird aufgerufen, um besondere Schließaktionen auszuführen, bevor die Netzwerkadressen in derselben Ressourcengruppe als inaktiv konfiguriert werden.

Monitorsteuerungs-Rückmeldemethoden

Eine Ressourcentypimplementierung kann optional ein Programm enthalten, das die Leistung einer Ressource überwacht, über deren Status berichtet oder bei Ressourcenversagen Aktionen ausführt. Die `Monitor_start`-, `Monitor_stop`- und `Monitor_check`-Methoden unterstützen die Implementierung eines Ressourcen-Monitors in einer Ressourcentypimplementierung.

`Monitor_start`
: Diese optionale Methode wird aufgerufen, um einen Monitor für die Ressource zu starten, nachdem die Ressource gestartet wurde.

`Monitor_stop`
: Diese optionale Methode wird aufgerufen, um einen Ressourcen-Monitor zu stoppen, bevor die Ressource gestoppt wird.

`Monitor_check`
: Diese optionale Methode wird aufgerufen, um die Zuverlässigkeit eines Knotens zu beurteilen, bevor eine Ressourcengruppe auf den Knoten verschoben wird. Die `Monitor_check`-Methode muss so implementiert werden, dass sie keine Konflikte mit anderen, gleichzeitig laufenden Methoden bewirkt.

KAPITEL 5

Beispieldatendienst

Dieses Kapitel beschreibt einen Sun Cluster-Beispieldatendienst, HA-DNS, für die `in.named`-Anwendung. Der `in.named`-Dämon ist die Solaris-Implementierung von DNS (Domain Name Service). Der Beispieldatendienst zeigt, wie eine Anwendung mithilfe der Ressourcenverwaltungs-API hoch verfügbar gemacht wird.

Die Ressourcenverwaltungs-API unterstützt eine Shell-Skriptschnittstelle und eine C-Programmschnittstelle. Die Beispielanwendung in diesem Kapitel wurde unter Verwendung der Shell-Skriptschnittstelle geschrieben.

In diesem Kapitel finden Sie folgende Informationen:

Überblick über den Beispieldatendienst

Der Beispieldatendienst wird gestartet, gestoppt, neu gestartet und verschiebt die DNS-Anwendung zwischen den Knoten des Clusters als Reaktion auf Cluster-Ereignisse wie Verwaltungsaktionen, Anwendungsfehler oder Knotenfehler.

Der Anwendungsneustart wird von PMF (Process Monitor Facility) verwaltet. Wenn die Anwendungsausfälle den Fehlschlagzähler im Fehlschlagzeitfenster überschreiten, führt der Fehler-Monitor für die Ressourcengruppe mit der Anwendungsressource ein Failover auf einen anderen Knoten aus.

Im Beispieldatendienst sorgt die `PROBE`-Methode für die Fehlerüberwachung. Diese Methode verwendet den `nslookup`-Befehl, um sicherzustellen, dass die Anwendung fehlerfrei läuft. Wenn das Testsignal feststellt, dass ein DNS-Dienst hängt, wird versucht, den Fehler durch einen lokalen Neustart der DNS-Anwendung zu korrigieren. Wenn dadurch der Fehler nicht behoben werden kann und das Testsignal wiederholt Probleme mit dem Dienst feststellt, wird versucht, ein Failover des Datendienstes auf einen anderen Knoten im Cluster auszuführen.

Dieser Beispieldatendienst setzt sich aus folgenden Komponenten zusammen:

- Eine Ressourcentyp-Registrierungsdatei, in der die statischen Datendiensteigenschaften definiert werden.
- Eine `Start`-Rückmeldemethode, die von RGM aufgerufen wird, um den `in.named`-Dämon zu starten, sobald die Ressourcengruppe mit dem HA-DNS-Datendienst online gebracht wird.
- Eine `Stop`-Rückmeldemethode, die von RGM aufgerufen wird, um den `in.named`-Dämon zu stoppen, wenn die Ressourcengruppe mit HA-DNS offline gebracht wird.
- Ein Fehler-Monitor, der die Verfügbarkeit des Dienstes prüft, indem er überprüft, ob der DNS-Server läuft. Der Fehler-Monitor wird durch eine benutzerdefinierte `PROBE`-Methode implementiert und von den Rückmeldemethoden `Monitor_start` und `Monitor_stop` gestartet bzw. gestoppt.
- Eine `Validate`-Rückmeldemethode, die von RGM aufgerufen wird, um zu validieren, dass das Konfigurationsverzeichnis für den Dienst zugänglich ist.
- Eine `Update`-Rückmeldemethode, die von RGM aufgerufen wird, um den Fehler-Monitor zu starten, wenn der Systemverwalter den Wert einer Ressourceneigenschaft ändert.

Definieren der Ressourcentyp-Registrierungsdatei

Die Ressourcentyp-Registrierungsdatei (RTR-Datei) in diesem Beispiel definiert die statische Konfiguration des DNS-Ressourcentyps. Ressourcen dieses Typs übernehmen die in der RTR-Datei definierten Eigenschaften.

Die Informationen in der RTR-Datei werden von RGM gelesen, wenn der Cluster-Verwalter den HA-DNS-Datendienst registriert.

Überblick über RTR-Dateien

Die RTR-Datei ist in einem klar definierten Format aufgebaut. In der Datei werden zuerst Ressourcentypeigenschaften definiert, dann systemdefinierte Ressourceneigenschaften und zuletzt Erweiterungseigenschaften. Weitere Informationen finden Sie in der Online-Dokumentation unter `rt_reg`(4) und unter „Einstellen der Ressourcen- und Ressourcentypeigenschaften" auf Seite 35.

Dieser Abschnitt beschreibt die spezifischen Eigenschaften in der RTR-Beispieldatei. Verschiedene Teile der Datei werden aufgelistet. Eine vollständige Auflistung des Inhalts der RTR-Beispieldatei finden Sie unter „Auflistung der Ressourcentyp-Registrierungsdatei" auf Seite 281.

Ressourcentypeigenschaften in der RTR-Beispieldatei

Die RTR-Beispieldatei beginnt mit Kommentaren, gefolgt von Ressourcentypeigenschaften zur Definition der HA-DNS-Konfiguration, wie in der folgenden Auflistung gezeigt.

```
#
# Copyright (c) 1998-2004 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;
```

Tipp – Als ersten Eintrag in der RTR-Datei müssen Sie die `Resource_type`-Eigenschaft deklarieren. Andernfalls schlägt die Registrierung des Ressourcentyps fehl.

Hinweis – RGM unterscheidet bei Eigenschaftsnamen nicht zwischen Groß- und Kleinschreibung. Die Konvention für Eigenschaften in von Sun gelieferten RTR-Dateien, mit Ausnahme der Methodennamen, ist Großschreibung des ersten Buchstabens und Kleinschreibung der restlichen Buchstaben des Namens. Methodennamen werden — ebenso wie Eigenschaftsattribute — ganz in Großbuchstaben geschrieben.

Es folgen einige Informationen zu diesen Eigenschaften.

- Der Ressourcentypname kann entweder nur unter Verwendung der `Resource_type`-Eigenschaft angegeben werden (`sample`), oder zusammen mit `Vendor_id` als Präfix zwischen "." als Trennzeichen vom Ressourcentyp (`SUNW.sample`).

  Wenn Sie `Vendor_id` verwenden, nehmen Sie das Börsensymbol für das Unternehmen, das den Ressourcentyp definiert. Der Ressourcentypname muss im Cluster einmalig sein.
- Die `RT_version`-Eigenschaft identifiziert die Version des Beispieldatendienstes entsprechend der Angabe des Herstellers.
- Die `API_version`-Eigenschaft identifiziert die Sun Cluster-Version. `API_version = 2` gibt zum Beispiel an, dass der Datendienst unter Sun Cluster Version 3.0 ausgeführt wird.
- `Failover = TRUE` gibt an, dass der Datendienst nicht in einer Ressourcengruppe ausgeführt werden kann, die auf mehreren Knoten gleichzeitig online sein kann.
- `RT_basedir` zeigt auf `/opt/SUNWsample/bin` als den Verzeichnispfad zu vollständigen relativen Pfaden, wie zum Beispiel Rückmeldemethodenpfaden.
- `Start`, `Stop`, `Validate` usw. geben die Pfade zu den entsprechenden Rückmeldemethodenprogrammen an, die von RGM aufgerufen werden. Diese Pfade sind relativ zu dem Verzeichnis, das durch `RT_basedir` angegeben wird.
- `Pkglist` identifiziert `SUNWsample` als das Paket, das die Beispiel-Datendienstinstallation enthält.

Ressourcentypeigenschaften, die nicht in dieser RTR-Datei angegeben sind, wie `Single_instance`, `Init_nodes` und `Installed_nodes`, rufen ihre Standardwerte ab. Eine vollständige Liste der Ressourcentypeigenschaften mit den jeweiligen Standardwerten finden Sie in „Ressourcentypeigenschaften" auf Seite 253.

Der Cluster-Verwalter kann die für Ressourcentypeigenschaften in der RTR-Datei angegebenen Werte nicht ändern.

Ressourceneigenschaften in der RTR-Beispieldatei

Der Konvention gemäß werden Ressourceneigenschaften in der RTR-Datei nach den Ressourcentypeigenschaften deklariert. Ressourceneigenschaften sind sowohl die von Sun Cluster bereitgestellten systemdefinierten Eigenschaften als auch vom Benutzer definierte Erweiterungseigenschaften. Für jeden Typ können Sie eine Reihe von Eigenschaftsattributen angeben, die Sun Cluster vorgibt, wie zum Beispiel "minimum", "maximum" und Standardwerte.

Systemdefinierte Eigenschaften in der RTR-Datei

Die folgende Auflistung zeigt die systemdefinierten Eigenschaften in der RTR-Beispieldatei.

```
# 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;
}
{
        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 = "";
}
```

Sun Cluster stellt zwar die systemdefinierten Eigenschaften bereit. Sie können jedoch mithilfe der Ressourceneigenschaftsattribute andere Standardwerte einstellen. Eine vollständige Auflistung der Attribute, die für Ressourceneigenschaften zur Verfügung stehen, finden Sie unter „Ressourceneigenschaftsattribute" auf Seite 278.

Beachten Sie folgende Aspekte der systemdefinierten Ressourceneigenschaften in der RTR-Beispieldatei:

- Sun Cluster stellt für alle Zeitüberschreitungen einen Mindestwert (1 Sekunde) und einen Standardwert (3600 Sekunden) bereit. In der RTR-Beispieldatei wird der Mindestwert zu 60 und der Standardwert zu 300 Sekunden geändert. Der

Cluster-Verwalter kann diesen Standardwert akzeptieren oder den Wert für die Zeitüberschreitung ändern (60 oder größer). Sun Cluster hat keinen zulässigen Höchstwert.

- Für die Eigenschaften `Thorough_Probe_Interval`, `Retry_count` und `Retry_interval` wird das `TUNABLE`-Attribut auf `ANYTIME` eingestellt. Diese Einstellung bedeutet, dass der Cluster-Verwalter den Wert der betreffenden Eigenschaften ändern kann, auch wenn der Datendienst gerade läuft. Diese Eigenschaften werden vom Fehler-Monitor verwendet, der für den Beispieldatendienst implementiert wurde. Der Beispieldatendienst implementiert eine `Update`-Methode zum Starten und Stoppen des Fehler-Monitors, wenn diese oder andere Ressourceneigenschaften durch eine Verwaltungsaktion geändert werden. Weitere Informationen finden Sie unter „`Update`-Methode" auf Seite 123.
- Ressourceneigenschaften werden folgendermaßen klassifiziert:
  - *Erforderlich* — Der Cluster-Verwalter muss einen Wert angeben, wenn er eine Ressource erstellt.
  - *Optional* — Wenn der Verwalter keinen Wert angibt, stellt das System einen Standardwert bereit.
  - *Bedingt* — RGM erstellt die Eigenschaft nur, wenn sie in der RTR-Datei deklariert wurde.

  Der Fehler-Monitor des Beispieldatendienstes verwendet die bedingten Eigenschaften `Thorough_probe_interval`, `Retry_count`, `Retry_interval` und `Network_resources_used`. Daher mussten diese vom Entwickler in der RTR-Datei deklariert werden. Weitere Informationen zur Klassifizierung von Eigenschaften finden Sie in der Online-Dokumentation unter `r_properties`(5) und unter „Ressourceneigenschaften" auf Seite 260.

Erweiterungseigenschaften in der RTR-Datei

Am Ende der RTR-Datei befinden sich die Erweiterungseigenschaften, die in der folgenden Auflistung gezeigt werden.

```
# 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 = 120;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Die RTR-Beispieldatei definiert zwei Erweiterungseigenschaften, `Confdir` und `Probe_timeout`. `Confdir` gibt den Pfad zum DNS-Konfigurationsverzeichnis an. Dieses Verzeichnis enthält die `in.named`-Datei, die der DNS für einen erfolgreichen Betrieb benötigt. Die `Start`- und `Validate`-Methoden des Beispieldatendienstes verwenden diese Eigenschaft, um vor dem Starten von DNS zu überprüfen, ob auf das Konfigurationsverzeichnis und die `in.named`-Datei zugegriffen werden kann.

Nach Konfigurieren des Datendienstes überprüft die `Validate`-Methode, ob das neue Verzeichnis zugänglich ist.

Bei der `PROBE`-Methode des Beispieldatendienstes handelt es sich nicht um eine Sun Cluster-Rückmeldemethode, sondern um eine benutzerdefinierte Methode. Daher stellt Sun Cluster keine `Probe_timeout`-Eigenschaft für sie bereit. Der Entwickler hat eine Erweiterungseigenschaft in der RTR-Datei definiert, mit deren Hilfe der Cluster-Verwalter einen Probe_timeout-Wert konfigurieren kann.

Bereitstellen gemeinsamer Funktionalität für alle Methoden

Dieser Abschnitt beschreibt die Funktionalität, die in allen Rückmeldemethoden des Beispieldatendienstes verwendet wird:

- „Identifizieren des Befehlsinterpreters und Exportieren des Pfads" auf Seite 98.
- „Deklarieren der Variablen `PMF_TAG` und `SYSLOG_TAG`" auf Seite 99.
- „Analysieren der Funktionsargumente" auf Seite 100.
- „Generieren von Fehlermeldungen" auf Seite 101.
- „Abrufen von Eigenschaftsinformationen" auf Seite 102.

Identifizieren des Befehlsinterpreters und Exportieren des Pfads

Die erste Zeile eines Shell-Skripts muss den Befehlsinterpreter identifizieren. Jedes der Methodenskripts im Beispieldatendienst identifiziert den Befehlsinterpreter wie folgt:

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

Alle Methoden-Skripts in der Beispielanwendung exportieren den Pfad zu den Sun Cluster-Binärdateien und -Bibliotheken, anstatt sich auf die PATH-Einstellungen der Benutzer zu verlassen.

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

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

Deklarieren der Variablen PMF_TAG und SYSLOG_TAG

Alle Methoden-Skripts mit Ausnahme von Validate verwenden pmfadm zum Starten bzw. Stoppen des Datendienstes oder des Monitors, indem sie den Ressourcennamen übergeben. Jedes Skript definiert eine Variable, PMF_TAG, die an pmfadm übergeben werden kann, um entweder den Datendienst oder den Monitor zu identifizieren.

Ebenso verwendet jedes Methodenskript den logger-Befehl, um Meldungen im Systemprotokoll zu protokollieren. Jedes Skript definiert eine Variable, SYSLOG_TAG, die mit der Option -t an logger übergeben werden kann, um den Ressourcentyp, die Ressourcengruppe und den Ressourcennamen der Ressource, für die eine Meldung protokolliert wird, zu identifizieren.

Alle Methoden definieren SYSLOG_TAG auf die gleiche Art und Weise, wie im folgenden Beispiel gezeigt. Die dns_probe-, dns_svc_start-, dns_svc_stop- und dns_monitor_check-Methoden definieren PMF_TAG wie folgt, wobei die Verwendung von pmfadm und logger der dns_svc_stop-Methode entnommen wird:

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

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # SIGTERM-Signal an den Datendienst senden und  80% des
   # gesamten Zeitüberschreitungswerts warten.
   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"
```

Die Methoden dns_monitor_start, dns_monitor_stop und dns_update definieren PMF_TAG wie folgt, wobei die Verwendung von pmfadm der dns_monitor_stop-Methode entnommen wird:

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

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

# Feststellen, ob der Monitor läuft, und ggf. Beenden erzwingen.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

Analysieren der Funktionsargumente

RGM ruft alle Rückmeldemethoden — mit Ausnahme der `Validate`-Methode — folgendermaßen auf.

Methodenname `-R` *Ressourcenname* `-T` *Ressourcentypname* `-G` *Ressourcengruppenname*

Der Methodenname ist der Pfadname des Programms, das die Rückmeldemethode implementiert. Ein Datendienst gibt den Pfadnamen für jede Methode in der RTR-Datei an. Diese Pfadnamen beziehen sich auf das Verzeichnis, das ebenfalls in der RTR-Datei von der `RT_basedir`-Eigenschaft angegeben wird. Zum Beispiel werden das Basisverzeichnis und die Methodennamen in der RTR-Datei des Beispieldatendienstes wie folgt angegeben.

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

Alle Rückmeldemethodenargumente werden als Werte mit Flags übergeben, wobei `-R` den Namen der Ressourceninstanz, `-T` den Ressourcentyp und `-G` die Gruppe angibt, in der die Ressource konfiguriert wird. Weitere Informationen zu Rückmeldemethoden finden Sie in der Online-Dokumentation unter `rt_callbacks`(1HA).

Hinweis – Die `Validate`-Methode wird mit zusätzlichen Argumenten aufgerufen, das heißt mit den Eigenschaftswerten der Ressource und Ressourcengruppe, in denen sie aufgerufen wird. Weitere Informationen finden Sie unter „Bearbeiten von Eigenschaftsaktualisierungen” auf Seite 118.

Jede Rückmeldemethode benötigt eine Funktion zum Analysieren der Argumente, die ihr übergeben werden. Da an alle Rückmeldemethoden die gleichen Argumente übergeben werden, stellt der Datendienst eine einzige Analysefunktion bereit, die für alle Rückmeldungen in der Anwendung eingesetzt wird.

Im Folgenden wird die `parse_args()`-Funktion gezeigt, die für alle Rückmeldemethoden in der Beispielanwendung verwendet wird.

```
##############################################################################
# Programmargumente analysieren.
#
function parse_args # [args ...]
{
      typeset opt

      while getopts 'R:G:T:' opt
      do
              case "$opt" in
              R)
                   # Name der DNS-Ressource.
                   RESOURCE_NAME=$OPTARG
                   ;;
              G)
                   # Name der Ressourcengruppe, in der die Ressource
                   # konfiguriert ist.
                   RESOURCEGROUP_NAME=$OPTARG
                   ;;
              T)
                   # Name des Ressourcentyps.
                   RESOURCETYPE_NAME=$OPTARG
                   ;;
              *)
                   logger -p ${SYSLOG_FACILITY}.err \
                   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                   "FEHLER: Option $OPTARG unbekannt"
                   exit 1
                       ;;
              esac
    done
}
```

Hinweis – Die `PROBE`-Methode in der Beispielanwendung ist zwar benutzerdefiniert, also keine Sun Cluster-Rückmeldemethode, wird jedoch mit den gleichen Argumenten wie die Rückmeldemethoden aufgerufen. Daher enthält diese Methode genau die gleiche Analysefunktion wie die anderen Rückmeldemethoden.

Die Analysefunktion wird in `MAIN` wie folgt aufgerufen:

```
parse_args "$@"
```

Generieren von Fehlermeldungen

Für Rückmeldemethoden wird empfohlen, `syslog` für die Ausgabe von Fehlermeldungen an die Endbenutzer zu verwenden. Alle Rückmeldemethoden im Beispieldatendienst verwenden die Funktion `scha_cluster_get()`, um die Nummer des als Cluster-Protokoll verwendeten `syslog` wie folgt abzurufen:

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

Der Wert wird in einer Shell-Variablen, `SYSLOG_FACILITY`, gespeichert und kann im `logger`-Befehl verwendet werden, um Meldungen im Cluster-Protokoll zu protokollieren. So ruft zum Beispiel die `Start`-Methode im Beispieldatendienst syslog ab und protokolliert eine Meldung, dass der Datendienst gestartet wurde:

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

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

Weitere Informationen finden Sie in der Online-Dokumentation unter `scha_cluster_get`(1HA).

Abrufen von Eigenschaftsinformationen

Die meisten Rückmeldemethoden benötigen Informationen über die Ressourcen- und Ressourcentypeigenschaften des Datendienstes. Die API stellt zu diesem Zweck die `scha_resource_get()`-Funktion bereit.

Es stehen zwei Arten von Ressourceneigenschaften zur Verfügung: systemdefinierte Eigenschaften und Erweiterungseigenschaften. Systemdefinierte Eigenschaften sind vordefiniert, während Sie Erweiterungseigenschaften in der RTR-Datei definieren.

Wenn Sie `scha_resource_get()` zum Abrufen des Wertes einer systemdefinierten Eigenschaft verwenden, geben Sie den Eigenschaftsnamen mit dem `-O`-Parameter an. Der Befehl gibt nur den *Wert* der Eigenschaft zurück. Im Beispieldatendienst benötigt zum Beispiel die `Monitor_start`-Methode den Speicherort des Testsignalprogramms, um es zu starten. Das Testsignalprogramm residiert im Basisverzeichnis für den Datendienst, auf das die `RT_basedir`-Eigenschaft zeigt. Daher ruft die `Monitor_start`-Methode den Wert von `RT_basedir` ab und legt ihn in der `RT_basedir`-Variablen ab:

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

Für Erweiterungseigenschaften müssen Sie mit dem `-O`-Parameter angeben, dass es sich um eine Erweiterungseigenschaft handelt, und den Eigenschaftsnamen als letzten Parameter angeben. Für Erweiterungseigenschaften gibt der Befehl sowohl den *Typ* als auch den *Wert* der Eigenschaft zurück. Im Beispieldatendienst ruft das Testsignalprogramm zum Beispiel den Typ und den Wert der `probe_timeout`-Erweiterungseigenschaft ab und verwendet dann `awk`, um nur den Wert in der `PROBE_TIMEOUT`-Shell-Variablen abzulegen:

```
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}’‘
```

Steuern des Datendienstes

Ein Datendienst muss eine `Start`- oder `Prenet_start`-Methode bereitstellen, um den Anwendungsdämon auf dem Cluster zu aktivieren, sowie eine `Stop`- oder `Postnet_stop`-Methode zum Stoppen des Anwendungsdämons auf dem Cluster. Der Beispieldatendienst implementiert eine `Start`- und eine `Stop`-Methode. „Bestimmen der zu verwendenden `Start`- und `Stop`-Methoden" auf Seite 47 enthält Informationen darüber, in welchen Fällen möglicherweise die Verwendung von `Prenet_start` und `Postnet_stop` vorzuziehen wäre.

`Start`-Methode

RGM ruft die `Start`-Methode auf einem Cluster-Knoten auf, wenn die Ressourcengruppe, die den Datendienst enthält, auf diesem Knoten online gebracht wird bzw. wenn die Ressourcengruppe bereits online ist und die Ressource aktiviert wird. In der Beispielanwendung aktiviert die `Start`-Methode den `in.named` (DNS)-Dämon auf diesem Knoten.

Dieser Abschnitt beschreibt die Hauptteile der `Start`-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die `parse_args()`-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden" auf Seite 98 beschrieben).

Eine vollständige Auflistung der `Start`-Methode ist in „`Start`-Methode" auf Seite 284 enthalten.

Überblick über `Start`

Vor dem Versuch, DNS zu starten, überprüft die `Start`-Methode des Beispieldatendienstes, ob das Konfigurationsverzeichnis und die Konfigurationsdatei (`named.conf`) zugänglich und verfügbar sind. Die Informationen in `named.conf` sind für einen erfolgreichen DNS-Betrieb entscheidend.

Diese Rückmeldemethode verwendet PMF (`pmfadm`) zum Starten des DNS-Dämons (`in.named`). Wenn DNS abstürzt oder nicht startet, versucht PMF eine vorgeschriebene Anzahl von Malen, den Dienst während eines festgelegten Zeitintervalls zu starten. Die Anzahl der Wiederholversuche und das Intervall werden von den Eigenschaften in der RTR-Datei des Datendienstes angegeben.

Überprüfen der Konfiguration

DNS benötigt für die Ausführung Informationen aus der `named.conf`-Datei im Konfigurationsverzeichnis. Daher führt die `Start`-Methode einige Gesundheits-Checks aus, um zu überprüfen, ob auf das Verzeichnis und die Datei zugegriffen werden kann, bevor DNS gestartet wird.

Die `Confdir`-Erweiterungseigenschaft stellt den Pfad zum Konfigurationsverzeichnis bereit. Die Eigenschaft selbst ist in der RTR-Datei definiert. Den Speicherort gibt jedoch der Cluster-Verwalter beim Konfigurieren des Datendienstes an.

Im Beispieldatendienst ruft die `Start`-Methode den Speicherort des Konfigurationsverzeichnisses mithilfe der Funktion `scha_resource_get()` ab.

Hinweis – Da `Confdir` eine Erweiterungseigenschaft ist, gibt `scha_resource_get()` sowohl den Typ als auch den Wert zurück. Der `awk`-Befehl ruft lediglich den Wert ab und legt ihn in einer Shell-Variablen ab, `CONFIG_DIR`.

```
# Den Wert von Confdir suchen, den der Cluster-Verwalter beim Hinzufügen
# der Ressource eingestellt hat.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get gibt sowohl den "Typ" als auch den "Wert" für die
# Erweiterungseigenschaften zurück. Nur den Wert der Erweiterungseigenschaft abrufen.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

Die `Start`-Methode verwendet dann den Wert von `CONFIG_DIR`, um zu überprüfen, ob das auf das Verzeichnis zugegriffen werden kann. Wenn kein Zugriff möglich ist, protokolliert `Start` eine Fehlermeldung und wird mit Fehlerstatus beendet. Weitere Informationen finden Sie unter „`Start`-Beendigungsstatus" auf Seite 105.

```
# Prüfen, ob Zugriff auf $CONFIG_DIR möglich ist.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Verzeichnis $CONFIG_DIR fehlt oder ist nicht eingehängt"
   exit 1
fi
```

Vor dem Starten des Anwendungsdämons führt diese Methode eine abschließende Prüfung durch, um zu überprüfen, ob die `named.conf`-Datei vorhanden ist. Wenn sie nicht vorhanden ist, protokolliert `Start` eine Fehlermeldung und wird im Fehlerstatus beendet.

```
# Wechsel zum $CONFIG_DIR-Verzeichnis, falls die
# Datendateien relative Pfadnamen enthalten.
cd $CONFIG_DIR

# Prüfen, ob die named.conf-Datei im $CONFIG_DIR-Verzeichnis vorhanden ist
```

```
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Datei $CONFIG_DIR/named.conf fehlt oder ist leer"
   exit 1
fi
```

Starten der Anwendung

Diese Methode verwendet PMF (`pmfadm`) zum Starten der Anwendung. Über den `pmfadm`-Befehl können Sie die Anzahl der Male einstellen, die eine Anwendung während eines angegebenen Zeitraums neu gestartet wird. Die RTR-Datei enthält dafür zwei Eigenschaften: `Retry_count` gibt die Anzahl von Malen an, für die der Neustart der Anwendung versucht wird, und `Retry_interval` den Zeitraum, in dem die Versuche stattfinden sollen.

Die `Start`-Methode ruft die Werte für `Retry_count` und `Retry_interval` mithilfe der Funktion `scha_resource_get()` ab und speichert sie in Shell-Variablen. Dann werden diese Werte an `pmfadm` übergeben. Dabei werden die Optionen `-n` und `-t` verwendet.

```
# Wert für Wiederholversuchszähler aus der RTR-Datei abrufen.
RETRY_CNT=‘scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘
# Wert für das Wiederholungsintervall aus der RTR-Datei abrufen. Der
# Wert ist in Sekunden angegeben und muss für die Übergabe an pmfadm in Minuten
# konvertiert werden. Beachten Sie, dass die Konversion aufrundet; 50 Sekunden werden
# zu einer Minute aufgerundet.
((RETRY_INTRVAL=‘scha_resource_get -O Retry_Interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME‘ / 60))

# in.named-Dämon unter PMF-Steuerung starten. Abstürzen lassen und bis zu
# $RETRY_COUNT Male in einem Zeitraum von $RETRY_INTERVAL abstürzen lassen
# und neu starten. Wenn er öfter abstürzt, versucht PMF keinen Neustart mehr.
# Wenn unter der Markierung <$PMF_TAG> bereits ein Prozess registriert ist, sendet PMF
# eine Warnmeldung, dass der Prozess bereits läuft.
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Meldung protokollieren, dass HA-DNS gestartet wurde.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS erfolgreich gestartet"
fi
exit 0
```

`Start`-Beendigungsstatus

Eine `Start`-Methode darf erst dann mit Erfolg beendet werden, wenn die zugrunde liegende Anwendung auch tatsächlich läuft und verfügbar ist, vor allem dann, wenn andere Datendienste von ihr abhängen. Eine Möglichkeit zum Überprüfen des Erfolgs

besteht darin, ein Testsignal an die Anwendung zu senden, um festzustellen, ob sie läuft, bevor die `Start`-Methode beendet wird. Vergewissern Sie sich bei komplexen Anwendungen wie zum Beispiel Datenbanken, dass der Wert für die `Start_timeout`-Eigenschaft in der RTR-Datei ausreichend hoch eingestellt wird, damit die Anwendung genügend Zeit zum Initialisieren und Wiederherstellen nach einem Absturz hat.

Hinweis – Da die Anwendungsressource (DNS) im Beispieldatendienst schnell startet, überprüft der Beispieldatendienst nicht, ob sie läuft, bevor die Methode mit Erfolg beendet wird.

Wenn diese Methode DNS nicht starten kann und mit einem Fehlerstatus beendet wird, prüft RGM die `Failover_mode`-Eigenschaft, die die Reaktion auf den Fehlerstatus festlegt. Der Beispieldatendienst stellt die `Failover_mode`-Eigenschaft nicht ausdrücklich ein. Daher hat sie den Standardwert `NONE`, es sei denn, der Cluster-Verwalter hat diesen Standardwert übersteuert und einen anderen Wert angegeben. In diesem Fall ist die einzige Aufgabe von RGM, den Zustand des Datendienstes einzustellen. Ein Bedienereingriff ist erforderlich, um auf demselben Knoten neu zu starten oder ein Failover auf einen anderen Knoten auszuführen.

`Stop`-Methode

Die `Stop`-Methode wird auf einem Cluster-Knoten aufgerufen, wenn die Ressourcengruppe mit der HA-DNS-Ressource auf diesem Knoten offline genommen wird bzw. wenn die Ressourcengruppe online bleibt, jedoch die Ressource deaktiviert wird. Diese Methode stoppt den `in.named` (DNS)-Dämon auf dem Knoten.

Dieser Abschnitt beschreibt die Hauptteile der `Stop`-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die `parse_args()`-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden" auf Seite 98 beschrieben).

Eine vollständige Auflistung der `Stop`-Methode ist in „`Stop`-Methode" auf Seite 287 enthalten.

Überblick über `Stop`

Zwei Punkte müssen beim Stoppen des Datendienstes vor allem beachtet werden. Als Erstes muss für ein ordnungsgemäßes Herunterfahren gesorgt werden. Das Senden eines `SIGTERM`-Signals über `pmfadm` ist die beste Möglichkeit, ein ordnungsgemäßes Herunterfahren zu erzielen.

Als Zweites muss sichergestellt werden, dass der Datendienst wirklich gestoppt wird, um zu vermeiden, dass er in den `Stop_failed`-Zustand versetzt wird. Die beste Möglichkeit hierzu ist das Senden eines `SIGKILL`-Signals über `pmfadm`.

Die Stop-Methode im Beispieldatendienst berücksichtigt diese beiden Punkte. Sie sendet zunächst ein SIGTERM-Signal. Wenn dieses Signal den Datendienst nicht stoppen kann, sendet die Methode ein SIGKILL-Signal.

Bevor versucht wird, DNS zu stoppen, überprüft die Stop-Methode, ob der Prozess tatsächlich läuft. Wenn der Prozess ausgeführt wird, verwendet Stop PMF (pmfadm), um ihn zu stoppen.

Diese Stop-Methode ist garantiert idempotent. Auch wenn RGM eine Stop-Methode nicht zum zweiten Mal aufrufen sollte, ohne zuvor den Datendienst über einen Aufruf der entsprechenden Start-Methode gestartet zu haben, kann eine Stop-Methode für eine Ressource aufgerufen werden, obwohl diese nie gestartet wurde oder von selbst ausfiel. Diese Stop-Methode wird also mit Erfolg beendet, selbst wenn DNS nicht läuft.

Stoppen der Anwendung

Die Stop-Methode bietet zwei Möglichkeiten zum Stoppen des Datendienstes: eine geordnete oder weiche Art, bei der ein SIGTERM-Signal über pmfadm verwendet wird, und eine plötzliche oder harte Art, bei der ein SIGKILL-Signal eingesetzt wird. Die Stop-Methode ruft den Stop_timeout-Wert ab (den Zeitraum, in dem die Stop-Methode einen Wert zurückgeben muss). Stop weist dann 80% dieser Zeit dem weichen Stoppvorgang und 15% dem harten Stoppvorgang zu (5% werden zurückgehalten), wie im folgenden Beispiel gezeigt.

```
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME
\

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

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

Die Stop-Methode verwendet pmfadm -q, um zu überprüfen, ob der DNS-Dämon läuft. Ist das der Fall, sendet Stop zunächst mit pmfadm -s ein TERM-Signal zum Beenden des DNS-Prozesses. Wenn dieses Signal den Prozess nach Ablauf von 80% des Zeitüberschreitungswertes nicht beenden konnte, sendet Stop ein SIGKILL-Signal. Wenn auch dieses Signal den Prozess nicht innerhalb von 15% des Zeitüberschreitungswertes beenden kann, protokolliert die Methode eine Fehlermeldung und wird mit Fehlerstatus beendet.

Wenn pmfadm den Prozess beendet, protokolliert die Methode eine Meldung, die besagt, dass der Prozess gestoppt wurde, und wird mit Erfolg beendet.

Wenn der DNS-Prozess nicht läuft, protokolliert die Methode eine diesbezügliche Meldung und wird dennoch mit Erfolg beendet. Das folgende Codebeispiel zeigt, wie Stop den pmfadm-Befehl zum Stoppen des DNS-Prozesses einsetzt.

```
# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG; then
   # Send a SIGTERM signal to the data service and wait for 80% of
```

```
the
   # total timeout value.
   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"

      # Since the data service did not stop with a SIGTERM signal, use
      # SIGKILL now and wait for another 15% of the total timeout value.
      pmfadm -s $PMF_TAG -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG]
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL"

          exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS is not started"

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service resource in STOP_FAILED State.

   exit 0

fi

# Could successfully stop DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
    "HA-DNS successfully stopped"
exit 0
```

`Stop`-Beendigungsstatus

Eine `Stop`-Methode darf erst dann mit Erfolg beendet werden, wenn die zugrundeliegende Anwendung auch tatsächlich gestoppt wurde, vor allem wenn andere Datendienste von ihr abhängen. Andernfalls können Daten beschädigt werden.

Vergewissern Sie sich für komplexe Anwendungen wie zum Beispiel Datenbanken, dass der Wert für die `Stop_timeout`-Eigenschaft in der RTR-Datei ausreichend hoch eingestellt wird, damit die Anwendung beim Stoppvorgang genügend Zeit zum Bereinigen hat.

Wenn diese Methode DNS nicht stoppen kann und mit Fehlerstatus beendet wird, prüft RGM die `Failover_mode`-Eigenschaft, die festlegt, welche Reaktion nun erfolgt. Der Beispieldatendienst stellt die `Failover_mode`-Eigenschaft nicht explizit ein. Daher hat sie den Standardwert `NONE`, es sei denn, der Cluster-Verwalter hat diesen übersteuert und einen anderen Wert angegeben. In diesem Fall ist die einzige Aufgabe von RGM, den Zustand des Datendienstes auf `Stop_failed` einzustellen. Ein Bedienereingriff ist erforderlich, um das Stoppen der Anwendung zu erzwingen und den `Stop_failed`-Zustand aufzuheben.

Definieren eines Fehler-Monitors

Die Beispielanwendung implementiert einen einfachen Fehler-Monitor, der die Zuverlässigkeit der DNS-Ressource (`in.named`) überwacht. Der Fehler-Monitor setzt sich aus folgenden Elementen zusammen:

- `dns_probe`, ein benutzerdefiniertes Programm, das `nslookup` verwendet, um zu überprüfen, ob die vom Beispieldatendienst gesteuerte DNS-Ressource läuft. Wenn DNS nicht läuft, versucht diese Methode einen lokalen Neustart. Je nach der Anzahl der Neustartversuche kann sie auch anfordern, dass RGM den Datendienst auf einen anderen Knoten verschiebt.
- `dns_monitor_start`, eine Rückmeldemethode zum Starten von `dns_probe`. Wenn die Überwachung aktiviert ist, ruft RGM automatisch `dns_monitor_start` auf, nachdem der Beispieldatendienst online gebracht wurde.
- `dns_monitor_stop`, eine Rückmeldemethode zum Stoppen von `dns_probe`. RGM ruft automatisch `dns_monitor_stop` auf, bevor der Beispieldatendienst offline gebracht wird.
- `dns_monitor_check`, eine Rückmeldemethode zum Aufrufen der `Validate`-Methode, die überprüft, ob das Konfigurationsverzeichnis verfügbar ist, wenn das `PROBE`-Programm für den Datendienst ein Failover auf einen neuen Knoten ausführt.

Testsignalprogramm

Das `dns_probe`-Programm implementiert einen ständig ausgeführten Prozess, der überprüft, ob die vom Beispieldatendienst gesteuerte DNS-Ressource läuft. Der `dns_probe`-Befehl wird von der `dns_monitor_start`-Methode ausgelöst, die wiederum automatisch von RGM aufgerufen wird, sobald der Beispieldatendienst online gebracht wurde. Der Datendienst wird von der `dns_monitor_stop`-Methode gestoppt, die RGM anschließend aufruft, bevor der Beispieldatendienst offline gebracht wird.

Dieser Abschnitt beschreibt die Hauptteile der PROBE-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die parse_args()-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden" auf Seite 98 beschrieben).

Eine vollständige Auflistung der PROBE-Methode finden Sie unter „PROBE-Programm" auf Seite 290.

Überblick über Testsignal

Das Testsignal läuft in einer Endlosschleife. Es verwendet nslookup, um zu überprüfen, ob die richtige DNS-Ressource läuft. Wenn DNS läuft, ruht das Testsignal während eines festgesetzten Zeitintervalls (eingestellt von der systemdefinierten Eigenschaft Thorough_probe_interval) und prüft dann erneut. Wenn DNS nicht läuft, versucht das Programm einen lokalen Neustart. Je nach der Anzahl der Neustartversuche kann es auch anfordern, dass RGM den Datendienst auf einen anderen Knoten verschiebt.

Abrufen von Eigenschaftswerten

Dieses Programm benötigt die Werte der folgenden Eigenschaften:

- Thorough_probe_interval – Zum Einstellen des Zeitraums, während dem das Testsignal ruht.
- Probe_timeout – Zum Durchsetzen des Zeitüberschreitungswertes für das Testsignal an den nslookup-Befehl, der den Test ausführt.
- Network_resources_used – Zum Abrufen der IP-Adresse, unter der DNS läuft.
- Retry_count und Retry_interval – Zum Festlegen der Anzahl von Neustartversuchen und des Zeitraums, über den die Versuche gezählt werden.
- RT_basedir – Zum Abrufen des Verzeichnisses, in dem das PROBE-Programm und das gettime-Dienstprogramm gespeichert sind

Die Funktion scha_resource_get() ruft die Werte dieser Eigenschaften ab und speichert sie folgendermaßen in Shell-Variablen:

```
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‘
```

Hinweis – Für systemdefinierte Eigenschaften wie `Thorough_probe_interval` gibt `scha_resource_get()` nur den Wert zurück. Für Erweiterungseigenschaften wie `Probe_timeout` gibt `scha_resource_get()` den Typ und den Wert zurück. Verwenden Sie den `awk`-Befehl, um nur den Wert abzurufen.

Überprüfen der Zuverlässigkeit des Dienstes

Das Testsignal selbst besteht aus einer `while`-Endlosschleife aus `nslookup`-Befehlen. Vor der while-Schleife wird eine temporäre Datei für die `nslookup`-Antworten eingerichtet. Die Variablen *probefail* und *retries* werden auf 0 initialisiert .

```
# Temporäre Datei für nslookup-Antworten konfigurieren.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

Die `while`-Schleife selbst hat folgende Aufgaben:

- Festlegen des Ruheintervalls für das Testsignal.
- Verwenden von `hatimerun` zum Auslösen von `nslookup` durch Übergabe des `Probe_timeout`-Werts und Identifizieren des Zielhosts.
- Festlegen der *probefail*-Variable, basierend auf dem Erfolg oder Fehlschlag des `nslookup`-Rückgabecodes.
- Wenn *probefail* auf 1 (Fehlschlag) eingestellt ist, wird überprüft, ob die Antwort auf `nslookup` vom Beispieldatendienst und nicht von einem anderen DNS-Server kam.

Es folgt der `while`-Schleifencode.

```
while :
do
   # Das Intervall, in dem das Testsignal ausgeführt werden muss, wird in der
   # Eigenschaft THOROUGH_PROBE_INTERVAL angegeben. Daher wird das Ruhen
   # des Testsignals auf eine Dauer von THOROUGH_PROBE_INTERVAL eingestellt.
   sleep $PROBE_INTERVAL

   # nslookup-Befehl für die IP-Adresse des DNS ausführen.
```

```
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
          > $DNSPROBEFILE 2>&1

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

   # Sicherstellen, dass die Antwort auf nslookup vom HA-DNS-
   # Server und nicht von einem anderen in der /etc/resolv.conf-Datei
   # genannten Namensserver stammt.
   if [ $probefail -eq 0 ]; then
# Namen des Servers abrufen,m der auf die nslookup-Abfrage geantwortet hat.
   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
```

Abwägen von Neustart und Failover

Wenn die *probefail*-Variable ungleich 0 (Erfolg) ist, bedeutet dies, dass die Zeitüberschreitung für den `nslookup`-Befehl abgelaufen war oder dass die Antwort von einem anderen Server als dem Beispieldienst-DNS kam. In beiden Fällen funktioniert der DNS-Server nicht wie erwartet, und der Fehler-Monitor ruft die Funktion `decide_restart_or_failover()` auf, um festzulegen, ob der Datendienst lokal neu gestartet wird oder ob RGM aufgefordert wird, den Datendienst auf einen anderen Knoten zu verschieben. Wenn die *probefail*-Variable 0 ist, wird eine Meldung generiert, die besagt, dass das Testsignal erfolgreich war.

```
   if [ $probefail -ne 0 ]; then
         decide_restart_or_failover
   else
         logger -p ${SYSLOG_FACILITY}.err\
         -t [$SYSLOG_TAG]\
         "${ARGV0} Testsignal für Ressource HA-DNS erfolgreich"
   fi
```

Die Funktion `decide_restart_or_failover()` verwendet ein Zeitfenster (`Retry_interval`) und einen Fehlschlagzähler (`Retry_count`), um festzulegen, ob DNS lokal neu gestartet oder RGM aufgefordert wird, den Datendienst auf einen anderen Knoten zu verschieben. Sie implementiert den folgenden bedingten Code (siehe die Codeauflistung für `decide_restart_or_failover()` in „PROBE-Programm" auf Seite 290).

- Wenn dies der erste Fehlschlag ist, wird der Datendienst neu gestartet. Es wird eine Fehlermeldung protokolliert und der Zähler in der `retries`-Variable weitergedreht.

- Wenn es sich nicht um den ersten Fehlschlag handelt, aber das Zeitfenster überschritten wurde, wird der Datendienst neu gestartet. Es wird eine Fehlermeldung protokolliert, der Zähler zurückgesetzt und das Fenster verschoben.
- Wenn das Zeitfenster noch nicht abgelaufen ist und der Wiederholversuchszähler überschritten wurde, wird ein Failover auf einen anderen Knoten ausgeführt. Wenn das Failover fehlschlägt, wird ein Fehler protokolliert und das Testsignalprogramm mit Status 1 (Fehlschlag) beendet.
- Wenn das Zeitfenster noch nicht abgelaufen ist und der Wiederholversuchszähler nicht überschritten wurde, wird der Datendienst neu gestartet. Es wird eine Fehlermeldung protokolliert und der Zähler in der `retries`-Variable weitergedreht.

Wenn die Anzahl der Neustarts während des Zeitintervalls den Grenzwert erreicht, fordert die Funktion bei RGM das Verschieben des Datendienstes auf einen anderen Knoten an. Wenn die Anzahl der Neustarts den Grenzwert noch nicht erreicht hat, bzw. wenn das Zeitintervall abgelaufen ist und die Zählung von vorn beginnt, versucht die Funktion, DNS auf demselben Knoten neu zu starten. Beachten Sie Folgendes für diese Funktion:

- Das `gettime`-Dienstprogramm wird zum Verfolgen der Zeit zwischen Neustarts verwendet. Dabei handelt es sich um ein C-Programm, das im (`RT_basedir`-) Verzeichnis residiert.
- Die systemdefinierten Ressourceneigenschaften `Retry_count` und `Retry_interval` legen die Anzahl der Neustartversuche und das Zeitintervall für die Zählung fest. Der Standardwert für diese Eigenschaften in der RTR-Datei liegt bei 2 Versuchen in einem Zeitraum von 5 Minuten (300 Sekunden). Der Cluster-Verwalter kann diese Werte jedoch ändern.
- Die `restart_service()`-Funktion wird aufgerufen, um zu versuchen, den Datendienst auf demselben Knoten neu zu starten. Weitere Informationen zu dieser Funktion finden Sie im nächsten Abschnitt, „Neustarten des Datendienstes" auf Seite 113.
- Die API-Funktion `scha_control()` bringt die Ressourcengruppe, die den Beispieldatendienst enthält, mit der Option `GIVEOVER` offline und auf einem anderen Knoten wieder online.

Neustarten des Datendienstes

Die `restart_service()`-Funktion wird von `decide_restart_or_failover()` aufgerufen, um einen Neustart des Datendienstes auf dem gleichen Knoten zu versuchen. Diese Funktion führt folgende Aufgaben aus.

- Sie stellt fest, ob der Datendienst noch unter PMF registriert ist. Wenn der Dienst noch registriert ist, geht die Funktion folgendermaßen vor:
  - Sie ruft den `Stop`-Methodennamen und den `Stop_timeout`-Wert für den Datendienst ab.

- Sie verwendet `hatimerun`, um die `Stop`-Methode für den Datendienst zu starten, indem sie den `Stop_timeout`-Wert übergibt.
- Wenn der Datendienst erfolgreich gestoppt wurde, ruft sie den `Start`-Methodennamen und den `Start_timeout`-Wert für den Datendienst ab.
- Sie verwendet `hatimerun`, um die `Start`-Methode für den Datendienst zu starten, indem sie den `Start_timeout`-Wert übergibt.

- Wenn der Datendienst nicht mehr unter PMF registriert ist, bedeutet dies, dass er die maximale Anzahl zulässiger Wiederholversuche unter PMF überschritten hat. Daher wird die `scha_control()`-Funktion mit der `GIVEOVER`-Option aufgerufen, um für den Datendienst ein Failover auf einen anderen Knoten auszuführen.

```
function restart_service
{

        # Um den Datendienst neu zu starten, wird zunächst überprüft, ob
        # der Datendienst selbst noch unter PMF registriert ist.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Da das Tag fur den Datendienst noch unter PMF registriert ist,
                # den Datendienst zunächst stoppen und dann wieder neu starten.

                # Stop-Methodenname und STOP_TIMEOUT-Wert für diese
                # Ressource abrufen.
                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-Methode fehlgeschlagen."
                        return 1
                fi

                # START-Methodenname und START_TIMEOUT-Wert für diese
                # Ressource abrufen.
                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-Methode fehlgeschlagen."
```

```
                return 1
            fi


        else
            # Das Fehlen des TAG für den Datendienst weist darauf
            # hin, dass der Datendienst bereits die maximale Anzahl
            # der unter PMF zulässigen Wiederholversuche überschritten hat.
            # Daher nicht versuchen, den Datendienst noch einmal neu
            # zu starten, sondern ein Failover auf einen anderen Knoten im
            # Cluster versuchen.
            scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                -R $RESOURCE_NAME
        fi

        return 0
}
```

Testsignal-Beendigungsstatus

Das `PROBE`-Programm des Datendienstes wird mit Fehlschlag beendet, wenn sowohl die lokalen Neustartversuche als auch die Failover-Versuche auf einen anderen Knoten fehlgeschlagen sind. Es protokolliert die Meldung “Failover-Versuch fehlgeschlagen”.

`Monitor_start`-Methode

RGM ruft die `Monitor_start`-Methode auf, um die `dns_probe`-Methode zu starten, nachdem der Beispieldatendienst online gebracht wurde.

Dieser Abschnitt beschreibt die Hauptteile der `Monitor_start`-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die `parse_args()`-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden” auf Seite 98 beschrieben).

Eine vollständige Auflistung der `Monitor_start`-Methode finden Sie unter „`Monitor_start`-Methode” auf Seite 295.

Überblick über `Monitor_start`

Diese Methode verwendet PMF (`pmfadm`) zum Starten des Testsignals.

Starten des Testsignals

Die `Monitor_start`-Methode ruft den Wert der `RT_basedir`-Eigenschaft ab, um den vollständigen Pfadnamen für das `PROBE`-Programm zu erstellen. Diese Methode startet das Testsignal mit der Endloswiederholversuchs-Option von `pmfadm` (-n -1, -t -1). Wenn also das Testsignal nicht startet, versucht PMF eine endlose Anzahl von Malen während eines endlosen Zeitraums zu starten.

```
# Wert der RT_BASEDIR-Eigenschaft der Ressource
# abrufen, um herauszufinden, wo das Testsignalprogramm residiert.
RT_BASEDIR=‘scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

# Testsignal für den Datendienst unter PMF starten. Option der endlosen Wiederholversuche
# zum Starten des Testsignals verwenden. Ressourcenname, -typ und -gruppe
# an das Testsignalprogramm übergeben.
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
   $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME
```

`Monitor_stop`-Methode

RGM ruft die `Monitor_stop`-Methode auf, um die Ausführung von `dns_probe` zu stoppen, wenn der Beispieldatendienst offline gebracht wird.

Dieser Abschnitt beschreibt die Hauptteile der `Monitor_stop`-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die `parse_args()`-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden” auf Seite 98 beschrieben).

Eine vollständige Auflistung der `Monitor_stop`-Methode finden Sie unter „`Monitor_stop`-Methode” auf Seite 297.

Überblick über `Monitor_stop`

Diese Methode verwendet PMF (`pmfadm`), um festzustellen, ob das Testsignal läuft und es gegebenenfalls zu stoppen.

Stoppen des Monitors

Die `Monitor_stop`-Methode verwendet `pmfadm -q`, um festzustellen, ob das Testsignal läuft und es gegebenenfalls unter Verwendung von `pmfadm -s` zu stoppen. Wenn das Testsignal bereits gestoppt wurde, wird die Methode dennoch mit Erfolg beendet, was die Idempotenz der Methode sicherstellt.

```
# Feststellen, ob der Monitor läuft, und ggf. Beenden erzwingen.
if pmfadm -q $PMF_TAG; then
   pmfadm -s $PMF_TAG KILL
   if [ $? -ne 0 ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor für Ressource $RESOURCE_NAME"\
             "konnte nicht gestoppt werden"
           exit 1
```

```
   else
         # Monitor konnte erfolgreich gestoppt werde. Meldung protokollieren.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor für " $RESOURCE_NAME \
            " erfolgreich gestoppt"
   fi
fi
exit 0
```

Achtung – Stellen Sie sicher, dass das `KILL`-Signal mit `pmfadm` verwendet wird, um das Testsignal zu stoppen, und nicht ein maskierbares Signal wie `TERM`. Andernfalls kann die `Monitor_stop`-Methode für unbegrenzte Zeit hängen, bis schließlich die Zeit überschritten ist. Der Grund für dieses Problem ist, dass die `PROBE`-Methode `scha_control()` aufruft, wenn der Datendienst neu gestartet oder ein Failover ausgeführt werden muss. Wenn `scha_control()` die `Monitor_stop`-Methode als Teil des Prozesses zum Offline-bringen des Datendienstes aufruft und `Monitor_stop` ein maskierbares Signal verwendet, hängt die Methode, während sie darauf wartet, dass `scha_control()` endet, während `scha_control()` hängt und darauf wartet, dass `Monitor_stop` endet.

`Monitor_stop`-Beendigungsstatus

Die `Monitor_stop`-Methode protokolliert eine Fehlermeldung, wenn sie die `PROBE`-Methode nicht stoppen kann. RGM versetzt den Beispieldatendienst in den `MONITOR_FAILED`-Zustand auf dem Primärknoten, was den Knoten zum Absturz bringen kann.

`Monitor_stop` darf erst beendet werden, wenn das Testsignal gestoppt wurde.

`Monitor_check`-Methode

RGM ruft die `Monitor_check`-Methode immer dann auf, wenn die `PROBE`-Methode versucht, für die Ressourcengruppe, die den Datendienst enthält, ein Failover auf einen neuen Knoten auszuführen.

Dieser Abschnitt beschreibt die Hauptteile der `Monitor_check`-Methode für die Beispielanwendung. Die Funktionalität, die allen Rückmeldemethoden gemeinsam ist, wird nicht beschrieben, wie zum Beispiel die `parse_args()`-Funktion und das Abrufen von syslog (in „Bereitstellen gemeinsamer Funktionalität für alle Methoden" auf Seite 98 beschrieben).

Eine vollständige Auflistung der `Monitor_check`-Methode finden Sie unter „`Monitor_check`-Methode" auf Seite 299.

Die `Monitor_check`-Methode muss so implementiert werden, dass sie keine Konflikte mit anderen, gleichzeitig laufenden Methoden bewirkt.

Die Monitor_check-Methode ruft die Validate-Methode auf, um zu überprüfen, ob das DNS-Konfigurationsverzeichnis auf dem neuen Knoten verfügbar ist. Die Confdir-Erweiterungseigenschaft zeigt auf das DNS-Konfigurationsverzeichnis. Daher ruft Monitor_check den Pfad und Namen für die Validate-Methode und den Wert für die Confdir-Methode ab. Dieser Wert wird an Validate übergeben, wie in der folgenden Auflistung gezeigt.

```
# Den vollständigen Pfad für die Validate-Methode aus
# der RT_BASEDIR-Eigenschaft des Ressourcentyps abrufen.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAMÈ

# Namen der Validate-Methode für diese Ressource abrufen.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Wert der Confdir-Eigenschaft abrufen, um den Datendienst zu starten.
# Mithilfe des eingegebenen Ressourcennamens und der Ressourcengruppe
# den Confdir-Wert abrufen, der bei Hinzufügen der Ressource eingestellt wurde.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
 -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get gibt sowohl den Typ als auch den Wert von Erweiterungs-
# eigenschaften zurück. awk verwenden, um nur den Wert der Erweiterungs-
# eigenschaft abzurufen.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Validate-Methode aufrufen, damit für den Datendienst ein erfolgreiches
# Failover auf den neuen Knoten ausgeführt werden kann.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

Unter „Validate-Methode" auf Seite 119 wird gezeigt, wie die Beispielanwendung die Eignung eines Knotens für das Hosten des Datendienstes überprüft.

Bearbeiten von Eigenschaftsaktualisierungen

Der Beispieldatendienst implementiert die Methoden Validate und Update, um die Eigenschaftsaktualisierung durch einen Cluster-Verwalter bearbeiten zu können.

`Validate`-Methode

RGM ruft die `Validate`-Methode auf, wenn eine Ressource erstellt wird und wenn die Eigenschaften einer Ressource bzw. deren Gruppe durch einen Verwaltungsbefehl aktualisiert werden. RGM ruft `Validate` auf, bevor die Erstellung bzw. Aktualisierung angewendet wird. Ein Fehlerbeendigungscode der Methode auf einem Knoten führt zum Abbruch der Erstellung bzw. Aktualisierung.

RGM ruft `Validate` nur dann auf, wenn Ressourcen- bzw. Gruppeneigenschaften über eine Verwaltungsaktion geändert werden, und nicht, wenn RGM Eigenschaften einstellt oder wenn ein Monitor die Ressourceneigenschaften `Status` und `Status_msg` einstellt.

Hinweis – Die `Monitor_check`-Methode ruft auch ausdrücklich jedes Mal dann die `Validate`-Methode auf, wenn die `PROBE`-Methode versucht, ein Failover für den Datendienst auf einen neuen Knoten auszuführen.

Überblick über `Validate`

RGM ruft `Validate` mit zusätzlichen Argumenten zu denjenigen auf, die von anderen Methoden übergeben wurden, einschließlich der aktualisierten Eigenschaften und Werte. Daher muss diese Methode im Beispieldatendienst eine andere `parse_args()`-Funktion implementieren, um die zusätzlichen Argumente zu bearbeiten.

Die `Validate`-Methode im Beispieldatendienst überprüft eine einzige Eigenschaft, die `Confdir`-Erweiterungseigenschaft. Diese Eigenschaft zeigt auf das DNS-Konfigurationsverzeichnis, das für einen erfolgreichen DNS-Betrieb entscheidend ist.

Hinweis – Da das Konfigurationsverzeichnis nicht geändert werden kann, während DNS läuft, wird die `Confdir`-Eigenschaft in der RTR-Datei als `TUNABLE = AT_CREATION` deklariert. Daher wird die `Validate`-Methode nie aufgerufen, um die `Confdir`-Eigenschaft nach einer Aktualisierung zu überprüfen, sondern nur bei Erstellung der Datendienstressource.

Wenn `Confdir` eine der Eigenschaften ist, die RGM an `Validate` übergibt, ruft die `parse_args()`-Funktion deren Wert ab und speichert ihn. `Validate` überprüft daraufhin, ob auf das Verzeichnis, auf das der neue Wert von `Confdir` zeigt, zugegriffen werden kann, und ob die `named.conf`-Datei in diesem Verzeichnis vorhanden ist und Daten enthält.

Wenn die `parse_args()`-Funktion den Wert von `Confdir` nicht aus den von RGM übergebenen Befehlszeilenargumenten abrufen kann, versucht `Validate` dennoch, die `Confdir`-Eigenschaft zu validieren. `Validate` verwendet `scha_resource_get`

`()`, um den Wert von `Confdir` aus der statischen Konfiguration abzurufen. Dann führt die Funktion die gleichen Prüfungen aus, um zu überprüfen, ob auf das Konfigurationsverzeichnis zugegriffen werden kann und keine leere `named.conf`-Datei enthält.

Wenn `Validate` mit Fehlschlag beendet wird, schlägt die Aktualisierung bzw. Erstellung aller Eigenschaften fehl, nicht nur diejenige von `Confdir`.

Analysefunktion der `Validate`-Methode

RGM übergibt an die `Validate`-Methode einen anderen Satz Parameter als an die anderen Rückmeldemethoden. Daher benötigt `Validate` eine andere Funktion für die Argumentenanalyse als die anderen Methoden. In der Online-Dokumentation unter `rt_callbacks`(1HA) finden Sie weitere Informationen zu den an `Validate` und die anderen Rückmeldemethoden übergebenen Parametern. Im Folgenden wird die `Validate` `parse_args()`-Funktion gezeigt.

```
#############################################################################
# Validate-Argumente analysieren.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Name der DNS-Ressource.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Name der Ressourcengruppe, in der die
                  # Ressource konfiguriert ist.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
         T)
                  # Name des Ressourcentyps.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
         r)
                  # Die Methode greift auf keine systemdefinierten
                  # Eigenschaften zu, so dass diese Option nicht besteht
                  ;;
         g)
                  # Die Methode greift auf keine Ressourcengruppen-
                  # eigenschaften zu, so dass diese Option nicht besteht
                  ;;
         c)
                  # Gibt an, dass die Validate-Methode bei Erstellen der Ressource
                  # aufgerufen wird, so dass dieses Flag nicht besteht.
                  ;;
```

```
        u)
                # Gibt die Aktualisierung einer Eigenschaft an, wenn die Ressource
                # bereits vorhanden ist. Wenn die Confdir-Eigenschaft aktualisiert wird,
                # sollte Confdir in den Befehlszeilenargumenten stehen. Andernfalls
                # muss die Methode ausdrücklich mit scha_resource_get danach
                # suchen.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Erweiterungseigenschaftsliste. Eigenschafts- und Wertepaare mit
                # "=" als Trennzeichen voneinander trennen.
                PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                VAL=`echo $OPTARG | awk -F= '{print $2}'`
                # Wenn die Confdir-Erweiterungseigenschaft an der Befehls-
                # zeile gefunden wird, Wert festhalten.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "FEHLER: Option $OPTARG unbekannt"
                exit 1
                ;;
        esac
    done
}
```

Genau wie die `parse_args()`-Funktion für andere Methoden verfügt diese Funktion über ein Flag (`R`) zum Erfassen des Ressourcennamens, (`G`) zum Erfassen des Ressourcengruppennamens und (`T`) zum Erfassen des Ressourcentyps, die von RGM übergeben werden.

Das `r`-Flag (für systemdefinierte Eigenschaften), `g`-Flag (für Ressourcengruppeneigenschaften) und das `c`-Flag (das angibt, dass die Validierung bei Ressourcenerstellung stattfindet) werden ignoriert, da diese Methode aufgerufen wird, um eine Erweiterungseigenschaft bei Aktualisierung der Ressource zu validieren.

Das `u`-Flag stellt den Wert der `UPDATE_PROPERTY`-Shell-Variablen auf `1` (`TRUE`) ein. Das `x`-Flag erfasst die Namen und Eigenschaften der aktualisierten Eigenschaften. Wenn `Confdir` eine der aktualisierten Eigenschaften ist, wird ihr Wert in der *CONFDIR*-Shell-Variablen abgelegt, und die Variable *CONFDIR_FOUND* wird auf `1` (`TRUE`) eingestellt.

Validieren von `Confdir`

In der `MAIN`-Funktion setzt `Validate` zunächst die *CONFDIR*-Variable auf die leere Zeichenkette sowie *UPDATE_PROPERTY* und *CONFDIR_FOUND* auf 0.

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

Anschließend ruft `Validate` die `parse_args()`-Funktion auf, um die von RGM übergebenen Argumente zu analysieren.

```
parse_args "$@"
```

Dann prüft `Validate`, ob `Validate` als Ergebnis der Aktualisierung von Eigenschaften aufgerufen wird und ob die `Confdir`-Erweiterungseigenschaft an der Befehlszeile stand. Dann überprüft `Validate`, ob die `Confdir`-Eigenschaft einen Wert hat. Andernfalls wird mit Fehlerstatus und einer Fehlermeldung beendet.

```
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

# Überprüfen, ob die Confdir-Eigenschaft einen Wert hat. Andernfalls Fehlschlag,
# und mit Status 1 beenden
if [[ -z $CONFDIR ]]; then
         logger -p ${SYSLOG_FACILITY}.err \
            "${ARGV0} Validate-Method für Ressource "$RESOURCE_NAME " fehlgeschlagen"
         exit 1
fi
```

Hinweis – Der obige Code prüft vor allem, ob `Validate` als Ergebnis einer Aktualisierung aufgerufen wird (`$UPDATE_PROPERTY == 1`) und ob die Eigenschaft *nicht* an der Befehlszeile gefunden wurde (`CONFDIR_FOUND == 0`); in diesem Fall wird der vorhandene Wert von `Confdir` mithilfe von `scha_resource_get()` abgerufen. Wenn `Confdir` an der Befehlszeile gefunden wurde (`CONFDIR_FOUND == 1`), stammt der Wert für *CONFDIR* von der `parse_args()`-Funktion, nicht von `scha_resource_get()`.

Die `Validate`-Methode verwendet dann den Wert von *CONFDIR*, um zu überprüfen, ob auf das Verzeichnis zugegriffen werden kann. Wenn kein Zugriff möglich ist, protokolliert `Validate` eine Fehlermeldung und wird mit Fehlerstatus beendet.

```
# Prüfen, ob Zugriff auf $CONFDIR möglich ist.
if [ ! -d $CONFDIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Verzeichnis $CONFDIR fehlt oder ist nicht eingehängt."
   exit 1
fi
```

Vor dem Validieren der Aktualisierung der `Confdir`-Eigenschaft führt `Validate` eine letzte Prüfung durch, um festzustellen, ob die `named.conf`-Datei vorhanden ist. Andernfalls protokolliert die Methode eine Fehlermeldung und wird mit Fehlerstatus beendet.

```
# Prüfen, od die named.conf-Datei im Confdir-Verzeichnis vorhanden ist
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Datei $CONFDIR/named.conf fehlt oder ist leer"
         exit 1
fi
```

Wenn diese letzte Prüfung erfolgreich war, protokolliert `Validate` eine Erfolgsmeldung und wird mit Erfolgsstatus beendet.

```
# Meldung protokollieren, die angibt, dass die Validate-Methode erfolgreich war.
logger -p ${SYSLOG_FACILITY}.err \
   -t [$SYSLOG_TAG] \
   "${ARGV0} Validate-Methode für Ressource "$RESOURCE_NAME \
   " erfolgreich beendet"

exit 0
```

`Validate`-Beendigungsstatus

Wenn `Validate` mit Erfolg (0) endet, wird `Confdir` mit dem neuen Wert erstellt. Wenn `Validate` mit Fehlschlag (1) endet, werden weder `Confdir` noch irgendeine andere Eigenschaft erstellt, und eine Meldung mit Angabe der Gründe wird an den Cluster-Verwalter gesendet.

`Update`-Methode

RGM ruft die `Update`-Methode auf, um eine laufende Ressource darüber zu benachrichtigen, dass ihre Eigenschaften geändert wurden. RGM ruft `Update` auf, nachdem eine Verwaltungsaktion die Eigenschaften einer Ressource bzw. deren Gruppe erfolgreich eingestellt hat. Die Methode wird auf denjenigen Knoten aufgerufen, auf denen die Ressource online ist.

Überblick über `Update`

Die `Update`-Methode aktualisiert keine Eigenschaften — das ist Aufgabe von RGM. Stattdessen benachrichtigt sie laufende Prozesse davon, dass eine Aktualisierung stattgefunden hat. Der einzige im Beispieldatendienst von einer Eigenschaftsaktualisierung betroffene Prozess ist der Fehler-Monitor. Daher wird dieser Prozess von der `Update`-Methode gestoppt und neu gestartet.

Die `Update`-Methode muss überprüfen, ob der Fehler-Monitor läuft und dann dessen Beenden mithilfe von `pmfadm` erzwingen. Die Methode ruft den Pfad des Testsignalprogramms ab, das den Fehler-Monitor implementiert, und startet ihn dann mithilfe von `pmfadm` neu.

Stoppen des Monitors mit `Update`

Die `Update`-Methode verwendet `pmfadm -q`, um zu überprüfen, ob der Monitor läuft. Wenn dies der Fall ist, erzwingt sie das Beenden mit `pmfadm -s TERM`. Wenn der Monitor erfolgreich beendet wurde, wird eine entsprechende Meldung an den Verwaltungsbenutzer gesendet. Wenn der Monitor nicht gestoppt werden kann, wird `Update` mit Fehlerstatus beendet und sendet eine Fehlermeldung an den Verwaltungsbenutzer.

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

# Beenden des bereits laufenden Monitors erzwingen
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
                 "${ARGV0} Monitor konnte nicht gestoppt werden"
       exit 1
    else
    # DNS konnte erfolgreich gestoppt werden. Meldung protokollieren.
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                 "Monitor für HA-DNS erfolgreich gestoppt"
    fi
```

Neustarten des Monitors

Um den Monitor neu zu starten, muss die `Update`-Methode das Skript finden, mit dem das Testsignalprogramm implementiert wird. Das Testsignalprogramm residiert im Basisverzeichnis des Datendienstes, auf das die `RT_basedir`-Eigenschaft zeigt. `Update` ruft den Wert von `RT_basedir` ab und speichert ihn in der `RT_BASEDIR`-Variablen, wie im Folgenden gezeigt.

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

Dann verwendet `Update` den Wert von `RT_BASEDIR` mit `pmfadm`, um das dns_probe-Programm neu zu starten. Wenn dieser Vorgang erfolgreich ist, wird `Update` mit Erfolg beendet und sendet eine entsprechende Meldung an den Verwaltungsbenutzer. Wenn `pmfadm` das Testsignalprogramm nicht starten kann,wird `Update` mit Fehlerstatus beendet und protokolliert eine Fehlermeldung.

`Update`-Beendigungsstatus

Ein Fehlschlagen der `Update`-Methode versetzt die Ressource in einen Zustand “Aktualisierung fehlgeschlagen”. Dieser Zustand hat keine Auswirkung auf die RGM-Verwaltung der Ressource, gibt jedoch den Fehlschlag der Aktualisierungsaktion über die syslog-Funktion an Verwaltungstools an.

KAPITEL 6

DSDL

Dieses Kapitel bietet einen Überblick über die Anwendungsprogrammierschnittstellen, aus denen sich die DSDL (Data Service Development Library, Datendienst-Entwicklungsbibliothek) zusammensetzt. Die DSDL ist in der `libdsdev.so`-Bibliothek implementiert und im Sun Cluster-Paket enthalten.

Dieses Kapitel behandelt die folgenden Themen:

Überblick über die DSDL

Die DSDL-API befindet sich auf einer Ebene über der RMAPI. Sie übersteuert die RMAPI jedoch nicht, sondern kapselt sie ein und erweitert die RMAPI-Funktionalität. Die DSDL vereinfacht die Datendienstentwicklung, indem sie vorentwickelte Lösungen für bestimmte Sun Cluster-Integrationsfragen bereitstellt. Damit können Sie einen Großteil der für die Entwicklung erforderlichen Zeit für Fragen der Hochverfügbarkeit und Skalierbarkeit Ihrer Anwendung aufwenden, ohne sich lange mit der Integration der Start-, Schließ- und Monitor-Verfahren für die Anwendung mit Sun Cluster aufzuhalten.

Verwalten von Konfigurationseigenschaften

Alle Rückmeldemethoden benötigen Zugriff auf die Konfigurationseigenschaften. Die DSDL unterstützt den Zugriff auf die Eigenschaften folgendermaßen:

- Initialisieren der Umgebung,
- Bereitstellen eines Satzes praktischer Funktionen zum Abrufen von Eigenschaftswerten.

Die `scds_initialize`-Funktion, die zu Beginn jeder Rückmeldemethode aufgerufen werden muss, führt folgende Aktionen aus:

- Sie prüft und verarbeitet die Befehlszeilenargumente (`argc` und `argv[]`), die RGM an die Rückmeldemethode übergibt. Das erspart Ihnen das Schreiben einer Befehlszeilen-Analysefunktion.
- Sie konfiguriert interne Datenstrukturen, die von anderen DSDL-Funktionen verwendet werden können. So speichern zum Beispiel die Funktionen, die Eigenschaftswerte von RGM abrufen, die Werte in diesen Strukturen. Auch Werte aus der Befehlszeile, die Vorrang vor den von RGM abgerufenen Werte haben, werden in den Datenstrukturen gespeichert.

Hinweis – Für die `Validate`-Methode analysiert scds_initialize die Eigenschaftswerte, die an die Befehlszeile übergeben werden. Damit ersparen Sie sich das Schreiben einer Analysefunktion für `Validate`.

Die `scds_initialize`-Funktion initialisiert auch die Protokollierumgebung und validiert die Testsignaleinstellungen des Fehler-Monitors.

Die DSDL stellt Funktionssätze zum Abrufen von Ressourcen-, Ressourcentyp- und Ressourcengruppeneigenschaften sowie von häufig verwendeten Erweiterungseigenschaften bereit. Diese Funktionen wenden folgende Konventionen für den standardmäßigen Zugriff auf Eigenschaften an:

- Jede Funktion übernimmt nur ein Handle-Argument (zurückgegeben von `scds_initialize`).
- Jede Funktion entspricht einer bestimmten Eigenschaft. Der Rückgabewerttyp der Funktion entspricht dem Typ des abgerufenen Eigenschaftswerts.
- Funktionen geben keine Fehler zurück, da die Werte von `scds_initialize` vorberechnet wurden. Funktionen rufen Werte von RGM ab, es sei denn, es wird ein neuer Wert an die Befehlszeile übergeben.

Starten und Stoppen eines Datendienstes

Eine `Start`-Methode hat die Aufgabe, die erforderlichen Aktionen zum Starten eines Datendienstes auf einem Cluster-Knoten auszuführen. In der Regel umfasst dieser Vorgang das Abrufen der Ressourceneigenschaften, Suchen der anwendungsspezifischen ausführbaren Dateien und Konfigurationsdateien sowie das Starten der Anwendung über die entsprechenden Befehlszeilenargumente.

Die `scds_initialize`-Funktion ruft die Ressourcenkonfiguration ab. Die `Start`-Methode kann die bereitgestellten Eigenschaftsfunktionen zum Abrufen der Werte spezifischer Eigenschaften wie `Confdir_list` verwenden, welche die Konfigurationsverzeichnisse und -dateien für die zu startende Anwendung identifizieren.

Eine `Start`-Methode kann `scds_pmf_start` aufrufen, um eine Anwendung unter Steuerung durch PMF (Process Monitor Facility) zu starten. Mit PMF können Sie die Überwachungsebene angeben, die auf den Prozess angewendet werden soll. Außerdem ermöglicht PMF den Neustart des Prozesses, falls er fehlschlagen sollte. Ein Beispiel für eine mit der DSDL implementierte `Start`-Methode finden Sie unter „`xfnts_start`-Methode" auf Seite 146.

Eine `Stop`-Methode muss idempotent sein, um mit Erfolg zu enden, auch wenn sie auf einem Knoten aufgerufen wird, auf dem die Anwendung nicht läuft. Wenn die `Stop`-Methode fehlschlägt, wird die zu stoppende Ressource in den `STOP_FAILED`-Zustand versetzt, was zu einem harten Neustart des Clusters führen kann.

Um einen `STOP_FAILED`-Zustand der Ressource zu vermeiden, muss die `Stop`-Methode die Ressource unbedingt stoppen. Die `scds_pmf_stop`-Funktion unternimmt in Phasen unterteilte Ressourcenstoppversuche. Sie versucht zunächst, die Ressource mit dem `SIGTERM`-Signal zu stoppen. Wenn dies nicht erfolgreich ist, wird ein `SIGKILL`-Signal verwendet. Weitere Details finden Sie unter `scds_pmf_stop`(3HA) .

Implementieren eines Fehler-Monitors

Die DSDL vereinfacht das Implementieren eines Fehler-Monitors erheblich, indem sie ein vordefiniertes Modell bereitstellt. Eine `Monitor_start`-Methode startet den Fehler-Monitor unter PMF-Steuerung, wenn die Ressource auf einem Knoten gestartet wird. Der Fehler-Monitor läuft in einer Schleife, solange die Ressource auf dem Knoten ausgeführt wird. Die Logik auf hoher Ebene eines DSDL-Fehler-Monitors ist Folgende:

- Die `scds_fm_sleep`-Funktion verwendet die Eigenschaft `Thorough_probe_interval`, um den Zeitabstand zwischen den Testsignalen zu bestimmen. Alle Anwendungsprozessfehler, die PMF während dieses Intervalls feststellt, führen zu einem Neustart der Ressource.
- Das Testsignal selbst gibt einen Wert zurück, der die Schwere der Fehler angibt. Die Werte reichen von `0` (kein Fehler) bis `100` (Totalfehlschlag).
- Der Rückgabewert des Testsignals wird an die `scds_action` -Funktion gesendet, die eine kumulative Fehlerhistorie innerhalb des Intervalls der `Retry_interval`-Eigenschaft unterhält.
- Die `scds_action`-Funktion bestimmt folgendermaßen, wie im Fall eines Fehlers verfahren wird.
  - Wenn der kumulative Fehler unter `100` liegt, geschieht nichts.
  - Wenn der kumulative Fehler `100` (Totalfehlschlag) erreicht, wird der Datendienst neu gestartet. Wenn `Retry_interval` überschritten ist, wird die Historie zurückgesetzt.
  - Wenn die Anzahl der Neustarts den in der `Retry_count`-Eigenschaft angegebenen Wert innerhalb der in `Retry_interval` angegebenen Zeit überschreitet, wird für den Datendienst ein Failover ausgeführt.

Zugreifen auf Netzwerkadressinformationen

Die DSDL stellt praktische Funktionen bereit, mit denen Netzwerkadressinformationen für Ressourcen und Ressourcengruppen zurückgegeben werden können. So ruft zum Beispiel `scds_get_netaddr_list` die Netzwerkadressressourcen ab, die von einer Ressource verwendet werden. Dadurch wird es dem Fehler-Monitor ermöglicht, die Anwendung zu testen.

Die DSDL enthält auch einen Satz Funktionen für die TCP-basierte Überwachung. In der Regel erstellen diese Funktionen eine einfache Socket-Verbindung mit einem Dienst, lesen und schreiben Daten an den Dienst und trennen dann die Verbindung mit dem Dienst. Das Testsignalergebnis kann an die DSDL-Funktion `scds_fm_action` gesendet werden, um über die auszuführende Aktion zu entscheiden.

Ein Beispiel für TCP-basierte Fehlerüberwachung finden Sie unter „`xfnts_validate`-Methode” auf Seite 161.

Beheben von Fehlern bei der Ressourcentypimplementierung

Die DSDL verfügt über integrierte Funktionen, mit denen Sie Datendienstfehler beheben können.

Das DSDL-Dienstprogramm `scds_syslog_debug()` bietet einen grundlegenden Rahmen, in dem der Ressourcentypimplementierung Fehlerbehebungsanweisungen hinzugefügt werden können. Die Fehlerbehebungs-*Ebene* (eine Zahl zwischen 1 und 9) kann pro Ressourcentypimplementierung und Cluster-Knoten dynamisch eingerichtet werden. Eine Datei mit dem Namen `/var/cluster/rgm/rt/`*RT-Name*`/loglevel`, die lediglich eine ganze Zahl zwischen 1 und 9 enthält, wird von allen Ressourcentyp-Rückmeldemethoden gelesen. Die DSDL-Routine `scds_initialize()` liest diese Datei und stellt die Fehlerbehebungsebene intern auf die angegebene Ebene ein. Die Standard-Fehlerbehebungsebene ist 0 und gibt an, dass der Datendienst keine Fehlerbehebungsmeldungen protokolliert.

Die Funktion `scds_syslog_debug()` verwendet die Rückgabe der Funktion `scha_cluster_getlogfacility()` mit einem Vorrang von `LOG_DEBUG`. Diese Fehlerbehebungsmeldungen können in `/etc/syslog.conf` konfiguriert werden.

Manche Fehlerbehebungsmeldungen können in Informationsmeldungen für den regulären Betrieb des Ressourcentyps umgewandelt werden (zum Beispiel mit dem Vorrang `LOG_INFO`). Dafür wird das Dienstprogramm `scds_syslog` verwendet. In der DSDL-Beispielanwendung in Kapitel 8 können Sie sehen, dass sehr viele `scds_syslog_debug`- und `scds_syslog`-Funktionen eingesetzt werden.

Aktivieren von hoch verfügbaren lokalen Dateisystemen

Der Ressourcentyp `HAStoragePlus` kann eingesetzt werden, um ein lokales Dateisystem in einer Sun Cluster-Umgebung hoch verfügbar zu machen. Die Partitionen des lokalen Dateisystems müssen sich auf globalen Plattengruppen befinden. Affinitäts-Switchover müssen aktiviert sein, und die Sun Cluster-Umgebung muss für Failover konfiguriert sein. Mit diesen Einstellungen kann der Benutzer jedes Dateisystem auf Multihostplatten für jeden Host verfügbar machen, der direkt mit den Multihostplatten verbunden ist. Für einige E/A-intensive Datendienste wird die Verwendung eines hoch verfügbaren lokalen Dateisystems dringend empfohlen. Unter „Enabling Highly Available Local File Systems" in *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* finden Sie Informationen zum Konfigurieren des `HAStoragePlus`-Ressourcentyps.

KAPITEL 7

Entwerfen von Ressourcentypen

Dieses Kapitel erläutert, wie die DSDL in der Regel beim Entwurf und der Implementierung von Ressourcentypen eingesetzt wird. Das Kapitel geht auch darauf ein, wie der Ressourcentyp entworfen werden muss, um die Ressourcenkonfiguration zu validieren sowie die Ressource zu starten, zu stoppen und zu überwachen. Zuletzt wird beschreiben, wie die DSDL beim Implementieren der Rückmeldemethoden des Ressourcentyps verwendet werden kann.

Weitere Informationen finden Sie in der Online-Dokumentation unter `rt_callbacks`(1HA).

Um diese Aufgaben ausführen zu können, benötigen Sie Zugriff auf die Eigenschaftseinstellungen der Ressource. Das DSDL-Dienstprogramm `scds_initialize()` vereinheitlicht den Zugriff auf die Ressourceneigenschaften. Diese Funktion ist dafür ausgelegt, zu Beginn jeder Rückmeldemethode aufgerufen zu werden. Die Dienstprogrammfunktion ruft alle Eigenschaften einer Ressource aus dem Cluster Framework ab und stellt sie der Familie der `scds_get`*name*`()` -Funktionen zur Verfügung.

Dieses Kapitel behandelt die folgenden Themen:

Die RTR-Datei

Die Ressourcentyp-Registrierungsdatei (RTR-Datei) ist ein wesentlicher Bestandteil eines Ressourcentyps. Diese Datei gibt die Details des Ressourcentyps für Sun Cluster an. Diese Details umfassen Informationen wie die Eigenschaften, die von der Implementierung benötigt werden, die Datentypen der Eigenschaften, die Standardwerte der Eigenschaften, den Dateisystempfad für die Rückmeldemethoden der Ressourcentypimplementierung sowie verschiedene Einstellungen für die systemdefinierten Eigenschaften.

Die Beispiel-RTR-Datei, die mit der DSDL geliefert wird, dürfte für die meisten Ressourcentypimplementierungen ausreichen. Sie brauchen lediglich einige grundlegende Elemente zu bearbeiten, wie zum Beispiel den Ressourcentypnamen und den Pfadnamen der Rückmeldemethoden für den Ressourcentyp. Wenn für die Implementierung des Ressourcentyps eine neue Eigenschaft benötigt wird, können Sie diese als Erweiterungseigenschaft in der Ressourcentyp-Registrierungsdatei (RTR-Datei) der Ressourcentypimplementierung deklarieren und dann über das DSDL-Dienstprogramm `scds_get_ext_property()` auf die neue Eigenschaft zugreifen.

Die `Validate`-Methode

Die `Validate`-Methode einer Ressourcentypimplementierung wird von RGM unter den folgenden beiden Bedingungen aufgerufen:

- Es wird eine neue Ressource des Ressourcentyps erstellt.
- Eine Eigenschaft der Ressource oder Ressourcengruppe wird aktualisiert.

Diese beiden Szenarios können durch die Befehlszeilenoption `-c` (creation, Erstellung) bzw. `-u` (update, Aktualisierung) unterschieden werden, die an die `Validate`-Methode der Ressource übergeben werden.

Die `Validate`-Methode wird auf jedem Knoten einer Knotengruppe aufgerufen, wobei die Knotengruppe durch den Wert der Ressourcentypeigenschaft `INIT_NODES` definiert wird. Wenn `INIT_NODES` auf `RG_PRIMARIES` eingestellt ist, wird `Validate` auf jedem Knoten aufgerufen, der Host (Primärknoten) der die Ressource enthaltenden Ressourcengruppe sein kann. Wenn `INIT_NODES` auf `RT_INSTALLED_NODES` eingestellt ist, wird `Validate` auf jedem Knoten aufgerufen, auf dem die Ressourcentypsoftware installiert ist. Das sind normalerweise alle Knoten im Cluster. Der Standardwert für `INIT_NODES` ist `RG_PRIMARIES` (siehe `rt_reg`(4). Zu dem Zeitpunkt, an dem die `Validate`-Methode aufgerufen wird, hat RGM die Ressource noch nicht erstellt (im Fall einer Rückmeldungserstellung) bzw. hat die

aktualisierten Werte der zu aktualisierenden Eigenschaften noch nicht angewendet (im Fall einer Rückmeldungsaktualisierung). Der Zweck der `Validate`-Rückmeldemethode einer Ressourcentypimplementierung ist es zu prüfen, ob die vorgeschlagenen Ressourceneinstellungen (angegeben durch die vorgeschlagenen Eigenschaftseinstellungen für die Ressource) für den Ressourcentyp akzeptabel sind.

Hinweis – Wenn Sie von `HAStoragePlus` verwaltete lokale Dateisysteme verwenden, wird mit dem Befehl `scds_hasp_check` der Zustand der `HAStoragePlus`-Ressource überprüft. Diese Informationen werden aus dem Zustand (online oder anderweitig) aller `SUNW.HAStoragePlus`(5)-Ressourcen abgerufen, von denen die Ressource abhängt, und zwar unter Verwendung der für die Ressource definierten Systemeigenschaften `Resource_dependencies` oder `Resource_dependencies_weak`. Unter `scds_hasp_check`(3HA) finden Sie eine vollständige Liste der vom `scds_hasp_check`-Aufruf zurückgegebenen Statuscodes.

Die DSDL-Funktion `scds_initialize()` verfährt in diesen Situationen folgendermaßen:

- Im Fall einer Ressourcenerstellung analysiert sie die vorgeschlagenen Ressourceneigenschaften, die an der Befehlszeile übergeben werden. Die vorgeschlagenen Werte von Ressourceneigenschaften stehen also dem Ressourcentypentwickler genau so zur Verfügung, als wäre die Ressource bereits im System erstellt worden.
- Im Fall einer Ressourcen- bzw. Ressourcengruppenaktualisierung werden die vorgeschlagenen Werte der vom Verwalter zu aktualisierenden Eigenschaften aus der Befehlszeile gelesen. Die restlichen Eigenschaften, deren Werte nicht aktualisiert werden, werden mithilfe der Ressourcenverwaltungs-API aus Sun Cluster gelesen. Ein Ressourcentypentwickler, der die DSDL verwendet, muss sich nicht um all diese Systemverwaltungsaufgaben zu kümmern. Die Validierung einer Ressource kann erfolgen, als ob dem Entwickler alle Ressourceneigenschaften zur Verfügung stünden.

Angenommen, die Funktion, welche die Validierung der Eigenschaften einer Ressource implementiert, wird als `svc_validate()` bezeichnet und verwendet die `scds_get_`*`Name`*`()`-Funktionsfamilie zum Untersuchen der Eigenschaft, die validiert werden soll. Ferner wird angenommen, dass eine akzeptable Ressourceneinstellung durch einen Rückgabecode von 0 von dieser Funktion dargestellt wird. Die `Validate`-Methode kann dann durch das folgende Codefragment dargestellt werden:

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

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

```
   return (1);   /* Initialisierungsfehler */
   }
   rc = svc_validate(handle);
   scds_close(&handle);
   return (rc);
}
```

Die Validierungsfunktion muss auch den Grund für den Fehlschlag der Ressourcenvalidierung protokollieren. Ohne auf weitere Einzelheiten einzugehen (im folgenden Kapitel sehen Sie eine realistischere Darstellung einer Validierungsfunktion) , kann ein einfaches Beispiel einer `svc_validate()`-Funktion folgendermaßen implementiert werden:

```
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);   /* Ungültige Ressourceneigenschaftseinstellung */
   }
   return (0);   /* Akzeptable Einstellung */
}
```

Der Ressourcentypentwickler muss sich also nur um die Implementierung der `svc_validate()`-Funktion kümmern. Ein typisches Beispiel für eine Ressourcentypimplementierung wäre die Prüfung, ob eine Anwendungskonfigurationsdatei mit dem Namen app.conf unter der `Confdir_list`-Eigenschaft vorhanden ist. Sie kann einfach durch einen `stat()`-Systemaufruf an den entsprechenden Pfadnamen, abgeleitet aus der `Confdir_list`-Eigenschaft, implementiert werden.

Die `Start`-Methode

Die `Start`-Rückmeldemethode einer Ressourcentypimplementierung wird von RGM auf einem bestimmten Cluster-Knoten aufgerufen, um die Ressource zu starten. Der Ressourcengruppenname, der Ressourcenname und der Ressourcentypname werden an die Befehlszeile übergeben. Die `Start`-Methode soll die erforderlichen Aktionen ausführen, um eine Datendienstressource auf dem Cluster-Knoten zu starten. In der Regel müssen hierfür die Ressourceneigenschaften abgerufen, die anwendungsspezifischen ausführbaren Dateien und/oder Konfigurationsdateien gesucht und die Anwendung mit den entsprechenden Befehlszeilenargumenten gestartet werden.

Bei Einsatz der DSDL wird die Ressourcenkonfiguration bereits von dem `scds_initialize()`-Dienstprogramm abgerufen. Die Startaktion für die Anwendung kann in einer `svc_start()`-Funktion enthalten sein. Eine weitere Funktion, `svc_wait()`, kann aufgerufen werden, um zu überprüfen, dass die Anwendung tatsächlich startet. Der vereinfachte Code für die `Start`-Methode sieht folgendermaßen aus:

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

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Initialisierungsfehler */
   }
   if (svc_validate(handle) != 0) {
   return (1);   /* Ungültige Einstellungen */
   }
   if (svc_start(handle) != 0) {
   return (1);   /* Start fehlgeschlagen */
   }
   return (svc_wait(handle));
}
```

Diese Start-Methodenimplementierung ruft `svc_validate()`auf, um die Ressourcenkonfiguration zu validieren. Falls sie fehlschlägt, entspricht entweder die Ressourcenkonfiguration nicht der Anwendungskonfiguration, oder es besteht zurzeit auf diesem Cluster-Knoten ein Problem bezüglich des Systems. Es kann zum Beispiel sein, dass ein für die Ressource erforderliches globales Dateisystem zurzeit auf diesem Cluster-Knoten nicht verfügbar ist. In diesem Fall ist jeder Startversuch der Ressource auf diesem Cluster-Knoten zwecklos. Stattdessen sollte RGM versuchen, die Ressource auf einem anderen Knoten zu starten. Beachten Sie jedoch, dass in obigem Beispiel davon ausgegangen wird, dass `svc_validate()` ausreichend konservativ ist, also nur die Ressourcen auf den Cluster-Knoten prüft, die für die Anwendung unbedingt erforderlich sind. Andernfalls kann der Start der Ressource auf allen Cluster-Knoten fehlschlagen, und sie wird in den `START_FAILED`-Zustand versetzt. In `scswitch`(1M) und *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* finden Sie eine Erläuterung dieses Zustands.

Die `svc_start()`-Funktion muss 0 für ein erfolgreiches Starten der Ressource auf dem Knoten zurückgeben. Wenn die Startfunktion auf ein Problem stößt, muss sie einen Wert ungleich Null zurückgeben. Bei Fehlschlagen dieser Funktion versucht RGM, die Ressource auf einem anderen Cluster-Knoten zu starten.

Um die DSDL so weit wie möglich zu nutzen, kann die `svc_start()`-Funktion das Dienstprogramm `scds_pmf_start()` verwenden, um die Anwendung unter PMF (Process Management Facility) zu starten. Dieses Dienstprogramm setzt auch die Fehlschlag-Rückmeldeaktion von PMF ein (siehe Aktions-Flag `-a` in `pmfadm`(1M)), um die Prozessfehlschlagerkennung zu implementieren.

Die Stop-Methode

Die Stop-Rückmeldemethode einer Ressourcentypimplementierung wird von RGM auf einem Cluster-Knoten aufgerufen, um die Anwendung zu stoppen. Für die Rückmeldesemantik der Stop-Methode gelten folgende Voraussetzungen:

- Die Stop-Methode muss *idempotent* sein, da die Stop-Methode von RGM auch dann aufgerufen werden kann, wenn die Start-Methode auf dem Knoten nicht erfolgreich beendet wurde. Daher muss die Stop-Methode erfolgreich sein (mit 0 beenden), auch wenn die Anwendung derzeit nicht auf dem Cluster-Knoten läuft und keine Aktion ausgeführt werden muss.
- Wenn die Stop-Methode des Ressourcentyps auf einem Cluster-Knoten fehlschlägt (nicht mit 0 endet), wird die zu stoppende Ressource in den STOP_FAILED-Zustand versetzt. Je nach der Failover_mode-Einstellung der Ressource kann dies zu einem harten Neustart des Cluster-Knotens durch RGM führen. Daher ist es wichtig, die Stop-Methode so zu entwerfen, dass sie versucht, die Anwendung wenn irgend möglich zu stoppen, bei Bedarf auch durch ein hartes Erzwingen der Anwendungsbeendigung (zum Beispiel mit SIGKILL), wenn andere Versuche fehlgeschlagen sind. Die Methode muss das Stoppen auch innerhalb eines bestimmten Zeitraums erzielen, da das Framework ein Überschreiten von Stop_timeout als Stopp-Fehlschlag wertet und die Ressource in den STOP_FAILED-Zustand versetzt.

Das DSDL-Dienstprogramm scds_pmf_stop() dürfte für die meisten Anwendungen ausreichend sein. Es versucht zunächst, die Anwendung weich zu stoppen (mithilfe von SIGTERM), wobei sie davon ausgeht, dass die Anwendung unter PMF über scds_pmf_start() gestartet wurde. Darauf folgt SIGKILL, um das Beenden des Prozesses zu erzwingen. Einzelheiten zu diesem Dienstprogramm finden Sie unter „PMF-Funktionen" auf Seite 217.

Dem bislang hier verwendeten Codemodell entsprechend, und ausgehend von der Annahme, dass die anwendungsspezifische Funktion zum Stoppen der Anwendung svc_stop() ist (ob die Implementierung von svc_stop() die scds_pmf_stop() -Methode verwendet, spielt hier keine Rolle, und hängt davon ab, ob die Methode unter PMF über die Start-Methode gestartet wurde), kann die Stop-Methode folgendermaßen implementiert werden:

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

Die svc_validate()-Methode wird in der Implementierung der Stop-Methode nicht verwendet, denn die Stop-Methode muss selbst bei einem aktuellen Systemproblem versuchen, die Anwendung auf diesem Knoten zu stoppen.

Die Monitor_start-Methode

RGM ruft die Monitor_start-Methode auf, um einen Fehler-Monitor für die Ressource zu starten. Fehler-Monitore überwachen die Fehlerfreiheit der Anwendung, die von der Ressource verwaltet wird. Ressourcentypimplementierungen implementieren einen Fehler-Monitor in der Regel als eigenen Dämon, der im Hintergrund ausgeführt wird. Die Monitor_start-Rückmeldemethode wird verwendet, um diesen Dämon mit den entsprechenden Argumenten zu starten.

Da beim Monitor-Dämon selbst ebenfalls Fehler auftreten können (er könnte zum Beispiel versagen und die Anwendung unüberwacht zurücklassen), wird empfohlen, den Monitor-Dämon über PMF zu starten. Das DSDL-Dienstprogramm scds_pmf_start() verfügt über integrierte Unterstützung für das Starten von Fehler-Monitoren. Dieses Dienstprogramm verwendet den relativen Pfadnamen (relativ zum RT_basedir für den Speicherort der Ressourcentyp-Rückmeldemethodenimplementierungen) des Monitor-Dämonprogramms. Es verwendet die von der DSDL verwalteten Erweiterungseigenschaften Monitor_retry_interval und Monitor_retry_count , um zu verhindern, dass der Dämon eine unbegrenzte Anzahl von Malen neu gestartet wird. Dabei ist die gleiche Befehlszeilensyntax, die für alle Rückmeldemethoden definiert ist (also -R *Ressource* -G *Ressourcengruppe* -T *Ressourcentyp*) auch für den Monitor-Dämon verbindlich, auch wenn dieser nie direkt von RGM aufgerufen wird. Dadurch wird es der Monitor-Dämonimplementierung selbst ermöglicht, das scds_initialize()-Dienstprogramm zum Konfigurieren der eigenen Umgebung einzusetzen. Die Hauptarbeit besteht im Entwerfen des Monitor-Dämons selbst.

Die Monitor_stop-Methode

RGM ruft die Monitor_stop-Methode zum Stoppen des Fehler-Monitor-Dämons auf, der über die Monitor_start-Methode gestartet wurde. Ein Fehlschlag dieser Rückmeldemethode wird genauso wie ein Fehlschlag der Stop-Methode behandelt; daher muss die Monitor_stop-Methode idempotent und robust wie die Stop-Methode sein.

Wenn Sie das Dienstprogramm scds_pmf_start() zum Starten des Fehler-Monitor-Dämons verwenden, müssen Sie scds_pmf_stop() verwenden, um ihn zu stoppen.

Die Monitor_check-Methode

Die Monitor_check-Rückmeldemethode für eine Ressource wird auf einem Knoten für die angegebene Ressource aufgerufen, um sicherzstellen, dass der Cluster-Knoten in der Lage ist, die Ressource zu unterstützen, dass heißt, dass die von der Ressource verwalteten Anwendungen auf dem Knoten erfolgreich ausgeführt werden können.). In der Regel muss dabei sichergestellt werden, dass die von der Anwendung benötigten Systemressourcen auch tatsächlich auf dem Cluster-Knoten verfügbar sind. Wie unter „Die Validate-Methode" auf Seite 132 erläutert, muss die vom Entwickler implementierte svc_validate()-Funktion zumindest diesen Punkt überprüfen.

Abhängig von der spezifischen Anwendung, die von der Ressourcentypimplementierung verwaltet wird, kann die Monitor_check-Methode für das Ausführen einiger weiterer Aufgaben geschrieben werden. Die Monitor_check-Methode muss so implementiert werden, dass sie nicht im Konflikt mit anderen gleichzeitig ausgeführten Methoden gerät. Wenn die Entwickler die DSDL einsetzen, wird empfohlen, für die Monitor_check-Methode die svc_validate()-Funktion zu nutzen, die eigens zur Implementierung der anwendungsspezifischen Validierung von Ressourceneigenschaften geschrieben wurde.

Die Update-Methode

RGM ruft die Update-Methode einer Ressourcentypimplementierung auf, um alle Änderngen anzuwenden, die vom Systemverwalter an der Konfiguration einer aktiven Ressource vorgenommen wurden. Die Update-Methode wird nur auf denjenigen Knoten aufgerufen, auf denen die Ressource aktuell online ist (falls zutreffend).

Die zuvor an der Ressourcenkonfiguration vorgenommenen Änderungen sind mit Sicherheit für die Ressourcentypimplementierung akzeptabel, da RGM die Validate-Methode des Ressourcentyps vor der Update-Methode ausführt. Die Validate-Methode wird aufgerufen, bevor die Ressourcen- bzw. Ressourcengruppeneigenschaften geändert werden, und die Validate-Methode kann die vorgeschlagenen Änderungen ablehnen. Die Update-Methode wird aufgerufen, nachdem die Änderungen angewendet wurden, damit die aktive (sich online befindende) Ressource auf die neuen Einstellungen aufmerksam gemacht wird.

Als Ressourcentypentwickler müssen Sie sich genau überlegen, welche Eigenschaften zur dynamischen Aktualisierung fähig sein sollen, und diese mit der TUNABLE = ANYTIME-Einstellung in der RTR-Datei markieren. In der Regel können

Sie angeben, dass jede Eigenschaft einer Ressourcentypimplementierung, die vom Fehler-Monitor-Dämon verwendet wird, dynamisch aktualisiert werden kann. Voraussetzung dafür ist, dass die `Update`-Methodenimplementierung zumindest den Monitor-Dämon neu startet.

Mögliche Kandidaten sind:

- `Thorough_Probe_Interval`
- `Retry_Count`
- `Retry_Interval`
- `Monitor_retry_count`
- `Monitor_retry_interval`
- `Probe_timeout`

Diese Eigenschaften beeinflussen, wie ein Fehler-Monitor-Dämon den Zustand des Dienstes überprüft, wie oft der Dämon Überprüfungen durchführt, das Historienintervall, das der Dämon zur Verfolgung von Fehlern verwendet, sowie die von PMF festgelegten Neustart-Schwellenwerte. Zum Implementieren dieser Eigenschaften wird in der DSDL das Dienstprogramm `scds_pmf_restart ()` bereitgestellt.

Wenn Sie eine Ressourceneigenschaft dynamisch aktualisieren müssen, die Änderung dieser Eigenschaft sich jedoch auf die laufende Anwendung auswirken könnte, müssen Sie die entsprechenden Aktionen implementieren, damit die Aktualisierungen der Eigenschaft für alle laufenden Instanzen dieser Anwendung korrekt angewendet werden. Derzeit ist dies nicht über die DSDL möglich. An `Update` werden die geänderten Eigenschaften nicht an der Befehlszeile übergeben (wie dies bei `Validate` der Fall ist).

Die `Init`-, `Fini`- und `Boot`-Methoden

Diese Methoden sind *einmalige Aktionsmethoden*, entsprechend der Definition in den Spezifikationen der Ressourcenverwaltungs-API. Die Beispielimplementierung für die DSDL zeigt die Verwendung dieser Methoden nicht. Alle Funktionen der DSDL stehen jedoch auch für diese Methoden zur Verfügung, wenn der Ressourcentypentwickler die Methoden benötigen sollte. In der Regel dienen die `Init`- und die `Boot`-Methode dem gleichen Zweck für eine Ressourcentypimplementierung, um eine *einmalige Aktion* zu implementieren. Die `Fini`-Methode führt in der Regel eine Aktion zum *Rückgängigmachen* der Aktionen einer `Init`- oder `Boot`-Methode aus.

Entwerfen des Fehler-Monitor-Dämons

Ressourcentypimplementierungen, welche die DSDL verwenden, verfügen in der Regel über einen Fehler-Monitor-Dämon mit folgenden Aufgaben:

- Periodische Überwachung der Fehlerfreiheit der verwalteten Anwendung. Dieser besondere Aspekt eines Monitor-Dämons hängt stark von der jeweiligen Anwendung ab und kann bei den einzelnen Ressourcentypen erhebliche Unterschiede aufweisen. Die DSDL verfügt über einige integrierte Dienstprogrammfunktionen, um Gesundheits-Checks für einfache, TCP-basierte Dienste auszuführen. Anwendungen mit ASCII-basierten Protokollen wie HTTP, NNTP, IMAP und POP3 können mithilfe dieser Dienstprogramme implementiert werden.
- Verfolgen der in der Anwendung aufgetretenen Probleme mithilfe der Ressourceneigenschaften `Retry_interval` und `Retry_count`. Bei Totalfehlschlag der Anwendung entscheiden, ob das PMF-Aktionsskript den Dienst neu starten soll oder ob die Anwendungsfehler so schnell aufeinander folgten, dass ein Failover in Betracht kommt. Die DSDL-Dienstprogramme `scds_fm_action()` und `scds_fm_sleep ()` sollen Ihnen bei der Implementierung dieses Mechanismus helfen.
- Ausführen der angemessenen Aktionen (in der Regel das Neustarten der Anwendung oder der Versuch, ein Failover der betroffenen Ressourcengruppe auszuführen). Das DSDL-Dienstprogramm `scds_fm_action()` implementiert einen solchen Algorithmus. Er berechnet zu diesem Zweck die aktuelle Akkumulation von Testsignalfehlschlägen in den letzten Retry_interval Sekunden.
- Aktualisieren des Ressourcenzustands, damit der fehlerfreie Zustand der Anwendung dem `scstat`-Befehl und der Cluster-Verwaltungs-GUI zur Verfügung steht.

Die DSDL-Dienstprogramme sind so ausgelegt, dass die Hauptschleife des Fehler-Monitor-Dämons durch den folgenden Pseudocode dargestellt werden kann.

Für Fehler-Monitore, die unter Verwendung der DSDL implementiert wurden, gilt:

- Die Erkennung eines Anwendungsprozessversagens durch `scds_fm_sleep()` erfolgt relativ schnell, da die Prozessausfallbenachrichtigung über PMF asynchron erfolgt. Im Vergleich zu einem Fall, in dem ein Fehler-Monitor immer wieder aktiv wird, um die Fehlerfreiheit des Dienstes zu prüfen und Anwendungsversagen festzustellen, wird die Fehlererkennungszeit deutlich reduziert, was die Verfügbarkeit des Dienstes steigert.
- Wenn RGM den Versuch, ein Failover des Dienstes über die `scha_control(3HA)` -API auszuführen, zurückweist, wird durch `scds_fm_action()` die aktuelle Fehlschlaghistorie *zurückgesetzt* (gelöscht). Der Grund dafür ist, dass die Fehlschlaghistorie bereits über `Retry_count` liegt. Wenn der Monitor-Dämon im nächsten Durchlauf aktiv wird und den Gesundheits-Check des Dämons nicht

erfolgreich beenden kann, versucht er erneut, `scha_control()` aufzurufen. Dieser Aufruf würde wahrscheinlich erneut zurückgewiesen, da die Situation, die zur Zurückweisung im letzten Durchlauf führte, noch gültig ist. Das Zurücksetzen der Fehlschlaghistorie stellt sicher, dass der Fehler-Monitor zumindest versucht, die Situation im nächsten Durchlauf lokal zu beheben (zum Beispiel über Anwendungsneustart).

- `scds_fm_action()` setzt die Anwendungsfehlschlaghistorie *nicht* zurück, wenn Neustartfehler auftreten, da normalerweise bald `scha_control()` versucht wird, wenn die Lage nicht behoben werden kann.
- Das Dienstprogramm `scds_fm_action()` aktualisiert den Ressourcenstatus zu `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_DEGRADED` oder `SCHA_RSSTATUS_FAULTED`, entsprechend der Fehlschlaghistorie. Dadurch steht dieser Status der Cluster-Systemverwaltung zur Verfügung.

In den meisten Fällen kann die anwendungsspezifische Gesundheits-Checkaktion in einem eigenständigen Dienstprogramm (zum Beispiel `svc_probe()`) implementiert und in diese generische Hauptschleife integriert werden.

```
for (;;) {

   / * Für die Dauer von thorough_probe_interval zwischen den
   *  einzelnen Testsignalen ruhen. */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));

   /* Jetzt alle verwendeten IP-Adressen testen. Schleife über:
   * 1. Allen verwendeten Netzwerkressourcen.
   * 2. Allen IP-Adressen in einer bestimmten Ressource.
   * Für jede getestete IP-Adresse
   * Fehlschlaghistorie berechnen. */
   probe_result = 0;
   /* Für alle Ressourcen wiederholen, um jede IP-Adresse
    * für den Aufruf von svc_probe() abzurufen */
   for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
   /* Hostname und Port für Überwachung der
   * Fehlerfreiheit erfassen.
   */
   hostname = netaddr->netaddrs[ip].hostname;
   port = netaddr->netaddrs[ip].port_proto.port;
   /*
   * HA-XFS unterstützt nur einen Port. Daher den
   * Port-Wert aus dem ersten Eintrag im
   * Port-Array abrufen.
   */
   ht1 = gethrtime(); /* Testsignal-Startzeit festhalten */
   probe_result = svc_probe(scds_handle,

   hostname, port, timeout);
   /*
   * Testsignalhistorie des Dienstes aktualisieren,
   * bei Bedarf Aktionen ausführen.
   * Testsignal-Endzeit festhalten.
```

```
*/
ht2 = gethrtime();
/* In Millisekunden konvertieren */
dt = (ulong_t)((ht2 - ht1) / 1e6);

/*
* Fehlschlaghistorie berechnen und
* bei Bedarf Aktionen ausführen
*/
(void) scds_fm_action(scds_handle,
probe_result, (long)dt);
}       /* Jede Netzwerkressource */
}       /* Endlos weitertesten */
```

KAPITEL 8

Beispielressourcentyp-Implementierung mit DSDL

Dieses Kapitel beschreibt einen Beispielressourcentyp, `SUNW.xfnts`, der mit der DSDL implementiert wird. Der Datendienst ist in C geschrieben. Die zugrunde liegende Anwendung ist X Font Server, ein TCP/IP-basierter Dienst.

In diesem Kapitel finden Sie folgende Informationen:

X Font Server

X Font Server ist ein einfacher, TCP/IP-basierter Dienst, der seinen Clients Schriftdateien zustellt. Clients stellen eine Verbindung mit dem Server her, um einen Schriftsatz anzufordern. Der Server liest die Schriftdateien von der Platte und stellt sie den Clients zu. Der X Font Server-Dämon besteht aus einer Server-Binärdatei, `/usr/openwin/bin/xfs`. Der Dämon wird in der Regel von `inetd` aus gestartet. In diesem Beispiel wird jedoch angenommen, dass der entsprechende Eintrag in der `/etc/inetd.conf`-Datei deaktiviert wurde (zum Beispiel durch den Befehl `fsadmin -d`). Daher befindet sich der Dämon unter alleiniger Steuerung durch Sun Cluster.

X Font Server-Konfigurationsdatei

Standardmäßig liest X Font Server die Konfigurationsinformationen aus der Datei `/usr/openwin/lib/X11/fontserver.cfg`. Der Katalogeintrag in diese Datei enthält eine Liste der Schriftverzeichnisse, die dem Dämon zur Verfügung stehen. Der Cluster-Verwalter kann die Schriftverzeichnisse im globalen Dateisystem ablegen. Dadurch wird die Verwendung von X Font Server unter Sun Cluster optimiert, da im System nur eine einzige Kopie der Schriftdatenbank verwaltet wird. Dann muss der Verwalter `fontserver.cfg` dahingehend bearbeiten, dass die neuen Pfade für die Schriftverzeichnisse enthalten sind.

Zur Vereinfachung der Konfiguration kann der Verwalter auch die Konfigurationsdatei selbst im globalen Dateisystem ablegen. Der `xfs`-Dämon stellt Befehlszeilenargumente bereit, mit denen der vorgegebene Standardspeicherort dieser Datei übersteuert wird. Der `SUNW.xfnts`-Ressourcentyp verwendet den folgenden Befehl zum Starten des Dämons unter Sun Cluster-Steuerung:

```
/usr/openwin/bin/xfs -config <Speicherort_der_CFG_Datei>/fontserver.cfg \
-port <Port-Nummer>
```

In der `SUNW.xfnts`-Ressourcentypimplementierung können Sie die `Confdir_list`-Eigenschaft zum Verwalten des Speicherortes der `fontserver.cfg`-Konfigurationsdatei verwenden.

TCP-Port-Nummer

Die TCP-Port-Nummer, die der `xfs`-Serverdämon abhört, ist normalerweise der “fs”-Port (üblicherweise als `7100` in der `/etc/services`-Datei definiert). Mit der Option `-port` an der `xfs`-Befehlszeile kann der Systemverwalter jedoch die Standardeinstellung übersteuern. Sie können die `Port_list`-Eigenschaft im `SUNW.xfnts`-Ressourcentyp verwenden, um den Standardwert einzustellen und um die Verwendung der Option `-port` an der `xfs`-Befehlszeile zu unterstützen. Der Standardwert dieser Eigenschaft wird als `7100/tcp` in der RTR-Datei definiert. In der `SUNW.xfnts-Start`-Methode wird `Port_list` an die Option `-port` an der `xfs`-Befehlszeile übergeben. Daher muss der Benutzer dieses Ressourcentyps keine Port-Nummer angeben—der Standard-Port ist `7100/tcp`—, er hat jedoch die Möglichkeit, bei der Konfiguration des Ressourcentyps einen anderen Port anzugeben, indem er einen anderen Wert für die `Port_list` -Eigenschaft angibt.

Namenskonventionen

Sie können die verschiedenen Teile des Beispielcodes identifizieren, indem Sie auf folgende Konventionen achten.

- RMAPI-Funktionen beginnen mit `scha_`.
- DSDL-Funktionen beginnen mit `scds_`.
- Rückmeldemethoden beginnen mit `xfnts_`.
- Benutzergeschriebene Funktionen beginnen mit `svc_`.

`SUNW.xfnts`-RTR-Datei

Dieser Abschnitt beschreibt mehrere Schlüsseleigenschaften in der `SUNW.xfnts`-RTR-Datei. Der Zweck der einzelnen Eigenschaften in der Datei wird nicht beschrieben. Diese Beschreibung finden Sie unter „Einstellen der Ressourcen- und Ressourcentypeigenschaften" auf Seite 35.

Die `Confdir_list`-Erweiterungseigenschaft identifiziert das Konfigurationsverzeichnis (bzw. eine Verzeichnisliste) folgendermaßen:

```
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "The Configuration Directory Path(s)";
}
```

Die `Confdir_list`-Eigenschaft gibt keinen Standardwert an. Der Cluster-Verwalter muss zum Zeitpunkt der Ressourcenerstellung ein Verzeichnis angeben. Dieser Wert kann später nicht mehr geändert werden, da seine Einstellbarkeit auf `AT_CREATION` beschränkt ist.

Die `Port_list`-Eigenschaft identifiziert folgendermaßen den Port, den der Serverdämon abhört:

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

Da für die Eigenschaft ein Standardwert deklariert wird, kann der Cluster-Verwalter zum Zeitpunkt der Ressourcenerstellung entscheiden, ob er einen neuen Wert angibt oder den Standardwert akzeptiert. Dieser Wert kann später nicht mehr geändert werden, da seine Einstellbarkeit auf `AT_CREATION` beschränkt ist.

`scds_initialize()`-Funktion

Die DSDL erfordert, dass jede Rückmeldemethode zu Methodenbeginn die `scds_initialize`(3HA)-Funktion aufruft. Diese Funktion führt folgende Vorgänge aus:

- Sie prüft und verarbeitet die Befehlszeilenargumente (`argc` und `argv`), die das Framework an die Datendienstmethode übergibt. Die Methode muss keine weiteren Verarbeitungsschritte für die Befehlszeilenargumente ausführen.
- Sie richtet interne Datenstrukturen ein, die von anderen DSDL-Funktionen verwendet werden können.
- Sie initialisiert die Protokollierumgebung.
- Sie validiert die Testsignaleinstellungen des Fehler-Monitors.

Sie verwendet die `scds_close()`-Funktion, um die Ressourcen zurückzufordern, die von `scds_initialize()` zugewiesen wurden.

`xfnts_start`-Methode

RGM ruft die `Start`-Methode auf einem Cluster-Knoten auf, wenn die Ressourcengruppe mit der Datendienstressource auf diesem Knoten online gebracht wird bzw. wenn die Ressource aktiviert wird. Im `SUNW.xfnts`-Beispielressourcentyp aktiviert die `xfnts_start`-Methode den `xfs`-Dämon auf diesem Knoten.

Die `xfnts_start`-Methode ruft `scds_pmf_start()` auf, um den Dämon unter PMF zu starten. PMF verfügt über automatische Fehlerbenachrichtigungs- und Neustartfunktionen und ist in den Fehler-Monitor integriert.

Hinweis – Der erste Aufruf in `xfnts_start` erfolgt an `scds_initialize()`. Diese Funktion führt einige notwendige *Systemverwaltungs*-Funktionen aus (weitere Einzelheiten hierzu finden Sie in der Online-Dokumentation unter „`scds_initialize()`-Funktion" auf Seite 146 und `scds_initialize`(3HA)).

Validieren des Dienstes vor dem Start

Vor dem Versuch, X Font Server zu starten, ruft die `xfnts_start`-Methode `svc_validate()` auf, um zu überprüfen, ob eine geeignete Konfiguration zur Unterstützung des `xfs`-Dämons eingerichtet ist (weitere Einzelheiten hierzu finden Sie unter „xfnts_validate-Methode" auf Seite 161):

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

Starten des Dienstes

Die `xfnts_start`-Methode ruft die `svc_start()`-Methode auf, die in `xfnts.c` definiert ist, um den `xfs`-Dämon zu starten. Dieser Abschnitt beschreibt `svc_start()`.

Folgender Befehl startet den `xfs`-Dämon:

`xfs -config` *Konfig_verzeichnis*`/fontserver.cfg -port` *Port_Nummer*

Die `Confdir_list`-Erweiterungseigenschaft identifiziert *Konfig_verzeichnis*, während die `Port_list`-Systemeigenschaft *Port_Nummer* identifiziert. Beim Konfigurieren des Datendienstes stellt der Cluster-Verwalter spezifische Werte für diese Eigenschaften bereit.

Die `xfnts_start`-Methode deklariert diese Eigenschaften als Zeichenketten-Arrays und ruft die Werte ab, die der Verwalter mit den Funktionen `scds_get_ext_confdir_list()` und `scds_get_port_list()` einstellt (beschrieben unter `scds_property_functions`(3HA)):

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

   /* 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);
   }
```

Beachten Sie, dass die `confdirs`-Variable auf das erste Element (0) im Array zeigt.

Die `xfnts_start`-Methode verwendet `sprintf`, um die Befehlszeile für `xfs` zu bilden:

```
/* 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);
```

Beachten Sie, dass die Ausgabe an `dev/null` umgeleitet wird, um die vom Dämon generierten Meldungen zu unterdrücken.

Die `xfnts_start`-Methode übergibt die `xfs`-Befehlszeile an `scds_pmf_start()`, um den Datendienst unter PMF-Steuerung zu starten:

```
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 ");
   }
```

Beachten Sie folgende Punkte bezüglich des Aufrufs an `scds_pmf_start()`.

- Der Parameter `SCDS_PMF_TYPE_SVC` identifiziert das zu startende Programm als Datendienstanwendung — mit dieser Methode kann auch ein Fehler-Monitor oder ein anderer Anwendungstyp gestartet werden.
- Der Parameter `SCDS_PMF_SINGLE_INSTANCE` identifiziert eine Ressource mit einer einzigen Instanz.
- Der `cmd`-Parameter ist die zuvor generierte Befehlszeile.
- Der letzte Parameter, `-1`, gibt die untergeordnete Überwachungsebene an. Der Wert -1 gibt an, dass PMF neben dem Prozess selbst auch alle untergeordneten Prozesse überwacht.

Vor der Rückgabe gibt `svc_pmf_start()` den der `portlist`-Struktur zugewiesenen Speicherplatz frei:

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

Rückgabe von `svc_start()`

Auch wenn `svc_start()` Erfolg zurückgibt, kann es sein, dass die zugrunde liegende Anwendung nicht gestartet wurde. Daher muss `svc_start()` die Anwendung testen, um sicherzustellen, dass sie läuft, bevor eine Erfolgsmeldung

zurückgegeben wird. Beim Testen muss beachtet werden, dass die Anwendung eventuell nicht sofort zur Verfügung steht, weil sie einige Zeit zum Starten benötigt. Die `svc_start()`-Methode ruft die in `xfnts.c` definierte `svc_wait()`-Funktion auf, um zu überprüfen, ob die Anwendung läuft:

```
/* 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.");
   }
```

Die `svc_wait()`-Funktion ruft `scds_get_netaddr_list`(3HA) auf, um die für das Testen der Anwendung erforderlichen Netzwerkadressressourcen abzurufen:

```
/* 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);
   }
```

Dann ruft `svc_wait()` die `start_timeout`- und `stop_timeout`-Werte ab:

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

Um die Zeit zu berücksichtigen, die der Server zum Starten benötigt, ruft `svc_wait()` die `scds_svc_wait()`-Methode auf und übergibt einen Zeitüberschreitungswert, der drei Prozent des `start_timeout`-Wertes entspricht. Dann ruft `svc_wait()` die `svc_probe()`-Methode auf, um zu überprüfen, ob die Anwendung gestartet wurde. Die `svc_probe()`-Methode stellt eine einfache Socketverbindung mit dem Server auf dem angegebenen Port her. Wenn die Verbindung mit dem Port fehlschlägt, gibt `svc_probe()` den Wert 100 für Totalfehlschlag zurück. Wenn die Verbindung hergestellt werden kann, aber die Verbindungstrennung vom Port fehlschlägt, gibt `svc_probe()` den Wert 50 zurück.

Bei Fehlschlag oder Teilfehlschlag von `svc_probe()` ruft `svc_wait()` die `scds_svc_wait()`-Methode mit einem Zeitüberschreitungswert von 5 auf. Die `scds_svc_wait()`-Methode beschränkt die Testhäufigkeit auf ein Testsignal alle fünf Sekunden. Diese Methode zählt auch die Anzahl der Startversuche für den Dienst. Wenn die Anzahl der Versuche den Wert der `Retry_count`-Ressourceneigenschaft innerhalb des von der `Retry_interval`-Ressourceneigenschaft angegebenen Zeitraums überschreitet, gibt die `scds_svc_wait()`-Funktion Fehlschlag zurück. In diesem Fall gibt die `svc_start()`-Funktion ebenfalls Fehlschlag zurück.

```
#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);
```

Hinweis – Vor der Beendigung ruft die `xfnts_start`-Methode `scds_close()` auf, um die von `scds_initialize()` zugewiesenen Ressourcen zurückzufordern. Weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_close`(3HA).

---

xfnts_stop-Methode

Da die xfnts_start-Methode scds_pmf_start() verwendet, um den Dienst unter PMF zu starten, verwendet die xfnts_stop-Methode scds_pmf_stop() zum Stoppen des Dienstes.

Hinweis – Der erste Aufruf in xfnts_stop erfolgt an scds_initialize(), um einige erforderliche *Systemverwaltungs*-Funktionen auszuführen (weitere Einzelheiten finden Sie unter „scds_initialize()-Funktion" auf Seite 146 und in der Online-Dokumentation unter scds_initialize(3HA)).

Die xfnts_stop-Methode ruft die in xfnts.c definierte svc_stop()-Methode folgendermaßen auf.

```
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 */
```

Bezüglich des Aufrufs in svc_stop() an die scds_pmf_stop()-Funktion ist Folgendes zu beachten.

- Der Parameter SCDS_PMF_TYPE_SVC identifiziert das zu stoppende Programm als Datendienstanwendung — mit dieser Methode kann auch ein Fehler-Monitor oder ein anderer Anwendungstyp gestoppt werden.
- Der Parameter SCDS_PMF_SINGLE_INSTANCE identifiziert das Signal.
- Der SIGTERM-Parameter identifiziert das Signal, mit dem die Ressourceninstanz gestoppt werden soll. Wenn dieses Signal die Instanz nicht stoppen kann, sendet scds_pmf_stop() das SIGKILL-Signal, um die Instanz zu stoppen. Wenn dieses Signal ebenfalls fehlschlägt, wird ein Zeitüberschreitungsfehler zurückgegeben. Weitere Einzelheiten finden Sie in der Online-Dokumentation unter scds_pmf_stop(3HA).
- Der Zeitüberschreitungswert ist der Wert der Stop_timeout-Eigenschaft der Ressource.

Hinweis – Vor der Beendigung ruft die `xfnts_stop`-Methode `scds_close()` auf, um die von `scds_initialize()` zugewiesenen Ressourcen zurückzufordern. Weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_close`(3HA).

`xfnts_monitor_start`-Methode

RGM ruft die `Monitor_start`-Methode auf einem Knoten auf, um den Fehler-Monitor zu starten, nachdem eine Ressource auf diesem Knoten gestartet wurde. Die `xfnts_monitor_start`-Methode verwendet `scds_pmf_start()`, um den Monitor-Dämon unter PMF zu starten.

Hinweis – Der erste Aufruf in `xfnts_monitor_start` erfolgt an `scds_initialize()`. Diese Funktion führt einige wichtige *Systemverwaltungs*-Funktionen aus (weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_initialize`(3HA)).

Die `xfnts_monitor_start`-Methode ruft die in `xfnts.c` definierte `mon_start`-Methode folgendermaßen auf:

```
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 */
}
```

Bezüglich des Aufrufs in `svc_mon_start()` an die `scds_pmf_start()`-Funktion ist Folgendes zu beachten.

- Der Parameter `SCDS_PMF_TYPE_MON` identifiziert das zu startende Programm als Fehler-Monitor — mit dieser Methode kann auch ein Datendienst oder ein anderer Anwendungstyp gestartet werden.
- Der Parameter `SCDS_PMF_SINGLE_INSTANCE` identifiziert eine Ressource mit einer einzigen Instanz.
- Der `xfnts_probe`-Parameter identifiziert den zu startenden Monitor-Dämon. Es wird davon ausgegangen, dass der Monitor-Dämon sich in demselben Verzeichnis wie die anderen Rückmeldeprogramme befindet.
- Der letzte Parameter, `0`, gibt die untergeordnete Überwachungsebene an — in diesem Fall wird nur der Monitor-Dämon überwacht.

Hinweis – Vor der Beendigung ruft die `xfnts_monitor_start`-Methode `scds_close()` auf, um die von `scds_initialize()` zugewiesenen Ressourcen zurückzufordern. Weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_close`(3HA).

`xfnts_monitor_stop`-Methode

Da die `xfnts_monitor_start`-Methode `scds_pmf_start()` zum Starten des Monitor-Dämons unter PMF verwendet, setzt `xfnts_monitor_stop` die `scds_pmf_stop()`-Methode zum Stoppen des Monitor-Dämons ein.

Hinweis – Der erste Aufruf in `xfnts_monitor_stop` erfolgt an `scds_initialize()`. Diese Funktion führt einige wichtige *Systemverwaltungs*-Funktionen aus (weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_initialize`(3HA)).

Die `xfnts_monitor_stop()`-Methode ruft die in `xfnts.c` definierte `mon_stop`-Methode folgendermaßen auf:

```
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 */
}
```

Bezüglich des Aufrufs in `svc_mon_stop()` an die `scds_pmf_stop()`-Funktion ist Folgendes zu beachten.

- Der Parameter `SCDS_PMF_TYPE_MON` identifiziert das zu stoppende Programm als Fehler-Monitor — mit dieser Methode kann auch ein Datendienst oder ein anderer Anwendungstyp gestoppt werden.
- Der Parameter `SCDS_PMF_SINGLE_INSTANCE` identifiziert eine Ressource mit einer einzigen Instanz.
- Der `SIGKILL`-Parameter identifiziert das Signal, mit dem die Ressourceninstanz gestoppt werden soll. Wenn dieses Signal die Instanz nicht stoppen kann, gibt `scds_pmf_stop()` einen Zeitüberschreitungsfehler zurück. Weitere Einzelheiten finden Sie in der Online-Dokumentation unter `scds_pmf_stop`(3HA).
- Der Zeitüberschreitungswert ist der Wert der `Monitor_stop_timeout`-Eigenschaft der Ressource.

Hinweis – Vor der Beendigung ruft die `xfnts_monitor_stop`-Methode `scds_close()` auf, um die von `scds_initialize()` zugewiesenen Ressourcen zurückzufordern. Weitere Einzelheiten finden Sie unter „`scds_initialize()` -Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_close`(3HA).

`xfnts_monitor_check`-Methode

RGM ruft die `Monitor_check`-Methode immer dann auf, wenn der Fehler-Monitor versucht, für die Ressourcengruppe der Ressorce ein Failover auf einen anderen Knoten auszuführen. Die `xfnts_monitor_check`-Methode ruft die `svc_validate()`-Methode auf, um zu überprüfen, ob eine geeignete Konfiguration zum Unterstützen des `xfs`-Dämons vorhanden ist (Einzelheiten finden Sie unter „`xfnts_validate`-Methode" auf Seite 161). Der Code für `xfnts_monitor_check` sieht folgendermaßen aus:

```
   /* 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);
}
```

SUNW.xfnts-Fehler-Monitor

RGM ruft die PROBE-Methode nicht direkt auf. Es wird vielmehr die Monitor_start -Methode aufgerufen, um den Monitor zu starten, nachdem eine Ressource auf einem Knoten gestartet wurde. Die xfnts_monitor_start-Methode startet den Fehler-Monitor unter PMF-Steuerung. Die xfnts_monitor_stop-Methode stoppt den Fehler-Monitor.

Der SUNW.xfnts-Fehler-Monitor führt folgende Aufgaben aus:

- Er überwacht in regelmäßigen Abständen die Fehlerfreiheit des xfs-Serverdämons mithilfe eigens zur Prüfung von einfachen TCP-basierten Diensten wie xfs entworfener Dienstprogramme.
- Er verfolgt Probleme der Anwendung innerhalb eines bestimmten Zeitfensters unter Verwendung der Eigenschaften Retry_count und Retry_interval und entscheidet, ob der Datendienst im Fall eines Totalfehlschlags der Anwendung neu gestartet oder ein Failover ausgeführt wird. Die Funktionen scds_fm_action() und scds_fm_sleep() unterstützen diesen Verfolgungs- und Entscheidungsmechanismus.
- Er implementiert die Failover- bzw. Neustartentscheidung mithilfe von scds_fm_action().
- Er aktualisiert den Ressourcenzustand und stellt ihn den Verwaltungstools und grafischen Benutzeroberflächen zur Verfügung.

xfonts_probe-Hauptschleife

Die xfonts_probe -Methode implementiert eine Schleife. Vor dem Implementieren der Schleife führt xfonts_probe folgende Aktionen aus:

- Sie ruft die Netzwerkadressressourcen für die xfnts-Ressource folgendermaßen ab:

```
/* 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);
   }
```

- Sie ruft `scds_fm_sleep()` auf und übergibt den Wert von `Thorough_probe_interval` als den Zeitüberschreitungswert. Das Testsignal ruht zwischen den einzelnen Testvorgängen für den Wert von `Thorough_probe_interval`.

```
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));
```

Die `xfnts_probe`-Methode implementiert die Schleife folgendermaßen:

```
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 */
```

Die `svc_probe()`-Funktion implementiert die Testsignallogik. Der Rückgabewert von `svc_probe()` wird an `scds_fm_action()` übergeben. Diese Funktion entscheidet, ob die Anwendung neu gestartet, ein Failover der Ressourcengruppe ausgeführt werden oder nichts geschehen soll.

`svc_probe()`-Funktion

Die `svc_probe()`-Funktion stellt eine einfache Socketverbindung mit dem angegebenen Port her, indem sie `scds_fm_tcp_connect()` aufruft. Wenn die Verbindung fehlschlägt, gibt `svc_probe()` den Wert `100` für Totalfehlschlag zurück. Wenn die Verbindung hergestellt werden kann, aber die Verbindungstrennung fehlschlägt, gibt `svc_probe()` den Wert `50` für Teilfehlschlag zurück. Wenn sowohl die Verbindung als auch die Verbindungstrennung erfolgreich verlaufen, gibt `svc_probe()` den Wert `0` für Erfolg zurück.

Der Code für `svc_probe()` sieht folgendermaßen aus:

```
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);
}
```

Nach Beendigung gibt `svc_probe()` einen Wert für Erfolg (`0`), Teilfehlschlag (`50`), oder Totalfehlschlag (`100`) zurück. Die `xfnts_probe`-Methode übergibt diesen Wert an `scds_fm_action()`.

Festlegen der Fehler-Monitor-Aktion

Die `xfnts_probe`-Methode ruft `scds_fm_action()` auf, um die auszuführende Aktion festzulegen. Die Logik in `scds_fm_action()` sieht folgendermaßen aus:

- Kumulative Fehlschlaghistorie innerhalb des Wertes der `Retry_interval`-Eigenschaft verwalten
- Wenn der kumulative Fehlschlag 100 (Totalfehlschlag) erreicht, wird der Datendienst neu gestartet. Wenn `Retry_interval` überschritten ist, wird die Historie zurückgesetzt.
- Wenn die Anzahl der Neustarts den in der `Retry_count`-Eigenschaft angegebenen Wert innerhalb der in `Retry_interval` angegebenen Zeit überschreitet, wird für den Datendienst ein Failover ausgeführt.

Angenommen, das Testsignal stellt eine Verbindung mit dem XFS-Server her, kann die Verbindung jedoch nicht trennen. Das bedeutet, dass der Server läuft, aber vielleicht hängt oder nur momentan überlastet ist. Bei Fehlschlag der Verbindungstrennung wird ein Teilfehlschlag (`50`) an `scds_fm_action()` gesendet. Dieser Wert liegt unter dem Schwellenwert für das Neustarten des Datendienstes. Der Wert wird jedoch in der Fehlerhistorie festgehalten.

Wenn während des nächsten Tests die Verbindungstrennung vom Server erneut fehlschlägt, wird ein Wert von `50` der von `scds_fm_action()` verwalteten Fehlschlaghistorie hinzugefügt. Der kumulative Fehlschlagwert beträgt nun `100`, so dass `scds_fm_action()` den Datendienst neu startet.

xfnts_validate-Methode

RGM ruft die Validate-Methode auf, wenn eine Ressource erstellt wird und wenn eine Verwaltungsaktion die Eigenschaften der Ressource bzw. deren Gruppe aktualisiert. RGM ruft Validate auf, bevor die Erstellung bzw. Aktualisierung angewendet wird. Ein Fehlerbeendigungscode der Methode auf einem Knoten führt zum Abbruch der Erstellung bzw. Aktualisierung.

RGM ruft Validate nur dann auf, wenn Ressourcen- bzw. Gruppeneigenschaften über eine Verwaltungsaktion geändert werden, und nicht, wenn RGM Eigenschaften einstellt oder wenn ein Monitor die Ressourceneigenschaften Status und Status_msg einstellt.

Hinweis – Die Monitor_check-Methode ruft auch ausdrücklich jedes Mal dann die Validate-Methode auf, wenn die PROBE-Methode versucht, ein Failover für den Datendienst auf einen neuen Knoten auszuführen.

RGM ruft Validate mit zusätzlichen Argumenten zu denjenigen auf, die von anderen Methoden übergeben wurden, einschließlich der aktualisierten Eigenschaften und Werte. Bei Aufruf von scds_initialize() zu Beginn von xfnts_validate werden alle Argumente analysiert, die RGM an xfnts_validate übergibt, und die Informationen werden im scds_handle-Parameter gespeichert. Die Unterroutinen, die xfnts_validate aufruft, verwenden diese Informationen.

Die xfnts_validate-Methode ruft svc_validate() auf, um Folgendes zu prüfen:

- Die Confdir_list-Eigenschaft wurde für die Ressource eingestellt und definiert ein einziges Verzeichnis.

```
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 */
   }
```

- Das von Confdir_list angegebene Verzeichnis enthält die fontserver.cfg-Datei.

```
(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);
   }
```

- Auf die Server-Dämon-Binärdatei kann auf dem Cluster-Knoten zugegriffen werden.

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
       scds_syslog(LOG_ERR,
           "Cannot access XFS binary : <%s> ", strerror(errno));
       return (1);
   }
```

- Die Port_list-Eigenschaft gibt einen einzigen Port an.

```
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
```

- Die Ressourcengruppe, die den Datendienst enthält, verfügt auch über mindestens eine Netzwerkadressressource.

```
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;
}
```

Vor der Rückgabe gibt `svc_validate()` alle zugewiesenen Ressourcen frei.

```
finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* return result of validation */
```

Hinweis – Vor der Beendigung ruft die `xfnts_validate`-Methode `scds_close()` auf, um die von `scds_initialize()` zugewiesenen Ressourcen zurückzufordern. Weitere Einzelheiten finden Sie unter „`scds_initialize()`-Funktion" auf Seite 146 und in der Online-Dokumentation unter `scds_close`(3HA).

`xfnts_update`-Methode

RGM ruft die `Update`-Methode auf, um eine laufende Ressource darüber zu benachrichtigen, dass ihre Eigenschaften geändert wurden. Die einzigen Eigenschaften des `xfnts`-Datendienstes, die geändert werden können, betreffen den Fehler-Monitor. Wenn daher eine Eigenschaft aktualisiert wird, ruft die `xfnts_update`-Methode `scds_pmf_restart_fm()` auf, um den Fehler-Monitor neu zu starten.

```
* 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.");
```

Hinweis – Der zweite Parameter von `scds_pmf_restart_fm()` identifiziert eindeutig die Instanz des Fehler-Monitors, die neu gestartet werden muss, wenn mehrere Instanzen vorhanden sind. Der Wert `0` im Beispiel gibt an, dass nur eine Instanz des Fehler-Monitors vorhanden ist.

KAPITEL 9

SunPlex Agent Builder

Dieses Kapitel beschreibt SunPlex Agent Builder und Cluster Agent Module für Agent Builder. Beides sind Tools, welche die Erstellung von Ressourcentypen bzw. Datendiensten automatisieren, die unter der Steuerung des Ressourcengruppen-Manager (RGM) ausgeführt werden sollen. Ein Ressourcentyp ist ein Wrapper um eine Anwendung. Dadurch kann die Anwendung in einer Cluster-Umgebung unter der Steuerung von RGM ausgeführt werden.

In diesem Kapitel werden die folgenden Themen behandelt:

Agent Builder Überblick

Agent Builder stellt eine bildschirmbasierte Benutzeroberfläche für die Eingabe von Informationen zur Anwendung und zum zu erstellenden Ressourcentyp bereit.

Hinweis – Wenn die grafische Benutzeroberfläche von Agent Builder nicht verfügbar ist, können Sie über eine Befehlszeilenoberfläche auf Agent Builder zugreifen. Siehe „So verwenden Sie die Befehlszeilenversion von Agent Builder" auf Seite 183.

Ausgehend von den Informationen, die Sie eingeben, generiert Agent Builder die folgende Software:

- Eine Reihe von C-, Korn-Shell- (`ksh`) oder GDS-Quelldateien (Generic Data Service, generischer Datendienst) für einen Failover- oder skalierbaren Ressourcentyp, der den Methodenrückmeldungen des Ressourcentyps entspricht; sowohl für Anwendungen mit Netzwerkunterstützung (Client-Server-Modell) als auch für Anwendungen ohne Netzwerkunterstützung (ohne Client)
- Eine angepasste RTR-Datei (Resource Type Registration, Ressourcentypregistrierung), wenn Sie C- bzw. Korn-Shell-Quellcode erstellen.
- Angepasste Dienstprogrammskripts zum Starten, Stoppen und Entfernen einer Instanz (Ressource) des Ressourcentyps sowie angepasste Online-Dokumentation zur Verwendung der einzelnen Dateien.
- Ein Solaris-Paket, das die Binärdateien (wenn Sie C-Quellcode generieren) eine RTR-Datei (wenn Sie C- bzw. Korn-Shell-Quellcode generieren), sowie die Dienstprogrammskripts enthält.

Agent Builder unterstützt Anwendungen mit Netzwerkunterstützung (Anwendungen, die über das Netzwerk mit den Clients kommunizieren) sowie Anwendungen ohne Netzwerkunterstützung (eigenständige Anwendungen). Mit Agent Builder können Sie auch einen Ressourcentyp für eine Anwendung generieren, die mehrere unabhängige Prozessbaumstrukturen aufweist, welche PMF (Process Monitor Facility) einzeln überwachen und neu starten muss. Siehe „Erstellen von Ressourcentypen mit mehreren unabhängigen Prozessbaumstrukturen" auf Seite 166.

Vor der Verwendung von Agent Builder

Der folgende Abschnitt enthält Informationen, die Sie vor der Verwendung von Agent Builder lesen sollten.

Erstellen von Ressourcentypen mit mehreren unabhängigen Prozessbaumstrukturen

Agent Builder kann Ressourcentypen für Anwendungen erstellen, die mehr als einen unabhängigen Prozessbaum aufweisen können. Diese Prozessbaumstrukturen sind unabhängig in dem Sinn, dass sie PMF-Monitoren einzeln starten und anhalten. PMF startet jede dieser Prozessbaumstrukturen mit ihrer eigenen Markierung.

Hinweis – Mithilfe von Agent Builder können Sie Ressourcentypen mit mehreren unabhängigen Prozessbäumen nur dann erstellen, wenn der angegebene generierte Quellcode C- oder GDS-Code darstellt. Für die Korn-Shell können diese Ressourcentypen nicht mit Agent Builder erstellt werden. Der Code zum Erstellen dieser Ressourcentypen für die Korn-Shell muss manuell geschrieben werden.

Im Fall einer Basisanwendung mit mehreren unabhängigen Prozessbaumstrukturen können Sie keinen einzelnen Befehl für das Starten der Anwendung angeben. Stattdessen muss eine Textdatei erstellt werden, in der jede Zeile den vollständigen Pfad zum Start-Befehl für eine der Prozessbaumstrukturen der Anwendung enthält. Diese Datei darf keine Leerzeilen enthalten. Sie können diese Textdatei im Textfeld "Start Command" im Bildschirm "Configure" angeben.

Wenn sichergestellt ist, dass diese Datei nicht über Ausführungsberechtigungen verfügt, kann Agent Builder die Datei, die dem Starten von mehreren Prozessbaumstrukturen dient, von einem einfachen ausführbaren Skript mit mehreren Befehlen unterscheiden. Wenn die Textdatei Ausführungsberechtigungen enthält, werden zwar die Ressourcen auf einem Cluster ohne Probleme oder Fehler hochgeladen. Alle Befehle werden jedoch unter einer einzigen PMF-Markierung gestartet, was die Möglichkeit ausschließt, die Prozessbaumstrukturen einzeln mit PMF zu überwachen und neu zu starten.

Verwenden von Agent Builder

Dieser Abschnitt beschreibt, wie Agent Builder verwendet wird, einschließlich der Aufgaben, die vor dem Einsatz von Agent Builder ausgeführt werden müssen. Der Abschnitt erläutert auch, auf welche Arten Agent Builder nach dem Generieren des Ressourcentypcodes eingesetzt werden kann.

Folgende Themen werden behandelt:

- „Bearbeiten des generierten Quellcodes" auf Seite 182
- „So verwenden Sie die Befehlszeilenversion von Agent Builder" auf Seite 183

Analysieren der Anwendung

Bevor Sie Agent Builder verwenden, müssen Sie entscheiden, ob die Anwendung den Kriterien für hohe Verfügbarkeit bzw. Skalierbarkeit entspricht. Agent Builder kann diese Analyse nicht ausführen, die lediglich auf den Laufzeiteigenschaften der Anwendung basiert. „Analysieren der Eignung einer Anwendung" auf Seite 29 enthält weitere Informationen zu diesem Thema.

Agent Builder ist möglicherweise nicht immer in der Lage, einen vollständigen Ressourcentyp für Ihre Anwendung zu erstellen. In den meisten Fällen kann Agent Builder jedoch zumindest eine Teillösung liefern. Kompliziertere Anwendungen benötigen zum Beispiel eventuell zusätzlichen Code, den Agent Builder nicht standardmäßig generiert, wie Code zum Hinzufügen von Validierungsprüfungen für zusätzliche Eigenschaften oder zum Einstellen von Parametern, die Agent Builder nicht vorgibt. In diesen Fällen müssen Sie Änderungen an dem generierten Quellcode oder an der RTR-Datei vornehmen. Agent Builder ist für diese flexible Arbeitsweise besonders geeignet.

Agent Builder fügt an bestimmten Stellen des generierten Quellcodes Kommentare ein. Hier können Sie Ihren eigenen spezifischen Ressourcentypcode hinzufügen. Nachdem Sie den Quellcode geändert haben, können Sie das von Agent Builder generierte `Makefile` verwenden, um den Quellcode neu zu kompilieren und das Ressourcentyppaket neu zu generieren.

Selbst wenn Sie den gesamten Ressourcentypcode selbst schreiben und keinen von Agent Builder generierten Code verwenden, können Sie das von Agent Builder bereitgestellte `Makefile` und die Struktur zum Erstellen eines Solaris-Pakets für Ihren Ressourcentyp nutzen.

Installieren und Konfigurieren von Agent Builder

Agent Builder muss nicht eigens installiert werden. Agent Builder ist im `SUNWscdev`-Paket enthalten, das im Rahmen einer Standardinstallation der Sun Cluster Software standardmäßig installiert wird. Weitere Informationen hierzu finden Sie im *Sun Cluster Handbuch Softwareinstallation für Solaris OS*.

Vor der Verwendung von Agent Builder sollte Folgendes überprüft werden.

- Java Runtime Environment ist in Ihrer `$PATH`-Variable enthalten. Agent Builder ist von Java abhängig (Java Development Kit, mindestens Version 1.3.1). Wenn Java nicht in `$PATH` enthalten ist, gibt `scdsbuilder` eine Fehlermeldung aus.
- Sie haben die Softwaregruppe "Entwicklersystemunterstützung" von Solaris 8 oder höher installiert.

- Der `cc`-Compiler ist in Ihrer `$PATH`-Variable enthalten. Agent Builder verwendet das erste Vorkommen von `cc` in Ihrer `$PATH`-Variable zur Identifizierung des Compilers, mit dem der C-Binärcode für den Ressourcentyp erstellt werden soll. Wenn `cc` nicht in `$PATH` enthalten ist, deaktiviert Agent Builder die Option zum Generieren von C-Code. Siehe „Verwenden des Bildschirms "Create"" auf Seite 174.

Hinweis – Mit Agent Builder kann auch ein anderer Compiler als der `cc`-Standard-Compiler verwendet werden. Zur Verwendung eines anderen Compilers müssen Sie in `$PATH` von `cc` eine symbolische Verknüpfung zu einem anderen Compiler wie `gcc` erstellen. Es kann auch die Compiler-Spezifikation im `Makefile` (aktuell `CC=cc`) in den vollständigen Pfad für einen anderen Compiler geändert werden. Ändern Sie zum Beispiel im von Agent Builder generierten `Makefile` den Eintrag `CC=cc` in `CC=`*Pfadname*`/gcc` . In diesem Fall können Sie Agent Builder nicht direkt ausführen, sondern müssen die Befehle `make` und `make pkg` ausführen, um Datendienstcode und ein Paket zu generieren.

Bildschirme in Agent Builder

Agent Builder ist ein Assistent, der zwei Arbeitsschritte mit jeweils einem Bildschirm umfasst. Agent Builder verfügt über die folgenden zwei Bildschirme, die Sie durch die Erstellung eines neuen Ressourcentyps führen:

1. **Create.** In diesem Bildschirm geben Sie grundlegende Informationen über den zu erstellenden Ressourcentyp wie dessen Namen und das Arbeitsverzeichnis für die erzeugten Dateien ein. Im Arbeitsverzeichnis wird die Ressourcentypvorlage erstellt und konfiguriert. Hier wird auch festgelegt, welche Art Ressource erstellt wird (Scalable oder Failover), ob die Basisanwendung Netzwerkunterstützung hat (das heißt, ob sie das Netzwerk für die Kommunikation mit den Clients verwendet), sowie der zu generierende Codetyp (C, Korn-Shell (`ksh`) oder GDS). Informationen zum GDS finden Sie unter Kapitel 10. Sie müssen alle erforderlichen Informationen in diesen Bildschirm eingeben und dann auf "Create" klicken, um die entsprechende Ausgabe zu erzeugen. Anschließend können Sie den Bildschirm "Configure" anzeigen.
2. **Configure.** In diesem Bildschirm müssen Sie die vollständige Befehlszeile angeben, die zum Starten Ihrer Basisanwendung an eine UNIX®-Shell übergeben werden soll. Optional können Sie auch Befehle zum Stoppen und Testen der Anwendung angeben. Wenn Sie diese Befehle nicht angeben, verwendet die erzeugte Ausgabe Signale zum Stoppen der Anwendung und stellt einen Standardtestmechanismus bereit. Eine Beschreibung des Befehls "Probe" finden Sie unter „Verwenden des Bildschirms "Configure"" auf Seite 176. Über diesen Bildschirm können auch die Zeitüberschreitungswerte für jeden dieser drei Befehle geändert werden.

Starten von Agent Builder

Hinweis – Wenn die grafische Benutzeroberfläche von Agent Builder nicht verfügbar ist, können Sie über eine Befehlszeilenoberfläche auf Agent Builder zugreifen. Siehe „So verwenden Sie die Befehlszeilenversion von Agent Builder" auf Seite 183.

Hinweis – Wenn Sie Agent Builder vom Arbeitsverzeichnis für einen vorhandenen Ressourcentyp aus starten, initialisiert Agent Builder die Bildschirme "Create" und "Configure" mit den Werten des vorhandenen Ressourcentyps.

Agent Builder wird durch die Eingabe des folgenden Befehls gestartet:

```
% /usr/cluster/bin/scdsbuilder
```

Der Bildschirm "Create" wird angezeigt.

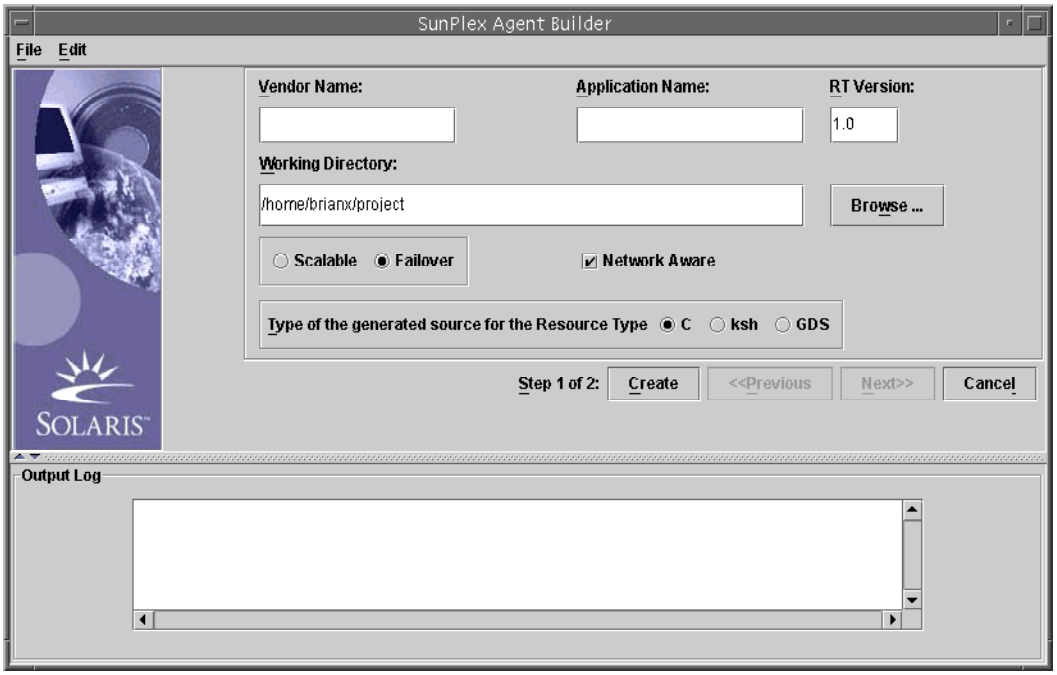


ABBILDUNG 9–1 Bildschirm "Create"

Navigieren in Agent Builder

In die Bildschirme "Create" und "Configure" können Informationen auf folgende Weisen eingegeben werden:

- Geben Sie Informationen in ein Feld ein.
- Durchsuchen Sie die Verzeichnisstruktur, und wählen Sie eine Datei bzw. ein Verzeichnis aus.
- Wählen Sie eine Reihe sich gegenseitig ausschließender Optionsfelder aus, beispielsweise durch Klicken auf "Scalable" oder "Failover".
- Aktivieren/Deaktivieren Sie ein Kontrollkästchen. Wenn Sie zum Beispiel "Network Aware" aktivieren, wird die Basisanwendung als Anwendung mit Netzwerkunterstützung identifiziert. Ein deaktiviertes Kontrollkästchen identifiziert dagegen eine Anwendung ohne Netzwerkunterstützung.

Mit den Optionen unten auf jedem Bildschirm können Sie die Aufgabe beenden, zum nächsten bzw. vorherigen Bildschirm gehen oder Agent Builder beenden. Agent Builder kennzeichnet die jeweils verfügbaren Optionen durch Hervorhebung. Nicht verfügbare Optionen werden abgeblendet dargestellt.

Wenn Sie zum Beispiel die Felder ausgefüllt und die gewünschten Optionen im Bildschirm "Create" aktiviert haben, klicken Sie unten im Bildschirm auf "Create". Die Schaltflächen "Previous" und "Next" sind abgeblendet dargestellt, da kein vorheriger Bildschirm vorhanden ist und Sie nicht zum nächsten Schritt weitergehen können, bevor dieser Schritt beendet ist.

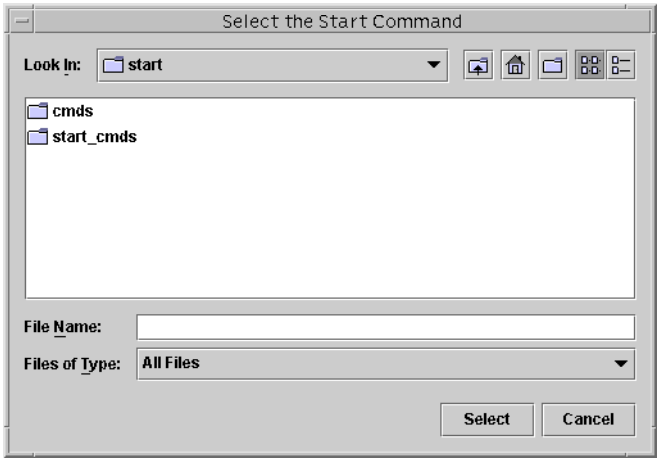
Agent Builder zeigt Fortschrittsmeldungen im Ausgabeprotokollbereich unten auf dem Bildschirm an. Wenn Agent Builder die Ausführung fertig gestellt hat, wird eine Erfolgsmeldung angezeigt, bzw. eine Warnmeldung, die zum Einsehen des Ausgabeprotokolls auffordert. Die Option "Next" ist hervorgehoben. Falls es sich um den letzten Bildschirm handelt, ist jedoch nur "Cancel" hervorgehoben.

Sie können jederzeit "Cancel" wählen, um Agent Builder zu beenden.

Auswählen

In bestimmte Agent Builder-Felder können Sie Informationen eingeben oder auf die Schaltfläche **"Browse"** klicken, um die Verzeichnisstruktur zu durchsuchen und eine Datei oder ein Verzeichnis auszuwählen.

Wenn Sie auf "Browse" klicken, wird ein Bildschirm entsprechend der folgenden Abbildung angezeigt.



Doppelklicken Sie auf einen Ordner, um ihn zu öffnen. Wenn Sie mit dem Cursor auf eine Datei zeigen, wird der Name der Datei im Feld "File Name" angezeigt. Klicken Sie auf "Select", wenn Sie die gewünschte Datei gefunden haben und mit dem Cursor darauf zeigen.

Hinweis – Wenn Sie nach einem Verzeichnis suchen, zeigen Sie mit dem Cursor auf das gewünschte Verzeichnis und klicken Sie dann auf "Open". Wenn keine Unterverzeichnisse vorhanden sind, schließt Agent Builder das Suchfenster und setzt den Namen des Verzeichnisses, auf dem sich der Cursor befindet, in das entsprechende Feld. Wenn das Verzeichnis Unterverzeichnisse hat, klicken Sie auf "Close", um das Suchfenster zu schließen und zum vorherigen Bildschirm zurückzukehren. Agent Builder setzt den Namen des Verzeichnisses, auf dem sich der Cursor befindet, in das entsprechende Feld.

Die Symbole in der oberen rechten Ecke des Bildschirms haben folgende Funktionen:

Dieses Symbol führt Sie in der Verzeichnisstruktur eine Ebene nach oben.

Dieses Symbol führt Sie zum Home-Ordner zurück.

Dieses Symbol erstellt einen neuen Ordner unter dem aktuell ausgewählten Ordner.

Dieses Symbol, das zum Umschalten zwischen verschiedenen Ansichten dient, wird für spätere Verwendung vorbehalten.

Menüs

Agent Builder verfügt über die Pulldown-Menüs "File" und "Edit".

Menü "File"

Das Menü "File" enthält zwei Optionen:

- **Load Resource Type.** Lädt einen bestehenden Ressourcentyp. Agent Builder stellt einen Suchbildschirm bereit, von dem aus Sie das Arbeitsverzeichnis für einen vorhandenen Ressourcentyp auswählen. Wenn ein Ressourcentyp in dem Verzeichnis vorhanden ist, von dem aus Sie Agent Builder starten, lädt Agent Builder automatisch den Ressourcentyp. "Load Resource Type" ermöglicht das Starten von Agent Builder von einem beliebigen Verzeichnis aus und die Auswahl eines vorhandenen Ressourcentyps als Vorlage für das Erstellen eines neuen Ressourcentyps. Siehe „So klonen Sie einen bestehenden Ressourcentyp" auf Seite 182.
- **Exit.** Beendet Agent Builder. Sie können das Tool auch beenden, indem Sie im Bildschirm "Create" oder "Configure" auf "Cancel" klicken.

Menü "Edit"

Das Menü "Edit" enthält die folgenden zwei Optionen:

- **Clear Output Log.** Löscht die Daten aus dem Ausgabeprotokoll. Jedes Mal, wenn Sie "Create" oder "Configure" auswählen, hängt Agent Builder an das Ausgabeprotokoll Statusmeldungen an. Wenn Sie wiederholt Änderungen am Quellcode vornehmen, die Ausgabe in Agent Builder neu generieren und die

Statusmeldungen trennen möchten, können Sie die Protokolldatei vor jeder Verwendung speichern und löschen.

- **Save Log File.** Speichert die Protokollausgabe in einer Datei. Agent Builder stellt einen Suchbildschirm bereit, in dem Sie ein Verzeichnis auswählen und einen Dateinamen angeben können.

Verwenden des Bildschirms "Create"

Bildschirm "Create"

Als erster Schritt für die Erstellung eines Ressourcentyps muss der Bildschirm "Create" ausgefüllt werden. Dieser Bildschirm wird beim Starten von Agent Builder angezeigt. Die folgende Abbildung zeigt den Bildschirm "Create" nach der Eingabe von Informationen in die Felder.

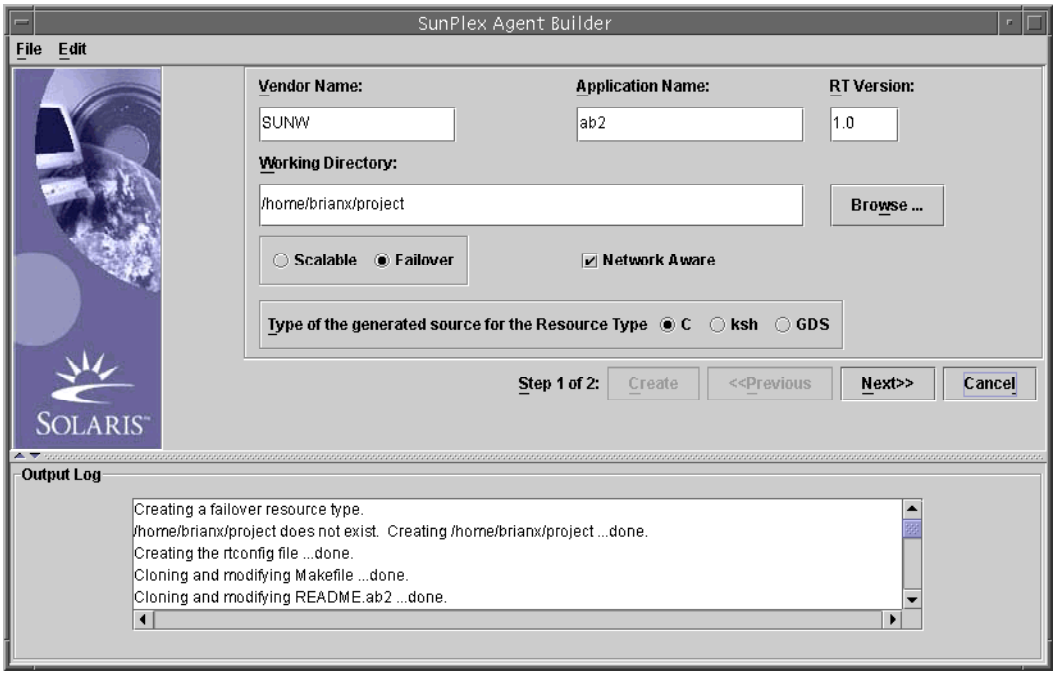


ABBILDUNG 9–2 Bildschirm "Create"

Der Bildschirm "Create" enthält die folgenden Felder, Optionsfelder und Kontrollkästchen:

- **Vendor Name.** Der Name des Herstellers des Ressourcentyps. In der Regel wird das Börsensymbol des Herstellers angegeben. Es ist jedoch ein beliebiger Name gültig, der den Hersteller eindeutig identifiziert. Es dürfen nur alphanumerische

Zeichen verwendet werden.

- **Application Name.** Der Name des Ressourcentyps. Es dürfen nur alphanumerische Zeichen verwendet werden.

Hinweis – Der vollständige Name des Ressourcentyps setzt sich aus Herstellername und Anwendungsname zusammen. Dieser vollständige Name darf nicht länger als neun Zeichen sein.

- **RT Version.** Die Typenversion der erzeugten Ressource. RT Version unterscheidet zwischen mehreren registrierten Versionen (Aufrüstungen) desselben Basisressourcentyps.

  Im Feld "RT Version" dürfen die folgenden Zeichen nicht verwendet werden: Leerzeichen, Tabulator, Schrägstrich (`/`), umgekehrter Schrägstrich (`\`), Sternchen (`*`), Fragezeichen (`?`), Komma (`,`), Strichpunkt (`;`), linke eckige Klammer (`[`) oder rechte eckige Klammer (`]`).

- **Working Directory.**Das Arbeitsverzeichnis, unter dem Agent Builder eine Verzeichnisstruktur erstellt, in der alle für den Zielressourcentyp erstellten Dateien abgelegt werden. In einem Arbeitsverzeichnis kann jeweils nur ein Ressourcentyp erstellt werden. Agent Builder initialisiert dieses Feld mit dem Pfad zu dem Verzeichnis, von dem aus Sie Agent Builder gestartet haben. Sie können jedoch einen anderen Namen eingeben oder auf "Browse" klicken, um ein anderes Verzeichnis zu suchen.

  Unter dem Arbeitsverzeichnis erstellt Agent Builder ein Unterverzeichnis mit dem Ressourcentypnamen. Wenn zum Beispiel `SUNW` der Herstellernamen und `ftp` der Anwendungsname ist, nennt Agent Builder das Unterverzeichnis `SUNWftp`.

  Agent Builder legt alle Verzeichnisse und Dateien für den Zielressourcentyp unter diesem Unterverzeichnis ab. Siehe „Verzeichnisstruktur" auf Seite 184.

- **Scalable oder Failover.** Geben Sie an, ob der Zielressourcentyp Failover oder skalierbar sein soll.

- **Network Aware.**Geben Sie an, ob die Basisanwendung über Netzwerkunterstützung verfügt, das heißt, ob sie das Netzwerk für die Kommunikation mit den Clients verwendet. Aktivieren Sie das Kontrollkästchen "Network Aware", wenn die Netzwerkunterstützung aktiviert werden soll. Andernfalls lassen Sie das Kontrollkästchen deaktiviert.

- **C, ksh.**Geben Sie die Sprache des generierten Quellcodes an. Obwohl sich diese Optionen gegenseitig ausschließen, können Sie in Agent Builder einen Ressourcentyp mit Korn-Shell-Code erstellen und dann dieselben Daten zur Erstellung von C-Code verwenden. Siehe „So klonen Sie einen bestehenden Ressourcentyp" auf Seite 182.

- **GDS.**Hiermit geben Sie an, dass dieser Dienst ein generischer Datendienst ist. Informationen zum Erstellen und Konfigurieren eines generischen Datendienstes finden Sie in Kapitel 10.

Hinweis – Wenn sich der `cc`-Compiler nicht in Ihrem `$PATH` befindet, wird das Optionsfeld "C" abgeblendet dargestellt und Sie können das Optionsfeld "ksh" auswählen. Informationen zum Angeben eines anderen Compilers finden Sie im Hinweis am Ende von „Installieren und Konfigurieren von Agent Builder" auf Seite 168.

Klicken Sie nach Eingabe der erforderlichen Informationen auf "Create". Das Ausgabeprotokollfenster unten im Bildschirm zeigt an, welche Aktionen Agent Builder ausführt. Sie können im Menü "Edit" den Befehl "Save Output Log" verwenden, um die Informationen im Ausgabeprotokoll zu speichern.

Nach Abschluss des Vorgangs zeigt Agent Builder entweder eine Erfolgsmeldung oder eine Warnmeldung an.

- Wenn Agent Builder diesen Schritt nicht ausführen konnte, lesen Sie die Informationen im Ausgabeprotokoll.
- Wenn Agent Builder den Schritt erfolgreich ausführt, können Sie auf "Next" klicken, um den Bildschirm "Configure" anzuzeigen, und die Erzeugung des Ressourcentyps abzuschließen.

Hinweis – Obwohl sich der Prozess zum Generieren eines vollständigen Ressourcentyps aus zwei Schritten zusammensetzt, können Sie Agent Builder nach Beenden des ersten Schritts (Create) beenden, ohne die eingegebenen Informationen oder die von Agent Builder geleistete Arbeit zu verlieren. Siehe „Wiederverwenden fertiger Arbeiten" auf Seite 181.

Verwenden des Bildschirms "Configure"

Bildschirm "Configure"

In der folgenden Abbildung wird der Bildschirm "Configure" gezeigt. Dieser Bildschirm wird angezeigt, nachdem Agent Builder die Erstellung des Ressourcentyps beendet hat und Sie im Bildschirm "Create" auf "Next" geklickt haben. Auf den Bildschirm "Configure" kann erst dann zugegriffen werden, wenn der Ressourcentyp erstellt worden ist.

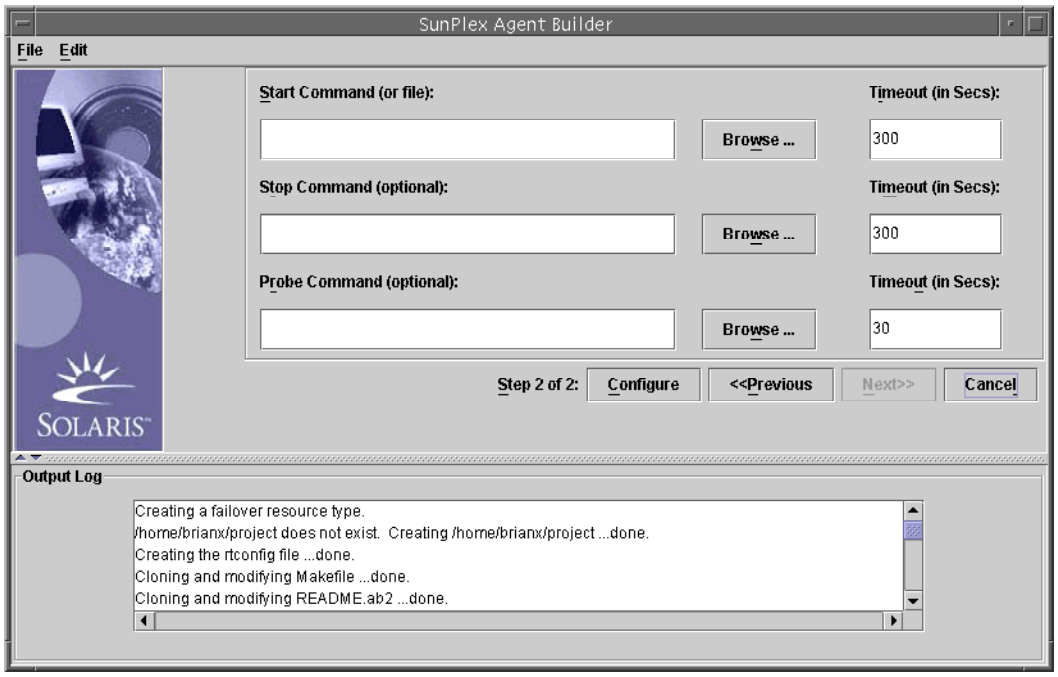


ABBILDUNG 9–3 Bildschirm "Configure"

Der Bildschirm "Configure" enthält folgende Felder:

- **Start Command.** Die vollständige Befehlszeile, die an eine beliebige UNIX-Shell übergeben werden kann, um die Basisanwendung zu starten. Sie müssen einen Startbefehl angeben. Sie können den Befehl in das dafür vorgesehene Feld eingeben oder die Schaltfläche "Browse" verwenden, um die Datei zu suchen, die den Startbefehl für die Anwendung enthält.

  Die vollständige Befehlszeile muss alle erforderlichen Parameter zum Starten der Anwendung enthalten, wie Hostnamen, Port-Nummern, einen Pfad zu Konfigurationsdateien usw. Sie können auch Eigenschaftsvariablen festlegen (siehe „Eigenschaftsvariablen" auf Seite 180). Wenn Ihre Korn-Shell-basierte Anwendung die Angabe eines Hostnamens in der Befehlszeile erfordert, können Sie die Variable `$hostnames` verwenden, die von Agent Builder definiert wird. Siehe „Verwenden der Korn-Shell-basierten `$hostnames`-Variablen in Agent Builder" auf Seite 179.

  Der Befehl darf nicht in doppelten Anführungszeichen stehen ("").

Hinweis – Wenn die Basisanwendung über mehrere unabhängige Prozessbaumstrukturen verfügt, von denen jede unter einer eigenen Markierung unter PMF-Steuerung (Process Monitor Facility) gestartet wird, kann kein einzelner Befehl angegeben werden. Stattdessen müssen Sie eine Textdatei mit den einzelnen Befehlen zum Starten jeder Prozessbaumstruktur erstellen und im Textfeld "Start Command" den Pfad zu dieser Datei angeben. In „Erstellen von Ressourcentypen mit mehreren unabhängigen Prozessbaumstrukturen" auf Seite 166 finden Sie Informationen zu einigen speziellen Eigenschaften, die diese Datei erfüllen muss, um ordnungsgemäß zu funktionieren.

- **Stop Command.** Die vollständige Befehlszeile, die an eine beliebige UNIX-Shell übergeben werden kann, um die Basisanwendung zu stoppen. Sie können den Befehl in das dafür vorgesehene Feld eingeben oder die Schaltfläche "Browse" verwenden, um die Datei zu suchen, die den Stopp-Befehl für die Anwendung enthält. Sie können auch Eigenschaftsvariablen festlegen (siehe „Eigenschaftsvariablen" auf Seite 180). Wenn Ihre Korn-Shell-basierte Anwendung die Angabe eines Hostnamens in der Befehlszeile erfordert, können Sie die Variable `$hostnames` verwenden, die von Agent Builder definiert wird. Siehe „Verwenden der Korn-Shell-basierten `$hostnames`-Variablen in Agent Builder" auf Seite 179.

  Dieser Befehl ist optional. Wenn Sie keinen Stopp-Befehl angeben, verwendet der generierte Code die folgenden Signale (in der `Stopp`-Methode), um die Anwendung zu stoppen.

  - Die `Stopp`-Methode sendet `SIGTERM`, um die Anwendung zu stoppen, und wartet 80% des Zeitüberschreitungswertes darauf, dass die Anwendung beendet wird.
  - Wenn das `SIGTERM`-Signal nicht erfolgreich ist, sendet die `Stopp`-Methode `SIGKILL`, um die Anwendung zu stoppen, und wartet während 15% des Zeitüberschreitungswertes darauf, dass die Anwendung beendet wird.
  - Ist `SIGKILL` nicht erfolgreich, wird die `Stopp`-Methode beendet. Die restlichen 5% des Zeitüberschreitungswertes gelten als Überlastung.

Achtung – Stellen Sie sicher, dass der Stopp-Befehl keine Antwort zurückgibt, bevor die Anwendung vollständig gestoppt wurde.

- **Probe Command.** Ein Befehl, der in regelmäßigen Abständen ausgeführt werden kann, um die Anwendung auf Fehler zu überprüfen und einen entsprechenden Beendigungsstatus zwischen `0` (Erfolg) und `100` (Totalfehlschlag) zurückzugeben. Dieser Befehl ist optional. Sie können den gesamten Pfad zum Befehl eingeben oder mit der Schaltfläche "Browse" die Datei suchen, die den Befehl für das Testen der Anwendung enthält.

  In der Regel wird ein einfacher Client der Basisanwendung angegeben. Wenn Sie keinen Testsignal-Befehl angeben, stellt der generierte Code einfach Verbindungen mit dem von der Ressource verwendeten Port her und trennt diese wieder. Wenn

dieser Vorgang erfolgreich verläuft, wird die Anwendung als fehlerfrei deklariert. Sie können auch Eigenschaftsvariablen festlegen (siehe „Eigenschaftsvariablen" auf Seite 180). Wenn Ihre Korn-Shell-basierte Anwendung die Angabe eines Hostnamens in der Probe-Befehlszeile erfordert, können Sie die Variable `$hostnames` verwenden, die von Agent Builder definiert wird. Siehe „Verwenden der Korn-Shell-basierten `$hostnames`-Variablen in Agent Builder" auf Seite 179.

- **Timeout.** Ein Zeitüberschreitungswert (in Sekunden) für jeden Befehl. Sie können einen neuen Wert angeben oder den von Agent Builder vorgegebenen Standardwert (300 Sekunden für "start" und "stop", 30 Sekunden für "probe") akzeptieren.

Verwenden der Korn-Shell-basierten `$hostnames`-Variablen in Agent Builder

Bei zahlreichen Anwendungen, besonders bei Anwendungen mit Netzwerkunterstützung, muss der Name des Hosts, auf dem die Anwendung Kundenanforderungen empfängt und beantwortet, an die Anwendung in der Befehlszeile übergeben werden. Daher ist der Hostname in vielen Fällen ein Parameter, der für Start-, Stopp- und Testsignal-Befehle für den Zielressourcentyp angegeben werden muss (im Bildschirm "Configure"). Der Name des Hosts, auf dem eine Anwendung Anforderungen empfängt, ist jedoch Cluster-spezifisch. Der Hostname wird bestimmt, wenn die Ressource in einem Cluster ausgeführt wird, und kann nicht bestimmt werden, wenn Agent Builder Ihren Ressourcentypcode erzeugt.

Zum Beheben dieses Problems stellt Agent Builder die `$hostnames`-Variable bereit, die Sie an der Befehlszeile für die Start-, Stopp- und Testsignal-Befehle angeben können.

Hinweis – Die `$hostnames`-Variable wird ausschließlich für die Verwendung mit Korn-Shell-basierten Diensten unterstützt. Für die Verwendung mit C- und GDS-basierten Diensten wird die `$hostnames`-Variable nicht unterstützt.

Sie geben die `$hostnames`-Variable genau so an, wie Sie dies mit einem tatsächlichen Hostnamen tun würden, zum Beispiel:

```
% /opt/network_aware/echo_server -p Port-Nummer -l $hostnames
```

Wenn eine Ressource des Zielressourcentyps auf einem Cluster ausgeführt wird, ersetzt der für diese Ressource konfigurierte `LogicalHostname` bzw. `SharedAddress`-Hostname für die Ressource (in der Ressourceneigenschaft `Network_resources_used` der Ressource) den Wert der `$hostnames`-Variablen.

Wenn Sie die Eigenschaft `Network_resources_used` mit mehreren Hostnamen konfigurieren, enthält die `$hostnames` alle Hostnamen, durch Komma voneinander getrennt.

Eigenschaftsvariablen

Durch Verwendung von Eigenschaftsvariablen können Sie die Werte von ausgewählten Sun Cluster-Ressourcen-, Ressourcentyp- und Ressourcengruppeneigenschaften aus dem RGM-Framework abrufen. Agent Builder durchsucht Ihr Start-, Testsignal- oder Stopp-Skript nach Eigenschaftsvariablen und ersetzt diese Variablen durch ihre Werte, bevor das Skript gestartet wird.

Hinweis – Eigenschaftsvariablen werden nicht für die Verwendung mit Korn-Shell-basierten Diensten unterstützt.

Liste der Eigenschaftsvariablen

Die folgende Liste enthält die Eigenschaftsvariablen, die Sie in Ihren Skripts verwenden können. Eine Beschreibung der Sun Cluster-Ressourcen-, Ressourcentyp- und Ressourcengruppeneigenschaften finden Sie unter Anhang A.

Die folgende Liste enthält die verfügbaren Ressourceneigenschaftsvariablen:

- `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`

Die folgende Liste enthält die verfügbaren Ressourcentyp-Eigenschaftsvariablen:

- `RT_API_VERSION`
- `RT_BASEDIR`
- `RT_FAILOVER`
- `RT_INSTALLED_NODES`
- `RT_NAME`
- `RT_RT_VERSION`
- `RT_SINGLE_INSTANCE`

Die folgende Liste enthält die verfügbaren Ressourcengruppen-Eigenschaftsvariablen:

- `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

Syntax der Eigenschaftsvariablen

Eigenschaftsvariablen werden durch Einfügen eines Prozentzeichens (%) vor einem Eigenschaftsnamen angegeben, wie im folgenden Beispiel dargestellt.

```
# /opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

Agent Builder kann diese Eigenschaftsvariablen interpretieren und das Skript starten. In diesem Beispiel wird das `echo_server`-Skript mit den folgenden Werten gestartet:

```
# /opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

So ersetzt Agent Builder Eigenschaftsvariablen

In der folgenden Liste wird beschrieben, wie Agent Builder die verschiedenen Eigenschaftsvariablen interpretiert:

- Eine Ganzzahl wird durch ihren tatsächlichen Wert ersetzt (z. B. `300`).
- Ein boolescher Wert wird durch die Zeichenkette `TRUE` oder `FALSE` ersetzt.
- Eine Zeichenkette wird durch die tatsächliche Zeichenkette ersetzt (z. B. `phys-node-1`).
- Eine Liste von Zeichenketten wird durch alle Mitglieder der Liste, jeweils durch ein Komma getrennt, ersetzt (z. B. `phys-node-1,phys-node-2,phys-node-3`).
- Eine Liste von Ganzzahlen wird durch alle Mitglieder in der Liste, jeweils durch ein Komma getrennt, ersetzt (z. B.`1,2,3`).
- Ein Aufzählungstyp wird durch seinen Wert in Form einer Zeichenkette ersetzt.

Wiederverwenden fertiger Arbeiten

In Agent Builder können Sie erstellte Elemente auf folgende Weisen wieder verwenden:

- Sie können einen mit Agent Builder erstellten vorhandenen Ressourcentyp klonen.
- Sie können den von Agent Builder erzeugten Quellcode bearbeiten und dann den Code zur Erstellung eines neuen Pakets erneut kompilieren.

▼ So klonen Sie einen bestehenden Ressourcentyp

Führen Sie dieses Verfahren aus, um einen von Agent Builder erstellten vorhandenen Ressourcentyp zu klonen.

1. **Laden Sie einen bestehenden Ressourcentyp mithilfe einer der folgenden Methoden in Agent Builder:**
   - Starten Sie Agent Builder über das Arbeitsverzeichnis (das die Datei `rtconfig` enthält) für einen bestehenden Ressourcentyp (der mit Agent Builder erstellt wurde). Agent Builder lädt die Werte für diesen Ressourcentyp in die Bildschirme "Create" und "Configure".
   - Verwenden Sie die Option "Load Resource Type" im Pulldown-Menü "File".
2. **Ändern Sie das Arbeitsverzeichnis im Bildschirm "Create".**

   Wählen Sie mithilfe von "Browse" ein Verzeichnis aus. Die Eingabe eines neuen Verzeichnisnamens ist nicht ausreichend. Nachdem Sie ein Verzeichnis ausgewählt haben, aktiviert Agent Builder wieder die Schaltfläche "Create".
3. **Nehmen Sie Änderungen vor.**

   Mit diesem Verfahren können Sie den für den Ressourcentyp generierten Codetyp ändern. Wenn Sie beispielsweise ursprünglich eine Korn-Shell-Version eines Ressourcentyps erstellt haben, nun aber eine C-Version benötigen, können Sie den bestehenden Korn-Shell-Ressourcentyp laden, die Sprache für die Ausgabe in "C" ändern und dann eine C-Version des Ressourcentyps erzeugen lassen.
4. **Erstellen Sie den geklonten Ressourcentyp.**

   Klicken Sie auf "Create", um den Ressourcentyp zu erstellen. Klicken Sie auf "Next", um den Bildschirm "Configure" anzuzeigen. Klicken Sie auf "Configure", um den Ressourcentyp zu konfigurieren, und klicken Sie dann zum Beenden auf "Cancel".

Bearbeiten des generierten Quellcodes

Um den Prozess für die Erstellung eines Ressourcentyps einfach zu halten, beschränkt Agent Builder die Anzahl der Eingaben. Dadurch werden natürlich auch die Einsatzmöglichkeiten für den generierten Ressourcentyp eingeschränkt. Um daher komplexere Funktionen wie zum Beispiel Validierungsprüfungen für zusätzliche Eigenschaften hinzuzufügen oder von Agent Builder nicht vorgegebene Parameter einzustellen, müssen Sie den generierten Quellcode oder die RTR-Datei ändern.

Die Quelldateien befinden sich im Verzeichnis *Installationsverzeichnis*/*RT_Name*/`src`. Agent Builder bettet an den Stellen im Quellcode, an denen Sie Code hinzufügen können, Kommentare ein. Diese Kommentare haben für C-Code folgende Form:

```
/* Vom Benutzer hinzugefügter Code -- BEGIN vvvvvvvvvvvvvvv */
/* Vom Benutzer hinzugefügter Code -- END   ^^^^^^^^^^^^^^^ */
```

Hinweis – Diese Kommentare sind im Korn-Shell-Quellcode identisch, jedoch wird der Beginn eines Kommentars durch ein Rautezeichen (#) angegeben.

Zum Beispiel deklariert *RT_Name*`.h` alle Dienstprogrammroutinen, welche die verschiedenen Programme verwenden. Am Ende der Deklarationenliste befinden sich Kommentare, mit denen Sie weitere Routinen deklarieren können, die Sie einem der Codes hinzugefügt haben können.

Agent Builder generiert auch das `Makefile` im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/src` mit entsprechenden Zielen. Verwenden Sie den `make`-Befehl, um den Quellcode neu zu kompilieren, und den `make pkg`-Befehl zur Neugenerierung des Ressourcentyppakets.

Die RTR-Datei befindet sich im Verzeichnis *Installationsverzeichnis*`/` *RT_Name*`/etc`. Sie können die RTR-Datei mit einem Standardtexteditor bearbeiten. Weitere Informationen zur RTR-Datei finden Sie unter „Einstellen der Ressourcen- und Ressourcentypeigenschaften" auf Seite 35 sowie Informationen zu den Eigenschaften unter Anhang A.

▼ So verwenden Sie die Befehlszeilenversion von Agent Builder

Die Befehlszeilenversion von Agent Builder verwendet denselben Basisprozess wie die Benutzeroberfläche. Hierbei geben Sie jedoch keine Informationen in die Benutzeroberfläche ein, sondern übergeben Parameter an die Befehle `scdscreate` und `scdsconfig`. Informationen hierzu finden Sie in der Online-Dokumentation unter `scdscreate`(1HA) und `scdsconfig`(1HA).

Führen Sie folgende Schritte aus, um die Befehlszeilenversion von Agent Builder zu verwenden.

1. **Erstellen Sie mit dem Befehl `scdscreate` eine Sun Cluster-Ressourcentypvorlage, um eine Anwendung mit hoher Verfügbarkeit oder Skalierbarkeit auszustatten.**

2. **Verwenden Sie `scdsconfig`, um die mit `scdscreate` erstellte Ressourcentypvorlage zu konfigurieren.**

   Sie können auch Eigenschaftsvariablen festlegen. Eine Beschreibung der Eigenschaftsvariablen finden Sie unter „Eigenschaftsvariablen" auf Seite 180.

3. **Ändern Sie die Verzeichnisse in das `pkg`-Unterverzeichnis im Arbeitsverzeichnis.**

4. **Verwenden Sie den Befehl `pkgadd` zur Installation der mit `scdscreate` erstellten Pakete.**

5. **Falls gewünscht, können Sie den generierten Quellcode bearbeiten.**

6. **Führen Sie das Start-Skript aus.**

Verzeichnisstruktur

Agent Builder erstellt eine Verzeichnisstruktur, in der alle für den Zielressourcentyp erstellten Dateien abgelegt werden. Das Arbeitsverzeichnis wird im Bildschirm "Create" angegeben. Sie müssen für jeden weiteren entwickelten Ressourcentyp ein eigenes Installationsverzeichnis angeben. Unter dem Arbeitsverzeichnis erstellt Agent Builder ein Unterverzeichnis, dessen Name eine Verkettung aus dem Herstellernamen und dem Ressourcentypnamen ist (aus dem Bildschirm "Create"). Wenn Sie zum Beispiel `SUNW` als Herstellername angeben und einen Ressourcentyp mit dem Namen `ftp` erstellen, erstellt Agent Builder ein Verzeichnis mit dem Namen `SUNWftp` unter dem Arbeitsverzeichnis.

Unter diesem Unterverzeichnis erstellt und füllt Agent Builder die in der folgenden Tabelle aufgelisteten Verzeichnisse aus.

Verzeichnisname	Inhalt
`bin`	Für C-Ausgabe; enthält die aus den Quelldateien kompilierten Binärdateien. Für Korn-Shell-Ausgabe; enthält dieselben Dateien wie das `src`-Verzeichnis.
`etc`	Enthält die RTR-Datei. Agent Builder verkettet den Herstellernamen mit dem Anwendungsnamen, getrennt durch einen Punkt (.), um den RTR-Dateinamen zu bilden. Wenn zum Beispiel der Herstellername `SUNW` und der Ressourcentypname `ftp` ist, lautet der Name der RTR-Datei `SUNW.ftp`.
`man`	Enthält angepasste Online-Dokumentation für die Dienstprogrammskripts `start`, `stop` und `remove`. Zum Beispiel `startftp`(1M), `stopftp`(1M) und `removeftp`(1M). Zum Anzeigen dieser Online-Dokumentation geben Sie den Pfad mit der Option `man -M` an. Beispiel: `# man -M` *Installationsverzeichnis*`/SUNWftp/man removeftp`
`pkg`	Enthält das fertige Paket.
`src`	Enthält die von Agent Builder generierten Quelldateien.
`util`	Enthält die von Agent Builder generierten Dienstprogrammskripts `start`, `stop` und `remove`. Siehe „Dienstprogrammskripts und Online-Dokumentation" auf Seite 186. Agent Builder hängt den Anwendungsnamen an die einzelnen Skriptnamen an, z. B. `startftp`, `stopftp`, `removeftp`.

Agent Builder-Ausgabe

Dieser Abschnitt beschreibt die von Agent Builder generierte Ausgabe.

Quell- und Binärdateien

Der Ressourcentyp-Manager (RGM), der Ressourcengruppen und damit auch Ressourcen auf einem Cluster verwaltet, arbeitet mit einem Rückmeldemodell. Wenn spezifische Ereignisse auftreten, wie zum Beispiel ein Knotenversagen, ruft RGM die Ressourcentypmethoden für jede der auf dem betroffenen Knoten ausgeführten Ressourcen auf. RGM ruft zum Beispiel die `Stop`-Methode auf, um eine auf dem betroffenen Knoten ausgeführte Ressource zu stoppen. Dann ruft das Programm die `Start`-Methode der Ressource auf, um sie auf einem anderen Knoten neu zu starten. Weitere Informationen zu diesem Modell finden Sie unter „RGM-Modell" auf Seite 21 und „Rückmeldemethoden" auf Seite 24 sowie in der Online-Dokumentation unter `rt_callbacks`(1HA).

Zur Unterstützung dieses Modells erzeugt Agent Builder im Verzeichnis *Installationsverzeichnis*`/` *RT_Name*`/bin` acht ausführbare C-Programme oder Korn-Shell-Skripts, die als Rückmeldemethoden dienen.

Hinweis – Streng genommen ist das Programm *RT_Name* `_probe`, das einen Fehler-Monitor implementiert, kein Rückmeldeprogramm. RGM ruft *RT_Name* `_probe` nicht direkt auf, sondern ruft *RT_Name* `_monitor_start` und *RT_Name* `_monitor_stop` auf, um den Fehler-Monitor durch Aufrufen von *RT_Name*`_probe` zu starten und zu stoppen.

Die acht von Agent Builder generierten Methoden sind hier aufgelistet:

- *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`

Spezielle Informationen zu jeder dieser Methoden finden Sie in der Online-Dokumentation unter `rt_callbacks`(1HA).

Im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/src` (C-Ausgabe) generiert Agent Builder die folgenden Dateien:

- Eine Header-Datei (*RT_Name* `.h`)
- Eine Quelldatei (*RT_Name* `.c`), die den für alle Methoden gemeinsamen Code enthält.
- Eine Objektdatei (*RT_Name* `.o`) für den gemeinsamen Code.
- Quelldateien (`*.c`) für jede der Methoden.
- Objektdateien (`*.o`) für jede der Methoden.

Agent Builder verknüpft die *RT_Name* `.o`-Datei mit jeder der `.o`-Dateien der Methode, um die ausführbaren Dateien im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/bin` zu erstellen.

Für die Korn-Shell-Ausgabe sind die Verzeichnisse *Installationsverzeichnis*`/`*RT_Name*`/bin` und *Installationsverzeichnis*`/`*RT_Name*`/src` identisch. Jedes Verzeichnis enthält die acht ausführbaren Skripts, die den sieben Rückmeldemethoden und der `Probe`-Methode entsprechen.

Hinweis – Die Korn-Shell-Ausgabe enthält zwei kompilierte Dienstprogramme (`gettime` und `gethostnames`), die von bestimmten Rückmeldemethoden zum Abrufen der Zeit und zum Testen benötigt werden.

Sie können den Quellcode bearbeiten und den `make`-Befehl zum Neukompilieren des Codes ausführen. Anschließend führen Sie den `make pkg` -Befehl aus, um ein neues Paket zu generieren. Als Unterstützung für die Änderung des Quellcodes bettet Agent Builder Kommentare an denjenigen Stellen im Quellcode ein, an denen Code hinzugefügt werden kann. Siehe „Bearbeiten des generierten Quellcodes" auf Seite 182.

Dienstprogrammskripts und Online-Dokumentation

Nach dem Generieren eines Ressourcentyps und Installieren des entsprechenden Pakets auf einem Cluster müssen Sie noch eine Instanz (Ressource) des Ressourcentyps auf einem Cluster ausführen können. Hierzu werden gewöhnlich Verwaltungsbefehle oder SunPlex-Manager eingesetzt. Zur Vereinfachung generiert Agent Builder jedoch ein angepasstes Dienstprogrammskript für diesen Zweck (das Start-Skript), sowie Skripts zum Stoppen und Entfernen einer Ressource aus dem Zielressourcentyp. Diese drei Skripts befinden sich im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/util` und führen folgende Aufgaben aus:

- **Start-Skript.**Registriert den Ressourcentyp und erstellt die erforderlichen Ressourcengruppen und Ressourcen. Es erstellt auch die Netzwerkadressressourcen (`LogicalHostname` oder `SharedAddress`), mit denen die Anwendung mit den Clients im Netzwerk kommunizieren kann.

- **Stopp-Skript.** Stoppt und deaktiviert die Ressource.
- **Remove-Skript.** Macht die Schritte des Start-Skripts wieder rückgängig. Das bedeutet, dass dieses Skript die Ressourcen, die Ressourcengruppen und den Zielressourcentyp stoppt und aus dem System entfernt.

Hinweis – Sie können das Remove-Skript nur für eine Ressource verwenden, die mit dem entsprechenden Start-Skript gestartet wurde, da diese Skripts interne Konventionen für die Namensgebung für Ressourcen und Ressourcengruppen verwenden.

Agent Builder benennt diese Skripts, indem der Anwendungsname an den Skriptnamen angehängt wird. Wenn der Anwendungsname zum Beispiel `ftp` lautet, heißen die Skripts `startftp`, `stopftp` und `removeftp`.

Agent Builder stellt im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/man/man1m` Online-Dokumentation für jedes der Dienstprogrammskripts bereit. Lesen Sie diese Online-Dokumentation vor dem Starten der Skripts, da sie die Parameter enthält, die Sie in das Skript übertragen müssen.

Zum Anzeigen der Online-Dokumentation geben Sie den Pfad zum man-Verzeichnis an, indem Sie die Option `-M` mit dem `man`-Befehl verwenden. Wenn zum Beispiel `SUNW` der Hersteller und `ftp` der Anwendungsname ist, verwenden Sie folgenden Befehl, um die `startftp` (1M)-Online-Dokumentation anzuzeigen:

```
% man -M Installationsverzeichnis/SUNWftp/man startftp
```

Die Dienstprogrammskripts der Online-Dokumentation stehen auch dem Cluster-Verwalter zur Verfügung. Wenn ein von Agent Builder generiertes Paket auf einem Cluster installiert ist, wird die Online-Dokumentation für die Dienstprogrammskripts im Verzeichnis `/opt/`*RT_Name*`/man` abgelegt. Verwenden Sie z. B. folgenden Befehl, um die Online-Dokumentation zu `startftp`(1M) anzuzeigen:

```
% man -M /opt/SUNWftp/man startftp
```

Unterstützungsdateien

Agent Builder legt Unterstützungsdateien, wie `pkginfo`, `postinstall`, `postremove` und `preremove` im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/etc` ab. Dieses Verzeichnis enthält auch die Ressourcentyp-Registrierungsdatei (RTR-Datei), welche die Ressourcen- und Ressourcentypeigenschaften deklariert, die dem Zielressourcentyp zur Verfügung stehen, und Eigenschaftswerte initialisiert, wenn eine Ressource bei einem Cluster registriert wird. Weitere Informationen finden Sie unter „Einstellen der Ressourcen- und Ressourcentypeigenschaften" auf Seite 35. Die RTR-Datei erhält den Namen *Herstellername*`.`*Ressourcentypname*, z. B. `SUNW.ftp`.

Sie können diese Datei mit einem Standard-Textverarbeitungsprogramm bearbeiten und Änderungen vornehmen, ohne den Quellcode neu zu kompilieren. Sie müssen jedoch das Paket mit dem `make pkg`-Befehl neu zusammenstellen.

Paketverzeichnis

Das Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/pkg` enthält ein Solaris-Paket. Der Paketname ist eine Verkettung des Herstellernamens mit dem Anwendungsnamen, zum Beispiel `SUNWftp`. Das `Makefile` im Verzeichnis *Installationsverzeichnis*`/`*RT_Name*`/src` unterstützt die Erstellung eines neuen Pakets. Wenn Sie zum Beispiel Änderungen an den Quelldateien vornehmen und den Code neu kompilieren, oder wenn Sie Änderungen an den Paket-Dienstprogrammskripts vornehmen, verwenden Sie den `make pkg`-Befehl, um ein neues Paket zu erstellen.

Wenn Sie ein Paket aus einem Cluster entfernen, kann der `pkgrm`-Befehl fehlschlagen, wenn Sie versuchen, den Befehl auf mehr als einem Knoten aus gleichzeitig auszuführen. Sie können dieses Problem auf zwei Arten lösen:

- Führen Sie das Skript `remove` *rt_name* auf einem Knoten im Cluster aus, bevor Sie `pkgrm` auf einem beliebigen Knoten ausführen.
- Führen Sie `pkgrm` auf einem der Knoten im Cluster aus. Dadurch werden die erforderlichen Bereinigungsvorgänge ausgeführt. Führen Sie dann `pkgrm` auf den restlichen Knoten aus; bei Bedarf auch gleichzeitig.

Wenn `pkgrm` fehlschlägt, weil Sie die Ausführung auf mehreren Knoten gleichzeitig versuchen, führen Sie den Befehl noch einmal auf einem Knoten und dann auf den restlichen Knoten aus.

Die Datei `rtconfig`

Wenn Sie C- oder Korn-Shell-Quellcode generieren, generiert Agent Builder eine Konfigurationsdatei `rtconfig`, welche die an den Bildschirmen "Create" und "Configure" eingegebenen Informationen enthält. Wenn Sie Agent Builder über das Arbeitsverzeichnis für einen bestehenden Ressourcentyp starten (oder einen bestehenden Ressourcentyp durch Auswahl von "Load Resource Type" im Pulldown-Menü "File" laden), liest Agent Builder die Datei `rtconfig` und fügt die Informationen, die Sie für den bestehenden Ressourcentyp angegeben haben, in die Bildschirme "Create" und "Configure" ein. Diese Funktion ist nützlich, wenn Sie einen vorhandenen Ressourcentyp klonen möchten. Siehe „So klonen Sie einen bestehenden Ressourcentyp” auf Seite 182.

Cluster Agent Module für Agent Builder

Cluster Agent Module für Agent Builder ist ein NetBeans™-Modul. Damit können die Benutzer des Sun Java Studio-Produkts (früher Sun ONE Studio) mithilfe einer integrierten Entwicklungsumgebung Ressourcentypen oder Datendienste für die Sun Cluster-Software erstellen. Das Cluster Agent Module verfügt über eine bildschirmbasierte Benutzeroberfläche für die Beschreibung der Art von Ressourcentyp, die erstellt werden soll.

Hinweis – Die Sun Java Studio-Dokumentation documentation enthält Informationen darüber, wie das Sun Java Studio-Produkt eingerichtet, installiert und verwendet wird.

▼ So wird das Cluster Agent Modul installiert und eingerichtet

Cluster Agent Module wird bei der Installation der Sun Cluster-Software installiert. Das Sun Cluster-Installationstool legt die Cluster Agent Module-Datei `scdsbuilder.jar` in `/usr/cluster/lib/scdsbuilder` ab. Um Cluster Agent Module zusammen mit der Sun Java Studio-Software zu verwenden, müssen Sie eine symbolische Verknüpfung zu dieser Datei erstellen.

Hinweis – Die Sun Cluster- und Sun Java Studio-Produkte sowie Java 1.4 müssen auf dem System, auf dem Cluster Agent Module ausgeführt werden soll, installiert und verfügbar sein.

---

1. **Aktivieren Sie alle Benutzer oder nur sich selbst für die Verwendung des Cluster Agent Module.**

   - Um alle Benutzer zu aktivieren, müssen Sie als Superbenutzer angemeldet sein bzw. eine äquivalente Rolle haben und die symbolische Verknüpfung im globalen Modulverzeichnis erstellen:

     ```
     # cd /opt/s1studio/ee/modules
     # ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

Hinweis – Wenn Sie die Sun Java Studio-Software in einem anderen Verzeichnis als `/opt/s1studio/ee` installiert haben, müssen Sie diesen Verzeichnispfad durch den von Ihnen verwendeten Pfad ersetzen.

- Wenn Sie nur sich selbst als Benutzer aktivieren möchten, erstellen Sie die symbolische Verknüpfung in Ihrem `modules`-Unterverzeichnis:

```
% cd ~Ihr_Stammverzeichnis/ffjuser40ee/modules
% ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

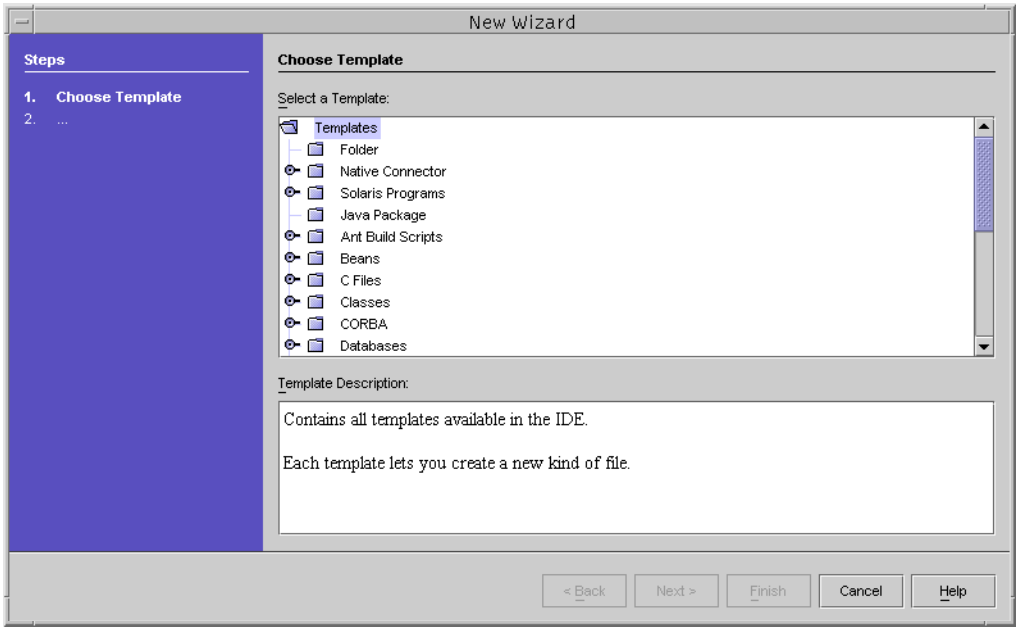
2. **Stoppen Sie die Sun Java Studio-Software und starten Sie sie erneut.**

▼ So starten Sie das Cluster Agent Module

Die folgenden Schritte beschreiben, wie Cluster Agent Module über die Sun Java Studio-Software gestartet wird.

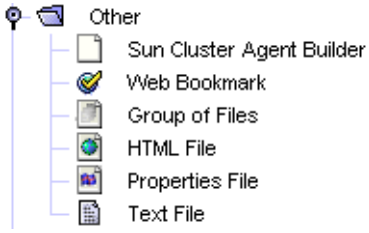
1. **Wählen Sie im Menü "File" von Sun Java Studio die Option "New" oder klicken Sie auf der Symbolleiste auf das entsprechende Symbol.**

Der Bildschirm "New Wizard" wird angezeigt.



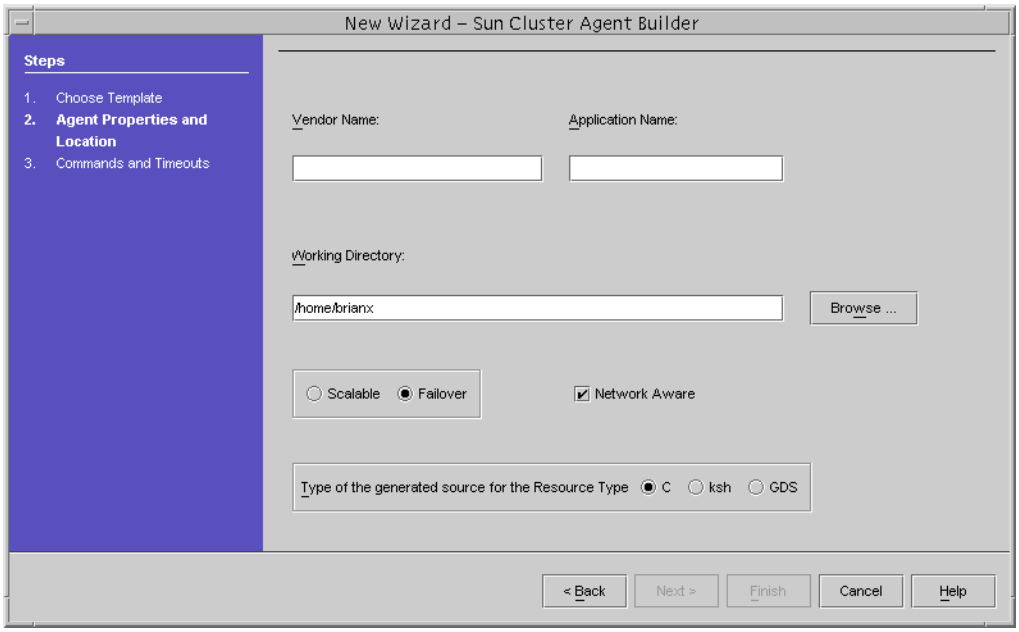
2. **Führen Sie im Fenster "Select a Template" gegebenenfalls einen Bildlauf nach unten aus und klicken Sie auf den Schlüssel neben dem Ordner "Other":**

Der Ordner "Other" wird geöffnet.



3. **Wählen Sie "Sun Cluster Agent Builder" im Ordner "Other" aus und klicken Sie auf "Next".**

   Cluster Agent Module für Sun Java Studio wird gestartet. Der erste Bildschirm "New Wizard - Sun Cluster Agent Builder" wird angezeigt.



Verwenden von Cluster Agent Module

Cluster Agent Module wird genauso wie die Agent Builder-Software verwendet. Die Benutzeroberflächen sind identisch. Die folgende Abbildung zeigt, dass zum Beispiel der Bildschirm "Create" in der Agent Builder-Software und der erste Bildschirm "New Wizard - Sun Cluster Agent Builder" im Cluster Agent Module die gleichen Felder und die gleiche Auswahl enthalten.

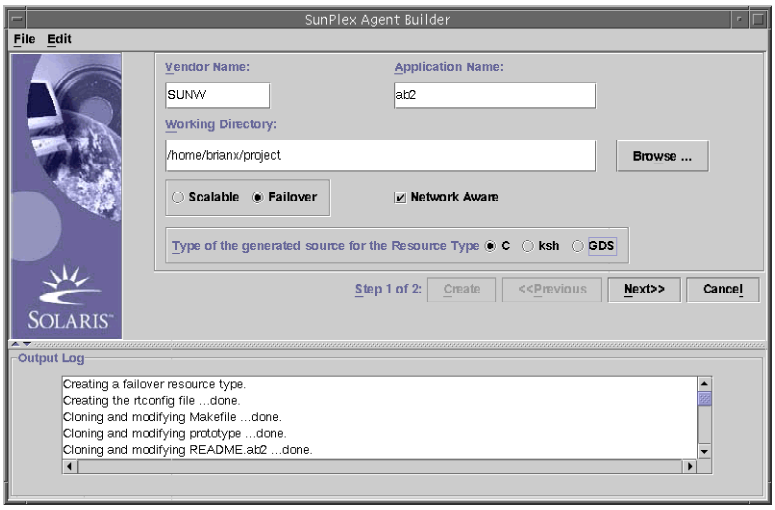


ABBILDUNG 9–4 Bildschirm "Create" in der Agent Builder-Software

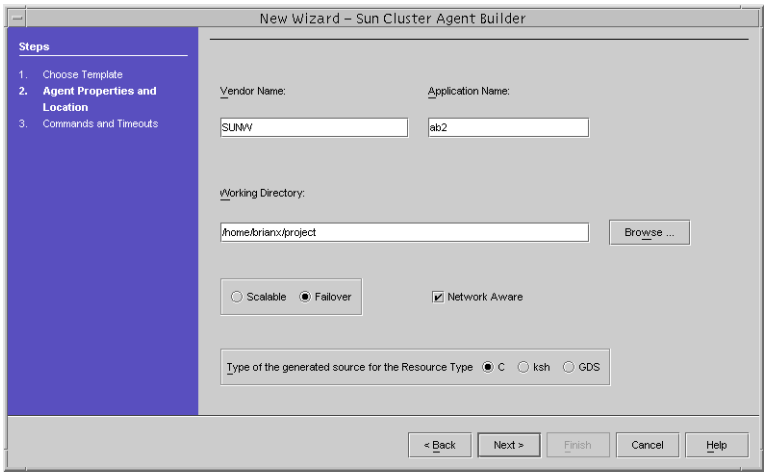


ABBILDUNG 9–5 Bildschirm "New Wizard - Sun Cluster Agent Builder" in Cluster Agent Module

Unterschiede zwischen Cluster Agent Module und Agent Builder

Trotz der Ähnlichkeiten zwischen Cluster Agent Module und Agent Builder bestehen einige kleine Unterschiede:

- In Cluster Agent Module wird der Ressourcentyp erst dann erstellt und konfiguriert, nachdem Sie im zweiten Bildschirm "New Wizard - Sun Cluster Agent Builder" auf "Finish" geklickt haben. Der Ressourcentyp wird *nicht* erstellt, wenn Sie im ersten Bildschirm "New Wizard - Sun Cluster Agent Builder" auf "Next" klicken.

  In Agent Builder wird der Ressourcentyp sofort erstellt, wenn Sie im Bildschirm "Create" auf "Create" klicken, und konfiguriert, wenn Sie im Bildschirm "Configure" auf "Configure" klicken.
- Die Informationen, die im Fenster "Output Log" in Agent Builder angezeigt werden, sind im Sun Java Studio-Produkt in einem eigenen Ausgabefenster enthalten.

KAPITEL 10

Generische Datendienste

Dieses Kapitel enthält Informationen zum GDS (Generic Data Service, generischer Datendienst) und beschreibt das Verfahren zur Erstellung eines Dienstes, der den GDS verwendet. Sie können diesen Dienst mithilfe von SunPlex Agent Builder oder mithilfe der Standardverwaltungsbefehle von Sun Cluster erstellen.

Dieses Kapitel umfasst die folgenden Themen:

Überblick über den GDS

Der GDS ist ein Mechanismus zur Einrichtung von hoher Verfügbarkeit bzw. Skalierbarkeit für einfache Anwendungen mit oder ohne Netzwerkunterstützung durch Integrierung in das Sun Cluster RGM-Framework (Resource Group Management). Dieser Mechanismus erfordert keine Agentencodierung, wie dies sonst beim Einrichten von hoher Verfügbarkeit bzw. Skalierbarkeit üblich ist.

Der GDS ist ein einzelner, vorkompilierter Datendienst. Der vorkompilierte Datendienst und seine Komponenten, die Implementierungen der Rückmeldemethoden (`rt_callbacks` (1HA)) und die Ressourcentyp-Registrierungsdatei (`rt_reg`(4)) können nicht geändert werden.

Vorkompilierter Ressourcentyp

Der Ressourcentyp des generischen Datendienstes `SUNW.gds` ist im `SUNWscgds`-Paket enthalten. Das Dienstprogramm `scinstall` installiert dieses Paket während der Cluster-Installation (siehe `scinstall`(1M) in der Online-Dokumentation). Das `SUNWscgds`-Paket umfasst die folgenden Dateien:

```
# 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
```

Vorteile und Nachteile der Verwendung des GDS

Der GDS verfügt gegenüber dem in SunPlex Agent Builder erzeugten Quellcodemodell (siehe `scdscreate`(1HA) in der Online-Dokumentation) oder gegenüber den Standardverwaltungsbefehlen von Sun Cluster über die folgenden Vorteile:

- Der GDS ist einfach zu verwenden.
- Der GDS und seine Methoden sind vorkompiliert und können daher nicht geändert werden.
- SunPlex Agent Builder kann zum Generieren von Skripts für Ihre Anwendung verwendet werden. Diese Skripts werden dann in ein Solaris-Paket integriert, das auf mehreren Clustern wiederverwendet werden kann.

Obwohl die Verwendung des GDS große Vorteile bietet, ist sein Einsatz in manchen Fällen ungeeignet. In den folgenden Fällen ist der GDS *kein* geeigneter Mechanismus:

- Es ist mehr Steuerung erforderlich, als bei Verwendung des vorkompilierten Ressourcentyps möglich ist (beispielsweise wenn Sie Erweiterungseigenschaften hinzufügen oder Standardwerte ändern müssen).
- Der Quellcode muss geändert werden, um spezielle Funktionen hinzuzufügen.

Erstellungsmöglichkeiten für einen Dienst, der den GDS verwendet

Es gibt zwei Möglichkeiten zur Erstellung eines Dienstes, der den GDS verwendet:

- Verwenden von SunPlex Agent Builder
- Verwenden der Standardverwaltungsbefehle von Sun Cluster

GDS und SunPlex Agent Builder

Verwenden Sie SunPlex Agent Builder und wählen Sie "GDS" als Typ für den generierten Quellcode aus. Die Benutzereingabe wird für das Generieren einer Reihe von Skripts verwendet, die Ressourcen für die entsprechende Anwendung konfigurieren.

GDS und die Standardverwaltungsbefehle von Sun Cluster

Bei dieser Methode wird der vorkompilierte Datendienstcode in `SUNWscgds` verwendet. Sie erfordert jedoch, dass der Systemadministrator zum Erstellen und Konfigurieren der Ressource die Standardverwaltungsbefehle von Sun Cluster verwendet. Informationen hierzu finden Sie unter `scrgadm`(1M) und `scswitch`(1M) in der Online-Dokumentation.

Auswahl der Methode für die Erstellung eines GDS-basierten Dienstes

Wie in den Verfahren „So verwenden Sie Sun Cluster-Verwaltungsbefehle zum Erstellen eines hoch verfügbaren Dienstes, der GDS verwendet" auf Seite 207 und „So verwenden Sie Sun Cluster-Verwaltungsbefehle zum Erstellen eines skalierbaren Dienstes, der GDS verwendet" auf Seite 208 gezeigt, bedeutet die Ausgabe der geeigneten `scrgadm`- und `scswitch`-Befehle erheblichen Schreibaufwand.

Die Verwendung des GDS mit SunPlex Agent Builder vereinfacht den Prozess, da dadurch die Skripts generiert werden, welche die `scrgadm`- und `scswitch`-Befehle für Sie ausgeben.

GDS-Ereignisprotokollierung

Der GDS ermöglicht die Protokollierung wichtiger Informationen, die er an die von ihm gestarteten Skripts übergibt. Zu diesen wichtigen Informationen gehören der Status der Start-, Testsignal- und Stopp-Methoden sowie die Eigenschaftsvariablen. Sie können diese Informationen für die Diagnose von Problemen oder Fehlern in Ihren Skripts verwenden oder sie zu anderen Zwecken einsetzen.

Mithilfe der Eigenschaft `Log_level`, die unter „`Log_level`-Eigenschaft" auf Seite 202 beschrieben ist, können Sie die Stufe oder den Typ der Meldungen festlegen, die der GDS protokollieren soll. Mögliche Einstellungen sind `NONE`, `INFO` oder `ERR`.

GDS-Protokolldateien

Die folgenden beiden GDS-Protokolldateien werden im Verzeichnis `/var/cluster/logs/DS/`*Name_der_Ressourcengruppe*`/`*Ressourcenname* abgelegt:

- `start_stop_log.txt`; diese Datei enthält Meldungen, die durch die Start- und Stopp-Methoden der Ressource protokolliert werden.
- `probe_log.txt`; diese Datei enthält Meldungen, die vom Ressourcen-Monitor protokolliert werden.

Das folgende Beispiel zeigt die Arten von Informationen, die `start_stop_log.txt` enthält:

```
10/20/2004 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
10/20/2004 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

Das folgende Beispiel zeigt die Arten von Informationen, die `probe_log.txt` enthält:

```
10/20/2004 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
10/20/2004 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2004 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2004 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

Erforderliche GDS-Eigenschaften

Wenn Ihre Anwendung keine Netzwerkunterstützung bietet, müssen Sie sowohl die Erweiterungseigenschaft `Start_command` als auch die Eigenschaft `Port_list` angeben. Wenn Ihre Anwendung Netzwerkunterstützung bietet, müssen Sie nur die Eigenschaft `Port_list` angeben.

`Start_command`-Erweiterungseigenschaft

Der Start-Befehl, den Sie in der `Start_command`-Erweiterungseigenschaft angeben, startet die Anwendung. Es muss sich um einen UNIX-Befehl mit Argumenten handeln, die direkt an eine Shell zum Starten der Anwendung übergeben werden können.

`Port_list`-Eigenschaft

Die `Port_list`-Eigenschaft gibt die Liste der Ports an, die die Anwendung abhört. Die `Port_list`-Eigenschaft muss im Start-Skript, das von SunPlex Agent Builder erstellt wird, bzw. mit dem `scrgadm`-Befehl angegeben werden, wenn Sie die Standardverwaltungsbefehle von Sun Cluster verwenden.

Optionale GDS-Eigenschaften

Die folgende Liste enthält optionale GDS-Eigenschaften:

- `Network_resources_used`
- `Stop_command` (Erweiterungseigenschaft)
- `Probe_command` (Erweiterungseigenschaft)
- `Start_timeout`
- `Stop_timeout`
- `Probe_timeout` (Erweiterungseigenschaft)
- `Child_mon_level` (Erweiterungseigenschaft, die nur mit den Standardverwaltungsbefehlen verwendet wird)
- `Failover_enabled` (Erweiterungseigenschaft)
- `Stop_signal` (Erweiterungseigenschaft)
- `Log_level` (Erweiterungseigenschaft)

Eigenschaft `Network_resources_used`

Der Standardwert für diese Eigenschaft ist Null. Diese Eigenschaft muss angegeben werden, wenn die Anwendung an eine oder mehrere spezifische Adressen gebunden werden muss. Wenn diese Eigenschaft ausgelassen oder als `Null` festgelegt wird, wird angenommen, dass die Anwendung alle Adressen abhören soll.

Vor Erstellen der GDS-Ressource muss bereits eine `LogicalHostname`- oder `SharedAddress`-Ressource konfiguriert worden sein. In *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* finden Sie Informationen zum Konfigurieren einer `LogicalHostname`- bzw. `SharedAddress`-Ressource.

Zum Festlegen eines Wertes geben Sie einen oder mehrere Ressourcennamen an. Jeder Ressourcenname kann eine oder mehrere Instanzen von `LogicalHostname` oder `SharedAddress` enthalten. Detaillierte Informationen hierzu finden Sie in der Online-Dokumentation unter `r_properties` (5).

`Stop_command`-Eigenschaft

Der Stopp-Befehl muss die Anwendung stoppen und erst dann einen Wert zurückgeben, wenn die Anwendung vollständig gestoppt wurde. Dabei muss es sich um einen vollständigen UNIX-Befehl handeln, der direkt an eine Shell zum Stoppen der Anwendung übergeben werden kann.

Wenn die `Stop_command`-Erweiterungseigenschaft bereitgestellt wird, startet die GDS-Stopp-Methode den Stopp-Befehl mit 80% der Stopp-Zeitüberschreitung. Unabhängig vom Ergebnis der Ausgabe des Stopp-Befehls sendet die GDS-Stopp-Methode `SIGKILL` mit 15% der Stopp-Zeitüberschreitung. Die restlichen 5% der Zeit werden für Systemverwaltungsaufwand reserviert.

Wenn der Stopp-Befehl ausgelassen wird, versucht der GDS, die Anwendung unter Verwendung des in `Stop_signal` angegebenen Signals anzuhalten.

`Probe_command`-Eigenschaft

Der Testsignal-Befehl prüft in regelmäßigen Abständen die Fehlerfreiheit einer bestimmten Anwendung. Es muss sich um einen UNIX-Befehl mit Argumenten handeln, die direkt an eine Shell zum Testen der Anwendung übergeben werden können. Der Testsignal-Befehl gibt einen Beendigungsstatus von `0` zurück, wenn die Anwendung fehlerfrei läuft.

Der Beendigungsstatus des Testsignal-Befehls dient zum Feststellen, wie schwerwiegend der Fehler der Anwendung ist. Dieser Beendigungsstatus, der als Testsignal-Status bezeichnet wird, muss eine Ganzzahl zwischen `0` (Erfolg) und `100` (Totalfehlschlag) sein. Der Testsignal-Status kann auch `201` lauten. Dieser Wert bewirkt, dass für die Anwendung sofort ein Failover durchgeführt wird, es sei denn, `Failover_enabled` ist auf `FALSE` eingestellt. Der Testsignal-Status wird innerhalb des GDS-Testalgorithmus verwendet (siehe `scds_fm_action`(3HA) in der Online-Dokumentation), um zu entscheiden, ob die Anwendung lokal neu gestartet oder ob ein Failover auf einen anderen Knoten ausgeführt werden soll. Wenn der Beendigungsstatus `201` ist, wird für die Anwendung ein sofortiges Failover ausgeführt.

Wenn der Testsignal-Befehl ausgelassen wird, verwendet der GDS ein eigenes einfaches Testsignal, das eine Verbindung zur Anwendung an die aus der Eigenschaft `Network_resources_used` oder aus der Ausgabe von `scds_get_netaddr_list` abgeleiteten IP-Adressen herstellt (siehe `scds_get_netaddr_list`(3HA) in der Online-Dokumentation). Wenn die Verbindung erfolgreich ist, wird sie sofort wieder getrennt. Wenn sowohl die Verbindungsherstellung als auch die Verbindungstrennung erfolgreich verlaufen, wird davon ausgegangen, dass die Anwendung fehlerfrei läuft.

Hinweis – Das von dem GDS bereitgestellte Testsignal soll lediglich ein einfacher Ersatz für das voll funktionsfähige anwendungsspezifische Testsignal sein.

`Start_timeout`-Eigenschaft

Diese Eigenschaft gibt die Startzeitüberschreitung für den Start-Befehl an. Weitere Informationen hierzu finden Sie in „`Start_command`-Erweiterungseigenschaft" auf Seite 198. Der Standardwert für `Start_timeout` ist 300 Sekunden.

`Stop_timeout`-Eigenschaft

Diese Eigenschaft gibt die Stopp-Zeitüberschreitung für den Stopp-Befehl an. Weitere Informationen hierzu finden Sie in „`Stop_command`-Eigenschaft" auf Seite 199. Der Standardwert für `Stop_timeout` ist 300 Sekunden.

`Probe_timeout`-Eigenschaft

Diese Eigenschaft gibt den Zeitüberschreitungswert für den Testsignal-Befehl an. Weitere Informationen hierzu finden Sie in „`Probe_command`-Eigenschaft" auf Seite 200. Der Standardwert für `Probe_timeout` ist 30 Sekunden.

Eigenschaft `Child_mon_level`

Hinweis – Bei Verwendung der Standardverwaltungsbefehle von Sun Cluster können Sie diese Option verwenden. Bei Verwendung von SunPlex Agent Builder kann diese Option jedoch nicht eingesetzt werden.

Diese Eigenschaft ermöglicht die Steuerung der Prozesse, die mittels PMF (Process Monitor Facility) überwacht werden. Sie hält die Ebene fest, bis zu der die verzweigten untergeordneten Prozesse überwacht werden. Diese Eigenschaft hat dieselbe Wirkung wie das `-C`-Argument für den `pmfadm`-Befehl. Informationen hierzu finden Sie in der Online-Dokumentation unter `pmfadm`(1M).

Wenn diese Eigenschaft ausgelassen bzw. der Standardwert auf `-1` gesetzt wird, hat dies die gleiche Wirkung wie das Auslassen der Option `-C` für den `pmfadm`-Befehl. Das bedeutet, dass alle untergeordneten Prozesse und deren Folgeprozesse überwacht werden.

`Failover_enabled`-Eigenschaft

Diese boolesche Erweiterungseigenschaft steuert das Failover-Verhalten der Ressource. Wenn die Erweiterungseigenschaft auf `true` eingestellt wird, wird für die Anwendung ein Failover ausgeführt, wenn die Anzahl der Neustarts den Wert von `retry_count` innerhalb der Anzahl Sekunden aus `retry_interval` überschreitet.

Wenn diese Eigenschaft auf `false` eingestellt ist, startet die Anwendung nicht neu bzw. führt ein Failover auf einem anderen Knoten aus, wenn die Anzahl der Neustarts den Wert von `retry_count` innerhalb der Anzahl Sekunden aus `retry_interval` übersteigt.

Diese Eigenschaft kann verwendet werden, um zu verhindern, dass die Anwendungsressource ein Failover der Ressourcengruppe einleitet. Der Standardwert für diese Eigenschaft ist `true`.

`Stop_signal`-Eigenschaft

Der GDS verwendet den Wert dieser Ganzzahl-Erweiterungseigenschaft, um das Signal zu bestimmen, das für das Stoppen der Anwendung über PMF verwendet wird. Eine Liste der Ganzzahlwerte, die angegeben werden können, finden Sie in der Online-Dokumentation unter `signal`(3HEAD). Der Standardwert ist `15` (`SIGTERM`).

`Log_level`-Eigenschaft

Diese Eigenschaft gibt die Stufe, den Typ oder die Diagnosemeldungen an, die vom GDS protokolliert werden. Mögliche Werte für diese Eigenschaft sind `NONE`, `INFO` oder `ERR`. Wenn Sie `NONE` angeben, werden vom GDS keine Diagnosemeldungen protokolliert. Bei Angabe von `INFO` werden nur informative Meldungen protokolliert. Und wenn Sie `ERR` angeben, werden nur Fehlermeldungen protokolliert. Der GDS protokolliert standardmäßig keine Diagnosemeldungen (`NONE`).

Verwenden von SunPlex Agent Builder zum Erstellen eines Dienstes, der GDS verwendet

Sie können SunPlex Agent Builder zur Erstellung des Dienstes einsetzen, der den GDS verwendet. SunPlex Agent Builder wird in Kapitel 9 genauer beschrieben.

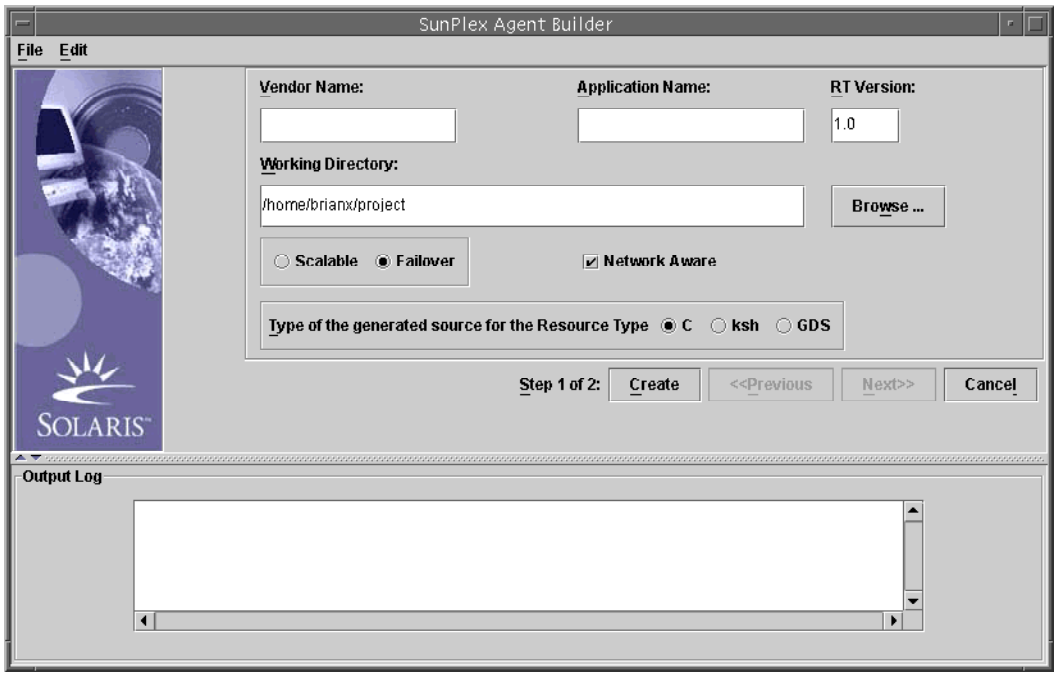
Erstellen und Konfigurieren der Skripts

▼ So starten Sie SunPlex Agent Builder und erstellen Sie die Skripts

1. **Nehmen Sie Superuser-Status oder eine entsprechende administrative Rolle an.**
2. **Starten Sie SunPlex Agent Builder.**

   ```
   # /usr/cluster/bin/scdsbuilder
   ```

3. **Der Bildschirm "Create" von SunPlex Agent Builder wird angezeigt.**



4. **Geben Sie den Herstellernamen ein.**
5. **Geben Sie den Anwendungsnamen ein.**

Hinweis – Die Kombination aus Herstellernamen und Anwendungsnamen darf neun Zeichen nicht überschreiten. Sie dient als Name für das Paket der Skripts.

---

6. **Gehen Sie zum Arbeitsverzeichnis.**

   Anstatt den Pfad einzugeben, können Sie das Verzeichnis auch über das Pulldown-Menü "Browse" auswählen.

7. **Wählen Sie aus, ob der Datendienst "Scalable" oder "Failover" ist.**

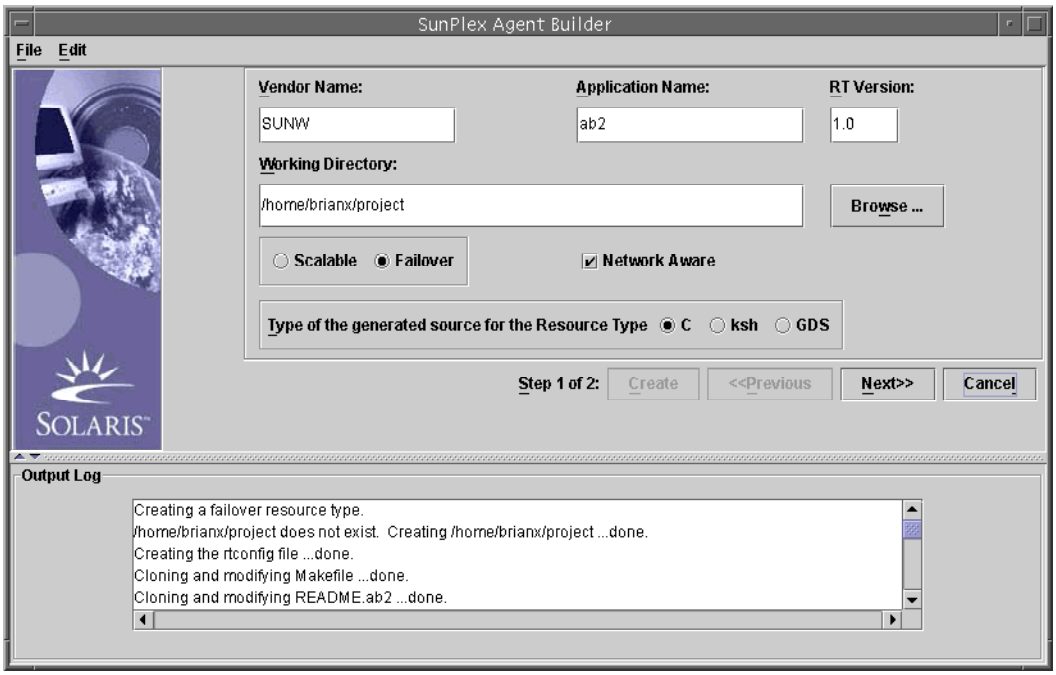
   Sie müssen "Network Aware" nicht auswählen, da die Netzwerkunterstützung bei Erstellung des GDS standardmäßig aktiviert ist.

8. **Wählen Sie "GDS" aus.**
9. **(Optional) Ändern Sie die RT-Version vom angezeigten Standardwert in einen anderen Wert.**

Hinweis – Im Feld "RT Version" dürfen die folgenden Zeichen nicht verwendet werden: Leerzeichen, Tabulator, Schrägstrich (`/`), umgekehrter Schrägstrich (`\`), Sternchen (`*`), Fragezeichen (`?`), Komma (`,`), Strichpunkt (`;`), linke eckige Klammer (`[`) oder rechte eckige Klammer (`]`).

10. **Klicken Sie auf "Fertig stellen".**

    Agent Builder erstellt die Skripts. Das Ergebnis der Erstellung des Dienstes wird im Fenster "Output Log" angezeigt.



"Create" ist abgeblendet dargestellt. Nun können Sie die Skripts konfigurieren.

11. **Klicken Sie auf "Next".**

    Der Bildschirm "Configuration" wird angezeigt.

▼ So konfigurieren Sie die Skripts

Nach Erstellung der Skripts müssen Sie den neuen Dienst konfigurieren.

1. **Geben Sie den Speicherort des Start-Befehls ein oder klicken Sie auf "Browse", um nach dem Start-Befehl zu suchen.**

Sie können auch Eigenschaftsvariablen festlegen. Eine Beschreibung der Eigenschaftsvariablen finden Sie unter „Eigenschaftsvariablen" auf Seite 180.

2. **(Optional) Geben Sie den Stopp-Befehl ein oder klicken Sie auf "Browse", um den Stopp-Befehl zu suchen.**

   Sie können auch Eigenschaftsvariablen festlegen. Eine Beschreibung der Eigenschaftsvariablen finden Sie unter „Eigenschaftsvariablen" auf Seite 180.

3. **(Optional) Geben Sie den Testsignal-Befehl ein oder klicken Sie auf "Browse", um den Testsignal-Befehl zu suchen.**

   Sie können auch Eigenschaftsvariablen festlegen. Eine Beschreibung der Eigenschaftsvariablen finden Sie unter „Eigenschaftsvariablen" auf Seite 180.

4. **(Optional) Geben Sie die Zeitüberschreitungswerte für die Start-, Stopp- und Testsignal-Befehle an.**

5. **Klicken Sie auf "Configure".**

   Agent Builder beginnt mit dem Konfigurieren der Skripts.

Hinweis – Agent Builder verkettet den Herstellernamen und den Anwendungsnamen zur Erstellung des Paketnamens.

Ein Paket für Skripts wird erstellt und im folgenden Verzeichnis abgelegt:

Arbeitsverzeichnis/*HerstellernameAnwendung*/pkg

Zum Beispiel: `/export/wdir/NETapp/pkg`

6. **Melden Sie sich als Superbenutzer an und installieren Sie das fertige Paket auf allen Knoten des Clusters.**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

7. **Die folgenden Dateien werden von `pkgadd` installiert:**

```
/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
```

Hinweis – Die Online-Dokumentation und Skriptnamen entsprechen dem zuvor eingegebenen Anwendungsnamen, mit vorangestelltem Skriptnamen (zum Beispiel `startapp`).

Zum Anzeigen der Online-Dokumentation müssen Sie den Pfad zu dieser Dokumentation angeben. Zum Anzeigen der Online-Dokumentationsseite `startapp`(1M) beispielsweise geben Sie Folgendes ein:

```
# man -M /opt/NETapp/man startapp
```

8. **Konfigurieren Sie die Ressourcen auf einem Knoten des Clusters, und starten Sie die Anwendung.**

   # **/opt/NETapp/util/startapp -h** *logischer_Hostname* **-p** *Port-_und_Protokollliste*

   Die Argumente für das `startapp`-Skript sind je nach Ressourcentyp unterschiedlich: Failover oder Scalable. Prüfen Sie die angepasste Online-Dokumentation oder führen Sie das `startapp`-Skript ohne Argumente aus, um eine Syntaxanweisung anzuzeigen.

```
# /opt/NETapp/util/startapp
 Der Ressourcenname von LogicalHostname bzw. SharedAddress muss
 angegeben werden.
Für Failover-Dienste:
 Syntax: startapp -h logischer_Hostname
           -p Port-_und_Protokollliste
           [-n IPIM-Gruppe/Adapterliste]
 Für Scalable-Dienste:
 Syntax: startapp -h gemeinsam_genutzer_Adressname
           -p Port-_und_Protokollliste
           [-l Lastausgleichsverfahren]
           [-n IPMP-Gruppe/Adapterliste]
           [-w Lastausgleichsgewichtung]
```

Ausgabe von SunPlex Agent Builder

SunPlex Agent Builder generiert basierend auf Ihren Eingaben während der Paketerstellung drei Skripts und eine Konfigurationsdatei. Die Konfigurationsdatei gibt die Namen der Ressourcengruppe und des Ressourcentyps an.

Die Skripts sind:

- Start-Skript: Wird zum Konfigurieren der Ressourcen und zum Starten der Anwendung verwendet, die von RGM gesteuert wird.
- Stop-Skript: Wird für das Stoppen der Anwendung und Herunterfahren von Ressourcen und Ressourcengruppen verwendet.
- Remove-Skript: Wird für das Entfernen der Ressourcen und Ressourcengruppen verwendet, die vom Start-Skript erstellt wurden.

Diese Skripts haben dieselbe Schnittstelle und das gleiche Verhalten wie die Dienstprogrammskripts, die von SunPlex Agent Builder für nicht-GDS-basierte Agenten generiert werden. Die Skripts werden in einem Solaris-Paket zusammengestellt, das auf mehreren Clustern wiederverwendet werden kann.

Sie können die Konfigurationsdatei anpassen, um eigene Namen für die Ressourcengruppen oder andere Parameter anzugeben, die normalerweise über den `scrgadm`-Befehl eingegeben werden. Wenn Sie die Skripts nicht anpassen, stellt SunPlex Agent Builder Standardwerte für die `scrgadm`-Parameter bereit.

Verwenden der Standardverwaltungsbefehle von Sun Cluster zum Erstellen eines Dienstes, der GDS verwendet

In diesem Abschnitt wird beschrieben, wie Parameter für den GDS eingegeben werden. Zur Verwendung und Verwaltung des GDS werden die bestehenden Sun Cluster-Verwaltungsbefehle wie `scrgadm` und `scswitch` eingesetzt.

Es ist nicht erforderlich, die in diesem Abschnitt aufgeführten Verwaltungsbefehle auf niedriger Ebene einzugeben, wenn die Skripts angemessene Funktionalität bereitstellen. Sie können jedoch die Verwaltungsbefehle auf niedriger Ebene eingeben, wenn eine feinere Steuerung der GDS-basierten Ressource erforderlich ist. Diese Befehle werden von den Skripts ausgeführt.

▼ So verwenden Sie Sun Cluster-Verwaltungsbefehle zum Erstellen eines hoch verfügbaren Dienstes, der GDS verwendet

1. **Nehmen Sie Superuser-Status oder eine entsprechende administrative Rolle an.**
2. **Registrieren Sie den Ressourcentyp `SUNW.gds`.**

   ```
   # scrgadm -a -t SUNW.gds
   ```

3. **Erstellen Sie die Ressourcengruppe, welche die `LogicalHostname` -Ressource und den Failover-Dienst selbst enthält.**

   ```
   # scrgadm -a -g haapp_rg
   ```

4. **Erstellen Sie die Ressource für die `LogicalHostname`-Ressource.**

```
# scrgadm -a -L -g haapp_rs -l hhead
```

5. **Erstellen Sie die Ressource für den Failover-Dienst selbst.**

```
# 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. **Bringen Sie die Ressourcengruppe `haapp_rg` online.**

```
# scswitch -Z -g haapp_rg
```

▼ So verwenden Sie Sun Cluster-Verwaltungsbefehle zum Erstellen eines skalierbaren Dienstes, der GDS verwendet

1. **Nehmen Sie Superuser-Status oder eine entsprechende administrative Rolle an.**

2. **Registrieren Sie den Ressourcentyp `SUNW.gds`.**

```
# scrgadm -a -t SUNW.gds
```

3. **Erstellen Sie die Ressourcengruppe für die `SharedAddress`-Ressource.**

```
# scrgadm -a -g sa_rg
```

4. **Erstellen Sie die `SharedAddress`-Ressource auf `sa_rg`.**

```
# scrgadm -a -S -g sa_rg -l hhead
```

5. **Erstellen Sie die Ressourcengruppe für den Scalable-Dienst.**

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

6. **Erstellen Sie die Ressourcengruppe für den Scalable-Dienst selbst.**

```
# 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. **Bringen Sie die Ressourcengruppe, welche die Netzwerkressourcen enthält, online.**

```
# scswitch -Z -g sa_rg
```

8. **Bingen Sie die Ressourcengruppe `app_rg` online.**

```
# scswitch -Z -g app_rg
```

Befehlszeilenschnittstelle für SunPlex Agent Builder

SunPlex Agent Builder verfügt auch über eine Befehlszeilenschnittstelle, welche dieselbe Funktionalität bietet wie die grafische Benutzeroberfläche. Diese Schnittstelle besteht aus den Befehlen `scdscreate` und `scdsconfig`. Informationen hierzu finden Sie in der Online-Dokumentation unter `scdscreate` (1HA) und `scdsconfig`(1HA).

▼ So erstellen Sie einen Dienst, der GDS verwendet, mit der Befehlszeilenversion von Agent Builder

In diesem Abschnitt wird beschrieben, wie Sie die unter „Verwenden von SunPlex Agent Builder zum Erstellen eines Dienstes, der GDS verwendet" auf Seite 202 gezeigten Schritte mithilfe der Befehlszeilenschnittstelle ausführen können.

1. **Nehmen Sie Superuser-Status oder eine entsprechende administrative Rolle an.**

2. **Erstellen Sie den Dienst.**

   Geben Sie für einen Failover-Dienst folgenden Befehl ein:

   ```
   # scdscreate -g -V NET -T app -d /export/Arbeitsverzeichnis
   ```

   Geben Sie für einen Scalable-Dienst folgenden Befehl ein:

   ```
   # scdscreate -g -s -V NET -T app -d /export/Arbeitsverzeichnis
   ```

   Hinweis – Der Parameter `-d` ist optional. Wenn Sie diesen Parameter nicht angeben, wird das aktuelle Verzeichnis zum Arbeitsverzeichnis.

3. **Konfigurieren Sie den Dienst.**

```
# scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" \
-m "/export/app/bin/probe" -d /export/Arbeitsverzeichnis
```

Sie können auch Eigenschaftsvariablen festlegen. Eine Beschreibung der Eigenschaftsvariablen finden Sie unter „Eigenschaftsvariablen" auf Seite 180.

Hinweis – Nur der `start`-Befehl ist erforderlich. Alle anderen Parameter sind optional.

---

4. **Installieren Sie das fertige Paket auf allen Knoten des Clusters.**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

5. **Die folgenden Dateien werden von `pkgadd` installiert:**

```
/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
```

Hinweis – Die Online-Dokumentation und Skriptnamen entsprechen dem zuvor eingegebenen Anwendungsnamen, mit vorangestelltem Skriptnamen (zum Beispiel `startapp`).

Zum Anzeigen der Online-Dokumentation müssen Sie den Pfad zu dieser Dokumentation angeben. Zum Anzeigen der Online-Dokumentationsseite `startapp`(1M) beispielsweise geben Sie Folgendes ein:

```
# man -M /opt/NETapp/man startapp
```

6. **Konfigurieren Sie die Ressourcen auf einem Knoten des Clusters, und starten Sie die Anwendung.**

```
# /opt/NETapp/util/startapp -h logischer_Hostname -p Port-_und_Protokollliste
```

Die Argumente für das `startapp`-Skript sind je nach Ressourcentyp unterschiedlich: Failover oder Scalable. Prüfen Sie die angepasste Online-Dokumentation oder führen Sie das `startapp`-Skript ohne Argumente

aus, um eine Syntaxanweisung anzuzeigen.

```
# /opt/NETapp/util/startapp
 Der Ressourcenname von LogicalHostname bzw. SharedAddress muss
 angegeben werden.
 Für Failover-Dienste:
 Syntax: startapp -h logischer_Hostname
           -p Port-_und_Protokollliste
           [-n IPIM-Gruppe/Adapterliste]
 Für Scalable-Dienste:
 Syntax: startapp -h gemeinsam_genutzer_Adressname
           -p Port-_und_Protokollliste
           [-l Lastausgleichsverfahren]
           [-n IPMP-Gruppe/Adapterliste]
           [-w Lastausgleichsgewichtung]
```

KAPITEL 11

DSDL-Referenz

Dieses Kapitel listet die API-Funktionen der DSDL (Data Service Development Library, Datendienst-Entwicklungsbibliothek) auf und beschreibt sie kurz. In den einzelnen Seiten der 3HA-Online-Dokumentation finden Sie eine vollständige Beschreibung jeder DSDL-Funktion. Die DSDL definiert lediglich eine C-Schnittstelle. Eine skriptbasierte DSDL-Schnittstelle ist nicht verfügbar.

Die DSDL bietet die folgenden Arten von Funktionen:

DSDL-Funktionen

Die folgenden Unterabschnitte enthalten einen kurzen Überblick über jede Kategorie der DSDL-Funktionen. Als maßgebliche Referenz für die DSDL-Funktionen wird jedoch auf die einzelnen Seiten der 3HA-Online-Dokumentation verwiesen.

Funktionen für allgemeine Zwecke

Die Funktionen in diesem Abschnitt decken einen großen Funktionsumfang ab. Diese Funktionen ermöglichen Ihnen das Ausführen der folgenden Vorgänge:

- Die DSDL-Umgebung initialisieren,
- Ressourcen, Ressourcentypen und Ressourcengruppennamen sowie Erweiterungseigenschaftswerte abrufen,

- Ein Failover für eine Ressourcengruppe ausführen und sie neu starten sowie eine Ressource neu starten,
- Fehler-Zeichenketten in Fehlermeldungen konvertieren,
- Einen Befehl im Rahmen einer Zeitüberschreitung ausführen.

Die folgenden Funktionen initialisieren die Aufrufmethode:

- `scds_initialize`(3HA) – ordnet Ressourcen zu und initialisiert die DSDL-Umgebung.
- `scds_close`(3HA) – gibt Ressourcen frei, die von `scds_initialize` zugeordnet wurden.

Die folgenden Funktionen rufen Informationen über Ressourcen, Ressourcentypen, Ressourcengruppen und Erweiterungseigenschaften ab.

- `scds_get_resource_name`(3HA) – ruft den Namen der Ressource für das aufrufende Programm ab.
- `scds_get_resource_type_name`(3HA) – ruft den Namen des Ressourcentyps für das aufrufende Programm ab.
- `scds_get_resource_group_name`(3HA) – ruft den Namen der Ressourcengruppe für das aufrufende Programm ab.
- `scds_get_ext_property`(3HA) – ruft den Wert der angegebenen Erweiterungseigenschaft ab.
- `scds_free_ext_property`(3HA) – gibt den durch `scds_get_ext_property` zugeordneten Speicher frei.

Die folgende Funktion ruft Statusinformationen über die `SUNW.HAStoragePlus` ab, die von einer Ressource verwendet werden.

- `scds_hasp_check`(3HA) – ruft Statusinformationen zu `SUNW.HAStoragePlus`(5)-Ressourcen ab, die von einer Ressource verwendet werden. Diese Informationen werden aus dem Zustand (online oder anderweitig) aller `SUNW.HAStoragePlus`-Ressourcen abgerufen, von denen die Ressource abhängt, und zwar unter Verwendung der für die Ressource definierten Systemeigenschaften `Resource_dependencies` oder `Resource_dependencies_weak`.

Die folgenden Funktionen führen ein Failover aus bzw. starten eine Ressource oder Ressourcengruppe neu.

- `scds_failover_rg`(3HA) – führt ein Failover für eine Ressourcengruppe aus.
- `scds_restart_rg`(3HA) – startet eine Ressourcengruppe neu.
- `scds_restart_resource`(3HA) – startet eine Ressource neu.

Die folgenden Funktionen führen einen Befehl im Rahmen einer Zeitüberschreitung aus und konvertieren einen Fehlercode in eine Fehlermeldung:

- `scds_timerun`(3HA) – führt einen Befehl unter einem Zeitüberschreitungswert aus.

- `scds_error_string`(3HA) – übersetzt einen Fehlercode in eine Fehlerzeichenkette.

Eigenschaftsfunktionen

Diese Funktionen stellen praktische APIs für den Zugriff auf bestimmte Eigenschaften der entsprechenden Ressource, Ressourcengruppe und des Ressourcentyps bereit, einschließlich einiger häufig verwendeter Erweiterungseigenschaften. Die DSDL stellt die `scds_initialize`-Funktion für die Analyse der Befehlszeilenargumente bereit. Die Bibliothek legt dann die verschiedenen Eigenschaften der entsprechenden Ressource, Ressourcengruppe bzw. des Ressourcentyps im *Cache-Speicher* ab.

`scds_property_functions`(3HA) beschreibt diese Funktionen. Diese Funktionen umfassen:

- `scds_get_rs_`*Eigenschaftsname*
- `scds_get_rg_`*Eigenschaftsname*
- `scds_get_rt_`*Eigenschaftsname*
- `scds_get_ext_`*Eigenschaftsname*

Funktionen für den Zugriff auf Netzwerkressourcen

Die in diesem Abschnitt aufgelisteten Funktionen rufen von Ressourcen und Ressourcengruppen verwendete Netzwerkressourcen auf, drucken sie und geben sie frei. Die `scds_get_*`-Funktionen in diesem Abschnitt bieten eine praktische Möglichkeit zum Abrufen von Netzwerkressourcen, ohne spezifische Eigenschaften wie `Network_resources_used` und `Port_list` abzufragen. Dabei werden die RMAPI-Funktionen eingesetzt. Die Funktionen `scds_print_`*Name*`()` drucken Werte aus den Datenstrukturen, die von den Funktionen `scds_get_`*Name*`()` zurückgegeben werden. Die Funktionen `scds_free_`*Name*`()` geben den Speicherplatz frei, der von den Funktionen `scds_get_`*Name*`()` zugewiesen wurde.

Zu den Funktionen, die Hostnamen verarbeiten, gehören:

- `scds_get_rg_hostnames`(3HA) – ruft eine Liste der Hostnamen auf, die von den Netzwerkressourcen in einer Ressourcengruppe verwendet werden.
- `scds_get_rs_hostnames`(3HA) – ruft eine Liste der Hostnamen auf, die von der Ressourcen verwendet werden.
- `scds_print_net_list`(3HA) – druckt den Inhalt der Hostnamenliste, die von `scds_get_rg_hostnames()` oder `scds_get_rs_hostnames()` ausgegeben wird.
- `scds_free_net_list`(3HA) – gibt den durch `scds_get_rg_hostnames()` oder `scds_get_rs_hostnames ()` zugeordneten Speicher frei.

Zu den Funktionen, die Port-Listen verarbeiten, gehören:

- `scds_get_port_list`(3HA) – ruft eine Liste der Port-Protokoll-Paare auf, die von einer Ressource verwendet werden.
- `scds_print_port_list`(3HA) – druckt den Ihnalt der Liste der Port-Protokoll-Paare, die von `scds_get_port_list ()` ausgegeben wird.
- `scds_free_port_list`(3HA) – gibt den durch`scds_get_port_list()` zugeordneten Speicher frei.

Zu den Funktionen, die Netzwerkadressen verarbeiten, gehören:

- `scds_get_netaddr_list`(3HA) – ruft eine Liste der Netzwerkadressen ab, die von einer Ressource verwendet werden.
- `scds_print_netaddr_list`(3HA) – druckt den Inhalt der Liste der Netzwerkadressen, die von `scds_get_netaddr_list` ausgegeben wird.
- `scds_free_netaddr_list`(3HA) – gibt den durch `scds_get_netaddr_list` zugeordneten Speicher frei.

Fehlerüberwachung mit TCP-Verbindungen

Die Funktionen in diesem Abschnitt aktivieren die TCP-basierte Überwachung. In der Regel verwendet ein Fehler-Monitor diese Funktionen, um eine einfache Socketverbindung mit einem Dienst herzustellen, Daten bezüglich des Dienstes zu lesen und zu schreiben, um dessen Status festzustellen, sowie die Verbindung mit dem Dienst zu trennen.

Diese Funktionen umfassen:

- `scds_fm_tcp_connect`(3HA) – stellt eine TCP-Verbindung zu einem Prozess her, der nur die IPv4-Adressierung verwendet.
- `scds_fm_net_connect`(3HA) – stellt eine TCP-Verbindung zu einem Prozess her, der die IPv4- oder IPv6-Adressierung verwendet.
- `scds_fm_tcp_read`(3HA) – verwendet eine TCP-Verbindung zum Lesen von Daten aus dem überwachten Prozess.
- `scds_fm_tcp_write`(3HA) – verwendet eine TCP-Verbindung zum Schreiben von Daten in einen überwachten Prozess.
- `scds_simple_probe`(3HA) – testet einen Prozess durch Herstellen und Beenden einer TCP-Verbindung zu diesem Prozess. Diese Funktion verarbeitet nur IPv4-Adressen.
- `scds_simple_net_probe`(3HA) – testet einen Prozess durch Herstellen und Beenden einer TCP-Verbindung zu diesem Prozess. Diese Funktion verarbeitet IPv4- und IPv6-Adressen.
- `scds_fm_tcp_disconnect`(3HA) – beendet die Verbindung zu einem überwachten Prozess, der die IPv4-Adressierung verwendet.

- `scds_fm_net_disconnect`(3HA) – beendet die Verbindung zu einem überwachten Prozess, der die IPv4- oder IPv6-Adressierung verwendet.

PMF-Funktionen

Diese Funktionen kapseln die PMF-Funktionalität ein. Das DSDL-Modell für die Überwachung über PMF erstellt und verwendet implizite *tag*-Werte für `pmfadm`(1M). PMF verwendet auch implizite Werte für `Restart_interval`, `Retry_count` und `action_script` (die Optionen `-t`, `-n` und `-a` für `pmfadm`). Am wichtigsten ist, dass die DSDL die Prozessausfallhistorie, die PMF feststellt, in die Anwendungsfehlerhistorie einbindet, die der Fehler-Monitor feststellt. So wird die Entscheidung über Neustart oder Failover getroffen.

Diese Funktionen umfassen:

- `scds_pmf_get_status`(3HA) – ermittelt, ob die angegebene Instanz unter PMF-Steuerung überwacht wird.
- `scds_pmf_restart_fm`(3HA) – startet den Fehler-Monitor unter Verwendung von PMF neu.
- `scds_pmf_signal`(3HA) – sendet das angegebene Signal an einen Prozessbaum, der unter PMF-Steuerung ausgeführt wird.
- `scds_pmf_start`(3HA) – führt ein festgelegtes Programm (einschließlich eines Fehler-Monitors) unter PMF-Steuerung aus.
- `scds_pmf_stop`(3HA) – beendet einen Prozess, der unter PMF-Steuerung ausgeführt wird.
- `scds_pmf_stop_monitoring`(3HA) – stoppt die Überwachung eines Prozesses, der unter PMF-Steuerung ausgeführt wird.

Fehler-Monitor-Funktionen

Die Funktionen in diesem Abschnitt stellen ein voreingestelltes Fehlerüberwachungsmodell bereit. Dabei wird die Fehlerhistorie festgehalten und zusammen mit den Eigenschaften `Retry_count` und `Retry_interval` ausgewertet.

Diese Funktionen umfassen:

- `scds_fm_sleep`(3HA) – wartet auf eine Nachricht an einem Steuerungs-Socket des Fehler-Monitors.
- `scds_fm_action`(3HA) – startet bestimmte Aktionen nach Abschluss eines Testvorgangs.
- `scds_fm_print_probes`(3HA) – schreibt die Teststatusinformationen in das Systemprotokoll.

Dienstprogrammfunktionen

Mit den Funktionen in diesem Abschnitt können Meldungen und Fehlerbehebungsmeldungen ins Systemprotokoll geschrieben werden. Diese Funktionen umfassen:

- `scds_syslog`(3HA) – schreibt Meldungen in das Systemprotokoll.
- `scds_syslog_debug`(3HA) – schreibt eine Debugging-Meldung in das Systemprotokoll.

KAPITEL 12

CRNP

Dieses Kapitel enthält Informationen zum CRNP (Cluster Reconfiguration Notification Protocol). Das CRNP ermöglicht die "Cluster-Unterstützung" von Failover- und Scalable-Anwendungen. Insbesondere stellt das CRNP einen Mechanismus bereit, mit dessen Hilfe sich die Anwendungen für Sun Cluster-Rekonfigurationsereignisse registrieren und anschließend asynchrone Benachrichtigungen darüber erhalten können. Datendienste, die innerhalb des Clusters, und Anwendungen, die außerhalb des Clusters ausgeführt werden, können sich für die Ereignisbenachrichtigung registrieren. Ereignisse werden generiert, wenn sich die Mitgliedschaft in einem Cluster ändert und wenn sich der Zustand einer Ressourcengruppe oder einer Ressource ändert.

Hinweis – Die `SUNW.Event`-Ressourcentypimplementierung bietet CRNP-Dienste mit hoher Verfügbarkeit in Sun Cluster. Eine detaillierte Beschreibung der `SUNW.Event`-Ressourcentypimplementierung finden Sie in der Online-Dokumentation unter `SUNW.Event`(5).

Überblick über CRNP

CRNP stellt Mechanismen und Dämonen bereit, die Cluster-Rekonfigurationsereignisse generieren, diese durch den Cluster routen und sie an interessierte Clients senden.

Der `cl_apid`-Dämon arbeitet interaktiv mit den Clients zusammen. Sun Cluster-Ressourcengruppen-Manager (RGM) generiert Cluster-Rekonfigurationsereignisse. Diese Dämone verwenden `sysevent`d(1M), um Ereignisse an jeden lokalen Knoten zu übertragen. Der `cl_apid`-Dämon verwendet XML (Extensible Markup Language) über TCP/IP, um mit den interessierten Clients zu kommunizieren.

Das folgende Diagramm stellt einen Überblick über den Ereignisfluss zwischen den CRNP-Komponenten dar. In diesem Diagramm wird ein Client auf Cluster-Knoten 2 ausgeführt, und der andere Client läuft auf einem Computer, der nicht zum Cluster gehört.

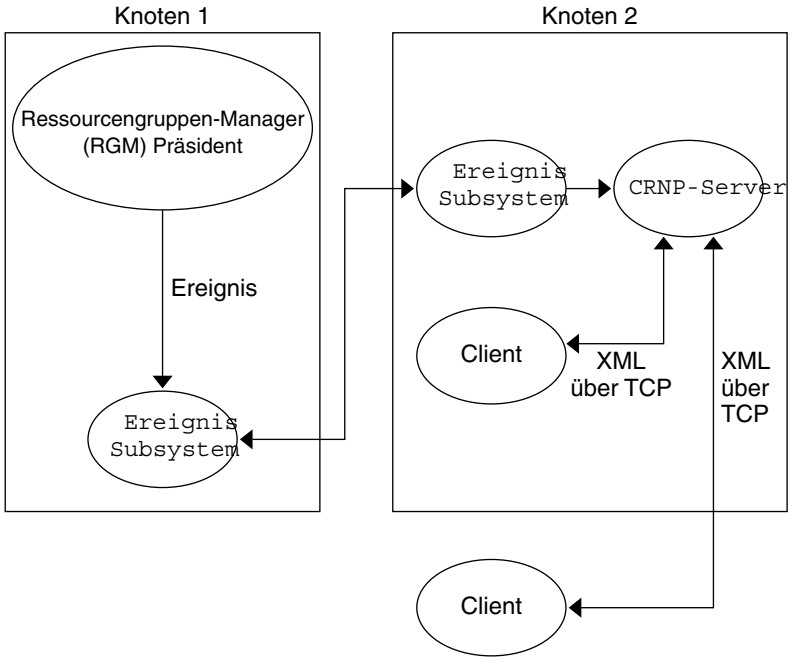


ABBILDUNG 12–1 Funktionsweise des CRNP

Überblick über das CRNP-Protokoll

CRNP definiert die Anwendungs-, Darstellungs und Sitzungsschichten des standardmäßigen OSI-Protokollstapels mit sieben Ebenen (OSI, Open System Interconnect). Die Transportschicht muss TCP und die Netzwerkebene muss IP sein. Das CRNP ist von den Datenverbindungs- und realen Schichten unabhängig. Alle Meldungen der Anwendungsschicht, die im CRNP ausgetauscht werden, basieren auf XML 1.0.

Semantik des CRNP-Protokolls

Die Clients leiten die Kommunikation ein, indem sie eine Registrierungsmeldung (`SC_CALLBACK_RG`) an den Server senden. Diese Registrierungsmeldung gibt die Ereignistypen an, für die Clients Benachrichtigungen erhalten möchten, sowie den Port, an den die Ereignisse zugestellt werden können. Das Quell-IP der Registrierungsverbindung und der angegebene Port bilden zusammen die Rückmeldeadresse.

Immer wenn im Cluster ein Ereignis eintritt, das für den Client von Interesse ist, kontaktiert der Server den Client über die Rückmeldeadresse (IP und Port) und stellt dem Client das Ereignis (`SC_EVENT`) zu. Der Server ist hoch verfügbar und wird im Cluster selbst ausgeführt. Der Server speichert Clientregistrierungen in einem Speicher, der auch bei einem Neustart des Clusters nicht gelöscht wird.

Clients können sich deregistrieren, indem sie eine Registrierungsmeldung (`SC_CALLBACK_RG` mit einer `REMOVE_CLIENT`-Meldung) an den Server senden. Nachdem der Client eine `SC_REPLY`-Meldung vom Server erhalten hat, beendet er die Verbindung.

Das folgende Diagramm zeigt den Kommunikationsfluss zwischen einem Client und einem Server.

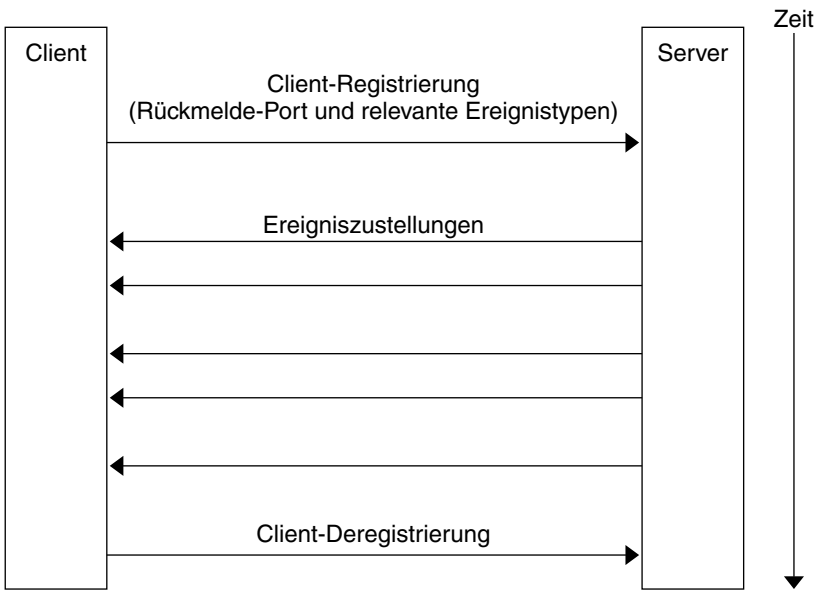


ABBILDUNG 12–2 Kommunikationsfluss zwischen einem Client und einem Server

Vom CRNP verwendete Meldungstypen

Das CRNP verwendet drei XML-basierte Meldungstypen, die in der folgenden Tabelle beschrieben werden. Weitere Einzelheiten zu diesen Meldungstypen werden weiter unten in diesem Kapitel beschrieben. Auch auf die Syntax wird später in diesem Kapitel noch genauer eingegangen.

Meldungstyp	Beschreibung
SC_CALLBACK_REG	Diese Meldung kann in vier Formen vorkommen: ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS und REMOVE_EVENTS. Jede dieser Formen enthält folgende Informationen: ■ Protokollversion ■ Rückmelde-Port in ASCII-Format (nicht Binärformat) Die Formen ADD_CLIENT, ADD_EVENTS und REMOVE_EVENTS enthalten daneben eine uneingeschränkte Liste von Ereignistypen. Jeder Typ enthält folgende Informationen: ■ Ereignisklasse ■ Ereignisunterklasse (optional) ■ Liste der Namens- und Wertepaare (optional) Die Ereignisklasse und die Ereignisunterklasse definieren zusammen einen einmaligen "Ereignistyp." Die DTD (Document Type Definition), von der aus die SC_CALLBACK_REG-Klassen generiert werden, ist SC_CALLBACK_REG. Diese DTD wird in Anhang F genauer beschrieben.

SC_EVENT	Diese Meldung enthält folgende Informationen:
	■ Protokollversion
	■ Ereignisklasse
	■ Ereignisunterklasse
	■ Hersteller
	■ Herausgeber
	■ Liste der Namens- und Wertepaare (0 oder mehr Namens- und Wertepaar-Datenstrukturen)
	  ■ Name (Zeichenkette)
	  ■ Wert (Zeichenkette oder Zeichenketten-Array)
	
	Die Werte in einem SC_EVENT werden nicht eingegeben. Die DTD (Document Type Definition), von der aus die SC_EVENT-Klassen generiert werden, ist SC_EVENT. Diese DTD wird in Anhang F genauer beschrieben.

SC_REPLY	Diese Meldung enthält folgende Informationen: ■ Protokollversion ■ Fehlercode ■ Fehlermeldung Die DTD (Document Type Definition), von der aus die SC_REPLY-Klassen generiert werden, ist SC_REPLY. Diese DTD wird in Anhang F genauer beschrieben.

Client-Registrierung beim Server

Dieser Abschnitt beschreibt, wie ein Verwalter den Server einrichtet, wie Clients identifiziert werden, wie Informationen über die Anwendungsschicht und die Sitzungsebene gesendet werden, sowie die Fehlerbedingungen.

Annahmen bezüglich der Serverkonfiguration durch Verwalter

Der Systemadministrator muss den Server mit einer hoch verfügbaren IP-Adresse, also einer IP-Adresse, die nicht an einen bestimmten Rechner im Cluster gebunden ist, sowie mit einer Port-Nummer konfigurieren. Er muss diese Netzwerkadresse an mögliche Clients veröffentlichen. Das CRNP definiert nicht, wie der Servername den Clients verfügbar gemacht wird. Die Verwalter können entweder einen Namensdienst einsetzen, mit dessen Hilfe die Clients die Netzwerkadresse des Servers dynamisch finden können, oder sie fügen den Netzwerknamen einer vom Client gelesenen Konfigurationsdatei hinzu. Der Server wird im Cluster als Failover-Ressourcentyp ausgeführt.

Client-Identifizierung durch den Server

Jeder Client wird durch seine Rückmeldeadresse, also die IP-Adresse und Port-Nummer, eindeutig identifiziert. Der Port wird in den `SC_CALLBACK_REG`-Meldungen angegeben, und die IP-Adresse wird aus der TCP-Registrierungsverbindung abgerufen. Das CRNP geht davon aus, dass alle folgenden `SC_CALLBACK_REG`-Meldungen mit der gleichen Rückmeldeadresse von demselben Client kommen, auch wenn der Quell-Port, von dem aus die Meldungen gesendet werden, unterschiedlich ist.

Senden von `SC_CALLBACK_REG`-Meldungen zwischen einem Client und dem Server

Ein Client leitet die Registrierung ein, indem er eine TCP-Verbindung mit der IP-Adresse und Port-Nummer des Servers herstellt. Wenn die TCP-Verbindung hergestellt und schreibbereit ist, muss der Client die Registrierungsmeldung senden. Die Registrierungsmeldung muss eine korrekt formatierte `SC_CALLBACK_REG`-Meldung sein, die weder vor noch nach der Meldung zusätzliche Bytes enthält.

Nach Schreiben aller Bytes an den Strom muss der Client die Verbindung aufrechterhalten, um die Antwort des Servers erhalten zu können. Wenn der Client die Meldung nicht korrekt formatiert, wird er vom Server nicht registriert, und dieser sendet eine Fehlerantwort an den Client. Wenn der Client die Socketverbindung beendet, bevor der Server eine Antwort gesendet hat, wird er dennoch vom Server ordnungsgemäß registriert.

Ein Client kann jederzeit Kontakt mit dem Server aufnehmen. Bei jeder Kontaktaufnahme mit dem Server muss der Client eine `SC_CALLBACK_REG`-Meldung senden. Wenn der Server eine Meldung erhält, die fehlerhaft, beschädigt oder ungültig ist, sendet er eine Fehlerantwort an den Client.

Ein Client kann keine `ADD_EVENTS`-, `REMOVE_EVENTS` - oder `REMOVE_CLIENT`-Meldung senden, bevor seine `ADD_CLIENT`-Meldung gesendet wurde. Er kann auch keine `REMOVE_CLIENT`-Meldung senden, bevor seine `ADD_CLIENT`-Meldung gesendet wurde.

Wenn ein Client eine `ADD_CLIENT`-Meldung sendet, obwohl er bereits registriert ist, kann der Server diese Meldung tolerieren. In diesem Fall ersetzt der Server die alte Client-Registrierung stillschweigend durch die neue Client-Registrierung, die in der zweiten `ADD_CLIENT`-Meldung angegeben ist.

In den meisten Fällen registriert sich ein Client einmal beim Server. Dies geschieht beim Starten des Clients mittels Senden einer `ADD_CLIENT`-Meldung. Ebenso deregistriert sich ein Client nur einmal, indem er eine `REMOVE_CLIENT`-Meldung an den Server sendet. Das CRNP bietet jedoch größere Flexibilität für diejenigen Clients, die ihre Ereignistypliste dynamisch ändern möchten.

Inhalt einer `SC_CALLBACK_REG`-Meldung

Jede `ADD_CLIENT`-, `ADD_EVENTS`- und `REMOVE_EVENTS`-Meldung enthält eine Ereignisliste. Die folgende Tabelle beschreibt die Ereignistypen, die das CRNP akzeptiert, einschließlich der erforderlichen Namens- und Wertepaare.

Wenn ein Client:

- eine `REMOVE_EVENTS`-Meldung sendet, die einen oder mehrere Ereignistypen angibt, für die der Client nicht zuvor registriert wurde, oder
- sich zweimal für den gleichen Ereignistyp registriert,

ignoriert der Server diese Meldungen stillschweigend.

Klasse und Unterklasse	Namens- und Wertepaare	Beschreibung
`EC_Cluster` `ESC_cluster_membership`	Erforderlich: Keine Optional: Keine	Wird für alle Änderungsereignisse bezüglich der Cluster-Mitgliedschaft registriert (Knotenversagen oder -beitritt)
`EC_Cluster` `ESC_cluster_rg_state`	Eines erforderlich, wie folgt: `rg_name` Werttyp: Zeichenkette Optional: Keine	Wird für alle Zustandsänderungsereignisse für Ressourcengruppe *Name* registriert
`EC_Cluster` `ESC_cluster_r_state`	Eines erforderlich, wie folgt: `r_name` Werttyp: Zeichenkette Optional: Keine	Wird für alle Zustandsänderungsereignisse für Ressource *Name* registriert
`EC_Cluster` Keine	Erforderlich: Keine Optional: Keine	Wird für alle Sun Cluster-Ereignisse registriert

Server-Antworten an den Client

Nach Verarbeiten der Registrierung sendet der Server die `SC_REPLY`-Meldung. Der Server sendet diese Meldung über die offene TCP-Verbindung des Clients, über die er die Registrierungsanforderung erhalten hatte. Daraufhin beendet der Server die Verbindung. Der Client muss die TCP-Verbindung so lange offen halten, bis er die `SC_REPLY`-Meldung vom Server erhalten hat.

Der Client führt zum Beispiel folgende Aktionen aus:

1. Herstellen einer TCP-Verbindung mit dem Server,
2. Warten, bis die Verbindung “schreibbereit” ist,
3. Senden einer `SC_CALLBACK_REG`-Meldung mit einer `ADD_CLIENT`-Meldung,
4. Warten auf eine `SC_REPLY`-Meldung,
5. Erhalten einer `SC_REPLY`-Meldung,
6. Erhalten einer Anzeige, dass der Server die Verbindung beendet hat (Lesen von 0 Bytes vom Socket),
7. Beenden der Verbindung.

Zu einem späteren Zeitpunkt führt der Client folgende Aktionen aus:

1. Herstellen einer TCP-Verbindung mit dem Server,
2. Warten, bis die Verbindung “schreibbereit” ist,
3. Senden einer SC_CALLBACK_REG -Meldung mit einer REMOVE_CLIENT-Meldung,
4. Warten auf eine SC_REPLY-Meldung,
5. Erhalten einer SC_REPLY-Meldung,
6. Erhalten einer Anzeige, dass der Server die Verbindung beendet hat (Lesen von 0 Bytes vom Socket),
7. Beenden der Verbindung.

Jedes Mal, wenn der Server eine SC_CALLBACK_REG-Meldung von einem Client erhält, sendet er eine SC_REPLY-Meldung über dieselbe offene Verbindung. Diese Meldung gibt an, ob der Vorgang erfolgreich war oder fehlgeschlagen ist. „SC_REPLY-XML-DTD ” auf Seite 344 enthält die XML-Dokumenttypdefinition einer SC_REPLY-Meldung, sowie die möglichen Fehlermeldungen für diese Meldung.

Inhalt einer SC_REPLY-Meldung

Eine SC_REPLY-Meldung gibt an, ob ein Vorgang erfolgreich war oder fehlgeschlagen ist. Diese Meldung enthält die Version der CRNP-Protokollmeldung, einen Statuscode und eine Statusmeldung, die den Statuscode detaillierter beschreibt. In der folgenden Tabelle werden die möglichen Werte für den Statuscode aufgelistet.

Statuscode	Beschreibung
OK	Die Meldung wurde erfolgreich verarbeitet.
RETRY	Die Client-Registrierung wurde vom Server aufgrund eines temporären Fehlers zurückgewiesen. Der Client sollte einen weiteren Registrierungsversuch mit anderen Parametern unternehmen.
LOW_RESOURCE	Die Cluster-Ressourcen sind ausgelastet, und der Client muss für einen erneuten Registrierungsversuch einen späteren Zeitpunkt abwarten. Eine andere Möglichkeit wäre, dass der Systemverwalter die entsprechenden Cluster-Ressourcen erhöht.
SYSTEM_ERROR	Ein schwerwiegendes Problem ist aufgetreten. Nehmen Sie Kontakt mit dem Systemverwalter für den Cluster auf.
FAIL	Die Autorisierung ist fehlgeschlagen, oder ein sonstiges Problem hat das Scheitern der Registrierung verursacht.
MALFORMED	Die XML-Anforderung war fehlerhaft und konnte nicht analysiert werden.

Statuscode	Beschreibung
INVALID	Die XML-Anforderung war ungültig (entspricht nicht der XML-Spezifikation).
VERSION_TOO_HIGH	Die Meldungsversion war für eine erfolgreiche Verarbeitung zu hoch.
VERSION_TOO_LOW	Die Meldungsversion war für eine erfolgreiche Meldungsverarbeitung zu niedrig.

Umgang des Clients mit Fehlerbedingungen

Unter normalen Bedingungen erhält ein Client, der eine SC_CALLBACK_REG-Meldung sendet, eine Antwort, die eine erfolgreiche oder fehlgeschlagene Registrierung angibt.

Bei der Client-Registrierung kann jedoch auf der Serverseite eine Fehlerbedingung auftreten, die den Server davon abhält, eine SC_REPLY-Meldung an den Client zu senden. In diesem Fall kann die Registrierung entweder vor Auftreten der Fehlerbedingung erfolgreich verlaufen sein, oder sie konnte noch nicht verarbeitet werden.

Da der Server auf dem Cluster als Failover- oder hoch verfügbarer Server auf dem Cluster eingesetzt werden muss, bedeutet diese Fehlerbedingung nicht, dass der Dienst beendet wird. Es kann sogar sein, dass der Server bald beginnt, Ereignisse an den neu registrierten Client zu senden.

Um diese Fehlerbedingungen zu beheben, muss der Client folgendermaßen verfahren:

- Er muss auf Anwendungsebene eine Zeitüberschreitung für die Registrierungsverbindung einsetzen, die auf eine SC_REPLY-Meldung wartet. Nach Ablauf der Zeitüberschreitung muss der Client einen erneuten Registrierungsversuch unternehmen.
- Er muss beginnen, an der Rückmelde-IP-Adresse und Port-Nummer Ereigniszustellungen abzuhören, bevor er sich für die Ereignisrückmeldungen registriert. Der Client muss parallel eine Registrierungs-Bestätigungsmeldung und Ereigniszustellungen abwarten. Wenn der Client Ereignisse erhält, bevor die Bestätigungsmeldung bei ihm eingegangen ist, sollte er die Registrierungsverbindung stillschweigend beenden.

Verfahren für Ereigniszustellungen vom Server an den Client

Sobald im Cluster Ereignisse generiert werden, stellt der CRNP-Server diese allen Clients zu, die Ereignisse des entsprechenden Typs angefordert haben. Die Zustellung besteht im Senden einer `SC_EVENT`-Meldung an die Rückmeldeadresse des Clients. Jedes Ereignis wird über eine neue TCP-Verbindung zugestellt.

Unmittelbar nach der Client-Registrierung für einen Ereignistyp sendet der Server über eine `SC_CALLBACK_REG`-Meldung mit einer `ADD_CLIENT`-Meldung bzw. einer `ADD_EVENT`-Meldung dem Client das neueste Ereignis des entsprechenden Typs. Der Client kann so den aktuellen Zustand des Systems feststellen, von dem die nachfolgenden Ereignisse kommen.

Wenn der Server eine TCP-Verbindung mit dem Client herstellt, sendet er genau eine `SC_EVENT`-Meldung über die Verbindung. Daraufhin gibt der Server eine Vollduplex-Beendigung aus.

Der Client führt zum Beispiel folgende Aktionen aus:

1. Abwarten einer vom Server hergestellten TCP-Verbindung,
2. Akzeptieren der eingehenden Verbindung vom Server,
3. Abwarten einer `SC_EVENT`-Meldung,
4. Lesen der `SC_EVENT`-Meldung,
5. Erhalten einer Anzeige, dass der Server die Verbindung beendet hat (Lesen von 0 Bytes vom Socket),
6. Beenden der Verbindung.

Wenn sich alle Clients registriert haben, müssen sie jederzeit ihre Rückmeldeadressen (IP-Adresse und Port-Nummer) abhören, um eine eingehende Verbindung für die Ereigniszustellung abzuwarten.

Wenn der Server keinen Kontakt mit dem Client aufnehmen kann, um ein Ereignis zuzustellen, versucht der Server eine vom Benutzer definierte Anzahl von Malen innerhalb eines definierten Zeitintervalls erneut, das Ereignis zuzustellen. Wenn alle Versuche fehlschlagen, wird der Client aus der Client-Liste des Servers entfernt. Der Client muss sich dann erneut registrieren, indem er eine weitere `SC_CALLBACK_REG`-Meldung mit einer `ADD_CLIENT`-Meldung sendet. Erst dann kann er weitere Ereignisse erhalten.

Garantie der Ereigniszustellung

Innerhalb des Clusters besteht eine Gesamtreihenfolge für die Ereignisgenerierung, die in der Reihenfolge der Zustellung an jeden Client eingehalten wird. Wenn also Ereignis A im Cluster vor Ereignis B generiert wird, erhält der Client X das Ereignis A vor dem Ereignis B. Diese Gesamtreihenfolge der Ereigniszustellung an *alle* Clients wird jedoch *nicht* eingehalten. Das heißt, dass Client Y beide Ereignisse A und B erhalten kann, bevor Client X das Ereignis A erhält. Dadurch wird sichergestellt, dass langsame Clients nicht die Zustellung an alle Clients aufhalten.

Alle Ereignisse, die der Server zustellt (mit Ausnahme des ersten Ereignisses für eine Unterklasse und von Ereignissen, die auf Serverfehler folgen), sind eine Reaktion auf die tatsächlichen Ereignisse, die der Cluster generiert, es sei denn, beim Server tritt ein Fehler auf, durch den er im Cluster generierte Ereignisse nicht erfasst. In diesem Fall generiert der Server ein Ereignis für jeden Ereignistyp, das den aktuellen Zustand des Systems für diesen Typ darstellt. Jedes Ereignis wird an Clients gesendet, die Interesse an diesem Ereignistyp registriert haben.

Die Ereigniszustellung folgt der "Mindestens einmal"-Semantik. Das heißt, dass der Server das Ereignis mehr als einmal an einen Client senden kann. Dies muss in Fällen zugelassen sein, in denen der Server vorübergehend heruntergefahren wird und nach erneutem Herauffahren nicht feststellen kann, ob der Client die neuesten Informationen erhalten hat.

Inhalt einer `SC_EVENT`-Meldung

Die `SC_EVENT`-Meldung enthält die eigentliche Meldung, die im Cluster generiert wird, übertragen in das `SC_EVENT`-XML-Meldungsformat. Die folgende Tabelle beschreibt die vom CRNP zugestellten Ereignistypen, einschließlich der Namens- und Wertepaare, Herausgeber und Hersteller.

Klasse und Unterklasse	Herausgeber und Hersteller	Namens- und Wertepaare	Anmerkungen
EC_Cluster ESC_cluster_membership	Herausgeber: rgm Hersteller: SUNW	Name: node_list Werttyp: Zeichenketten-Array Name: state_list Werttyp: Zeichenketten-Array	Die Positionen der Array-Elemente für state_list sind mit denjenigen der node_list synchronisiert. Das heißt, dass der Zustand für die im node_list-Array zuerst aufgelistete Knotenliste der erste Zustand im state_list-Array ist. Die state_list enthält nur in ASCII dargestellte Ziffern. Jede Ziffer stellt die aktuelle Zusammensetzungsnummer für diesen Knoten im Cluster dar. Wenn die Nummer die gleiche wie die in einer vorhergehenden Meldung erhaltene Nummer ist, hat sich die Beziehung des Knotens zum Cluster nicht geändert (gelöscht, beigetreten bzw. erneut beigetreten). Wenn die Zusammensetzungsnummer –1 ist, so ist der Knoten kein Cluster-Mitglied. Wenn die Zusammensetzungsnummer keine negative Zahl ist, handelt es sich beim Knoten um ein Cluster-Mitglied. Zusätzliche Namen, die mit ev_ beginnen, und deren zugeordnete Werte können vorhanden sein, sind aber nicht für die Verwendung durch den Client vorgesehen.

Klasse und Unterklasse	Herausgeber und Hersteller	Namens- und Wertepaare	Anmerkungen
EC_Cluster ESC_cluster_rg_state	Herausgeber: rgm Hersteller: SUNW	Name: rg_name Werttyp: Zeichenkette Name: node_list Werttyp: Zeichenketten-Array Name: state_list Werttyp: Zeichenketten-Array	Die Positionen der Array-Elemente für state_list sind mit denjenigen der node_list synchronisiert. Das heißt, dass der Zustand für die im node_list-Array zuerst aufgelistete Knotenliste der erste Zustand im state_list-Array ist. Die state_list enthält Zeichenkettendarstellungen des Zustands der Ressourcengruppe. Gültige Werte sind Werte, die mit den Befehlen scha_cmds(1HA) abgerufen werden können. Zusätzliche Namen, die mit ev_ beginnen, und deren zugeordnete Werte können vorhanden sein, sind aber nicht für die Verwendung durch den Client vorgesehen.

Klasse und Unterklasse	Herausgeber und Hersteller	Namens- und Wertepaare	Anmerkungen
`EC_Cluster` `ESC_cluster_r_state`	Herausgeber: rgm Hersteller: SUNW	Drei erforderlich, wie folgt: Name: `r_name` Werttyp: Zeichenkette Name: `node_list` Werttyp: Zeichenketten-Array Name: `state_list` Werttyp: Zeichenketten-Array	Die Positionen der Array-Elemente für `state_list` sind mit denjenigen der `node_list` synchronisiert. Das heißt, dass der Zustand für den im `node_list`-Array zuerst aufgelisteten Knoten der erste Zustand im `state_list`-Array ist. Die `state_list` enthält Zeichenkettendarstellungen des Zustands der Ressource. Gültige Werte sind Werte, die mit den Befehlen `scha_cmds`(1HA) abgerufen werden können. Zusätzliche Namen, die mit `ev_` beginnen, und deren zugeordnete Werte können vorhanden sein, sind aber nicht für die Verwendung durch den Client vorgesehen.

Authentisierung von Clients und Server durch das CRNP

Der Server authentifiziert einen Client mit einer Form von TCP-Wrappern. Die Quell-IP-Adresse der Registrierungsmeldung, die auch als Rückmelde-IP-Adresse dient, an die Ereignisse zugestellt werden, muss sich in der Liste der zulässigen Clients für den Server befinden. Die Quell-IP-Adresse und Registrierungsmeldung darf sich nicht in der Liste der abgewiesenen Clients befinden. Wenn sich die Quell-IP-Adresse und Registrierung nicht in der Liste befinden, weist der Server die Anforderung zurück und gibt eine Fehlerantwort an den Client aus.

Wenn der Server eine `SC_CALLBACK_REG ADD_CLIENT`-Meldung erhält, muss die Quell-IP-Adresse der nachfolgenden `SC_CALLBACK_REG`-Meldungen für diesen Client derjenigen der ersten Meldung entsprechen. Wenn der CRNP-Server eine `SC_CALLBACK_REG` erhält, die dieser Anforderung nicht entspricht, hat der Server zwei Möglichkeiten:

- Er ignoriert die Anforderung und sendet eine Fehlerantwort an den Client; oder
- Er nimmt an, dass die Anforderung von einem neuen Client stammt (abhängig vom Inhalt der `SC_CALLBACK_REG`-Meldung).

Dieser Sicherheitsmechanismus trägt dazu bei, Dienstverweigerungsangriffe abzuwehren, bei denen versucht wird, einen berechtigten Client zu deregistrieren.

Clients sollten den Server auf ähnliche Weise authentisieren. Die Clients brauchen nur die Ereigniszustellungen von einem Server zu akzeptieren, dessen Quell-IP-Adresse und Port-Nummer der vom Client für die Registrierung verwendeten IP-Adresse und Port-Nummer entsprechen.

Da davon ausgegangen wird, dass sich die Clients des CRNP-Dienstes hinter einem Firewall befinden, der den Cluster schützt, stellt das CRNP keine weiteren Sicherheitsmechanismen bereit.

Erstellen einer Java-Anwendung, die CRNP verwendet

Das folgende Beispiel zeigt, wie eine einfache Java-Anwendung mit dem Namen `CrnpClient`, die das CRNP verwendet, entwickelt werden kann. Die Anwendung wird für Ereignisrückmeldungen bei dem CRNP-Server auf dem Cluster registriert, hört Ereignisrückmeldungen ab und verarbeitet die Ereignisse, indem sie deren Inhalt druckt. Vor der Beendigung deregistriert die Anwendung ihre Anforderung von Ereignisrückmeldungen.

Beachten Sie folgende Punkte beim Untersuchen des vorliegenden Beispiels.

- Die Beispielanwendung führt XML-Generierung und -Analyse mit JAXP (Java API for XML Processing) aus. Dieses Beispiel zeigt nicht, wie JAXP verwendet wird. JAXP wird unter `http://java.sun.com/xml/jaxp/index.html` genauer beschrieben.
- Dieses Beispiel stellt Teile einer vollständigen Anwendung vor. Die ganze Anwendung finden Sie in Anhang G. Um bestimmte Konzepte deutlicher darzustellen, weicht das in diesem Kapitel beschriebene Beispiel leicht von der in Anhang G vorgestellten vollständigen Anwendung ab.
- Der Kürze halber werden die Kommentare im Beispielcode in diesem Kapitel ausgelassen. Die vollständige Anwendung in Anhang G enthält Kommentare.

- Die in diesem Beispiel gezeigte Anwendung reagiert auf die meisten Fehlerbedingungen, indem die Anwendung einfach beendet wird. Eine reale Anwendung muss Fehlern gegenüber robuster reagieren können.

▼ Konfigurieren der Umgebung

Zunächst muss die Umgebung konfiguriert werden.

1. **Laden Sie JAXP und die richtige Version des Java-Compilers und der Virtual Machine herunter, und installieren Sie sie.**

   Anweisungen hierzu finden Sie unter `http://java.sun.com/xml/jaxp/index.html`.

   Hinweis – Für dieses Beispiel ist Java 1.3.1 bzw. eine höhere Java-Version erforderlich.

   ---

2. **Vergewissern Sie sich, dass Sie einen `classpath` in der Kompilierungsbefehlszeile angegeben haben, damit der Compiler die JAXP-Klassen finden kann. Geben Sie vom Verzeichnis der Quelldatei aus Folgendes ein:**

   ```
   % 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 . QUELL_DATEINAME.java
   ```

   wobei *JAXP_ROOT* der absolute bzw. relative Pfad zu dem Verzeichnis ist, in dem sich die JAXP-JAR-Dateien befinden und *QUELL_DATEINAME* der Name der Java-Quelldatei.

3. **Beim Ausführen der Anwendung geben Sie den `classpath` an, damit die Anwendung die richtigen JAXP-Klassendateien laden kann. Beachten Sie, dass der erste Pfad in `classpath` das aktuelle Verzeichnis ist:**

   ```
   java -cp .:JAXP_ROOT/dom.jar:JAXP_ROOTjaxp-api. \
   jar:JAXP_ROOTsax.jar:JAXP_ROOTxalan.jar:JAXP_ROOT/xercesImpl \
   .jar:JAXP_ROOT/xsltc.jar QUELL_DATEINAME ARGUMENTE
   ```

   Damit ist die Umgebung konfiguriert, und Sie können die Anwendung entwickeln.

▼ Erste Schritte

In diesem Teil des Beispiels erstellen Sie eine Basisklasse mit dem Namen `CrnpClient`, deren Hauptmethode die Befehlszeilenargumente analysiert und ein `CrnpClient`-Objekt erstellt. Dieses Objekt übergibt die Befehlszeilenargumente an die Klasse, wartet, bis der Benutzer die Anwendung beendet, ruft `shutdown` für `CrnpClient` auf und wird dann beendet.

Der Konstruktor der `CrnpClient`-Klasse muss folgende Aufgaben ausführen:

- Einrichten der XML-Verarbeitungsobjekte.
- Erstellen eines Threads, der Ereignisrückmeldungen abhört.
- Kontaktaufnahme mit dem CRNP-Server und Registrierung für Ereignisrückmeldungen.

- **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

  Das folgende Beispiel zeigt den Hauptcode für die `CrnpClient`-Klasse. Die Implementierungen der vier Hilfsmethoden, die im Konstruktor referenziert werden, sowie die Methoden zum Herunterfahren werden später erläutert. Beachten Sie, dass der Code gezeigt wird, mit dem alle benötigten Pakete importiert werden.

```
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("Drücken Sie die Eingabetaste, um die Demo zu beenden...");
                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;
}
```

Mitgliedsvariablen werden weiter unten detaillierter behandelt.

▼ Analyse der Befehlszeilenargumente

- **Anhand des Codes in Anhang G können Sie sehen, wie die Befehlszeilenargumente analysiert werden.**

▼ Definieren des Ereignisempfangs-Threads

Im Code müssen Sie sicherstellen, dass der Ereignisempfang über einen eigenen Thread ausgeführt wird, so dass die Anwendung anderweitig weiterarbeiten kann, wenn der Thread blockiert ist und auf Ereignisrückmeldungen wartet.

Hinweis – Das Einrichten des XML wird weiter unten erläutert.

1. **Definieren Sie im Code eine `Thread`-Unterklasse mit dem Namen `EventReceptionThread`, die ein `ServerSocket` erstellt und die an dieses Socket ankommenden Ereignisse abwartet.**

   In diesem Teil des Beispielcodes werden Ereignisse weder gelesen noch verarbeitet. Das Lesen und Verarbeiten von Ereignissen wird später behandelt. `EventReceptionThread` erstellt ein `ServerSocket` über eine Internetworking-Protokolladresse mit Platzhalter. `EventReceptionThread` ist auch mit dem `CrnpClient`-Objekt referenziert, so dass `EventReceptionThread` Ereignisse zur Verarbeitung an das `CrnpClient`-Objekt senden kann.

```
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();
                                // Ereignis aus sock-Strom erstellen und verarbeiten
                                sock.close();
                        }
                        // UNERREICHBAR

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* private Mitgliedsvariablen */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. **Nachdem Sie nun gesehen haben, wie die `EventReceptionThread`-Klasse funktioniert, erstellen Sie ein `createEvtRecepThr`-Objekt:**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
```

```
        evtThr.start();
}
```

▼ Registrieren und Deregistrieren von Rückmeldungen

Die Registrierung besteht aus folgenden Schritten:

- Öffnen eines Basis-TCP-Sockets für das Internetworking-Protokoll und den Port für die Registrierung.
- Erstellen der XML-Registrierungsmeldung.
- Senden der XML-Registrierungsmeldung an das Socket.
- Lesen der XML-Antwortmeldung vom Socket.
- Schließen des Sockets.

1. **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

   Das folgende Beispiel zeigt die Implementierung der `registerCallbacks`-Methode der `CrnpClient`-Klasse, die vom `CrnpClient`-Konstruktor aufgerufen wird. Die Aufrufe an `createRegistrationString()` und `readRegistrationReply()` werden später eingehender beschrieben.

   `regIp` und `regPort` sind Objektmitglieder, die vom Konstruktor konfiguriert werden.

```
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. **Implementieren Sie die `unregister`-Methode. Diese Methode wird von der `shutdown`-Methode von `CrnpClient` aufgerufen. Die Implementierung von `createUnregistrationString` wird später eingehender beschrieben.**

```
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();
}
```

▼ Generieren des XML

Nach dem Konfigurieren der Anwendungsstruktur und Schreiben des gesamten Netzwerkcodes können Sie nun den Code schreiben, der XML generiert und analysiert. Schreiben Sie zunächst den Code, der die `SC_CALLBACK_REG` -XML-Registrierungsmeldung generiert.

Eine `SC_CALLBACK_REG`-Meldung besteht aus einem Registrierungstyp (`ADD_CLIENT`, `REMOVE_CLIENT`, `ADD_EVENTS` oder `REMOVE_EVENTS`), einem Rückmelde-Port und einer Liste der interessierenden Ereignisse. Jedes Ereignis besteht aus einer Klasse und einer Unterklasse, gefolgt von einer Liste der Namens- und Wertepaare.

In diesem Teil des Beispiels schreiben Sie eine `CallbackReg`-Klasse, die den Registrierungstyp, den Rückmelde-Port und die Liste der Registrierungsereignisse speichert. Diese Klasse kann sich auch an eine `SC_CALLBACK_REG`-XML-Meldung serialisieren.

Eine interessante Methode dieser Klasse ist die `convertToXml`-Methode, die eine `SC_CALLBACK_REG`-XML-Meldungszeichenkette aus den Klassenmitgliedern erstellt. Eine detailliertere Beschreibung des Codes dieser Methode finden Sie in der JAXP-Dokumentation unter `http://java.sun.com/xml/jaxp/index.html`.

Im Folgenden wird die Implementierung der `Event`-Klasse gezeigt. Beachten Sie, dass die `CallbackReg`-Klasse eine `Event`-Klasse verwendet, die ein Ereignis speichert und dieses Ereignis in ein XML-`Element` konvertieren kann.

1. Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.

```
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("Fehler, ungültiger 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 mit angegebenen Optionen kann nicht erstellt werden
                pce.printStackTrace();
                System.exit(1);
        }

        // Root-Element erstellen
        Element root = (Element) document.createElement(
            "SC_CALLBACK_REG");

        // Attribute hinzufügen
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Ereignisse hinzufügen
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(
                    document));
        }
        document.appendChild(root);
```

```
                // Das Ganze in eine Zeichenkette konvertieren
                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. **Implementieren Sie die `Event`- und `NVPair`-Klassen.**

   Beachten Sie, dass die `CallbackReg`-Klasse eine `Event`-Klasse verwendet, die wiederum eine `NVPair`-Klasse verwendet.

```
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;
}
```

▼ Erstellen der Registrierungs- und Deregistrierungsmeldungen

Nach dem Erstellen der Helper-Klassen für die Generierung von XML-Meldungen können Sie die Implementierung der `createRegistrationString`-Methode schreiben. Diese Methode wird von der `registerCallbacks`-Methode aufgerufen (siehe Beschreibung unter „Registrieren und Deregistrieren von Rückmeldungen" auf Seite 239).

`createRegistrationString` erstellt ein `CallbackReg`-Objekt und richtet dessen Registrierungstyp und Port ein. Danach erstellt `createRegistrationString` verschiedene Ereignisse unter Verwendung der Helper-Methoden `createAllEvent`, `createMembershipEvent`, `createRgEvent` und `createREvent`. Jedes Ereignis wird diesem Objekt nach Erstellung des `CallbackReg`-Objekts hinzugefügt. Zuletzt ruft `createRegistrationString` die `convertToXml`-Methode des `CallbackReg`-Objekts auf, um die XML-Meldung im `String`-Format abzurufen.

Beachten Sie, dass die `regs`-Mitgliedsvariable die Befehlszeilenargumente speichert, die der Benutzer der Anwendung angibt. Das fünfte und alle folgenden Argumente geben die Ereignisse an, für die eine Anwendung registriert werden soll. Das vierte Argument gibt den Registrierungstyp an; es wird in diesem Beispiel jedoch übergangen. Der vollständige Code in Anhang G zeigt die Verwendung dieses vierten Arguments.

1. **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

```
private String createRegistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);

        cbReg.setRegType(CallbackReg.ADD_CLIENT);

        // Ereignisse hinzufügen
        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. **Erstellen Sie die Deregistrierungs-Zeichenkette.**

   Das Erstellen einer Deregistrierungs-Zeichenkette ist einfacher als das Erstellen der Registrierungszeichenkette, da keine Ereignisse berücksichtigt werden müssen:

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
```

```
        return (xmlStr);
}
```

▼ Konfigurieren des XML-Parsers

Bisher wurden der Netzwerk- und der XML-Generierungscode für die Anwendung erstellt. Der letzte Schritt besteht in der Analyse und Verarbeitung der Registrierungsantwort und der Ereignisrückmeldungen. Der `CrnpClient`-Konstruktor ruft eine `setupXmlProcessing`-Methode auf. Diese Methode erstellt ein `DocumentBuilderFactory`-Objekt und stellt verschiedene Analyseeigenschaften für dieses Objekt ein. Eine detailliertere Beschreibung dieser Methode finden Sie in der JAXP-Dokumentation unter `http://java.sun.com/xml/jaxp/index.html`.

- **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        // Eine Validierung ist nicht erforderlich.
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // Kommentare und Leerzeichen sollen ignoriert werden
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // CDATA-Abschnitte mit TEXT-Knoten verbinden.
        dbf.setCoalescing(true);
}
```

▼ Analysieren der Registrierungsantwort

Für die Analyse der `SC_REPLY`-XML-Meldung, die der CRNP-Server als Antwort auf eine Registrierungs- bzw. Deregistrierungsmeldung sendet, benötigen Sie eine `RegReply`-Helper-Klasse. Diese Klasse kann aufbauend auf einem XML-Dokument erstellt werden. Die Klasse ermöglicht den Zugang zum Statuscode und zur Statusmeldung. Um den XML-Strom vom Server zu analysieren, müssen Sie ein neues XML-Dokument erstellen und die Analysemethode dieses Dokuments verwenden. Eine detailliertere Beschreibung dieser Methode finden Sie in der JAXP-Dokumentation unter `http://java.sun.com/xml/jaxp/index.html`.

1. **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

   Beachten Sie, dass die `readRegistrationReply`-Methode die neue `RegReply`-Klasse verwendet.

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Dokument-Builder erstellen
```

```
                DocumentBuilder db = dbf.newDocumentBuilder();
                db.setErrorHandler(new DefaultHandler());

                // Eingabedatei analysieren
                Document doc = db.parse(stream);

                RegReply reply = new RegReply(doc);
                reply.print(System.out);
        }
```

2. **Implementieren Sie die `RegReply`-Klasse.**

   Beachten Sie, dass die `retrieveValues`-Methode der DOM-Struktur im XML-Dokument folgt und den Statuscode und die Statusmeldung abruft. Weitere Einzelheiten finden Sie in der JAXP-Dokumentation unter `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;

                // SC_REPLY-Element suchen.
                nl = doc.getElementsByTagName("SC_REPLY");
                if (nl.getLength() != 1) {
                        System.out.println("Analysefehler:"
                            + "SC_REPLY-Knoten kann nicht gefunden werden.");
                        return;
                }

                n = nl.item(0);
```

```
                // Wert des statusCode-Attributs abrufen
                statusCode = ((Element)n).getAttribute("STATUS_CODE");

                // SC_STATUS_MSG-Element suchen
                nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
                if (nl.getLength() != 1) {
                        System.out.println("Analysefehler:  "
                            + "SC_STATUS_MSG-Knoten kann nicht gefunden werden.");
                        return;
                }
                // TEXT-Abschnitt abrufen, falls vorhanden.
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                // Kein Fehler, falls nicht vorhanden; einfach stillschweigend zurückgehen.
                        return;
                }

                // Wert abrufen
                statusMsg = n.getNodeValue();
        }

        private String statusCode;
        private String statusMsg;
}
```

▼ Analysieren der Rückmeldeereignisse

Der letzte Schritt besteht in der Analyse und Verarbeitung der Rückmeldeereignisse selbst. Um diese Aufgabe zu unterstützen, ändern Sie die `Event`-Klasse, die in „Generieren des XML" auf Seite 240 erstellt wurde, damit diese Klasse in der Lage ist, ein `Event` basierend auf einem XML-Dokument zu erstellen und ein XML-`Element` zu erstellen. Diese Änderung erfordert einen zusätzlichen Konstruktor (XML-Dokument), eine `retrieveValues`-Methode, das Hinzufügen zweier weiterer Mitgliedsvariablen (`vendor` und `publisher`), Zugangsmethoden für alle Felder und schließlich eine Druckmethode.

1. **Erstellen Sie den Java-Code, der die vorstehende Logik implementiert.**

   Beachten Sie, dass dieser Code dem Code für die `RegReply`-Klasse ähnelt, die unter „Analysieren der Registrierungsantwort" auf Seite 246 beschrieben wurde.

```
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;

        // SC_EVENT-Element suchen.
        nl = doc.getElementsByTagName("SC_EVENT");
        if (nl.getLength() != 1) {
           System.out.println("Analysefehler: "
               + "SC_EVENT-Knoten kann nicht gefunden werden.");
           return;
        }

        n = nl.item(0);

        //
        // Werte der Attribute CLASS, SUBCLASS,
        // VENDOR und PUBLISHER abrufen.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Alle NW-Paare abrufen
        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. **Implementieren Sie die zusätzlichen Konstruktoren und Methoden für die `NVPair`-Klasse, welche die XML-Analyse unterstützen.**

   Die Änderungen an der `Event`-Klasse, die in Schritt 1 gezeigt werden, machen vergleichbare Änderungen an der `NVPair`-Klasse erforderlich.

```
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;

        // NAME-Element suchen
        nl = elem.getElementsByTagName("NAME");
        if (nl.getLength() != 1) {
           System.out.println("Analysefehler: "
               + "NAME-Knoten kann nicht gefunden werden.");
           return;
        }
        // TEXT-Abschnitt abrufen
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
           System.out.println("Analysefehler: "
               + "TEXT-Abschnitt konnte nicht gefunden werden.");
           return;
        }

        // Wert abrufen
        name = n.getNodeValue();

        // Jetzt das Wertelement abrufen
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
           System.out.println("Analysefehler: "
               + "VALUE-Knoten konnte nicht gefunden werden.");
```

```
                    return;
                }
                // TEXT-Abschnitt abrufen
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                System.out.println("Analysefehler "
                            + "TEXT-Abschnitt konnte nicht gefunden werden.");
                        return;
                }

                // Wert abrufen
                value = n.getNodeValue();
                }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3. **Implementieren Sie die `while`-Schleife in `EventReceptionThread`, die Rückmeldeereignisse abwartet (`EventReceptionThread` wird unter „Definieren des Ereignisempfangs-Threads" auf Seite 237 beschrieben).**

```
while(true) {
                Socket sock = listeningSock.accept();
                Document doc = db.parse(sock.getInputStream());
                Event event = new Event(doc);
                client.processEvent(event);
                sock.close();
        }
```

▼ Ausführen der Anwendung

- **Führen Sie die Anwendung aus.**

  `# java CrnpClient` *crnpHost crnpPort localPort* `...`

  Der vollständige Code für die `CrnpClient`-Anwendung ist in Anhang G enthalten.

ANHANG **A**

Standardeigenschaften

Dieser Anhang beschreibt die standardmäßigen Ressourcentypen, Ressourcengruppen und Ressourceneigenschaften. Daneben beschreibt er die Ressourceneigenschaftsattribute, die für das Ändern systemdefinierter Eigenschaften und das Erstellen von Erweiterungseigenschaften zur Verfügung stehen.

Dieser Anhang enthält folgende Abschnitte.

Ressourcentypeigenschaften

Die folgenden Informationen beschreiben die Ressourcentypeigenschaften, die von Sun Cluster definiert werden. Die Eigenschaftswerte werden in folgende Kategorien unterteilt:

- **Erforderlich** – Die Eigenschaft erfordert einen expliziten Wert in der RTR-Datei (Ressourcentyp-Registrierung). Andernfalls kann das Objekt, zu dem die Eigenschaft gehört, nicht erstellt werden. Ein Leerzeichen bzw. eine leere Zeichenkette sind als Wert nicht zulässig.
- **Bedingt** – Um vorhanden sein zu können, muss die Eigenschaft in der RTR-Datei deklariert werden. Andernfalls erstellt RGM die Eigenschaft nicht und die Eigenschaft steht den Verwaltungsdienstprogrammen nicht zur Verfügung. Ein Leerzeichen bzw. eine leere Zeichenkette sind zulässig. Wenn die Eigenschaft in der RTR-Datei deklariert, jedoch kein Wert angegeben ist, stellt RGM einen Standardwert bereit.

- **Bedingt/Explizit** – Um vorhanden sein zu können, muss die Eigenschaft in der RTR-Datei mit einem expliziten Wert deklariert werden. Andernfalls erstellt RGM die Eigenschaft nicht und die Eigenschaft steht den Verwaltungsdienstprogrammen nicht zur Verfügung. Ein Leerzeichen bzw. eine leere Zeichenkette sind nicht zulässig.
- **Optional** — Die Eigenschaft kann in der RTR-Datei deklariert werden. Wenn die Eigenschaft nicht in der RTR-Datei deklariert ist, erstellt RGM die Eigenschaft und stellt einen Standardwert bereit. Wenn die Eigenschaft in der RTR-Datei deklariert, jedoch kein Wert angegeben ist, stellt RGM denselben Standardwert bereit, als wäre die Eigenschaft nicht in der RTR-Datei deklariert.

Mit Ausnahme der Eigenschaften `Installed_nodes` und `RT_system` können Verwaltungsdienstprogramme keine Ressourcentypeigenschaften aktualisieren, die nicht in der RTR-Datei deklariert werden können, sondern vom Verwalter eingestellt werden müssen.

Die Eigenschaftsnamen werden zuerst angezeigt, gefolgt von einer Beschreibung.

`API_version` (`Ganzzahl`)
Die Version der Ressourcenverwaltungs-API, die von dieser Ressourcentypimplementierung verwendet wird.

Die nachfolgenden Informationen geben die maximale `API_version` an, die von den verschiedenen Versionen von Sun Cluster unterstützt wird.

Bis Version 3.1	2
Version 3.1 10/03	3
Version 3.1 4/04	4
3.1 9/04	5

Wenn Sie in der RTR-Datei für `API_version` einen Wert größer 2 deklarieren, wird verhindert, dass der Ressourcentyp auf einer Version von Sun Cluster installiert wird, der eine niedrigere maximale Version unterstützt. Wenn Sie für einen Ressourcentyp beispielsweise `API_version=5` deklarieren, kann dieser Ressourcentyp nicht auf einer Version von Sun Cluster installiert werden, die vor 3.1 9/04 herausgegeben wurde.

Kategorie: Optional

Standard: 2

Einstellbar: Nein

`Boot` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM auf einem Knoten aufruft, der dem Cluster beitritt bzw. erneut beitritt, wenn eine Ressource dieses Typs bereits verwaltet wird. Diese Methode hat die Aufgabe, Initialisierungsaktionen für Ressourcen dieses Typs, vergleichbar mit denen der `Init`-Methode, auszuführen.

Kategorie: Bedingt/Explizit

Standard: Keine

Einstellbar: Nein

`Failover` (`Boolescher Wert`)
`TRUE` gibt an, dass Ressourcen dieses Typs nicht in einer Gruppe konfiguriert werden können, die auf mehreren Knoten gleichzeitig online sein kann.

Kategorie: Optional

Standard: `FALSE`

Einstellbar: Nein

`Fini` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, wenn eine Ressource dieses Typs aus der RGM-Verwaltung entfernt wird.

Kategorie: Bedingt/Explizit

Standard: Keine

Einstellbar: Nein

`Init` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, wenn eine Ressource dieses Typs unter die Verwaltung durch RGM gestellt wird.

Kategorie: Bedingt/Explizit

Standard: Keine

Einstellbar: Nein

`Init_nodes` (`Aufzählung`)
Die Werte können `RG_primaries` (nur diejenigen Knoten, die Master der Ressourcen sein können) oder `RT_installed_nodes` (alle Knoten, auf denen der Ressourcentyp installiert ist) sein. Gibt die Knoten an, auf denen RGM die Methoden `Init`, `Fini`, `Boot` und `Validate` aufruft.

Kategorie: Optional

Standard: `RG_primaries`

Einstellbar: Nein

`Installed_nodes` (`Zeichenketten-Array`)
Eine Liste der Cluster-Knotennamen, auf denen der Ressourcentyp ausgeführt werden kann. RGM erstellt diese Eigenschaft automatisch. Der Cluster-Verwalter kann den Wert einstellen. Diese Eigenschaft kann nicht in der RTR-Datei deklariert werden.

Kategorie: Kann vom Cluster-Verwalter konfiguriert werden.

Standard: Alle Cluster-Knoten

Einstellbar: Ja

`Is_logical_hostname` (`Boolescher Wert`)
`TRUE` gibt an, dass dieser Ressourcentyp eine Version des Ressourcentyps `LogicalHostname` ist, der Failover-IP-Adressen verwaltet.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nein

`Is_shared_address` (`Boolescher Wert`)
`TRUE` gibt an, dass dieser Ressourcentyp eine Version des Ressourcentyps `SharedAddress` ist, der Failover-IP-Adressen verwaltet.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nein

`Monitor_check` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, bevor ein vom Monitor angefordertes Failover einer Ressource dieses Typs ausgeführt wird.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Monitor_start` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, um einen Fehler-Monitor für eine Ressource dieses Typs zu starten.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Monitor_stop` (`Zeichenkette`)
Eine Rückmeldemethode, die erforderlich ist, wenn `Monitor_start` eingestellt ist: der Pfad zu dem Programm, das RGM aufruft, um einen Fehler-Monitor für eine Ressource dieses Typs zu stoppen.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Pkglist` (`Zeichenketten-Array`)
Eine optionale Liste der Pakete, die in der Ressourcentypinstallation inbegriffen sind.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Postnet_stop` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM nach Aufruf der `Stop`-Methode einer beliebigen Netzwerkadressressource aufruft, von der eine Ressource dieses Typs abhängt. Nachdem die Netzwerkschnittstellen als inaktiv konfiguriert wurden, muss diese Methode `Stop`-Aktionen ausführen.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Prenet_start` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM vor Aufruf der `Start`-Methode einer beliebigen Netzwerkadressressource aufruft, von der eine Ressource dieses Typs abhängt. Diese Methode ist für `Start`-Aktionen vorgesehen, die ausgeführt werden müssen, bevor die Netzwerkschnittstellen konfiguriert werden.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Resource_type` (`Zeichenkette`)
Der Name des Ressourcentyps. Geben Sie Folgendes ein, um die Namen der aktuell registrierten Ressourcentypen anzuzeigen:

```
scrgadm -p
```

In Sun Cluster 3.1 und höheren Versionen muss ein Ressourcentypname auch die Version enthalten:

```
vendor_id.resource_type:version
```

Die drei Komponenten des Ressourcentypnamens sind Eigenschaften, die in der RTR-Datei als *Vendor_id*, *Resource_type* und *RT_version* angegeben sind. Der `scrgadm`-Befehl fügt einen Punkt (.) und einen Doppelpunkt (:) als Trennzeichen ein. Das `RT_version`-Suffix des Ressourcentypnamens hat den gleichen Wert wie die `RT_version`-Eigenschaft. Um sicherzustellen, dass *Vendor_id* einmalig ist, wird empfohlen, das Börsensymbol für das Unternehmen, das den Ressourcentyp erstellt, zu verwenden. Ressourcentypnamen, die vor Sun Cluster 3.1 erstellt wurden, verwenden weiterhin folgende Syntax:

```
vendor_id.resource_type
```

Kategorie: Erforderlich

Standard: Leerzeichenkette

Einstellbar: Nein

`RT_basedir` (`Zeichenkette`)
Der Verzeichnispfad, der zum Vervollständigen von relativen Pfaden für Rückmeldemethoden verwendet wird. Dieser Pfad muss auf den Installationsspeicherort für die Ressourcentyppakete eingestellt sein. Dabei muss es sich um einen vollständigen, mit einem Schrägstrich (/) beginnenden Pfad handeln. Diese Eigenschaft ist nicht erforderlich, wenn alle Methodenpfadnamen absolut sind.

Kategorie: Erforderlich, falls nicht alle Methodenpfadnamen absolut sind.

Standard: Kein Standard

Einstellbar: Nein

`RT_description` (`Zeichenkette`)
Eine kurze Beschreibung des Ressourcentyps.

Kategorie: Bedingt

Standard: Leerzeichenkette

Einstellbar: Nein

`RT_system` (`Boolescher Wert`)
Bei Einstellung auf `TRUE` für einen Ressourcentyp gibt diese Eigenschaft an, dass die zulässigen `scrgadm`(1M)-Vorgänge für diesen Ressourcentyp beschränkt sind. Ein Ressourcentyp, dessen `RT_system`-Wert auf `TRUE` eingestellt ist, wird als Systemressourcentyp bezeichnet. Die Bearbeitung der `RT_system`-Eigenschaft selbst ist nie beschränkt, unabhängig vom aktuellen Wert von `RT_system`.

Kategorie: Optional

Standard: `FALSE`

Einstellbar: Ja

`RT_version` (`Zeichenkette`)
Ab Sun Cluster 3.1 eine erforderliche Versionszeichenkette für diese Ressourcentypimplementierung. `RT_version` ist die Suffixkomponente des vollständigen Ressourcentypnamens. Die `RT_version`-Eigenschaft, die in Sun Cluster 3.0 optional war, ist in Sun Cluster 3.1 und späteren Versionen erforderlich.

Kategorie: Optional/Explizit oder Erforderlich

Standard: Kein Standard

Einstellbar: Nein

`Single_instance` (`Boolescher Wert`)
`TRUE` gibt an, dass nur eine Ressource dieses Typs im Cluster vorhanden sein darf. RGM lässt Cluster-weit jeweils nur die Ausführung einer Ressource dieses Typs zu.

Kategorie: Optional

Standard: FALSE

Einstellbar: Nein

`Start` (`Zeichenkette`)
Eine Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, um eine Ressource dieses Typs zu starten.

Kategorie: Erforderlich, es sei denn, in der RTR-Datei wird eine `Prenet_start` -Methode deklariert.

Standard: Kein Standard

Einstellbar: Nein

`Stop` (`Zeichenkette`)
Eine Rückmeldemethode: der Pfad zu dem Programm, das von RGM zum Stoppen einer Ressource dieses Typs aufgerufen wird.

Kategorie: Erforderlich, falls die RTR-Datei keine `Postnet_stop`-Methode deklariert.

Standard: Kein Standard

Einstellbar: Nein

`Update` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das RGM aufruft, wenn Eigenschaften einer laufenden Ressource dieses Typs geändert werden.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Validate` (`Zeichenkette`)
Eine optionale Rückmeldemethode: der Pfad zu dem Programm, das aufgerufen wird, um die Werte der Eigenschaften von Ressourcen dieses Typs zu prüfen.

Kategorie: Bedingt/Explizit

Standard: Kein Standard

Einstellbar: Nein

`Vendor_ID` (`Zeichenkette`)
Siehe die `Resource_type`-Eigenschaft.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: Nein

Ressourceneigenschaften

Dieser Abschnitt enthält eine Beschreibung der von Sun Cluster definierten Ressourceneigenschaften. Die Eigenschaftswerte werden in folgende Kategorien unterteilt:

- **Erforderlich** – Der Verwalter muss einen Wert angeben, wenn er eine Ressource mit einem Verwaltungsdienstprogramm erstellt.
- **Optional** – Wenn der Verwalter beim Erstellen einer Ressourcengruppe keinen Wert angibt, stellt das System einen Standardwert bereit.
- **Bedingt** – RGM erstellt die Eigenschaft nur dann, wenn sie in der RTR-Datei deklariert wurde. Andernfalls ist die Eigenschaft nicht vorhanden, und sie steht den Systemverwaltern nicht zur Verfügung. Eine bedingte Eigenschaft, die in der RTR-Datei deklariert wurde, ist optional oder erforderlich, je nachdem, ob in der RTR-Datei ein Standardwert angegeben ist. Weitere Details sind in der Beschreibung für jede bedingte Eigenschaft enthalten.
- **Nur-Abfrage** – Kann nicht direkt durch ein Verwaltungstool eingestellt werden.

"Tunable" gibt an, ob und wann Sie Ressourceneigenschaften aktualisieren können:

`NONE` oder `FALSE`	Nie
`TRUE` oder `ANYTIME`	Jederzeit
`AT_CREATION`	Beim Hinzufügen der Ressource zu einem Cluster
`WHEN_DISABLED`	Wenn die Ressource deaktiviert ist

Die Eigenschaftsnamen werden zuerst angezeigt, gefolgt von einer Beschreibung.

`Affinity_timeout` (`Ganzzahl`)
: Länge der Zeit in Sekunden, während der Verbindungen von einer bestimmten Client-IP-Adresse für jeden Dienst in der Ressource an denselben Serverknoten gesendet werden.

  Diese Eigenschaft ist nur relevant, wenn `Load_balancing_policy` entweder `Lb_sticky` oder `Lb_sticky_wild` ist. Außerdem muss `Weak_affinity` auf `FALSE` (Standardwert) eingestellt sein.

  Diese Eigenschaft wird nur für Scalable-Dienste verwendet.

  Kategorie: Optional

  Standard: Kein Standard

  Einstellbar: `ANYTIME`

`Cheap_probe_interval` (`Ganzzahl`)
Die Anzahl Sekunden zwischen den Aufrufen eines schnellen Fehlertests der Ressource. Diese Eigenschaft wird durch RGM erstellt und steht dem Verwalter nur zur Verfügung, wenn sie in der RTR-Datei deklariert wird.

Diese Eigenschaft ist optional, wenn in der RTR-Datei ein Standardwert angegeben ist. Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable` -Wert für die Eigenschaft `WHEN_DISABLED`.

Diese Eigenschaft ist erforderlich, wenn sie in der RTR-Datei deklariert und das Attribut `Default` nicht festgelegt ist.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

Erweiterungseigenschaften
Erweiterungseigenschaften, wie sie in der RTR-Datei des Ressourcentyps deklariert sind. Die Implementierung des Ressourcentyps definiert diese Eigenschaften. „Ressourceneigenschaftsattribute" auf Seite 278 enthält Informationen zu den einzelnen Attributen, die Sie für Erweiterungseigenschaften festlegen können.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: Abhängig von der spezifischen Eigenschaft.

`Failover_mode` (`Aufzählung`)
`NONE`, `SOFT` und `HARD` wirken sich nur auf das Failover-Verhalten aus, wenn eine Start-Methode (`Prenet_start` oder `Start`) fehlschlägt. Sobald die Ressource erfolgreich gestartet wurde, haben `NONE`, `SOFT` und `HARD` jedoch keine Auswirkung auf nachfolgende Neustarts der Ressource oder auf das Übergabeverhalten, das der Ressourcen-Monitor mit `scha_control`(1HA) oder `scha_control`(3HA) initialisiert. `NONE` (Standardeinstellung) gibt an, dass RGM beim Fehlschlagen einer Methode den Ressourcestatus angeben und auf eine Benutzereingabe warten soll. `SOFT` gibt an, dass RGM die Gruppe der Ressource beim Fehlschlagen einer `Start`-Methode auf einen anderen Knoten verlagern soll. Wenn eine `Stopp`- oder `Monitor_stop`-Methode fehlschlägt, versetzt RGM die Ressource in den `Stop_failed`-Status und die Ressourcengruppe in den `Error_stop_failed`-Status. Anschließend wartet RGM auf eine Benutzereingabe. Bei `Stopp`- oder `Monitor_stop`-Fehlern sind die Auswirkungen von `NONE` und `SOFT` dieselben. `HARD` gibt an, dass RGM die Gruppe beim Fehlschlagen einer `Start`-Methode verlagern soll. Wenn eine `Stopp`- oder `Monitor_stop`-Methode fehlschlägt, soll RGM die Ressource durch Abbruch des Cluster-Knotens stoppen. `HARD`, `NONE` und `SOFT` beeinflussen das Failover-Verhalten, wenn eine `Start`- oder `Prenet_start`-Methode fehlschlägt.

Im Gegensatz zu `NONE`, `SOFT` und `HARD` beeinflussen `RESTART_ONLY` und `LOG_ONLY` das gesamte Failover-Verhalten, einschließlich Monitor-initialisierte (`scha_control`) Neustarts von Ressourcen und Ressourcengruppen sowie

Übergaben, die vom Ressourcen-Monitor initialisiert wurden (`scha_control`). `RESTART_ONLY` gibt an, dass der Monitor `scha_control` zum Neustarten einer Ressource ausführen kann, jedoch Versuche zum Neustarten oder Übergeben einer Ressourcengruppe mit `scha_control` fehlschlagen. RGM ermöglicht `Retry_count` Neustarts innerhalb von `Retry_interval` . Wenn `Retry_count` überschritten wird, sind keine Ressourcen-Neustarts mehr zulässig. Wenn `Failover_mode` auf `LOG_ONLY` eingestellt ist, sind keine Ressourcen-Neustarts oder -Übergaben zulässig. Die Einstellung von `Failover_mode` auf `LOG_ONLY` entspricht der Einstellung von `Failover_mode` auf `RESTART_ONLY` bei einem `Retry_count` von null. Wenn eine Start-Methode fehlschlägt, haben `RESTART_ONLY` und `LOG_ONLY` dieselbe Wirkung wie `NONE`: Es erfolgt kein Failover und die Ressource wird in den `Start_failed`-Status versetzt.

Kategorie: Optional

Standard: Kein Standard

Einstellbar: `ANYTIME`

`Load_balancing_policy` (`Zeichenkette`)
Eine Zeichenkette, welche das verwendete Lastausgleichsverfahren definiert. Diese Eigenschaft wird nur für Scalable-Dienste verwendet. RGM erstellt diese Eigenschaft automatisch, wenn die `Scalable`-Eigenschaft in der RTR-Datei deklariert ist. `Load_balancing_policy` kann die folgenden Werte aufweisen:

`Lb_weighted` (Standardwert). Die Last wird auf mehrere Knoten verteilt, entsprechend den in der Eigenschaft `Load_balancing_weights` eingestellten Gewichtungen.

`Lb_sticky`. Ein bestimmter Client (identifiziert durch die Client-IP-Adresse) des Scalable-Dienstes wird immer an denselben Cluster-Knoten gesendet.

`Lb_sticky_wild`. Die IP-Adresse eines bestimmten Client, der eine Verbindung zu einer IP-Adresse eines Platzhalter-Sticky-Dienstes herstellt, wird unabhängig von der Port-Nummer, zu der die IP-Adresse führt, stets an denselben Cluster-Knoten gesendet.

Kategorie: Bedingt/Optional

Standard: `Lb_weighted`

Einstellbar: `AT_CREATION`

`Load_balancing_weights` (`Zeichenketten-Array`)
Nur für skalierbare Ressourcen. RGM erstellt diese Eigenschaft automatisch, wenn die `Scalable`-Eigenschaft in der RTR-Datei deklariert ist. Das Format ist *Gewichtung@Knoten, Gewichtung@Knoten*, wobei *Gewichtung* eine Ganzzahl ist, die den relativen Anteil der Last angibt, die auf den angegebenen *Knoten* verteilt wird. Der Lastanteil, der auf einen Knoten verteilt wird, ist die Gewichtung dieses Knotens, geteilt durch die Summe aller Gewichtungen. So gibt zum Beispiel

`1@1,3@2` an, dass Knoten 1 1/4 der Last erhält, und Knoten 2 3/4. Die leere Zeichenkette (“”) ist der Standardwert und stellt eine gleichmäßige Verteilung ein. Jeder Knoten, dem keine ausdrückliche Gewichtung zugewiesen wurde, erhält eine Standardgewichtung von 1.

Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable`-Wert für die Eigenschaft `ANYTIME`. Eine Änderung dieser Eigenschaft ändert die Verteilung nur für neue Verbindungen.

Kategorie: Bedingt/Optional

Standard: Die Leerzeichenkette (“”)

Einstellbar: `ANYTIME`

Methoden`_timeout` für jede Rückmeldemethode im Typ (`Ganzzahl.`)
Der Zeitraum in Sekunden, nach dem RGM entscheidet, dass der Aufruf einer Methode fehlgeschlagen ist.

Kategorie: Bedingt/Optional

Standard: 3600 (eine Stunde), wenn die Methode selbst in der RTR-Datei deklariert ist

Einstellbar: `ANYTIME`

`Monitored_switch` (`Aufzählung`)
Wird von RGM auf `Enabled` oder `Disabled` eingestellt, wenn der Cluster-Verwalter den Monitor mit einem Verwaltungsdienstprogramm aktiviert oder deaktiviert. Im Fall von `Disabled` wird die `Start`-Methode des Monitors nicht mehr aufgerufen, bis er wieder aktiviert ist. Wenn die Ressource keine Monitor-Rückmeldemethode hat, ist diese Eigenschaft nicht vorhanden.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nie

`Network_resources_used` (`Zeichenketten-Array`)
Eine Liste logischer Hostnamen- bzw. gemeinsam genutzter Netzwerkadressressourcen, die von der Ressource verwendet werden. Für Scalable-Dienste muss sich diese Eigenschaft auf gemeinsam genutzte Adressressourcen beziehen, die in einer eigenen Ressourcengruppe vorhanden sind. Für Failover-Dienste bezieht sich diese Eigenschaft auf logische Hostnamen- oder gemeinsam genutzte Adressressourcen, die in derselben Ressourcengruppe vorhanden sind. RGM erstellt diese Eigenschaft automatisch, wenn die `Scalable`-Eigenschaft in der RTR-Datei deklariert ist. Wenn `Scalable` nicht in der RTR-Datei deklariert ist, steht `Network_resources_used` nicht zur Verfügung, es sei denn, sie wird explizit in der RTR-Datei deklariert.

Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable`-Wert für die Eigenschaft `AT_CREATION`.

Hinweis – In der Online-Dokumentation unter `SUNW.Event`(5) ist beschrieben, wie diese Eigenschaft für CRNP festgelegt wird.

Kategorie: Bedingt/Erforderlich

Standard: Kein Standard

Einstellbar: `AT_CREATION`

`Num_resource_restarts` auf jedem Cluster-Knoten (`Ganzzahl`)
Sie können diese Eigenschaft nicht direkt festlegen, da sie von RGM auf die Anzahl der `scha_control`-, `Resource_restart`- oder `Resource_is_restarted`-Aufrufe eingestellt wird, die für diese Ressource auf diesem Knoten innerhalb der letzten *n* Sekunden durchgeführt wurden. *n* ist der Wert der `Retry_interval`-Eigenschaft der Ressource. Der Ressourcen-Neustartzähler wird von RGM stets auf null zurückgesetzt, wenn von dieser Ressource eine `scha_control`-Übergabe durchgeführt wird, unabhängig davon, ob der Übergabeversuch gelingt oder fehlschlägt.

Wenn für einen Ressourcentyp die `Retry_interval` -Eigenschaft nicht deklariert wurde, dann steht die Eigenschaft `Num_resource_restarts` den Ressourcen dieses Typs nicht zur Verfügung.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nein

`Num_rg_restarts` auf jedem Cluster-Knoten (`Ganzzahl`)
Sie können diese Eigenschaft nicht direkt festlegen, da sie von RGM auf die Anzahl der `scha_control -Restart`-Aufrufe eingestellt wird, welche von dieser Ressource für ihre eigenen Ressourcengruppen auf diesem Knoten innerhalb der letzten *n* Sekunden durchgeführt wurden, wobei *n* dem Wert der Eigenschaft `Retry_interval` der Ressource entspricht. Wenn ein Ressourcentyp die `Retry_interval`-Eigenschaft nicht deklariert, ist die `Num_rg_restarts`-Eigenschaft für Ressourcen dieses Typs nicht verfügbar.

Kategorie: Siehe Beschreibung

Standard: Kein Standard

Einstellbar: Nein

`On_off_switch` (`Aufzählung`)
Wird von RGM auf `Enabled` oder `Disabled` eingestellt, wenn der Cluster-Verwalter die Ressource mit einem Verwaltungsdienstprogramm aktiviert oder deaktiviert. Wenn diese Eigenschaft deaktiviert wird, wird eine Ressource in den Offline-Status versetzt und es werden keine Rückmeldungen aufgerufen, bis sie wieder aktiviert wird.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nie

`Port_list` (`Zeichenketten-Array`)
Eine Liste mit Port-Nummern, die der Server abhört. Auf jede Port-Nummer folgt ein Schrägstrich (/) und das Protokoll, das von diesem Port verwendet wird. Zum Beispiel: `Port_list=80/tcp` oder `Port_list=80/tcp6,40/udp6`. Sie können die folgenden Protokollwerte angeben:

- `tcp` für TCP IPv4
- `tcp6` für TCP IPv6
- `udp` für UDP IPv4
- `udp6` für UDP IPv6 Wenn die `Scalable`-Eigenschaft in der RTR-Datei deklariert ist, erstellt RGM `Port_list` automatisch. Andernfalls steht diese Eigenschaft nicht zur Verfügung, es sei denn, sie wird explizit in der RTR-Datei deklariert.

Das Konfigurieren dieser Eigenschaft für Apache wird im *Handbuch zum Sun Cluster-Datendienst für Apache für Solaris OS* beschrieben.

Kategorie: Bedingt/Erforderlich

Standard: Kein Standard

Einstellbar: `AT_CREATION`

`R_description` (`Zeichenkette`)
Eine kurze Beschreibung der Ressource.

Kategorie: Optional

Standard: Die Leerzeichenkette

Einstellbar: `ANYTIME`

`Resource_dependencies` (`Zeichenketten-Array`)
Eine Liste der Ressourcen in denselben oder verschiedenen Gruppen, von denen diese Ressource stark abhängig ist. Diese Ressource kann nicht gestartet werden, wenn eine der Ressourcen in der Liste nicht online ist. Wenn diese Ressource und eine der Ressourcen in der Liste zur selben Zeit gestartet werden, wartet RGM, bis die Ressource in der Liste gestartet wurde, bevor diese Ressource gestartet wird. Wenn die Ressource in der Liste `Resource_dependencies` dieser Ressource nicht gestartet wird, bleibt diese Ressource ebenfalls offline. Die Ressource in der Liste dieser Ressource wird möglicherweise nicht gestartet, weil die Ressourcengruppe für die Ressource in der Liste offline bleibt, oder weil die Ressource in der Liste den Status `Start_failed` aufweist. Wenn diese Ressource aufgrund ihrer Abhängigkeit von einer Ressource in einer anderen Ressourcengruppe, die nicht gestartet werden kann, offline bleibt, wird die Gruppe dieser Ressource in den Status `Pending_online_blocked` versetzt.

Wenn diese Ressource zur selben Zeit wie die Ressourcen in der Liste offline geschaltet wird, wird diese Ressource vor jenen in der Liste gestoppt. Wenn diese Ressource jedoch online bleibt oder nicht gestoppt werden kann, wird eine

Ressource in der Liste, die sich in einer anderen Ressourcengruppe befindet, dennoch gestoppt. Die Ressourcen in der Liste können nur deaktiviert werden, wenn diese Ressource zuerst deaktiviert wird.

In einer Ressourcengruppe weisen Anwendungsressourcen standardmäßig eine starke Abhängigkeit von Netzwerkadressenressourcen auf. Unter `Implicit_network_dependencies` in „Ressourcengruppeneigenschaften" auf Seite 272 finden Sie weitere Informationen hierzu.

Innerhalb einer Ressourcengruppe werden `Prenet_start`-Methoden in Abhängigkeitsreihenfolge vor `Start`-Methoden ausgeführt. `Postnet_stop`-Methoden werden in Abhängigkeitsreihenfolge nach `Stopp`-Methoden ausgeführt. Bei unterschiedlichen Ressourcengruppen wartet die abhängige Ressource darauf, dass die andere Ressource, von der sie abhängt, `Prenet_start` und `Start` beendet hat, bevor sie `Prenet_start` ausführt. Die andere Ressource hingegen wartet darauf, dass die abhängige Ressource `Stop` und `Postnet_stop` beendet hat, bevor sie `Stop` ausführt.

Kategorie: Optional

Standard: Die leere Liste

Einstellbar: `ANYTIME`

`Resource_dependencies_restart` (`Zeichenketten-Array`)
Eine Liste der Ressourcen in denselben oder verschiedenen Gruppen, von denen der Neustart dieser Ressource abhängig ist.

Diese Eigenschaft funktioniert wie `Resource_dependencies` mit der Ausnahme, dass diese Ressource neu gestartet wird, wenn eine der Ressourcen in der Neustart-Abhängigkeitsliste neu gestartet wird. Der Neustart dieser Ressource erfolgt, sobald die Ressource in der Liste wieder online ist.

Kategorie: Optional

Standard: Die leere Liste

Einstellbar: `ANYTIME`

`Resource_dependencies_weak` (`Zeichenketten-Array`)
Eine Liste der Ressourcen in denselben oder verschiedenen Gruppen, von denen diese Ressource schwach abhängig ist. Eine schwache Abhängigkeit bestimmt die Reihenfolge der Methodenaufrufe. RGM ruft die `Start`-Methoden der Ressourcen in dieser Liste auf, bevor die `Start`-Methode dieser Ressource aufgerufen wird. RGM ruft die `Stopp`-Methoden dieser Ressource an, bevor die `Stopp`-Methoden der Ressourcen in der Liste aufgerufen werden. Die Ressource kann auch gestartet werden, wenn die Ressourcen in der Liste nicht gestartet werden können oder offline bleiben.

Wenn diese Ressource und eine Ressource in ihrer `Resource_dependencies_weak`-Liste gleichzeitig gestartet werden, wartet RGM, bis die Ressource in der Liste gestartet wurde, bevor diese Ressource gestartet wird. Wenn die Ressource in der Liste nicht gestartet wird, —

beispielsweise, wenn die Ressourcengruppe für die Ressource in der Liste offline bleibt oder die Ressource in der Liste den Status `Start_failed` aufweist — wird diese Ressource dennoch gestartet. Die Ressourcengrupe dieser Ressource wird möglicherweise vorübergehend in den `Pending_online_blocked`-Status versetzt, während die Ressourcen in der `Resource_dependencies_weak`-Liste dieser Ressource gestartet werden. Wenn alle Ressourcen in der Liste gestartet wurden oder nicht gestartet werden konnten, wird diese Ressource gestartet und ihre Gruppe wieder in den `Pending_online`-Status versetzt.

Wenn diese Ressource zur selben Zeit wie die Ressourcen in der Liste offline geschaltet wird, wird diese Ressource vor jenen in der Liste gestoppt. Wenn diese Ressource online bleibt oder nicht gestoppt werden kann, wird eine Ressource in der Liste dennoch gestoppt. Sie können die Ressourcen in der Liste nur deaktivieren, wenn diese Ressource zuerst deaktiviert wird.

Innerhalb einer Ressourcengruppe werden `Prenet_start`-Methoden in Abhängigkeitsreihenfolge vor `Start`-Methoden ausgeführt. `Postnet_stop`-Methoden werden in Abhängigkeitsreihenfolge nach `Stopp`-Methoden ausgeführt. Bei unterschiedlichen Ressourcengruppen wartet die abhängige Ressource darauf, dass die andere Ressource, von der sie abhängt, `Prenet_start` und `Start` beendet hat, bevor sie `Prenet_start` ausführt. Die andere Ressource hingegen wartet darauf, dass die abhängige Ressource `Stop` und `Postnet_stop` beendet hat, bevor sie `Stop` ausführt.

Kategorie: Optional

Standard: Die leere Liste

Einstellbar: `ANYTIME`

`Resource_name` (`Zeichenkette`)
Der Name der Ressourceninstanz. Dieser Name muss in der Cluster-Konfiguration einmalig sein und kann nach Erstellen einer Ressource nicht mehr geändert werden.

Kategorie: Erforderlich

Standard: Kein Standard

Einstellbar: Nie

`Resource_project_name` (`Zeichenkette`)
Der Solaris-Projektname, der dieser Ressource zugeordnet ist. Diese Eigenschaft wird verwendet, um Solaris-Ressourcenverwaltungsfunktionen wie CPU-Anteile und Ressourcen-Pools auf Cluster-Datendienste anzuwenden. Wenn RGM Ressourcen online bringt, werden die entsprechende Prozesse unter diesem Projektnamen gestartet. Wenn diese Eigenschaft nicht angegeben ist, wird der Projektname von der Eigenschaft `RG_project_name` der Ressourcengruppe mit der Ressource übernommen (siehe `rg_properties` (5)). Wenn keine der Eigenschaften angegeben ist, verwendet RGM den vordefinierten Projektnamen `default`. Der angegebene Projektname muss in der Projektdatenbank vorhanden sein, und der Benutzer `root` muss als Mitglied des benannten Projekts konfiguriert sein. Diese Eigenschaft wird nur in Solaris 9 oder höher unterstützt.

Hinweis – Änderungen an dieser Eigenschaft werden beim nächsten Starten der Ressource wirksam.

Kategorie: Optional

Standard: Null

Einstellbar: ANYTIME

`Resource_state` auf jedem Cluster-Knoten (`Aufzählung`)
Der von RGM festgelegte Zustand der Ressource auf jedem Cluster-Knoten. Mögliche Zustände sind `Online`, `Offline`, `Start_failed`, `Stop_failed`, `Monitor_failed`, `Online_not_monitored`, `Starting` und `Stopping`.

Sie können diese Eigenschaft nicht konfigurieren.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nie

`Retry_count` (`Ganzzahl`)
Anzahl der Male, die ein Monitor im Fehlerfall versucht, eine Ressource neu zu starten. Diese Eigenschaft wird durch RGM erstellt und steht dem Verwalter nur zur Verfügung, wenn sie in der RTR-Datei deklariert wird. `Retry_count` ist optional, wenn in der RTR-Datei ein Standardwert angegeben ist.

Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable`-Wert für die Eigenschaft `WHEN_DISABLED`.

Diese Eigenschaft ist erforderlich, wenn sie in der RTR-Datei deklariert und das Attribut `Default` nicht festgelegt ist.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

`Retry_interval` (`Ganzzahl`)
Die Anzahl Sekunden, während der die Versuche, eine fehlgeschlagene Ressource neu zu starten, gezählt werden. Der Ressourcen-Monitor verwendet diese Eigenschaft zusammen mit `Retry_count` . Diese Eigenschaft wird durch RGM erstellt und steht dem Verwalter nur zur Verfügung, wenn sie in der RTR-Datei deklariert wird. `Retry_interval` ist optional, wenn in der RTR-Datei ein Standardwert angegeben ist.

Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable`-Wert für die Eigenschaft `WHEN_DISABLED`.

Diese Eigenschaft ist erforderlich, wenn sie in der RTR-Datei deklariert und das Attribut `Default` nicht festgelegt ist.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

`Scalable` (`Boole'scher Wert`)
Gibt an, ob die Ressource skalierbar ist, d. h., ob die Ressource die Netzwerk-Lastausgleichsfunktionen von Sun Cluster nutzt.

Wenn diese Eigenschaft in der RTR-Datei deklariert ist, erstellt RGM automatisch die folgenden Scalable-Diensteigenschaften für Ressourcen dieses Typs: `Affinity_timeout`, `Load_balancing_policy`, `Load_balancing_weights`, `Network_resources_used`, `Port_list`, `UDP_affinity` und `Weak_affinity` . Diese Eigenschaften haben Standardwerte, wenn sie nicht explizit in der RTR-Datei deklariert werden. Der Standardwert für `Scalable` ist `True`, wenn die Eigenschaft in der RTR-Datei deklariert ist.

Wenn diese Eigenschaft in der RTR-Datei deklariert ist, darf ihr kein anderes Tunable-Attribut als `AT_CREATION` zugewiesen werden.

Wenn diese Eigenschaft nicht in der RTR-Datei deklariert ist, ist die Ressource nicht skalierbar. In diesem Fall können Sie keine Einstellungen dafür festlegen und von RGM werden keine Scalable-Diensteigenschaften eingestellt. Sie können die Eigenschaften `Network_resources_used` und `Port_list` jedoch explizit in der RTR-Datei deklarieren, da diese Eigenschaften in einem Nicht-Scalable-Dienst ebenso nützlich sein können wie in einem Scalable-Dienst.

Sie verwenden diese Ressourceneigenschaft wie folgt in Verbindung mit der `Failover`-Ressourcentypeigenschaft:

Die Verwendung dieser Ressourceneigenschaft in Verbindung mit der `Failover`-Ressourcentypeigenschaft ist unter `r_properties` (5) genauer beschrieben.

Kategorie: Optional

Standard: Kein Standard

Einstellbar: `AT_CREATION`

`Status` auf jedem Cluster-Knoten (`Aufzählung`)
Wird vom Ressourcen-Monitor mithilfe von `scha_resource_setstatus`(1HA) oder `scha_resource_setstatus`(3HA) eingestellt. Mögliche Werte sind `OK`, `degraded`, `faulted`, `unknown` und `offline`. Wenn eine Ressource online oder offline geschaltet wird, stellt RGM den `Status`-Wert automatisch ein, falls der `Status`-Wert nicht vom Monitor oder den Methoden der Ressource eingestellt wird.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nie

`Status_msg` auf jedem Cluster-Knoten (`Zeichenkette`)
Wird vom Ressourcen-Monitor zur gleichen Zeit wie die `Status`-Eigenschaft eingestellt. Wenn eine Ressource online oder offline geschaltet wird, setzt RGM diese Eigenschaft automatisch zurück, falls sie nicht von den Methoden der Ressource eingestellt wird.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nie

`Thorough_probe_interval` (`Ganzzahl`)
Die Anzahl Sekunden zwischen den Aufrufen eines Überlauftests der Ressource. Diese Eigenschaft wird durch RGM erstellt und steht dem Verwalter nur zur Verfügung, wenn sie in der RTR-Datei deklariert wird. `Thorough_probe_interval` ist optional, wenn in der RTR-Datei ein Standardwert angegeben ist.

Wenn das `Tunable`-Attribut nicht in der Ressourcentypdatei angegeben ist, lautet der `Tunable`-Wert für die Eigenschaft `WHEN_DISABLED`.

Diese Eigenschaft ist erforderlich, wenn das `Default`-Attribut nicht in der Eigenschaftsdeklaration in der RTR-Datei angegeben ist.

Kategorie: Bedingt

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

`Type` (`Zeichenkette`)
Der Ressourcentyp, von dem die Ressource eine Instanz darstellt.

Kategorie: Erforderlich

Standard: Kein Standard

Einstellbar: Nie

`Type_version` (`Zeichenkette`)
Gibt an, welche Version des Ressourcentyps der Ressource aktuell zugeordnet ist. RGM erstellt diese Eigenschaft automatisch. Sie kann nicht in der RTR-Datei deklariert werden. Der Wert dieser Eigenschaft entspricht der `RT_version`-Eigenschaft des Ressourcentyps. Bei Erstellung einer Ressource wird die `Type_version`-Eigenschaft nicht ausdrücklich angegeben. Sie kann jedoch als Suffix des Ressourcentypnamens angezeigt werden. Wenn eine Ressource bearbeitet wird, kann `Type_version` einen neuen Wert erhalten.

Die Einstellbarkeit dieser Eigenschaft wird aus den folgenden Quellen abgeleitet:

- Der aktuellen Version des Ressourcentyps,

- Der `#$upgrade_from`-Anweisung in der RTR-Datei.

Kategorie: Siehe Beschreibung

Standard: Kein Standard

Einstellbar: Siehe Beschreibung

`UDP_affinity` (`Boole'scher Wert`)
Wenn der Wert "true" ist, wird der ganze UDP-Verkehr eines bestimmten Clients an denselben Serverknoten gesendet, der aktuell den ganzen TCP-Verkehr für den Client bearbeitet.

Diese Eigenschaft ist nur relevant, wenn `Load_balancing_policy` entweder `Lb_sticky` oder `Lb_sticky_wild` ist. Außerdem muss `Weak_affinity` auf `FALSE` (Standardwert) eingestellt sein.

Diese Eigenschaft wird nur für Scalable-Dienste verwendet.

Kategorie: Optional

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

`Weak_affinity` (`Boole'scher Wert`)
Wenn "true" eingestellt ist, wird die schwache Form der Client-Affinität aktiviert. Die schwache Form der Client-Affinität ermöglicht, dass Verbindungen von einem bestimmten Client an denselben Serverknoten gesendet werden können. Hierbei gelten jedoch folgende Ausnahmebedingungen:

- Ein Server-Listener wird gestartet, beispielsweise als Reaktion auf den Neustart eines Fehler-Monitors, auf den Failover oder die Umschaltung einer Ressource oder auf die Rückkehr eines Knotens in einen Cluster nach einem Fehler.
- `Load_balancing_weights` für die skalierbare-Ressource wird aufgrund einer Verwaltungsaktion geändert.

Eine schwache Affinität sorgt für eine niedrige Überlaufalternative zur Standardform, und zwar sowohl bezüglich des Speicherverbrauchs als auch der Prozessorzyklen.

Diese Eigenschaft ist nur relevant, wenn `Load_balancing_policy` entweder `Lb_sticky` oder `Lb_sticky_wild` ist.

Diese Eigenschaft wird nur für Scalable-Dienste verwendet.

Kategorie: Optional

Standard: Kein Standard

Einstellbar: `WHEN_DISABLED`

Ressourcengruppeneigenschaften

Die folgenden Informationen beschreiben die Ressourcengruppeneigenschaften, die von Sun Cluster definiert werden. Die Eigenschaftswerte werden in folgende Kategorien unterteilt:

- **Erforderlich** — Der Verwalter muss einen Wert angeben, wenn er eine Ressourcengruppe mit einem Verwaltungsdienstprogramm erstellt.
- **Optional** – Wenn der Verwalter beim Erstellen einer Ressourcengruppe keinen Wert angibt, stellt das System einen Standardwert bereit.
- **Nur-Abfrage** – Kann nicht direkt durch ein Verwaltungstool eingestellt werden.

Jede Beschreibung gibt an, ob die Eigenschaft aktualisiert werden kann (Ja) oder nicht (Nein), nachdem sie erstmals eingestellt wurde.

Die Eigenschaftsnamen werden zuerst angezeigt, gefolgt von einer Beschreibung.

`Auto_start_on_new_cluster` (`Boole'scher Wert`)
Diese Eigenschaft lässt kein automatisches Starten der Ressourcengruppe zu, wenn ein neuer Cluster gebildet wird.

Wenn `TRUE` eingestellt ist, versucht Ressourcengruppen-Manager, die Ressourcengruppe automatisch zu starten, um `Desired_primaries` zu erzielen, wenn der Cluster neu gestartet wird. Wenn `FALSE` eingestellt ist, startet die Ressourcengruppe nicht automatisch, wenn der Cluster neu gebootet wird.

Kategorie: Optional

Standard: `TRUE`

Einstellbar: Ja

`Desired_primaries` (`Ganzzahl`)
Die Anzahl der Knoten, auf denen die Gruppe gleichzeitig online gebracht werden soll.

Wenn die `RG_mode`-Eigenschaft `Failover` ist, dann darf der Wert dieser Eigenschaft nicht größer als 1 sein. Wenn die `RG_mode`-Eigenschaft `Scalable` ist, dann ist ein Wert größer als 1 zulässig.

Kategorie: Optional

Standard: `1`

Einstellbar: Ja

`Failback` (`Boole'scher Wert`)
Ein boole'scher Wert, der angibt, ob der Satz der Knoten, auf denen die Gruppe online ist, neu berechnet wird, wenn sich die Cluster-Mitgliedschaft ändert. Eine Neuberechnung kann dazu führen, dass RGM die Gruppe auf weniger bevorzugten Knoten offline und auf stärker bevorzugten Knoten online bringt.

Kategorie: Optional

Standard: `FALSE`

Einstellbar: Ja

`Global_resources_used` (`Zeichenketten-Array`)
Gibt an, ob Cluster-Dateisysteme von einer Ressource in dieser Ressourcengruppe verwendet werden. Zulässige Werte, die der Verwalter angeben kann, sind ein Sternchen (*) für alle globalen Ressourcen und die leere Zeichenkette ("") für keine globalen Ressourcen.

Kategorie: Optional

Standard: Alle globalen Ressourcen

Einstellbar: Ja

`Implicit_network_dependencies` (`Boole'scher Wert`)
Ein boole'scher Wert, der bei Einstellung auf `TRUE` angibt, dass RGM starke Abhängigkeiten von Nicht-Netzwerkadressressourcen von Netzwerkadressressourcen innerhalb der Gruppe erzwingen soll. Netzwerkadressressourcen umfassen die logische Hostnamen- und gemeinsam genutzten Adressressourcentypen.

In einer Scalable-Ressourcengruppe hat diese Eigenschaft keine Wirkung, da eine solche Gruppe keine Netzwerkadressressourcen enthält.

Kategorie: Optional

Standard: `TRUE`

Einstellbar: Ja

`Maximum_primaries` (`Ganzzahl`)
Die maximale Anzahl der Knoten, auf denen die Gruppe gleichzeitig online sein kann.

Wenn die `RG_mode`-Eigenschaft `Failover` ist, dann darf der Wert dieser Eigenschaft nicht größer als 1 sein. Wenn die `RG_mode`-Eigenschaft `Scalable` ist, dann ist ein Wert größer als 1 zulässig.

Kategorie: Optional

Standard: `1`

Einstellbar: Ja

`Nodelist` (`Zeichenketten-Array`)
Eine Liste der Cluster-Knoten, auf denen die Gruppe in der Reihenfolge ihres Vorrangs online gebracht werden kann. Diese Knoten werden als potenzielle Primärknoten bzw. Master der Ressourcengruppe bezeichnet.

Kategorie: Optional

Standard: Die Liste aller Cluster-Knoten

Einstellbar: Ja

`Pathprefix` (`Zeichenkette`)
Ein Verzeichnis im Cluster-Dateisystem, in dem Ressourcen in der Gruppe wesentliche Verwaltungsdateien schreiben können. Für einige Ressourcen kann diese Eigenschaft erforderlich sein. `Pathprefix` muss für jede Ressourcengruppe einmalig sein.

Kategorie: Optional

Standard: Die Leerzeichenkette

Einstellbar: Ja

`Pingpong_interval` (`Ganzzahl`)
Ein nicht negativer Ganzzahlwert (in Sekunden), der von RGM verwendet wird, um zu bestimmen, wo die Ressourcengruppe online geschaltet werden soll. Bedingungen, unter denen diese Eigenschaft erforderlich sein kann:

- Wenn eine Neukonfiguration erfolgt.
- Ein `scha_control -O GIVEOVER`-Befehl oder eine `scha_control()` -Funktion mit dem `SCHA_GIVEOVER` -Argument wird ausgeführt. Falls eine Neukonfiguration erfolgt, wenn die Ressourcengruppe mehr als einmal innerhalb der letzten `Pingpong_interval` -Sekunden auf einem bestimmten Knoten nicht online geschaltet werden konnte, kommt dieser Knoten nicht als Host für die Ressourcengruppe in Frage und RGM sucht nach einem anderen Master. Die Ressourcengruppe kann nicht online geschaltet werden, weil die `Start-` oder `Prenet_start`-Methode der Ressource nicht mit null beendet wurde oder eine Zeitüberschreitung verursacht hat.

  Wenn ein Aufruf des `scha_control`-Befehls bzw. der Funktion der Ressource die Ressourcengruppe auf einem bestimmten Knoten innerhalb der letzten `Pingpong_interval`-Sekunden offline bringt, kommt dieser Knoten für das Hosten der Ressourcengruppe infolge eines späteren Aufrufs von `scha_control()` durch einen anderen Knoten nicht in Frage.

Kategorie: Optional

Standard: 3600 (eine Stunde)

Einstellbar: Ja

`Resource_list` (`Zeichenketten-Array`)
Die Liste der in dieser Gruppe enthaltenen Ressourcen. Der Verwalter stellt diese Eigenschaft nicht direkt ein. RGM aktualisiert sie immer dann, wenn der Verwalter der Ressourcengruppe Ressourcen hinzufügt bzw. sie daraus entfernt.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nein

`RG_affinities` (`Zeichenkette`)
RGM soll nach einer Ressourcengruppe auf einem Knoten suchen, der zurzeit Master einer bestimmten anderen Ressourcengruppe ist (positive Affinität), oder nach einer Ressourcengruppe auf einem Knoten, der zurzeit nicht Master einer bestimmten Ressourcengruppe ist (negative Affinität).

`RG_affinities` kann auf die folgenden Zeichenketten eingestellt werden:

- ++ (starke positive Affinität)
- + (schwache positive Affinität)
- - (schwache negative Affinität)
- -- (starke negative Affinität)
- +++ (starke positive Affinität mit Failover-Delegierung)

`RG_affinities=+RG2,--RG3` gibt beispielsweise an, dass diese Ressourcengruppe eine schwache positive Affinität für RG2 und eine starke negative Affinität für RG3 aufweist.

Die Verwendung von `RG_affinities` ist unter „VAdministering Data Service Resources" im *Sun Cluster Data Services Planning and Administration Guide for Solaris OS* beschrieben.

Kategorie: Optional

Standard: Die Leerzeichenkette

Einstellbar: Ja

`RG_dependencies` (`Zeichenketten-Array`)
Optionale Liste von Ressourcengruppen, welche die bevorzugte Reihenfolge für die Online- oder Offline-Schaltung anderer Gruppen auf demselben Knoten angibt. Der Graf aller starken `RG_affinities` (positiv und negativ) zusammen mit `RG_dependencies` darf keine Zirkelbezüge enthalten.

Beispiel: Die Ressourcengruppe RG2 ist in der Liste `RG_dependencies` von Ressourcengruppe RG1 enthalten. Das bedeutet, RG1 weist eine Ressourcengruppenabhängigkeit von RG2 auf. Die folgende Liste fasst die Auswirkungen dieser Ressourcengruppenabhängigkeit zusammen:

- Wenn ein Knoten dem Cluster beitritt, werden die `Boot`-Methoden auf diesem Knoten nicht auf Ressourcen in RG1 ausgeführt, bis alle `Boot`-Methoden auf diesem Knoten auf den Ressourcen in RG2 ausgeführt wurden.
- Wenn RG1 und RG2 auf demselben Knoten gleichzeitig den Status `Pending_online` aufweisen, werden die Start-Methoden (`Prenet_start` oder `Start`) auf keinen Ressourcen in RG1 ausgeführt, bis alle Ressourcen in RG2 ihre Start-Methoden ausgeführt haben.

- Wenn RG1 und RG2 auf demselben Knoten gleichzeitig den Status `Pending_offline` aufweisen, werden die Stopp-Methoden (`Stop` oder `Postnet_stop`) auf keinen Ressourcen in RG2 ausgeführt, bis alle Ressourcen in RG1 ihre Stopp-Methoden ausgeführt haben.
- Ein Versuch zum Umschalten der Primärknoten von RG1 oder RG2 schlägt fehl, wenn dadurch RG1 auf einem Knoten online und RG2 auf allen Knoten offline geschaltet ist. Weitere Informationen hierzu finden Sie unter `scswitch`(1M) und `scsetup`(1M) in der Online-Dokumentation.
- Das Einstellen der `Desired_primaries`-Eigenschaft für RG1 auf einen Wert größer null ist nicht zulässig, wenn `Desired_primaries` für RG2 auf null eingestellt ist.
- Das Einstellen der `Auto_start_on_new_cluster`-Eigenschaft für RG1 auf `TRUE` ist nicht zulässig, wenn `Auto_start_on_new_cluster` für RG2 auf `FALSE` eingestellt ist.

Kategorie: Optional

Standard: Die leere Liste

Einstellbar: Ja

`RG_description` (`Zeichenkette`)
Eine kurze Beschreibung der Ressourcengruppe.

Kategorie: Optional

Standard: Die Leerzeichenkette

Einstellbar: Ja

`RG_is_frozen` (`Boole'scher Wert`)
Ein boole'scher Wert, der angibt, ob ein globales Gerät, von dem eine Ressourcengruppe abhängt, umgeschaltet wird. Wenn diese Eigenschaft auf `TRUE` eingestellt ist, wird das globale Gerät umgeschaltet. Wenn diese Eigenschaft auf `FALSE` eingestellt ist, wird kein globaley Gerät umgeschaltet. Die Abhängigkeit einer Ressourcengruppe von globalen Geräten ist durch die Eigenschaft `Global_resources_used` festgelegt.

Die Eigenschaft `RG_is_frozen` wird nicht direkt eingestellt. RGM aktualisiert die Eigenschaft `RG_is_frozen`, wenn sich der Status der globalen Geräte ändert.

Kategorie: Optional

Standard: Kein Standard

Einstellbar: Nein

`RG_mode` (`Aufzählung`)
Gibt an, ob die Ressourcengruppe eine Failover- oder Scalable-Gruppe ist. Wenn der Wert `Failover` ist, stellt RGM die `Maximum_primaries`-Eigenschaft der Gruppe auf 1 und beschränkt die Ressourcengruppe auf einen einzigen Knoten als Master.

Wenn der Wert dieser Eigenschaft `Scalable` ist, lässt RGM für die `Maximum_primaries`-Eigenschaft einen Wert größer als 1 zu. Das bedeutet, dass die Gruppe mehrere Knoten gleichzeitig als Master haben kann. RGM lässt nicht zu, dass eine Ressource, deren `Failover`-Eigenschaft `TRUE` ist, einer Ressourcengruppe hinzugefügt wird, deren `RG_mode` auf `Scalable` eingestellt ist.

Wenn `Maximum_primaries` gleich 1 ist, lautet die Standardeinstellung `Failover`. Wenn `Maximum_primaries` größer als 1 ist, lautet die Standardeinstellung `Scalable`.

Kategorie: Optional

Standard: Ist vom Wert von `Maximum_primaries` abhängig.

Einstellbar: Nein

`RG_name` (`Zeichenkette`)
Der Name der Ressourcengruppe. Dieser Name muss im Cluster einmalig sein.

Kategorie: Erforderlich

Standard: Kein Standard

Einstellbar: Nein

`RG_project_name` (`Zeichenkette`)
Der Solaris-Projektname, der dieser Ressourcengruppe zugeordnet ist. Diese Eigenschaft wird verwendet, um Solaris-Ressourcenverwaltungsfunktionen wie CPU-Anteile und Ressourcen-Pools auf Cluster-Datendienste anzuwenden. Wenn RGM Ressourcengruppen online bringt, werden die entsprechenden Prozesse unter diesem Projektnamen gestartet, wenn für die Ressourcen die Eigenschaft `Resource_project_name` nicht eingestellt ist. Der angegebene Projektname muss in der Projektdatenbank vorhanden sein, und der Benutzer `root` muss als Mitglied des benannten Projekts konfiguriert sein.

Diese Eigenschaft wird nur in Solaris 9 oder höher unterstützt.

Hinweis – Änderungen an dieser Eigenschaft werden beim nächsten Starten der Ressource wirksam.

Kategorie: Optional

Standard: Die Textzeichenkette "`default`"

Einstellbar: `ANYTIME`

`RG_state` auf jedem Cluster-Knoten (`Aufzählung`)
Wird von RGM auf `Unmanaged`, `Online`, `Offline`, `Pending_online`, `Pending_offline`, `Pending_online_blocked` , `Error_stop_failed`, `Online_faulted` oder `Pending_online_blocked` eingestellt, um den Status der Gruppe auf jedem Cluster-Knoten zu beschreiben.

Sie können diese Eigenschaft nicht konfigurieren. Sie können diese Eigenschaft jedoch indirekt festlegen, indem Sie `scswitch`(1M) aufrufen oder die äquivalenten `scsetup` (1M)- oder SunPlex Manager-Befehle verwenden.

Kategorie: Nur-Abfrage

Standard: Kein Standard

Einstellbar: Nein

`RG_system` (`Boole'scher Wert`)
Wenn die `RG_system`-Eigenschaft für eine Ressourcengruppe auf `TRUE` eingestellt ist, sind bestimmte Vorgänge für die Ressourcengruppe und die darin enthaltenen Ressourcen beschränkt. Diese Beschränkung soll eine versehentliche Änderung oder Löschung wichtiger Ressourcengruppen und Ressourcen verhindern. Nur `scrgadm` (1M)- und `scswitch`(1M)-Befehle sind von dieser Eigenschaft betroffen. Vorgänge für `scha_control` (1HA) und `scha_control`(3HA) sind nicht betroffen.

Vor Ausführung eines beschränkten Vorgangs für eine Ressourcengruppe (oder für die Ressourcen einer Ressourcengruppe) müssen Sie zuerst die `RG_system`-Eigenschaft der Ressourcengruppe auf `FALSE` einstellen. Gehen Sie beim Ändern oder Löschen einer Ressourcengruppe, die Cluster-Dienste unterstützt, oder beim Ändern oder Löschen von Ressourcen in solchen Ressourcengruppen sorgfältig vor .

Eine Ressourcengruppe, deren `RG_system`-Wert auf `TRUE` eingestellt ist, wird als Systemressourcengruppe bezeichnet. Die Bearbeitung der `RG_system`-Eigenschaft selbst ist nie beschränkt, unabhängig vom aktuellen Wert von `RG_system`. Eine detaillierte Beschreibung dieser Beschränkungen finden Sie in der Online-Dokumentation unter `rg_properties` (5).

Kategorie: Optional

Standard: `FALSE`

Einstellbar: Ja

Ressourceneigenschaftsattribute

Die folgende Informationen beschreiben die Ressourceneigenschaftsattribute, die für das Ändern systemdefinierter Eigenschaften bzw. das Erstellen von Erweiterungseigenschaften verwendet werden können.

Achtung – `Null` oder die leere Zeichenkette (“”) können nicht als Standardwert für `boolean`-, `enum`- oder `int`-Typen angegeben werden.

Die Eigenschaftsnamen werden zuerst angezeigt, gefolgt von einer Beschreibung.

`Array_maxsize`
Für den `stringarray`-Typ die zulässige Höchstzahl von Array-Elementen.

`Array_minsize`
Für den `stringarray`-Typ die zulässige Mindestanzahl von Array-Elementen.

`Default`
Gibt einen Standardwert für die Eigenschaft an.

`Description`
Eine Zeichenkettenanmerkung, die eine kurze Beschreibung der Eigenschaft gibt. Das `Description`-Attribut kann in der RTR-Datei nicht für systemdefinierte Eigenschaften eingestellt werden.

`Enumlist`
Für einen `enum`-Typ ein Satz von Zeichenkettenwerten, die für die Eigenschaft zulässig sind.

`Extension`
Die Verwendung gibt an, dass der RTR-Dateieintrag eine Erweiterungseigenschaft deklariert, die durch die Ressourcentypimplementierung definiert wurde. Andernfalls ist der Eintrag eine systemdefinierte Eigenschaft.

`Max`
Für einen `int`-Typ der zulässige Höchstwert für die Eigenschaft.

`Maxlength`
Für `string`- und `stringarray`-Typen die zulässige Höchstlänge der Zeichenkette.

`Min`
Für einen `int`-Typ der zulässige Mindestwert für die Eigenschaft.

`Minlength`
Für `string`- und `stringarray`-Typen die zulässige Mindestlänge der Zeichenkette.

`Property`
Der Name der Ressourceneigenschaft.

`Tunable`
Gibt an, wann der Cluster-Verwalter den Wert dieser Eigenschaft in einer Ressource einstellen kann. Kann auf `NONE` oder `FALSE` eingestellt werden, damit der Verwalter die Eigenschaft nicht einstellen kann. Werte, die dem Verwalter das Festlegen von Einstellungen ermöglichen, sind `TRUE` bzw. `ANYTIME` (jederzeit), `AT_CREATION` (nur bei Erstellung der Ressource) oder `WHEN_DISABLED` (wenn

die Ressource offline ist). Zur Einrichtung anderer Einstellbarkeitsbedingungen, beispielsweise “bei deaktivierter Überwachung” oder “wenn offline”, stellen Sie dieses Attribut auf `ANYTIME` ein und validieren Sie den Status der Ressource mit der `Validate`-Methode.

Der Standardwert ist für jede Standardressourceneigenschaft verschieden, wie im folgenden Abschnitt zu sehen ist. Der Standardwert für die Einstellbarkeit einer Erweiterungseigenschaft ist `TRUE` (`ANYTIME`), falls in der RTR-Datei nichts anderes festgelegt ist.

Typ der Eigenschaft
Zulässige Typen sind: `string`, `boolean`, `int`, `enum` und `stringarray`. Das type-Attribut kann in der RTR-Datei nicht für systemdefinierte Eigenschaften eingestellt werden. Der Typ legt akzeptable Eigenschaftswerte und die typspezifischen Attribute fest, die in dem RTR-Dateieintrag zulässig sind. Ein `enum`-Typ ist ein Satz von Zeichenkettenwerten.

ANHANG **B**

Codeauflistungen für Beispieldatendienste

Dieser Anhang enthält den vollständigen Code für jede Methode im Beispieldatendienst. Daneben listet er den Inhalt der Ressourcentyp-Registrierungsdatei auf.

Der Anhang enthält die folgenden Codeauflistungen.

Auflistung der Ressourcentyp-Registrierungsdatei

Die RTR-Datei (Resource Type Registration, Ressourcentypregistrierung) enthält die Deklarationen der Ressourcen- und Ressourcentypeigenschaften, welche die anfängliche Konfiguration des Datendienstes zu dem Zeitpunkt definieren, an dem der Cluster-Verwalter den Datendienst registriert.

BEISPIEL B–1 `SUNW.Sample`-RTR-Datei

```
#
# Copyright (c) 1998-2004 by Sun Microsystems, Inc.
# All rights reserved.
```

BEISPIEL B–1 SUNW.Sample-RTR-Datei *(Fortsetzung)*

```
#
# 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;

# 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;
```

BEISPIEL B–1 SUNW.Sample-RTR-Datei *(Fortsetzung)*

```
}
{
        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;
}

# 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;
```

BEISPIEL B–1 SUNW.Sample-RTR-Datei *(Fortsetzung)*

```
        TUNABLE = AT_CREATION;
        DEFAULT = "";
}

#
# 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)";
}
```

Start-Methode

RGM ruft die Start-Methode auf einem Cluster-Knoten auf, wenn die Ressourcengruppe mit der Datendienstressource auf diesem Knoten online gebracht wird oder wenn die Ressource aktiviert wird. In der Beispielanwendung aktiviert die Start-Methode den in.named (DNS)-Dämon auf diesem Knoten.

BEISPIEL B–2 dns_svc_start-Methode

```
#!/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
```

BEISPIEL B–2 dns_svc_start-Methode *(Fortsetzung)*

```
# 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.


#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
```

BEISPIEL B–2 dns_svc_start-Methode *(Fortsetzung)*

```
# 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

# 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 \
```

BEISPIEL B–2 dns_svc_start-Methode *(Fortsetzung)*

```
    /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] \
           "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Stop-Methode

Die Stop-Methode wird auf einem Cluster-Knoten aufgerufen, wenn die Ressourcengruppe mit der HA-DNS-Ressource auf diesem Knoten offline gebracht bzw. deaktiviert wird. Diese Methode stoppt den in.named (DNS)-Dämon auf dem Knoten.

BEISPIEL B–3 dns_svc_stop-Methode

```
#!/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
```

BEISPIEL B–3 dns_svc_stop-Methode *(Fortsetzung)*

```
                        ;;
                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=$RESOURCE_NAME.named
SYSLOG_TAG=$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] \
```

BEISPIEL B–3 dns_svc_stop-Methode *(Fortsetzung)*

```
          "${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.
      pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL"

          exit 1
      fi
fi
else
   # 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
```

gettime-Dienstprogramm

Das gettime-Dienstprogramm ist ein C-Programm, das vom PROBE-Programm für das Verfolgen der Zeit verwendet wird, die zwischen den Neustarts des Testsignals verstreicht. Sie müssen dieses Programm kompilieren und im gleichen Verzeichnis wie die Rückmeldemethoden ablegen, also in dem Verzeichnis, auf das die RT_basedir -Eigenschaft zeigt.

BEISPIEL B–4 gettime.c-Dienstprogramm

```
#
# 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"
```

BEISPIEL B–4 `gettime.c`-Dienstprogramm *(Fortsetzung)*

```
#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
{
    printf("%d\n", time(0));
    exit(0);
}
```

PROBE-Programm

Das PROBE-Programm prüft die Verfügbarkeit des Datendienstes mithilfe von `nslookup`(1M)-Befehlen. Die Rückmeldemethode `Monitor_start` startet dieses Programm, und die Rückmeldemethode `Monitor_start` stoppt es.

BEISPIEL B–5 `dns_probe`-Programm

```
#!/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)
```

BEISPIEL B–5 dns_probe-Programm *(Fortsetzung)*

```
                        # 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
}

#############################################################################
# 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 \
```

BEISPIEL B–5 dns_probe-Programm *(Fortsetzung)*

```
                        -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
                # 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
```

BEISPIEL B–5 dns_probe-Programm *(Fortsetzung)*

```
      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
         # 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 "$@"
```

BEISPIEL B–5 dns_probe-Programm *(Fortsetzung)*

```
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 \
-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
```

BEISPIEL B–5 dns_probe-Programm *(Fortsetzung)*

```
   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
   # /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
```

Monitor_start-Methode

Diese Methode startet das PROBE-Programm für den Datendienst.

BEISPIEL B–6 dns_monitor_start-Methode

```
#!/bin/ksh
#
# Monitor start Method for HA-DNS.
```

BEISPIEL B–6 dns_monitor_start-Methode *(Fortsetzung)*

```
#
# 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.

#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`
```

BEISPIEL B–6 dns_monitor_start-Methode *(Fortsetzung)*

```
# 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 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
```

Monitor_stop-Methode

Diese Methode stoppt das PROBE-Programm für den Datendienst.

BEISPIEL B–7 dns_monitor_stop-Methode

```
#!/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
```

BEISPIEL B–7 dns_monitor_stop-Methode *(Fortsetzung)*

```
                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

# 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
```

BEISPIEL B–7 dns_monitor_stop-Methode *(Fortsetzung)*

```
fi
exit 0
```

Monitor_check-Methode

Diese Methode überprüft das Vorhandensein des Verzeichnisses, auf das die Confdir-Eigenschaft zeigt. RGM ruft jedesmal Monitor_check auf, wenn die PROBE-Methode ein Failover des Datendienstes auf einen neuen Knoten ausführt, und prüft auch Knoten, die potenzielle Master sind.

BEISPIEL B–8 dns_monitor_check-Methode

```
#!/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
      ;;
```

BEISPIEL B–8 dns_monitor_check-Methode *(Fortsetzung)*

```
      *)
      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 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] \
```

BEISPIEL B–8 dns_monitor_check-Methode *(Fortsetzung)*

```
      "${ARGV0} Monitor check for DNS successful."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS not successful."
   exit 1
fi
```

Validate-Methode

Diese Methode überprüft das Vorhandensein des Verzeichnisses, auf das die Confdir-Eigenschaft zeigt. RGM ruft diese Methode auf, wenn der Datendienst erstellt wird und wenn der Cluster-Verwalter Datendiensteigenschaften aktualisiert. Die Monitor_check-Methode ruft diese Methode immer dann auf, wenn der Fehler-Monitor ein Failover des Datendienstes auf einen neuen Knoten ausführt.

BEISPIEL B–9 dns_validate-Methode

```
#!/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 ...]
```

BEISPIEL B–9 dns_validate-Methode *(Fortsetzung)*

```
{
   typeset opt

   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
                       ;;
```

BEISPIEL B–9 dns_validate-Methode *(Fortsetzung)*

```
                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        "ERROR: Option $OPTARG unknown"
                        exit 1
                        ;;
                esac
   done
}

###############################################################################
# MAIN
#
###############################################################################

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

# 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
```

BEISPIEL B–9 dns_validate-Methode *(Fortsetzung)*

```
fi

# Check that the named.conf file is present in the Confdir directory.
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
```

Update-Methode

RGM ruft die Update-Methode auf, um eine laufende Ressource von der Änderung ihrer Eigenschaften zu benachrichtigen.

BEISPIEL B–10 dns_update-Methode

```
#!/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.
```

BEISPIEL B–10 dns_update-Methode *(Fortsetzung)*

```
                        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

# 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
```

BEISPIEL B–10 dns_update-Methode *(Fortsetzung)*

```
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                  "${ARGV0} Could not restart monitor for HA-DNS "
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully restarted"
   fi
fi
exit 0
```

ANHANG **C**

Auflistung von Beispielen für DSDL-Ressourcentypcode

Dieser Anhang listet den vollständigen Code für jede Methode im `SUNW.xfnts`-Ressourcentyp auf. Er enthält die Auflistung für `xfnts.c` mit Code für die Unterroutinen, die von den Rückmeldemethoden aufgerufen werden. Die Codeauflistungen in diesem Anhang sind folgende:

`xfnts.c`

Diese Datei implementiert die Unterroutinen, die von den `SUNW.xfnts`-Methoden aufgerufen werden.

BEISPIEL C–1 `xfnts.c`

```
/*
 * Copyright (c) 1998-2004 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
 * to probe the health of the data service. The probe just returns either
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
 * 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


/*
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
 * svc_validate():
 *
 * 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) {
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
      /*
       * 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) {
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
      scds_syslog(LOG_ERR,
          "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;
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
   scha_err_t   err;

   /* 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():
 *
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
 * Stop the XFS server
 * 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) {
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
      scds_syslog(LOG_ERR,
          "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);
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
      }

   /* 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>.
 */
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
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
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
 * 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,
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
            "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.
     */
```

BEISPIEL C–1 xfnts.c *(Fortsetzung)*

```
   (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);
}
```

xfnts_monitor_check-Methode

Diese Methode überprüft, ob die Basiskonfiguration des Ressourcentyps gültig ist.

BEISPIEL C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2004 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 */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
```

BEISPIEL C–2 xfnts_monitor_check.c *(Fortsetzung)*

```
{
      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);
}
```

xfnts_monitor_start-Methode

Diese Methode startet die xfnts_probe-Methode.

BEISPIEL C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2004 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.
 */

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

BEISPIEL C–3 xfnts_monitor_start.c *(Fortsetzung)*

```
   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);
}
```

xfnts_monitor_stop-Methode

Diese Methode stoppt die xfnts_probe-Methode.

BEISPIEL C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2004 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.
 */

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

BEISPIEL C–4 xfnts_monitor_stop.c *(Fortsetzung)*

```
   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);
}
```

xfnts_probe-Methode

Die xfnts_probe-Methode prüft die Verfügbarkeit der Anwendung und entscheidet, ob ein Failover ausgeführt oder der Datendienst neu gestartet wird. Die Rückmeldemethode xfnts_monitor_start startet dieses Programm, und die Rückmeldemethode xfnts_monitor_stop stoppt es.

BEISPIEL C–5 xfnts_probe.c+

```
/*
 * Copyright (c) 1998-2004 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>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"
```

BEISPIEL C–5 xfnts_probe.c+ *(Fortsetzung)*

```
/*
 * 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
    * 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 (;;) {
```

BEISPIEL C–5 xfnts_probe.c+ *(Fortsetzung)*

```
/*
 * 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();

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

   /*
    * Compute failure history and take
    * action if needed
    */
   (void) scds_fm_action(scds_handle,
```

BEISPIEL C–5 `xfnts_probe.c+` *(Fortsetzung)*

```
            probe_result, (long)dt);
      }   /* Each net resource */
   }    /* Keep probing forever */
}
```

xfnts_start-Methode

RGM ruft die `Start`-Methode auf einem Cluster-Knoten auf, wenn die Ressourcengruppe mit der Datendienstressource auf diesem Knoten online gebracht wird oder wenn die Ressource aktiviert wird. Die `xfnts_start`-Methode aktiviert den `xfs`-Dämon auf diesem Knoten.

BEISPIEL C–6 `xfnts_start.c`

```
/*
 * Copyright (c) 1998-2004 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;
   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);
```

BEISPIEL C–6 `xfnts_start.c` *(Fortsetzung)*

```
   }

   /* 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);
}
```

Die `xfnts_stop`-Methode

RGM ruft die `Stop`-Methode auf einem Cluster-Knoten auf, wenn die Ressourcengruppe mit der HA-XFS-Ressource auf diesem Knoten offline gebracht wird oder wenn die Ressource deaktiviert wird. Diese Methode stoppt den `xfs`-Dämon auf diesem Knoten.

BEISPIEL C–7 `xfnts_stop.c`

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
```

BEISPIEL C–7 `xfnts_stop.c` *(Fortsetzung)*

```
 * 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);
}
```

Die `xfnts_update`-Methode

RGM ruft die `Update`-Methode auf, um eine laufende Ressource von der Änderung ihrer Eigenschaften zu benachrichtigen. Das Programm ruft `Update` auf, nachdem eine Verwaltungsaktion erfolgreich die Eigenschaften einer Ressource bzw. deren Gruppe eingestellt hat.

BEISPIEL C–8 `xfnts_update.c`

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
```

BEISPIEL C–8 xfnts_update.c *(Fortsetzung)*

```
 * Copyright (c) 1998-2004 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.
    */

   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);
```

BEISPIEL C–8 `xfnts_update.c` *(Fortsetzung)*

```
   return (0);
}
```

Codeauflistung für die `xfnts_validate`-Methode

Diese Methode überprüft das Vorhandensein des Verzeichnisses, auf das die `Confdir_list`-Eigenschaft zeigt. RGM ruft diese Methode auf, wenn der Datendienst erstellt wird und wenn der Cluster-Verwalter Datendiensteigenschaften aktualisiert. Die `Monitor_check`-Methode ruft diese Methode immer dann auf, wenn der Fehler-Monitor ein Failover für den Datendienst auf einen neuen Knoten ausführt.

BEISPIEL C–9 `xfnts_validate.c`

```
/*
 * Copyright (c) 1998-2004 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.
 */

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);
```

BEISPIEL C–9 `xfnts_validate.c` *(Fortsetzung)*

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

   /* Return the result of validate method */
   return (rc);

}
```

ANHANG **D**

Zulässige RGM-Namen und -Werte

Dieser Anhang listet die Anforderungen für zulässige Zeichen in RGM-Namen und -Werten (Resource Group Manager) auf.

Gültige Namen für RGM

RGM-Namen werden in die folgenden Kategorien unterteilt:

- Ressourcengruppennamen
- Ressourcentypnamen
- Ressourcennamen
- Eigenschaftsnamen
- Aufzählungsliteralnamen

Regeln für alle Namen mit Ausnahme der Ressourcentypnamen

Mit Ausnahme der Ressourcentypnamen müssen alle Namen folgenden Regeln entsprechen:

- Sie müssen in ASCII geschrieben sein.
- Sie müssen mit einem Buchstaben beginnen.
- Sie können Groß- und Kleinbuchstaben, Ziffern, Bindestriche (-) und Unterstriche (_) enthalten.
- Sie dürfen nicht mehr als 255 Zeichen enthalten.

Format von Ressourcentypnamen

Das Format des vollständigen Namen eines Ressourcentyps ist auf folgende Weise vom Ressourcentyp abhängig:

- Wenn die Ressourcentyp-Registrierungsdatei (RTR-Datei) des Ressourcentyps die Anweisung `#$upgrade` enthält, lautet das Format wie folgt:

  Hersteller-ID.*Basis-RT-Name*:*Version*

- If the resource type's RTR file does *not* contain the `#$upgrade` directive, the format is as follows:

  Hersteller-ID.*Basis-RT-Name*

Ein Punkt dient als Trennzeichen zwischen *Hersteller-ID* und *Basis-RT-Name*. Ein Strichpunkt dient als Trennzeichen zwischen *Basis-RT-Name* und *Version*.

Die variablen Elemente in diesem Format sind:

Hersteller-ID	Gibt das Hersteller-ID-Präfix an. Das Hersteller-ID-Präfix ist der Wert der Ressourcentypeigenschaft `Vendor_id` in der RTR-Datei.
Basis-RT-Name	Gibt den Basis-Ressourcentypnamen an. Der Basis-Ressourcentypname ist der Wert der Ressourcentypeigenschaft `Resource_type` in der RTR-Datei.
Version	Gibt das Versionssuffix an. Das Versionssuffix ist der Wert der Ressourcentypeigenschaft `RT_version` in der RTR-Datei. Das Versionssuffix ist *nur* Teil des vollständigen Ressourcentypnamens, wenn die RTR-Datei die Anweisung `#$upgrade` enthält. Die Anweisung `#$upgrade` wurde in Version 3.1 von Sun Cluster eingeführt.

Hinweis – Wenn nur eine Version eines Basis-Ressourcentypnamens registriert ist, müssen Sie in `scrgadm` (1M)-Befehlen nicht den vollständigen Namen verwenden. Sie können das Hersteller-ID-Präfix, das Versionsnummernsuffix oder beides gleichzeitg weglassen.

Weitere Informationen zu den Ressourcentypeigenschaften finden Sie unter „Ressourcentypeigenschaften" auf Seite 253.

BEISPIEL D–1 Vollständiger Name eines Ressourcentyps mit der Anweisung `#$upgrade`

Dieses Beispiel zeigt den vollständigen Namen eines Ressourcentyps, für den die Eigenschaften in der RTR-Datei wie folgt festgelegt sind:

- `Vendor_id=SUNW`
- `Resource_type=sample`
- `RT_version=2.0`

BEISPIEL D–1 Vollständiger Name eines Ressourcentyps mit der Anweisung `#$upgrade` *(Fortsetzung)*

Der vollständige Name des in dieser RTR-Datei definierten Ressourcentyps lautet:

```
SUNW.sample:2.0
```

BEISPIEL D–2 Vollständiger Name eines Ressourcentyps ohne die Anweisung `#$upgrade`

Dieses Beispiel zeigt den vollständigen Namen eines Ressourcentyps, für den die Eigenschaften in der RTR-Datei wie folgt festgelegt sind:

- `Vendor_id=SUNW`
- `Resource_type=nfs`

Der vollständige Name des in dieser RTR-Datei definierten Ressourcentyps lautet:

```
SUNW.nfs
```

RGM-Werte

RGM-Werte werden in zwei Kategorien unterteilt: Eigenschaftswerte und Beschreibungswerte. Beide Kategorien teilen sich dieselben Regeln. Diese lauten wie folgt:

- Werte müssen in ASCII geschrieben sein.
- Die Höchstlänge eines Wertes ist 4 MB minus 1, also 4.194.303 Byte.
- Werte dürfen keines der folgenden Zeichen enthalten:
  - Null
  - Zeilenumbruch
  - Komma
  - Strichpunkt

ANHANG **E**

Anforderungen für Anwendungen ohne Cluster-Unterstützung

Eine gewöhnliche Anwendung ohne Cluster-Unterstützung muss bestimmte Anforderungen erfüllen, um als hoch verfügbare Anwendung (HA-Anwendung) eingesetzt zu werden. Abschnitt „Analysieren der Eignung einer Anwendung" auf Seite 29 listet diese Anforderungen auf. Dieser Anhang enthält weitere Einzelheiten zu bestimmten Elementen in der Liste.

Eine Anwendung wird hoch verfügbar, indem ihre Ressourcen in Ressourcengruppen konfiguriert werden. Die Anwendungsdaten werden in einem hoch verfügbaren globalen Dateiystem abgelegt, in dem ein noch betriebsbereiter Server auf sie zugreifen kann, falls ein anderer Server ausfällt. Informationen über Cluster-Dateisysteme finden Sie im *Sun Cluster Concepts Guide for Solaris OS*.

Für den Netzwerkzugriff durch Clients im Netzwerk wird eine logische Netzwerk-IP-Adresse in logischen Hostnamenressourcen konfiguriert, die sich in derselben Ressourcengruppe wie die Datendienstressource befindet. Die Datendienstressource und die Netzwerkadressressourcen führen das Failover gemeinsam aus. Anschlie[00df]end greifen die Netzwerk-Clients des Datendienstes auf die Datendienstressource auf ihrem neuen Host zu.

Multihost-Daten

Die Geräte der hoch verfügbaren globalen Dateisysteme haben mehrere Hosts (Multihost). Wenn also ein realer Host abstürzt, kann einer der noch funktionsfähigen Hosts auf das Gerät zugreifen. Damit eine Anwendung hoch verfügbar sein kann, müssen auch ihre Daten hoch verfügbar sein, sich also in den globalen HA-Dateisystemen befinden.

Das globale Dateisystem wird in Gerätegruppen eingehängt, die als unabhängige Einheiten erstellt werden. Sie können festlegen, dass einige Gerätegruppen als eingehängte globale Dateisysteme verwendet werden und andere als im raw-Modus betriebene Geräte für die Verwendung mit einem Datendienst wie HA Oracle.

Eine Anwendung kann über Befehlszeilenschalter oder Konfigurationsdateien verfügen, die auf den Speicherort der Datendateien zeigen. Wenn die Anwendung fest verdrahtete Pfadnamen verwendet, können Sie die Pfadnamen in symbolische Verknüpfungen ändern, die auf die Dateien in einem globalen Dateisystem zeigen, ohne dabei den Anwendungscode zu ändern. Weitere Einzelheiten zur Verwendung von symbolischen Verknüpfungen finden Sie unter „Verwenden von symbolischen Verknüpfungen für Multihost-Datenablage" auf Seite 336.

Im schlimmsten Fall muss der Anwendungsquellcode geändert werden, um einen Mechanismus bereitzustellen, der auf den tatsächlichen Datenspeicherort zeigt. Eine Möglichkeit hierfür ist das Implementieren zusätzlicher Befehlszeilenschalter.

Sun Cluster unterstützt die Verwendung von UNIX® UFS-Dateisystemen und im raw-Modus betriebenen HA-Geräten, die in einem Datenträger-Manager konfiguriert werden. Bei der Installation und Konfiguration muss der Systemverwalter angeben, welche Plattenressourcen für UFS-Dateisysteme bzw. für im raw-Modus betriebene Geräte verwendet werden. In der Regel werden im raw-Modus betriebene Geräte nur von Datenbank- und Multimedia-Servern verwendet.

Verwenden von symbolischen Verknüpfungen für Multihost-Datenablage

In manchen Fällen sind die Pfadnamen der Datendateien einer Anwendung fest verdrahtet, und es ist kein Mechanismus zum Übersteuern dieser fest verdrahteten Pfadnamen vorhanden. Um Änderungen am Anwendungscode zu vermeiden, können manchmal symbolische Verknüpfungen verwendet werden.

Beispiel: Angenommen, die Anwendung benennt ihre Datendateien mit dem fest verdrahteten Pfadnamen `/etc/mydatafile`. Diesen Pfad können Sie in eine symbolische Verknüpfung ändern, deren Wert auf eine Datei in einem der Dateisysteme des logischen Hosts zeigt. Sie können zum Beispiel eine symbolische Verknüpfung mit `/global/phys-schost-2/mydatafile` herstellen.

Bei der Verwendung von symbolischen Verknüpfungen kann jedoch ein Problem auftreten, wenn die Anwendung bzw. eines ihrer Verwaltungsverfahren neben dem Inhalt auch den Namen der Datendatei ändert. Angenommen, die Anwendung nimmt zunächst eine Aktualisierung vor, indem sie eine neue temporäre Datei `/etc/mydatafile.new` erstellt. Dann benennt sie die temporäre Datei mit dem Systemaufruf `rename(2)` bzw. dem Programm `mv(1` um, um ihr den echten Dateinamen zu geben. Durch das Erstellen der temporären Datei mit anschließender Umbenennung in die echte Datei versucht der Datendienst sicherzustellen, dass der Datendateiinhalt immer richtig gebildet wird.

Leider wird die symbolische Verknüpfung jedoch durch die `rename(2)`-Aktion zerstört. Der Name `/etc/mydatafile` ist nun eine reguläre Datei in demselben Dateisystem wie das `/etc`-Verzeichnis, nicht im globalen Dateisystem des Clusters. Da das `/etc`-Dateisystem für jeden Host privat ist, stehen die Daten nach einem Failover bzw. Switchover nicht zur Verfügung.

Das zugrunde liegende Problem in diesem Fall ist, dass die vorhandene Anwendung die symbolische Verknüpfung nicht wahrnimmt. Sie wurde nicht im Hinblick auf symbolische Verknüpfungen geschrieben. Wenn symbolische Verknüpfungen für die Umleitung des Datenzugriffs auf die Dateisysteme des logischen Hosts verwendet werden sollen, muss sich die Anwendungsimplementierung so verhalten, dass sie die symbolischen Verknüpfungen nicht löscht. Symbolische Verknüpfungen sind also keine völlig zufriedenstellende Lösung für das Problem, wie Daten im globalen Cluster-Dateisystem abgelegt werden können.

Hostnamen

Sie müssen festlegen, ob Situationen möglich sind, in denen der Datendienst den Hostnamen des Servers kennen muss, auf dem er ausgeführt wird. Wenn dies der Fall ist, muss der Datendienst möglicherweise dahingehend geändert werden, dass er anstelle des realen Hostnamens einen logischen Hostnamen verwendet (das heißt, einen Hostnamen, der in eine logische Hostnamenressource konfiguriert wurde, die in derselben Ressourcengruppe wie die Anwendungsressource residiert).

Manchmal gibt der Server im Client/Server-Protokoll für einen Datendienst als Teil des Meldungsinhalts an den Client seinen eigenen Hostnamen zurück. Bei diesen Protokollen kann es sein, dass der Client von diesem zurückgegebenen Hostnamen abhängt, ihn also für die Verbindungsherstellung mit dem Server verwenden muss. Damit der zurückgegebene Hostname auch nach einem Failover bzw. Switchover noch verwendet werden kann, muss es sich um einen logischen Hostnamen der Ressourcengruppe handeln, und nicht um den Namen des realen Hosts. In diesem Fall müssen Sie den Datendienstcode dahingehend ändern, dass der logische Hostname an den Client zurückgegeben wird.

Multihomed Hosts

Der Begriff *Multihomed Host* beschreibt einen Host, der sich in mehr als einem öffentlichen Netzwerk befindet. Ein solcher Host hat mehrere Hostnamen und IP-Adressen. Für jedes Netzwerk verfügt er über ein Hostname-IP-Adressenpaar. Sun Cluster ist dafür ausgelegt, das Vorhandensein eines Hosts in einer beliebigen Anzahl von Netzwerken zuzulassen, einschließlich einem einzigen Netzwerk (der Nicht-Multihomed-Fall). Genau wie der reale Hostname über mehrere Hostname-IP-Adressenpaare verfügt, kann jede Ressourcengruppe mehrere Hostnamen-IP-Adressenpaare haben, also eines für jedes öffentliche Netzwerk. Wenn Sun Cluster eine Ressourcengruppe von einem realen Host auf einen anderen verschiebt, wird gleichzeitig der vollständige Satz Hostname-IP-Adressenpaare für diese Ressourcengruppe mit verschoben.

Der Satz Hostname-IP-Adressenpaare für eine Ressourcengruppe wird als logische Hostnamenressource innerhalb der Ressourcengruppe konfiguriert. Diese Netzwerkadressressourcen werden vom Systemverwalter beim Erstellen und Konfigurieren der Ressourcengruppe angegeben. Die Sun Cluster-Datendienst-API bietet Möglichkeiten zum Abfragen dieser Hostname-IP-Adressenpaare.

Die meisten handelsüblichen Datendienst-Dämone, die für das Solaris-Betriebssystem geschrieben wurden, verarbeiten Multihomed Hosts bereits korrekt. Viele Datendienste führen ihre gesamte Netzwerkkommunikation durch Binden an die Solaris-Platzhalteradresse `INADDR_ANY` durch. Durch das Binden wird automatisch bewirkt, dass die Datendienste alle IP-Adressen für alle Netzwerkschnittstellen verarbeiten. `INADDR_ANY` sorgt für eine effektive Bindung an alle aktuell auf dem Rechner konfigurierten IP-Adressen. Bei einem Datendienst, der `INADDR_ANY` verwendet, ist im Allgemeinen keine Änderung erforderlich, um die logischen Netzwerkadressen von Sun Cluster verarbeiten zu können.

Binden an `INADDR_ANY` im Vergleich zu Binden an spezifische IP-Adressen

Auch wenn Nicht-Multihomed Hosts verwendet werden, ermöglicht das Sun Cluster-Konzept der logischen Netzwerkadressen dem Rechner, mit mehr als einer IP-Adresse zu arbeiten. Der Rechner verfügt über eine IP-Adresse für seinen eigenen realen Host und über weitere IP-Adressen für jede Netzwerkadressressource (logische Hostname-Ressource), die er aktuell unterstützt. Wenn ein Rechner als Master einer Netzwerkadressressource eingesetzt wird, erhält er dynamisch weitere IP-Adressen. Sobald er nicht mehr Master einer Netzwerkadressressource ist, gibt er die IP-Adressen dynamisch wieder auf.

Einige Datendienste funktionieren in einer Sun Cluster-Umgebung nicht richtig, wenn sie an `INADDR_ANY` gebunden sind. Solche Datendienste müssen den Satz der IP-Adressen, an die sie gebunden sind, dynamisch ändern, sobald die Ressourcengruppe unterstützt bzw. nicht mehr unterstützt wird. Eine Möglichkeit zum Erzielen dieser Neubindung besteht darin, dass die Start- und Stop-Methoden dieser Datendienste das Stoppen der Datendienst-Dämone erzwingen und sie neu starten.

Die Ressourceneigenschaft `Network_resources_used` ermöglicht dem Endbenutzer das Konfigurieren eines spezifischen Satzes Netzwerkressourcen, an die eine Anwendungsressource zu binden ist. Für Ressourcentypen, bei denen diese Funktion erforderlich ist, muss die Eigenschaft `Network_resources_used` in der entsprechenden RTR-Datei deklariert werden.

Wenn RGM die Ressourcengruppe online bringt bzw. offline nimmt, folgt er einer bestimmten Reihenfolge für das Anmelden, Abmelden und Aktiv- bzw. Inaktiv-Konfigurieren im Zusammenhang mit dem Aufruf der Datendienst-Ressourcenmethoden. Weitere Einzelheiten finden Sie unter „Bestimmen der zu verwendenden `Start`- und `Stop`-Methoden" auf Seite 47.

Bei Rückgabe der `Stop`-Methode muss der Datendienst die Verwendung der Netzwerkadressen der Ressourcengruppe eingestellt haben. Analog dazu muss der Datendienst bei Rückgabe der `Start`-Methode mit der Verwendung der Netzwerkadressen begonnen haben.

Wenn der Datendienst anstelle von einzelnen IP-Adressen an `INADDR_ANY` gebunden wird, spielt die Reihenfolge, in der die Datendienstressourcenmethoden und die Netzwerkadressmethoden aufgerufen werden, keine Rolle mehr.

Wenn die Stop- und Start-Methoden ihre Aufgabe ausführen, indem sie das Stoppen der Datendienst-Dämone erzwingen und diese neu starten, beendet bzw. beginnt der Datendienst die Verwendung der Netzwerkadressen zu den richtigen Zeitpunkten.

Client-Wiederholversuch

Für einen Netzwerk-Client wirkt ein Failover bzw. Switchover wie ein Absturz des logischen Hosts, gefolgt von einem schnellen Neustart. Im Idealfall sind die Client-Anwendung und das Client/Server-Protokoll so strukturiert, dass mehrere Wiederholversuche unternommen werden. Wenn die Anwendung und das Protokoll bereits den Absturz und Neustart eines einzelnen Servers unterstützen, können sie auch Takeover bzw. Switchover der Ressourcengruppe unterstützen. Einige Anwendungen unternehmen möglicherweise eine endlose Anzahl an Wiederholversuchen. Komplexere Anwendungen benachrichtigen den Benutzer darüber, dass ein lang andauernder Wiederholvorgang läuft, so dass der Benutzer entscheiden kann, ob er fortfährt oder den Prozess abbricht.

ANHANG **F**

Dokumenttypdefinitionen für CRNP

Dieser Anhang führt die DTDs (Document Type Definitions, Dokumenttypdefinitionen) für CRNP (Cluster Reconfiguration Notification Protocol) auf.

SC_CALLBACK_REG XML DTD

Hinweis – Die NVPAIR-Datenstruktur, die sowohl von SC_CALLBACK_REG als auch von SC_EVENT verwendet wird, wird nur einmal definiert.

```
<!— SC_CALLBACK_REG XML format specification
        Copyright 2001-2004 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
>
```

NVPAIR-XML-DTD

```
<!— NVPAIR XML format specification

        Copyright 2001-2004 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)>
```

SC_REPLY-XML-DTD

```
<!— SC_REPLY XML format specification

        Copyright 2001-2004 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)>
```

SC_EVENT-XML-DTD

Hinweis – Die NVPAIR-Datenstruktur, die sowohl von SC_CALLBACK_REG als auch von SC_EVENT verwendet wird, wird nur einmal definiert.

```
<!— SC_EVENT XML format specification

        Copyright 2001-2004 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 sysseventd 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
>
```

ANHANG **G**

`CrnpClient.java`-Anwendung

Dieser Anhang enthält die vollständige `CrnpClient.java`-Anwendung, auf die in Kapitel 12 näher eingegangen wird.

Inhalt von `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

E

F

G

H

N

O

P

Q

R

S

T

U

V

W

X

Z