Sun Cluster 3.1 10/03: Guía del desarrollador de los servicios de datos

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

Referencia: 817–4265–10
Noviembre 2003, Revisión A

040109@7518

Contenido

Prefacio

El manual *Sun Cluster 3.1 10/03: Guía del desarrollador de los servicios de datos* contiene información sobre la utilización de la API de gestión de recursos para desarrollar los servicios de datos de Sun Cluster.

Quién debe utilizar este manual

Este documento está destinado a desarrolladores experimentados que conozcan perfectamente el software y el hardware de Sun. La información de este manual presupone un conocimiento del sistema operativo Solaris™.

Organización de este manual

Sun Cluster 3.1 10/03: Guía del desarrollador de los servicios de datos incluye los siguientes capítulos y apéndices:

- El Capítulo 1 proporciona información general sobre los conceptos necesarios para desarrollar un servicio de datos.
- El Capítulo 2 proporciona información detallada sobre cómo desarrollar un servicio de datos.
- El Capítulo 4 proporciona una referencia a las funciones de acceso y métodos de llamada que componen la API de la gestión de recursos (RMAPI).
- El Capítulo 5 proporciona un servicio de datos Sun Cluster de ejemplo para la aplicación `in.named()`.

- El Capítulo 6 proporciona información general de las interfaces de programación de aplicaciones que forman la Biblioteca de desarrollo de los servicios de datos (DSDL)
- El Capítulo 7 explica la utilización habitual de DSDL para diseñar e implementar tipos de recursos.
- El Capítulo 8 describe un tipo de recurso de ejemplo, implementado con DSDL.
- El Capítulo 9 describe SunPlex Agent Builder.
- El Capítulo 10 explica cómo crear un servicio genérico de datos.
- El Capítulo 11 describe las funciones de la API de DSDL.
- El Apéndice A describe el tipo de recursos estándar, el grupo y las propiedades de recursos.
- El Apéndice B proporciona el código completo de cada método del servicio de datos de ejemplo.
- El Apéndice C enumera el código completo de cada método del tipo de recursos `SUNW.xfnts()`.
- El Apéndice D enumera los requisitos de los caracteres legales de los nombres y valores del Gestor de grupos de recursos (RGM).
- El Apéndice E enumera los requisitos de las aplicaciones ordinarias independientes del clúster que puedan ofrecer una alta disponibilidad.

Documentación relacionada

Aplicación	Título	Número de referencia
Conceptos	*Sun Cluster 3.1: Guía de conceptos*	817-4259-10
Instalación del software	*Sun Cluster 3.1 10/03: Guía de instalación del software*	817-4253-10
Administración	*Sun Cluster 3.1 10/03: Guía de administración del sistema*	817-4247-10
Desarrollo de API	*Sun Cluster 3.1 10/03: Guía del desarrollador de los servicios de datos*	817-4265-10
Mensajes de error	*Sun Cluster 3.1 10/03 Error Messages Guide*	817-0521
Hardware	*Sun Cluster 3.x Hardware Administration Manual*	817-0168
	Sun Cluster 3.x Hardware Administration Collection en `http://docs.sun.com/db/coll/1024.1`	

Aplicación	Título	Número de referencia
Servicios de datos	*Sun Cluster 3.1 Data Service Planning and Administration Guide*	817-3305
	Sun Cluster 3.1 Data Services 10/03 Collection en `http://docs.sun.com/db/coll/573.10`	
Páginas de comando man	*Sun Cluster 3.1 10/03 Reference Manual*	817–0522
Notas sobre la versión	*Sun Cluster 3.1 10/03: Notas sobre la versión*	817–4854
	Sun Cluster 3.x Release Notes Supplement	816-3381

Obtención de ayuda

Si tiene problemas durante la instalación o utilización de Sun Cluster, póngase en contacto con su proveedor de servicios y déle la información siguiente:

- Su nombre y dirección de correo electrónico (si estuviera disponible)
- El nombre, dirección y número de teléfono de su empresa
- Los modelos y números de serie de sus sistemas
- El número de versión del sistema operativo; por ejemplo Solaris 10
- El número de versión de Sun Cluster (por ejemplo, Sun Cluster 3.1)

Utilice los comandos siguientes para recopilar información de su sistema para el proveedor de asistencia técnica:

Comando	Función
`prtconf -v`	Muestra el tamaño de la memoria del sistema y ofrece información sobre los dispositivos periféricos
`psrinfo -v`	Muestra información acerca de los procesadores
`showrev -p`	Indica las modificaciones instaladas
`prtdiag -v`	Muestra información de diagnóstico del sistema
`/usr/cluster/bin/scinstall -pv`	Muestra información sobre la versión y el paquete de Sun Cluster.

Tenga también a punto el contenido del archivo `/var/adm/messages`.

Acceso a la documentación de Sun en línea

La sede web docs.sun.com[SM] permite acceder a la documentación técnica de Sun en línea. Puede explorar el archivo docs.sun.com, buscar el título de un manual o un tema específicos. El URL es `http://docs.sun.com`.

Convenciones tipográficas

La tabla siguiente describe los cambios tipográficos utilizados en este manual.

TABLA P–1 Convenciones tipográficas

Tipo de letra o símbolo	Significado	Ejemplo
`AaBbCc123`	Nombres de los comandos, archivos y directorios; la salida por pantalla del computador	Modifique el archivo `.login`. Utilice el comando `ls -a` para mostrar todos los archivos. `nombre_sistema% tiene correo.`
`AaBbCc123`	Lo que usted escribe, contrastado con la salida por pantalla del computador	`nombre_máquina%` **`su`** `Password:`
AaBbCc123	Plantilla de línea de comandos: sustitúyala por un nombre o valor real	El comando necesario para eliminar un archivo es `rm` *nombrearchivo*.
AaBbCc123	Títulos de los manuales, palabras o términos nuevos o palabras destacables	Consulte el capítulo 6 de la *Guía del usuario*. Éstas se denominan opción de *clase*. *No* guarde el archivo.

Indicadores de los shells en ejemplos de comandos

La tabla siguiente muestra los indicadores predeterminados del sistema y de superusuario para los shells Bourne, Korn y C.

TABLA P–2 Indicadores de los shell

Shell	Indicador
Indicador del shell C	`nombre_sistema%`
Indicador de superusuario en el shell C	`nombre_sistema#`
Indicador de los shells Bourne y Korn	`$`
Indicador de superusuario en los shell Bourne y Korn	`#`

CAPÍTULO 1

Información general sobre la gestión de recursos

Este manual proporciona directrices para crear un tipo de recurso para una aplicación de software como Oracle, Sun™ One Web Server, DNS, etc; está destinado a desarrolladores de tipos de recursos.

Este capítulo proporciona información general sobre los conceptos que es necesario comprender para desarrollar un servicio de datos e incluye la información siguiente.

Nota – Este manual utiliza los términos *tipo de recursos* y *servicios de datos* indistintamente. El término *agente*, aunque se usa pocas veces en este manual, equivale a *tipo de recurso* y *servicio de datos*.

Entorno de aplicaciones de Sun Cluster

El sistema Sun Cluster permite ejecutar y administrar las aplicaciones como recursos escalables y de alta disponibilidad. El recurso del clúster conocido como Gestor de grupos de recursos, o RGM, proporciona el mecanismo de alta disponibilidad y escalabilidad. Los elementos que componen la interfaz de programación de este recurso son los siguientes.

- Un conjunto de métodos de rellamada que permiten que RGM controle una aplicación del clúster.

- API de gestión de recursos (RMAPI), un conjunto de comandos y funciones de API de bajo nivel que se pueden utilizar para escribir métodos de rellamada y que se implementan en la biblioteca `libscha.so`.
- Los recursos de gestión de procesos para supervisar y reiniciar procesos en el clúster.
- Biblioteca de desarrollo del servicio de datos (DSDL), un conjunto de funciones de biblioteca que encapsula la API de bajo nivel y la función de gestión de procesos a un nivel superior y agrega ciertas funciones adicionales para facilitar la escritura de los métodos de rellamada. Estas funciones se implementan en la biblioteca `libdsdev.so`.

La figura siguiente muestra las relaciones entre estos elementos.

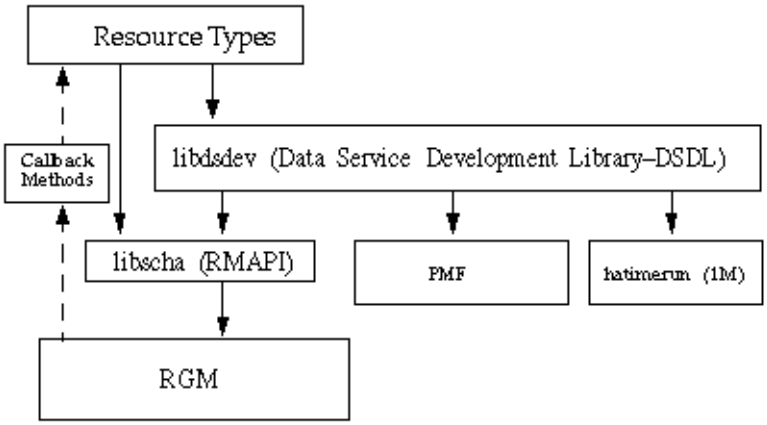


FIGURA 1–1 Arquitectura de programación

En el paquete Sun Cluster se incluye SunPlex Agent Builder™, una herramienta que automatiza el proceso de creación de un servicio de datos (consulte el Capítulo 9). Agent Builder genera código de servicio de datos en C (se emplean las funciones de DSDL para escribir los métodos de rellamada) o en el shell Korn (`ksh`) (con comandos de API de bajo nivel para escribir los métodos de rellamada).

RGM se ejecuta como daemon en cada nodo del clúster e inicia y detiene automáticamente los recursos en los nodos seleccionados, de acuerdo con políticas preconfiguradas; hace que un recurso tenga una alta disponibilidad en caso de fallo de un nodo o reinicio, deteniendo el recurso del nodo afectado e iniciándolo en otro; además, inicia y detiene automáticamente supervisores específicos de recursos, que pueden detectar fallos de los recursos y reubicar éstos en otros nodos o pueden supervisar otros aspectos del rendimiento de los recursos.

RGM admite recursos a prueba de fallos, que pueden estar en línea en un sólo nodo en cada momento, y recursos escalables, que pueden estar en línea en varios nodos simultáneamente.

Modelo de RGM

Esta sección introduce cierta terminología fundamental y explica con más detalle el Gestor de grupos de recursos y sus interfaces asociadas.

RGM maneja tres tipos principales de objetos interrelacionados: tipos de recursos, recursos y grupos de recursos. Un ejemplo, que se explica a continuación, nos puede servir para presentar estos objetos.

Un desarrollador implementa un tipo de recurso, `ha-oracle`, que proporciona a una aplicación Oracle DBMS una alta disponibilidad. Un usuario final define bases de datos diferentes para márqueting, ingeniería y finanzas, cada una de las cuales es un recurso de tipo `ha-oracle`. El administrador del clúster pone estos recursos en grupos de recursos separados para que puedan funcionar en nodos diferentes y realizar una operación de recuperación de fallos independiente. Un desarrollador crea un segundo tipo de de recurso, ha-calendar, para implementar un servidor de agendas de alta disponibilidad que exige una base de datos Oracle. El administrador del clúster pone el recurso de la agenda financiera en el mismo grupo de recursos que el recurso de la base de datos financiera para que ambos se ejecuten en el mismo nodo y realicen operaciones de recuperación de fallos conjuntamente.

Tipos de recursos

Un tipo de recurso consiste en una aplicación de software que se va a ejecutar en el clúster, programas de control que RGM utiliza como métodos de rellamada para gestionar la aplicación como un recurso de clúster y un conjunto de propiedades que forman parte de la configuración estática de un clúster. RGM utiliza propiedades de tipo de recurso para gestionar los recursos de un tipo concreto.

Nota – Además de una aplicación de software, un tipo de recurso puede representar otros recursos de sistema, como direcciones de red.

El desarrollador de los tipos de recursos especifica las propiedades de éstos y fija sus valores en un archivo de registro de tipo de recurso (RTR) que sigue un formato bien definido, descrito en «Establecimiento del recurso y las propiedades del tipo de

recurso» en la página 34 y en la página de comando man `rt_reg`(4). Consulte también «Definición del archivo de registro del tipo de recurso» en la página 90 si desea ver una descripción de un archivo de registro de tipo de recurso.

La Tabla A–1 enumera las propiedades de los tipos de recursos.

El administrador del clúster instala y registra la implementación del tipo de recurso y la aplicación subyacente en un clúster. El procedimiento de registro introduce en la configuración del clúster la información del archivo de registro del tipo de recurso. *Sun Cluster 3.1 Data Service Planning and Administration Guide* describe el procedimiento de registro de un servicio de datos.

Recursos

Un recurso hereda las propiedades y valores de su tipo de recurso. Además, un desarrollador puede declarar propiedades de recurso en el archivo de registro del tipo de recurso. Consulte la Tabla A–2 para ver una lista de las propiedades de recursos.

El administrador del clúster puede cambiar los valores de ciertas propiedades, en función de cómo se hayan especificado en el archivo de registro del tipo de recurso (RTR). Por ejemplo, las definiciones de propiedades pueden especificar un rango de valores permitidos y especificar cuándo se puede ajustar la propiedad: en el momento de la creación, en cualquier momento o nunca. Dentro de estas especificaciones, el administrador del clúster puede cambiar las propiedades con comandos de administración.

El administrador del clúster puede crear muchos recursos del mismo tipo, cada uno con su propio nombre y conjunto de valores de propiedad, de modo que se pueda ejecutar más de una instancia de la aplicación subyacente en el clúster. Cada instalación requiere un nombre único dentro del clúster.

Grupos de recursos

Cada recurso se debe configurar en un grupo de recursos. RGM pone todos los recursos de un grupo en línea y fuera de línea simultáneamente en el mismo nodo, para ello, invoca métodos de rellamada de los recursos individuales del grupo.

Los nodos en los que un grupo de recursos está actualmente en línea se denominan sus *principales* o *nodos principales*. Un grupo de recursos está *controlado* por cada uno de sus principales. Cada grupo de recursos tiene una propiedad `Nodelist` asociada, fijada por el administrador del clúster, que identifica todos los *principales potenciales* o maestros del grupo de recursos.

Un grupo de recursos tiene también un conjunto de propiedades que incluye propiedades de configuración que puede establecer el administrador de clúster y propiedades dinámicas que fija RGM y reflejan el estado activo del grupo de recursos.

RGM define dos tipos de grupos de recursos, a prueba de fallos y escalables; los primeros pueden estar en línea sólo en un nodo en cada momento, en tanto que los segundos pueden estar en línea en varios nodos simultáneamente. RGM proporciona un conjunto de propiedades para respaldar la creación de cada tipo de grupo de recursos. Consulte «Transferencia de un servicio de datos a un clúster» en la página 34 y «Implementación de los métodos de rellamada» en la página 43 para obtener más detalles sobre estas propiedades.

Consulte la Tabla A–3 para ver una lista de las propiedades de grupos de recursos.

Gestor de grupos de recursos

El Gestor de grupos de recursos (RGM) se implementa como daemon, `rgmd`, que se ejecuta en cada nodo miembro del clúster. Todos los procesos `rgmd` se comunican entre sí y actúan juntos como un único recurso que ocupa todo el clúster.

RGM admite las siguientes funciones:

- Siempre que un nodo se inicia o se produce una caída de sistema, RGM intenta mantener la disponibilidad de todos los grupos de recursos, poniéndolos automáticamente en línea en los maestros apropiados.
- Si falla un recurso determinado, el programa de supervisión puede solicitar que se reinicie el grupo de recursos en el mismo maestro o que se conmute a otro nuevo.
- El administrador del clúster puede emitir un comando administrativo para solicitar una de las acciones siguientes:
  - Cambiar el maestro de un grupo de recursos
  - Habilitar o inhabilitar un recurso determinado de un grupo de recursos
  - Crear, suprimir o modificar un recurso, grupo de recursos o tipo de recurso

Siempre que RGM activa cambios en la configuración, coordina las acciones en todos los nodos del clúster. Este tipo de actividad se denomina reconfiguración. Para provocar un cambio de estado en un recurso individual, RGM invoca un método de rellamada específico del tipo de recurso en ese recurso.

Métodos de rellamada

La estructura Sun Cluster utiliza un mecanismo de rellamada para proporcionar comunicación entre un servicio de datos y RGM. La estructura define un conjunto de métodos de rellamada, que incluye los argumentos, los valores de retorno y las circunstancias en las que RGM llama a cada método.

Un servicio de datos se crea mediante la codificación de métodos de rellamada individuales y la implementación de cada método como un programa de control al que RGM puede llamar. Es decir, el servicio de datos no consiste en un único ejecutable, sino en varias secuencias ejecutables (`ksh`) o binarios (C), cada uno de los cuales se puede llamar directamente desde RGM.

Los métodos de rellamada se registran en RGM mediante el archivo de registro del tipo de recurso (RTR) en el que se identifica el programa para cada método implementado para el servicio de datos. Cuando un administrador del sistema registra el servicio de datos en un clúster, RGM lee el archivo RTR, el cual proporciona la identidad de los programas de rellamada, entre otros datos.

Los únicos métodos de rellamada necesarios para un tipo de recurso son un método de inicio (`Start` o `Prenet_start`) y un método de parada (`Stop` o `Postnet_stop`).

Los métodos de rellamada se pueden agrupar en las categorías siguientes:

- Métodos de control e inicialización
  - `Start` y `Stop` inician y detienen los recursos en un grupo que se está poniendo en línea o fuera de línea.
  - `Init`, `Fini` y `Boot` ejecutan el código de inicialización y finalización en los recursos.
- Métodos de soporte administrativo
  - `Validate` verifica las propiedades que se han establecido en una acción administrativa.
  - `Update` actualiza la configuración de la propiedad de un recurso en línea.
- Métodos relacionados con la red
  - `Prenet_start` y `Postnet_stop` realizan acciones especiales de encendido y apagado antes de que se asignen las direcciones de red del mismo grupo de recursos, o después de que se hayan desasignado.
- Métodos de control de supervisión
  - `Monitor_start` y `Monitor_stop` inician o detienen la supervisión de un recurso.
  - `Monitor_check` evalúa la fiabilidad de un nodo antes de poner un grupo de recursos en él.

Consulte el Capítulo 4 y la página de comando man `rt_callbacks`(1HA) para obtener más información sobre los métodos de rellamada. Consulte también el Capítulo 5 y el Capítulo 8 para conocer los métodos de rellamada de los servicios de datos de ejemplo.

Interfaces de programación

Para escribir código de servicio de datos, la arquitectura de gestión de recursos proporciona una API de bajo nivel, o básica, una biblioteca de nivel superior, creada sobre la API básica, y una herramienta, SunPlex Agent Builder, que genera automáticamente un servicio de datos a partir de las entradas básicas que se aportan.

RMAPI

RMAPI (API de la gestión de recursos) proporciona un conjunto de rutinas de bajo nivel que permiten que los servicios de datos accedan a información sobre los recursos, tipos y grupos de recursos del sistema, soliciten un reinicio o una recuperación de fallos local y fijen el estado del recurso. A estas funciones se accede a través de la biblioteca `libscha.so`. RMAPI proporciona estos métodos de rellamada en forma de comandos de shell y de funciones de C. Consulte `scha_calls`(3HA) y el Capítulo 4 para obtener más información sobre las rutinas de RMAPI. Consulte también el Capítulo 5 si desea ver ejemplos sobre cómo usar estas rutinas en métodos de rellamada de servicios de datos de ejemplo.

Biblioteca de desarrollo del servico de datos (DSDL)

Sobre RMAPI se crea DSDL, que proporciona una estructura integrada de nivel superior, que conserva al mismo tiempo el modelo de rellamada a método subyacente de RGM. DSDL reúne varias funciones para el desarrollo de servicios de datos, por ejemplo:

- `libscha.so`: API de gestión de recursos de bajo nivel
- PMF: la función de gestión de procesos, que proporciona una forma de supervisar los procesos y sus descendientes, y de reiniciarlos si se terminan (consulte `pmfadm`(1M) y `rpc.pmfd`(1M)).
- `hatimerun`: una función para ejecutar programas en un tiempo de espera determinado (consulte `hatimerun`(1M).

Para la mayoría de las aplicaciones, DSDL proporciona casi todas las funciones necesarias para crear un servicio de datos, si no todas. Tenga presente, sin embargo, que DSDL no sustituye la API de bajo nivel, sino que la encapsula y la amplía. De hecho, muchas funciones de DSDL llaman a las funciones de `libscha.so`. Del mismo modo, es posible llamar directamente a las funciones `libscha.so` mientras se usa DSDL para codificar el grueso del servicio de datos. La biblioteca `libdsdev.so` contiene las funciones de DSDL.

Consulte el Capítulo 6 y la página de comando man de `scha_calls`(3HA) para obtener más información sobre DSDL.

SunPlex Agent Builder

Agent Builder es una herramienta que automatiza la creación de un servicio de datos. Tras la introducción de la información básica sobre la aplicación objetivo y el servicio de datos que se va a crear, Agent Builder genera un servicio de datos, junto con el código fuente y ejecutable (shells C o Korn), un archivo RTR personalizado y un paquete Solaris.

Con la mayoría de las aplicaciones es posible usar Agent Builder para generar un servicio de datos completo, apenas con unos cambios mínimos. Las aplicaciones con requisitos más complejos, como añadir comprobaciones de validación de propiedades adicionales, pueden requerir un trabajo que Agent Builder no es capaz realizar. Sin embargo, aun en estos casos se puede utilizar para generar el grueso del código y codificar el resto manualmente. Como mínimo, Agent Builder se puede utilizar para generar el paquete Solaris.

Interfaz administrativa del gestor de grupos de recursos

Sun Cluster proporciona una interfaz gráfica de usuario y un conjunto de comandos para administrar un clúster.

SunPlex Manager

SunPlex Manager es una herramienta basada en web que permite realizar las tareas siguientes:

- Instalar un clúster
- Administrar un clúster
- Crear y configurar recursos y grupos de recursos
- Configurar servicios de datos con el software Sun Cluster

Consulte *Sun Cluster 3.1 10/03 Software Installation Guide* para obtener instrucciones sobre cómo instalar SunPlex Manager y cómo usarlo para instalar el software del clúster. SunPlex Manager proporciona ayuda en línea para las tareas administrativas más exclusivas.

Comandos administrativos

Los comandos de Sun Cluster para administrar objetos de RGM son `scrgadm`(1M), `scswitch`(1M) y `scstat`(1M) `-g`.

El comando `scrgadm` permite ver, crear, configurar y suprimir el tipo, el grupo y los objetos de recursos que utiliza RGM. El comando forma parte de la interfaz administrativa del clúster y no se debe utilizar en el mismo contexto de programación que la interfaz de aplicaciones descrita en este capítulo. Sin embargo, `scrgadm` es la herramienta que sirve para construir la configuración del clúster en la que opera la API. Comprender la interfaz administrativa permite conocer la interfaz de aplicaciones. Consulte la página de comando man `scrgadm`(1M) para obtener detalles sobre las tareas administrativas que puede realizar el comando.

El comando `scswitch` pone los grupos de recursos en línea y fuera de línea en los nodos especificados y habilita o inhabilita un recurso o su supervisor. Consulte la página de comando man `scswitch`(1M) para obtener detalles sobre las tareas administrativas que puede realizar el comando.

El comando `scstat -g` muestra el estado dinámico actual de todos los grupos de recursos y recursos.

CAPÍTULO 2

Desarrollo de un servicio de datos

Este capítulo proporciona información detallada sobre cómo desarrollar un servicio de datos.

En este capítulo se tratan los temas siguientes:

Análisis de la validez de la aplicación

El primer paso en la creación de un servicio de datos es determinar si la aplicación de destino satisface los requisitos para poder tener una alta disponibilidad y escalabilidad. Si la aplicación no cumple todos los requisitos, es posible que se pueda modificar el código fuente de la aplicación para que los cumpla.

La siguiente lista resume los requisitos que debe cumplir una aplicación para poder tener una alta disponibilidad y escalabilidad. Si necesita más detalles o tiene que modificar el código fuente de la aplicación, consulte el Apéndice B.

Nota – Un servicio escalable debe cumplir todas las condiciones que se indican a continuación para ofrecer una alta disponibilidad, además de ciertos criterios adicionales.

- Tanto las aplicaciones habilitadas para red (modelo cliente-servidor) como las no habilitadas para red (sin cliente) son candidatas potenciales para convertirse en aplicaciones de alta disponibilidad y escalabilidad en el entorno Sun Cluster. Sin embargo, Sun Cluster no puede proporcionar una mayor disponibilidad en entornos en los que hay un reparto de tiempo, en los que las aplicaciones se ejecutan en un servidor al que se accede mediante `telnet` o `rlogin`.
- La aplicación debe ser tolerante a caídas del sistema. Es decir, debe recuperar datos del disco (si fuera necesario) cuando se reinicie después de una terminación de nodo inesperada. Además, el tiempo de recuperación tras una caída del sistema debe ser limitado. La tolerancia a caídas del sistema es un prerrequisito para hacer que una aplicación tenga una alta disponibilidad, porque la capacidad de recuperar el disco y reiniciar la aplicación es un problema que afecta a la integridad de los datos. El servicio de datos no es necesario para poder recuperar las conexiones.
- La aplicación no debe depender del nombre físico de servidor del nodo en el que se ejecuta. Consulte «Nombres de sistema» en la página 321 para obtener información adicional.
- La aplicación debe funcionar correctamente en entornos en los que se asignan múltiples direcciones IP; por ejemplo, entornos con sistemas principales con varios directorios iniciales, en los que el nodo está en más de una red pública y los entornos con nodos en los que se asignan múltiples interfaces lógicas en una sola interfaz de hardware.
- Para tener una alta disponibilidad, los datos de la aplicación deben residir en el sistema de archivos del clúster; consulte «Datos de sistemas múltiples» en la página 319.

  Si la aplicación utiliza un nombre de ruta invariable para la ubicación de los datos, es posible cambiar la ruta por un enlace simbólico que señale hacia una ubicación del sistema de archivos del clúster, sin cambiar el código fuente de la aplicación. Consulte «Utilización de vínculos simbólicos para la colocación de datos de varios sistemas» en la página 320 para obtener información adicional.
- Las bibliotecas y binarios de aplicación pueden residir localmente, en cada nodo, o en el sistema de archivos del clúster. La ventaja de residir en el sistema de archivos del clúster es que basta una única instalación. La desventaja es que el despliegue de las actualizaciones se vuelve problemático, porque los binarios están en uso mientras se ejecuta la aplicación bajo el control de RGM.

- El cliente debería tener cierta capacidad para reintentar automáticamente una consulta si se agota el tiempo de espera del primer intento. Si la aplicación y el protocolo ya pueden manejar el caso de la caída y reinicio de un único servidor, también podrán encargarse de que se produzca una recuperación de fallos o una conmutación del grupo de recursos que los contiene. Consulte «Reintento del cliente» en la página 323 para obtener información adicional.
- La aplicación no debe tener zócalos de dominio Unix ni conducciones con nombres en el sistema de archivos del clúster.

Además, los servicios escalables deben cumplir los requisitos siguientes.

- La aplicación debe poder ejecutarse en varias instancias, que trabajen, todas ellas, con los mismos datos de aplicación del sistema de archivos de clúster.
- La aplicación debe ofrecer coherencia de los datos para un acceso simultáneo desde múltiples nodos.
- La aplicación debe implementar un bloqueo suficiente, con un mecanismo visible desde cualquier punto, como el sistema de archivos de clúster.

Para un servicio escalable, las características de la aplicación determinan también la política de equilibrio de cargas. Por ejemplo, la política de equilibrio de cargas `LB_WEIGHTED`, que permite que cualquier instancia responda a las solicitudes del cliente, no funciona para una aplicación que utilice una memoria caché integrada en el servidor para las conexiones de cliente. En este caso, se debe especificar una política de equilibrio de cargas que restrinja el tráfico de un cliente determinado a una instancia de la aplicación. Las políticas de equilibrio de cargas `LB_STICKY` y `LB_STICKY_WILD` envían repetidamente todas las solicitudes de un cliente a la misma instancia de la aplicación, siempre que puedan utilizar una memoria caché integrada. Tenga presente que si hay múltiples solicitudes de cliente, provenientes de clientes distintos, RGM distribuye las solicitudes entre las instancias del servicio. Consulte «Implementación de un recurso a prueba de fallos» en la página 51 para obtener más información sobre el establecimiento de la política de equilibrio de cargas de los servicios de datos escalables.

Elección de la interfaz

El paquete de soporte del desarrollador de Sun Cluster (`SUNWscdev`) proporciona dos conjuntos de interfaces de codificación de los métodos de los servicios de datos:

- API de la gestión de recursos (RMAPI), un conjunto de rutinas de bajo nivel (en la biblioteca `libscha.so`)
- Biblioteca de desarrollo de los servicios de datos (DSDL), un conjunto de funciones de nivel superior (en la biblioteca `libdsdev.so`) que encapsula la función de RMAPI y proporciona funciones adicionales

El paquete de soporte del desarrollador de Sun Cluster incluye también SunPlex Agent Builder, una herramienta que automatiza la creación de un servicio de datos.

El enfoque recomendado para desarrollar un servicio de datos es:

1. Decidir si se va a codificar con el shell C o el Korn. Si opta por el shell Korn, no podrá utilizar la DSDL, que proporciona sólo una interfaz para C.
2. Ejecutar Agent Builder, especificar las entradas solicitadas y generar un servicio de datos, que incluya código fuente y ejecutable, un archivo RTR y un paquete.
3. Si el servicio de datos generado requiere personalización, puede agregar código de DSDL a los archivos fuente generados. Agent Builder indica, con comentarios, los lugares específicos, dentro de los archivos fuente, en los que se puede agregar un código propio.
4. Si el código requiere una personalización avanzada para admitir la aplicación objetivo, puede agregar funciones de RMAPI al código fuente.

En la práctica, es posible adoptar diferentes enfoques para crear un servicio de datos. Por ejemplo, en lugar de agregar un código propio en los puntos específicos del código generado por Agent Builder, puede sustituir completamente uno de los métodos generados o el programa de supervisión generado por un programa que se haya escrito desde cero con las funciones DSDL o RMAPI. Sin embargo, al margen de la opción elegida, prácticamente en cualquier caso es práctico utilizar Agent Builder, por las razones siguientes:

- El código que genera Agent Builder, aunque es genérico por naturaleza, se ha comprobado en muchos servicios de datos.
- Agent Builder genera un archivo RTR, un makefile, un paquete para el recurso y otros archivos de soporte para el servicio de datos. Aunque no utilice ninguno de los archivos del código del servicio de datos, utilizar los otros archivos puede ahorrar una gran cantidad de trabajo.
- Puede modificar el código generado.

Nota – A diferencia de RMAPI, que proporciona un conjunto de funciones C y un conjunto de comandos para utilizar en secuencias, DSDL proporciona sólo una interfaz de función C. Por tanto, si especifica una salida del shell Korn (`ksh`) en Agent Builder, el código fuente generado realiza llamadas a RMAPI, porque no hay comandos `ksh` de DSDL.

Configuración del entorno de desarrollo para escribir un servicio de datos

Antes de empezar el desarrollo de un servicio de datos, debe tener instalado el paquete de desarrollo de Sun Cluster (`SUNWscdev`) para tener acceso a los archivos de biblioteca y cabecera de Sun Cluster. Aunque este paquete ya está instalado en todos los nodos del clúster, generalmente se realiza el desarrollo en una máquina separada, de desarrollo sin clúster, no en un nodo del clúster. En ese caso, debe utilizar `pkgadd` para instalar el paquete `SUNWscdev` en la máquina de desarrollo.

Cuando compile y vincule el código, deberá establecer opciones concretas para identificar los archivos de cabecera y biblioteca. Una vez finalizado el desarrollo (en un nodo sin clúster), puede transferir el servicio de datos terminado a un clúster, para ejecutarlo y comprobarlo.

Nota – Asegúrese de que esté usando una versión de desarrollo de Solaris 5.8 o superior.

Utilice los procedimientos de esta sección para:

- Instalar el paquete de desarrollo de Sun Cluster (`SUNWscdev`) y establecer las opciones de compilación y vinculación adecuadas
- Transferir el servicio de datos a un clúster

▼ Configuración del entorno de desarrollo

Este procedimiento explica cómo instalar el paquete `SUNWscdev` y establecer las opciones de compilación y vinculación para el desarrollo del servicio de datos.

1. **Conviértase en superusuario o asuma un papel equivalente y cambie el directorio al directorio del CD-ROM que desee.**

   # **cd** *directorio_CD-ROM*

2. **Instale el paquete SUNWscdev en el directorio actual.**

   # **pkgadd -d . SUNWscdev**

3. **En `Makefile`, especifique las opciones de compilación y vinculación que identifican los archivos de biblioteca e inclusión del código del servicio de datos.**

   Especifique las opciones `-I`, para identificar los archivos de cabecera de Sun Cluster, `-L`, para especificar la ruta de búsqueda de la biblioteca de tiempo de

compilación en el sistema de desarrollo, y `-R`, para especificar la ruta de búsqueda de biblioteca del enlazador del tiempo de ejecución del clúster.

```
# Makefile para un servicio de datos de ejemplo
...

-I /usr/cluster/include

-L /usr/cluster/lib

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

Transferencia de un servicio de datos a un clúster

Cuando haya terminado el desarrollo de un servicio de datos en una máquina de desarrollo, debe transferirlo a un clúster para realizar comprobaciones. Para reducir las posibilidades de error, la mejor forma de realizar la transferencia es empaquetar el código del servicio de datos con el archivo RTR e instalar el paquete en todos los nodos del clúster.

Nota – Tanto si usa `pkgadd` como cualquier otro método para instalar el servicio de datos, deberá poner el servicio de datos en todos los nodos de clúster. Agent Builder empaqueta automáticamente juntos el archivo RTR y el código del servicio de datos.

Establecimiento del recurso y las propiedades del tipo de recurso

Sun Cluster proporciona un conjunto de propiedades de tipo de recurso y de recurso que se utilizan para definir la configuración estática de un servicio de datos. Las propiedades de tipo de recurso especifican el tipo de recurso y su versión, la versión de la API, etc., así como las rutas a cada uno de los métodos de rellamada. La Tabla A–1 enumera todas las propiedades del tipo de recurso.

Las propiedades de recurso, como `Failover_mode`, `Thorough_probe_interval`, y los tiempos de espera del método definen también la configuración estática del recurso. Las propiedades dinámicas del recurso, como `Resource_state` y `Status`, reflejan el estado activo de un recurso gestionado. La Tabla A–2 describe las propiedades del recurso.

Las propiedades del recurso y del tipo de recurso se declaran en el archivo de registro del tipo de recurso (RTR), que es un componente esencial del servicio de datos y que define la configuración inicial del servicio de datos en el momento en que el administrador del clúster registra el servicio de datos en Sun Cluster.

Se recomienda utilizar Agent Builder para generar el archivo RTR para el servicio de datos porque Agent Builder declara un conjunto de propiedades útiles y necesarias para cualquier servicio de datos. Por ejemplo, algunas propiedades (como `Resource_type`) se deben declarar en el archivo RTR si se quiere que el registro del servicio de datos resulte satisfactorio. Otras propiedades, aunque no sean necesarias, no estarán disponibles para el administrador del sistema si no se declaran en el archivo RTR; otras propiedades estarán disponibles, tanto si se declaran como si no, porque RGM las define y proporciona un valor predeterminado. Para evitar este nivel de complejidad, puede utilizar Agent Builder sencillamente para garantizar la creación de un archivo RTR adecuado que, más tarde, podrá editar para cambiar algún valor concreto, si fuera necesario.

En el resto de esta sección se muestra el desarrollo de un archivo RTR de ejemplo, creado por Agent Builder.

Declaración de las propiedades del tipo de recurso

El administrador del clúster no puede configurar las propiedades del tipo de recurso que el usuario declare en el archivo RTR, ya que forman parte de la configuración permanente del tipo de recurso.

Nota – Hay una propiedad del tipo de recurso, `Installed_nodes`, que el administrador del sistema puede configurar. De hecho, sólo la puede configurar éste, y no es posible declararla en el archivo RTR.

La sintaxis de las declaraciones del tipo de recurso es:

nombre_propiedad = *valor*;

Nota – RGM no diferencia entre mayúsculas y minúsculas en los nombres de las propiedades. La convención para las propiedades en los archivos RTR suministrados por Sun, salvo los nombres de métodos, es la utilización de mayúscula en el inicio del nombre y minúscula en el resto. Los nombres de métodos (y los atributos de las propiedades) contienen sólo mayúsculas.

A continuación figuran las declaraciones de tipo de recurso del archivo RTR de un servicio de datos de ejemplo (`smpl`):

```
# Plantilla de Sun Cluster Data Services Builder versión 1.0
# Información de registro y recursos para ejemplo
#
#NOTA: las palabras clave no diferencian mayúsculas y minúsculas
#se pueden usar indistintamente.
#
Resource_type = "smpl";
Vendor_id = SUNW;
RT_description = "Servicio de ejemplo de 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;
```

Consejo – Debe declarar la propiedad `Resource_type` como la primera entrada del archivo RTR, de lo contrario, el registro del tipo de recurso no será satisfactorio.

El primer conjunto de declaraciones del tipo de recurso proporciona información básica sobre el tipo de recurso, como se indica a continuación:

`Resource_type` y `Vendor_id` — Indique un nombre para el tipo de recurso. Puede especificarlo sólo con la propiedad `Resource_type` (`smpl`) o con `Vendor_id` como un prefijo con un "." que lo separe del tipo de recurso (`SUNW.smpl`), como en el ejemplo. Si utiliza `Vendor_id`, use el símbolo bursátil de la empresa que define el tipo de recurso. El nombre de éste debe ser exclusivo, dentro del clúster.

Nota – Por convención el nombre del tipo de recurso (*Resource_typeVendor_id*) se usa como nombre del paquete. Los nombres de los paquetes están limitados a nueve caracteres, por lo que se recomienda limitar el número total de caracteres en estas dos propiedades a nueve caracteres o menos, aunque RGM no imponga este límite. Agent Builder, por el contrario, genera explícitamente el nombre del paquete a partir del nombre del tipo de recurso, por lo que aplica el límite de nueve caracteres.

`Rt_version`	Identifica la versión del servicio de datos de ejemplo.
`API_version`	Identifica la versión de la API. Por ejemplo, `API_version = 2`, indica que el servicio de datos se ejecuta en Sun Cluster, versión 3.0.
`Failover = TRUE`	Indica que el servicio de datos no se puede ejecutar en un grupo de recursos que pueda estar en línea en varios nodos al mismo tiempo, es decir, especifica un servicio de datos a prueba de fallos. Consulte «Transferencia de un servicio de datos a un clúster» en la página 34 para obtener más información.
`Start`, `Stop`, `Validate`, etc.	Proporciona las rutas a los programas de métodos de rellamada correspondientes llamados por RGM. Estas rutas son relativas al directorio que especifica `RT_basedir`.

Las restantes declaraciones del tipo de recurso proporcionan información de configuración:

`Init_nodes = RG_PRIMARIES`	Especifica que RGM llama a los métodos `Init`, `Boot`, `Fini` y `Validate` sólo en los nodos que pueden controlar el servicio de datos. Los nodos especificados por `RG_PRIMARIES` son un subconjunto de todos los nodos en los que está instalado el servicio de datos. Establezca el valor en `RT_INSTALLED_NODES` para especificar que RGM llame a estos métodos en todos los nodos en los que está instalado el servicio de datos.
`RT_basedir`	Apunta a `/opt/SUNWsmpl/bin` como la ruta del directorio para completar las rutas relativas, como las rutas de los métodos de rellamada.

`Start`, `Stop`, `Validate`, etc.	Proporciona las rutas a los programas de métodos de rellamada correspondientes llamados por RGM. Estas rutas son relativas al directorio que especifica `RT_basedir`.

Declaración de las propiedades del recurso

Al igual que las propiedades del tipo de recursos, las del recurso se declaran en el archivo RTR. Por convención, las declaraciones de las propiedades del recurso van después de las del tipo de recurso en el archivo RTR. La sintaxis de las declaraciones de los recursos es un conjunto de pares de valores de atributos entre llaves:

```
{
    Atributo = Valor;
    Atributo = Valor;
               .
               .
               .
    Atributo = Valor;
}
```

Para las propiedades del recurso proporcionadas por Sun Clúster, denominadas propiedades *definidas por el sistema*, es posible modificar determinados atributos en el archivo RTR. Por ejemplo, Sun Cluster proporciona propiedades de tiempo de espera del método para cada uno de los métodos de rellamada y especifica valores predeterminados. En el archivo RTR se pueden especificar valores predeterminados diferentes.

También se pueden definir nuevas propiedades de recursos en el archivo RTR, denominadas propiedades de *extensión*, con un conjunto de atributos de propiedades suministrado por Sun Cluster. La Tabla A–4 enumera los atributos para cambiar y definir las propiedades de los recursos. Las declaraciones de las propiedades de extensión van después de las declaraciones de propiedades definidas por el sistema en el archivo RTR.

El primer conjunto de propiedades de los recursos definidas por el sistema especifica los valores de tiempo de espera de los métodos de rellamada:

```
...

# Las declaraciones de propiedades de los recursos aparecen como una lista de
# entradas entre llaves después de las declaraciones del tipo de recurso.
# La declaración del nombre de la propiedad debe ser el primer atributo
# después de la llave de apertura de una entrada de propiedad del recurso.
#
# Establezca el valor mínimo y predeterminado de los tiempos de espera
# de los métodos.
{
        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;
        }
```

El nombre de la propiedad (`PROPERTY` = *valor*) debe ser el primer atributo para cada declaración de la propiedad del recurso. Las propiedades del recurso se pueden configurar dentro de los límites que definen los atributos de propiedad del archivo RTR. Por ejemplo, el valor predeterminado para cada tiempo de espera del método del ejemplo es 300 segundos. Un administrador puede cambiar este valor; sin embargo, el valor mínimo permitido, especificado con el atributo `MIN`, es 60 segundos. Consulte la Tabla A–4 si desea ver una lista completa de los atributos de los recursos.

El siguiente conjunto de propiedades del recurso define las propiedades que tienen usos específicos en el servicio de datos.

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

```
        MAX=3600;
        DEFAULT=60;
        TUNABLE = ANYTIME;
}

# El número de reintentos que se debe realizar en un periodo determinado antes de
# concluir que no se puede iniciar correctamente la aplicación en este nodo.
{
        PROPERTY = Retry_Count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Establezca Retry_Interval como un múltiplo de 60, ya que se convierte de segundos
# a minutos, redondeando hacia arriba. Por ejemplo, un valor de 50 (segundos)
# se convierte en 1 minuto. Utilice esta propiedad para cronometrar el número de
# reintentos (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 = ;
}
```

Estas declaraciones de propiedades de los recursos agregan el atributo `TUNABLE`, que limita las ocasiones en las que el administrador del sistema puede cambiar sus valores. `AT_CREATION` significa que el administrador sólo puede especificar el valor cuando el recurso se crea y que no lo puede cambiar más tarde.

Para la mayoría de las propiedades puede aceptar los valores predeterminados como Agent Builder los genera, salvo que tenga algún motivo para cambiarlos. A continuación se incluye información sobre estas propiedades (para obtener información adicional, consulte «Propiedades de recurso» en la página 248 o la página de comando man `r_properties`(5)):

`Failover_mode`
: Indica si RGM debería reubicar el grupo de recursos o abortar el nodo en caso de un fallo de un método `Start` o `Stop`.

`Thorough_probe_interval`, `Retry_count`, `Retry_interval`
: Utilizado en el supervisor de fallos. `Tunable` es como `Anytime`, por lo que un administrador de sistema los puede ajustar si el supervisor de fallos no funciona de forma óptima.

`Network_resources_used`
: Una lista de recursos de dirección compartida o nombre lógico de sistema utilizados por el servicio de datos. Agent Builder declara esta propiedad para que un administrador del sistema pueda especificar una lista de recursos, si los hubiera, al configurar el servicio de datos.

`Scalable`
: Se fija en `FALSE` para indicar que este recurso no utiliza el recurso de conexión a red de clúster (dirección compartida). Esta configuración cuadra con la propiedad de tipo de recurso `Failover` fijada en `TRUE`, para indicar un servicio a prueba de fallos. Consulte «Transferencia de un servicio de datos a un clúster» en la página 34 y «Implementación de los métodos de rellamada» en la página 43 para obtener información adicional sobre cómo utilizar esta propiedad.

`Load_balancing_policy`, `Load_balancing_weights`
: Declara automáticamente estas propiedades; sin embargo, no tienen ninguna función en un tipo de recurso a prueba de fallos.

`Port_list`
: Identifica la lista de puertos en los que recibe el servidor. Agent Builder declara esta propiedad para que un administrador del sistema pueda especificar una lista de puertos al configurar el servicio de datos.

Declaración de las propiedades de extensión

Al final del archivo RTR de ejemplo hay propiedades de extensión, como muestra el listado siguiente

```
# Propiedades de extensión
#
```

```
# El administrador del clúster debe establecer el valor de esta propiedad para que apunte
# al directorio que contiene los archivos de configuración que utiliza la aplicación.
# Para esta aplicación, smpl, especifique la ruta del archivo de configuración de
# PXFS (generalmente, named.conf).
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "La ruta del directorio de configuración";
}

# Estas dos propiedades controlan el inicio del supervisor de fallos.
{
        PROPERTY = Monitor_retry_count;
        EXTENSION;
        INT;
        DEFAULT = 4;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Número de reinicios PMF permitidos para el supervisor de fallos.";
}
{
        PROPERTY = Monitor_retry_interval;
        EXTENSION;
        INT;
        DEFAULT = 2;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Ventana de tiempo (minutos) para reinicios del supervisor de fallos.";
}
# Valor de tiempo de espera para el análisis en segundos.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 120;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Valor de tiempo de espera para el análisis (segundos)";
}

# Nivel de supervisión de procesos subordinados para PMF (opción -C  de pmfadm).
# Default = -1 significa que no se usará la opción -C de pmfadm.
# Un valor 0 o mayor indica el nivel deseado de supervisión del
# proceso subordinado.
{
        PROPERTY = Child_mon_level;
        EXTENSION;
        INT;
        DEFAULT = -1;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Nivel de supervisión de subordinados para PMF";
}
# Código añadido por el usuario -- BEGIN VVVVVVVVVVV
# Código añadido por el usuario -- END   ^^^^^^^^^^^^
```

Agent Builder crea algunas propiedades de extensión, útiles para la mayoría de los servicios de datos:

`Confdir_list`
: Especifica la ruta al directorio de configuración de la aplicación, que es una información útil para muchas aplicaciones. El administrador del sistema puede indicar la ubicación de este directorio al configurar el servicio de datos.

`Monitor_retry_count`, `Monitor_retry_interval`, `Probe_timeout`
: Controla los reinicios del supervisor de fallos, no del daemon de servidor.

`Child_mon_level`
: Fija el nivel de supervisión que debe ejercer PMF. Consulte `pmfadm`(1M) para obtener más información.

Puede crear propiedades de extensión adicionales en la zona delimitada por los comentarios del *código añadido por el usuario*.

Implementación de los métodos de rellamada

Esta sección proporciona información relativa a la implantación de los métodos de rellamada en general.

Acceso a la información de las propiedades de los recursos y grupos de recursos

Generalmente, los métodos de rellamada requieren acceso a las propiedades del recurso. RMAPI proporciona comandos de shell y funciones de C que se pueden usar en métodos de rellamada para acceder a las propiedades de los recursos, tanto las de extensión como las definidas por el sistema. Consulte las páginas de comando man `scha_resource_get`(1HA) y `scha_resource_get`(3HA).

DSDL proporciona un conjunto de funciones de C (una para cada propiedad) para acceder a propiedades definidas por el sistema y una función para acceder a las propiedades de extensión. Consulte las páginas de comando man `scds_property_functions`(3HA) y `scds_get_ext_property`(3HA).

No se puede utilizar el mecanismo de propiedad para almacenar información de estado dinámico para un servicio de datos porque no hay funciones de API disponibles para establecer las propiedades de recursos (aparte de para establecer `Status` y `Status_msg`). En su lugar, se debe almacenar la información de estado dinámico en los archivos globales.

Nota – El administrador del clúster puede establecer determinadas propiedades de recurso con el comando `scrgadm` o, si los tiene disponibles, con un comando o una interfaz administrativos gráficos. Sin embargo, no invoque `scrgadm` desde ningún método de rellamada porque `scrgadm` falla durante la reconfiguración de clúster, es decir, cuando RGM invoca el método.

Idempotencia de métodos

En general, RGM no llama a un método más de una vez seguida en el mismo recurso y con los mismos argumentos. Sin embargo, si un método `Start` falla, RGM podría llamar a un método `Stop` en un recurso, aunque éste no se haya iniciado nunca. Del mismo modo, un daemon de recurso podría terminarse de forma autónoma y RGM podría aún invocar su método `Stop` en él. Lo mismo se aplica a los métodos `Monitor_start` y `Monitor_stop`.

Por esta causa, se debe aplicar la idempotencia a los métodos `Stop` y `Monitor_stop`. Repetidas llamadas de `Stop` o `Monitor_stop` en el mismo recurso con los mismos parámetros logran los mismos resultados que una sola llamada.

Una implicación de la idempotencia es que `Stop` y `Monitor_stop` debe devolver 0 (satisfactorio) aunque el recurso o supervisor ya esté detenido y no haya ninguna tarea que realizar.

Nota – Los métodos `Init`, `Fini`, `Boot` y `Update` debe ser también idempotentes. No es necesario que lo sea el método `Start`.

Servicio genérico de datos

Un servicio genérico de datos (GDS) es un mecanismo para hacer que las aplicaciones simples tengan una alta disponibilidad o escalabilidad, cuando se conectan al marco del gestor de grupos de recursos de Sun Cluster. Este mecanismo no requiere codificación de un agente, que es el método habitual para dotar una aplicación de alta disponibilidad y escalabilidad.

El modelo GDS se basa en un tipo de recurso compilado previamente, SUNW.gds, para interactuar con el marco de RGM

Consulte el Capítulo 10 para obtener información adicional.

Control de una aplicación

Los métodos de rellamada permiten que RGM controle el recurso subyacente (aplicación) cuando los nodos estén uniéndose o separándose del clúster.

Inicio y parada de un recurso

Una implementación de un tipo de recurso requiere, como mínimo, un método `Start` y un método `Stop`. RGM invoca programas del método del tipo de recurso en los momentos pertinentes y en los nodos adecuados para poner los grupos de recursos en línea y fuera de línea. Por ejemplo, tras la caída de un nodo del clúster, RGM mueve todos los nodos que ese nodo controla a uno nuevo. Debe implementar un método `Start` para que RGM pueda reiniciar cada recurso en el nodo principal superviviente.

No se debe devolver un método `Start` hasta que se haya iniciado el recurso y esté disponible en el nodo local. Asegúrese de que en los métodos `Start` se haya especificado un tiempo suficiente para los tipos de recursos que requieren un periodo de inicialización prolongado (establezcalos valores predeterminado y mínimo para la propiedad `Start_timeout` en el archivo de registro de tipo de recurso).

Debe implementar un método `Stop` para situaciones en las que RGM ponga un grupo de recursos fuera de línea. Por ejemplo, suponga que un grupo de recursos se pone fuera de línea en el Nodo1 y se pone en línea en el Nodo2. Mientras pone el grupo de recursos fuera de línea, RGM llama al método `Stop` para que los recursos del grupo detengan todas sus actividades en el Nodo1. Después de que los métodos `Stop` de todos los recursos se hayan terminado en el Nodo1, RGM vuelve a poner en línea el grupo de recursos en el Nodo2.

Un método `Stop` no se debe devolver hasta que el recurso haya detenido por completo toda su actividad en el nodo local y se haya apagado totalmente. La manera más segura de implementar un método `Stop` es dar tiempo a que todos los procesos del nodo local relacionados con el recurso se completen. Para los que requieren mucho tiempo para apagarse se debería establecer un tiempo de espera suficientemente prolongado en su método `Stop`. Establezca la propiedad `Stop_timeout` en el archivo de registro del tipo de recurso.

Cuando un método `Stop` no es satisfactorio o su tiempo de espera se agota, el grupo de recursos entra en un estado de error que requiere la intervención del operador. Para evitar esto, las implementaciones de los métodos `Stop` y `Monitor_stop` deberían intentar la recuperación de cualquier condición de error posible. Idealmente, estos métodos deberían salir con un estado de error 0 (satisfactorio), una vez que se haya detenido satisfactoriamente toda la actividad del recurso y su supervisor en el nodo local.

Elección de los métodos `Start` y `Stop` que se van a utilizar

Esta sección ofrece algunos consejos sobre cuándo se deben utilizar los métodos `Start` y `Stop` en lugar de los métodos `Prenet_start` y `Postnet_stop`. Debe conocer perfectamente el protocolo de conexión de red de cliente-servidor del servicio de datos y del cliente para decidir qué métodos se deben emplear.

Es posible que los servicios que utilizan recursos de dirección de red necesiten que se realicen las operaciones de inicio o parada en un orden determinado, dependiendo de la configuración de dirección de nombre lógico de servidor. Los métodos de rellamada opcionales `Prenet_start` y `Postnet_stop` permiten que la implementación de un tipo de recurso realice acciones especiales de encendido y apagado antes y después de asignar o desasignar las direcciones de red del mismo grupo de recursos.

RGM invoca métodos para conectar (pero no configurar) las direcciones de red antes de invocar el método `Prenet_start` de los servicios de datos. RGM invoca métodos para desconectar las direcciones de red, después de invocar los métodos `Postnet_stop` de los servicios de datos. RGM pone un grupo de recursos en línea con la siguiente secuencia de acciones:

1. Conectar direcciones de red.
2. Invocar el método `Prenet_start` de los servicios de datos (si lo hubiera).
3. Configurar las direcciones de red.
4. Invocar el método `Start` de los servicios de datos (si lo hubiera).

El proceso para poner un recurso fuera de línea es el contrario:

1. Invocar el método `Stop` del servicio de datos (si lo hubiera).
2. Desconfigurar las direcciones de red.
3. Invocar el método `Postnet_stop` de los servicios de datos (si lo hubiera).
4. Desconectar las direcciones de red.

En el proceso de decidir cuál de los métodos `Start`, `Stop`, `Prenet_start` o `Postnet_stop` se va a usar, se ha de tener en cuenta primero el lado del servidor. Cuando se pone en línea un grupo de recursos que contiene recursos de aplicación del servicio de datos y de dirección de red, RGM invoca métodos para configurar las direcciones de red antes de llamar a los métodos `Start` del recurso del servicio de datos. Por tanto, si un servicio de datos requiere que se configuren las direcciones de red en el momento del inicio, utilice el método `Start` para iniciar el servicio de datos.

Del mismo modo, al poner fuera de línea un grupo de recursos que contenga recursos del servicio de datos y de direcciones de red, RGM invoca métodos para desconfigurar las direcciones de red después de invocar las métodos `Stop` del recurso del servicio de datos. Por tanto, si un servicio de datos requiere que se desconfiguren direcciones de red en el momento de la parada, utilice el método `Stop` para detener el servicio de datos.

Por ejemplo, para iniciar o detener un servicio de datos, es posible que haya que invocar las bibliotecas o utilidades administrativas del servicio de datos. En ocasiones, el servicio de datos tiene bibliotecas o utilidades administrativas que utilizan una interfaz de red cliente-servidor para realizar la administración. Es decir, una utilidad administrativa realiza una llamada al daemon del servidor, por lo que es posible que la dirección de red tenga que estar activa para utilizar la biblioteca o utilidad administrativa. En este caso, utilice los métodos `Start` y `Stop`.

Si el servicio de datos requiere que se desconfiguren las direcciones de red en el momento en que se inicia y se detiene, utilice los métodos `Prenet_start` y `Postnet_stop` para iniciar y detener el servicio de datos. Tenga en cuenta si el software cliente responderá de modo diferente en función de que sea la dirección de red o el servicio de datos el que se ponga en línea primero, tras una reconfiguración del clúster (`scha_control()` con el argumento `SCHA_GIVEOVER` o una conmutación con `scswitch`). Por ejemplo, la implementación de cliente puede hacer unos reintentos mínimos, y desistir pronto, una vez haya determinado que el puerto del servicio de datos no está disponible.

Si el servicio de datos no requiere que la dirección de red se configure al inicio, inícielo antes de que se configure la interfaz de red. Esto garantiza que el servicio de datos podrá responder inmediatamente a las solicitudes de cliente, en cuanto se haya configurado la dirección de red, y es menos probable que los clientes dejen de realizar nuevos intentos. En este caso, utilice el método `Prenet_start` en lugar del método `Start` para iniciar el servicio de datos.

Si utiliza el método `Postnet_stop`, el recurso del servicio de datos seguirá activo en el punto en que se haya desconfigurado la dirección de red. El método `Postnet_stop` sólo se invoca después de la desconfiguración de la dirección de red. Así, el puerto de servicio UDP o TCP del servicio de datos o su número de programa RPC, siempre aparecen como disponibles para los clientes de la red, salvo cuando la dirección de red tampoco responda.

La decisión de utilizar los métodos `Start` y `Stop`, en lugar de `Prenet_start` y `Postnet_stop`, o de utilizar ambos métodos, debe tener en cuenta los requisitos y el comportamiento del servidor y el cliente.

Métodos `Init`, `Fini` y `Boot`

Tres métodos opcionales, `Init`, `Fini` y `Boot`, permiten que RGM ejecute el código de inicialización y finalización en un recurso. RGM invoca el método `Init` para realizar una inicialización del recurso una sola vez, cuando éste pase a ser gestionado, tanto cuando el grupo de recursos al que pertenece pasa de un estado no gestionado a un estado gestionado como cuando se crea en un grupo de recursos que ya está gestionado.

RGM invoca el método `Fini` para reorganizar después del recurso, cuando éste pase a no estar gestionado, ya sea cuando el grupo de recursos al que pertenece pase a un estado no gestionado ya sea cuando se elimina de un grupo de recursos gestionado. La reorganización debe ser idempotente, es decir, que si se ha realizado ya, `Fini` devuelve 0 (satisfactorio).

RGM invoca el método `Boot` en nodos que se han unido al clúster recientemente, es decir, que han sido arrancados o rearrancados.

El método `Boot` suele realizar la misma inicialización que `Init`. Ésta debe ser idempotente, es decir, que si ya se ha inicializado el recurso en el nodo local, `Boot` y `Init` devuelven 0 (satisfactorio).

Supervisión de un recurso

Generalmente, se implementan los supervisores para que realicen análisis periódicos de fallos en los recursos, para detectar si funcionan correctamente. Si falla un análisis de fallos, el supervisor puede intentar un reinicio local o solicitar una operación de recuperación de fallos del grupo de recursos afectado, invocando las funciones `scha_control()` de RMAPI o `scds_fm_action()` de DSDL.

También se puede supervisar el rendimiento de un recurso y ajustar o realizar un informe del rendimiento. Escribir un supervisor de fallos específico del tipo de recurso es totalmente opcional. Incluso si decide no escribir un supervisor de fallos así, el tipo de recurso estará bajo la supervisión básica del clúster que realiza el propio Sun Cluster que detecta fallos del hardware del sistema, fallos graves del sistema operativo principal y los fallos de comunicación de un sistema en sus propias redes públicas.

Aunque RGM no invoca directamente al supervisor de recursos, permite el inicio automático de los supervisores de los recursos. Cuando se pone un recurso fuera de línea, RGM invoca el método `Monitor_stop` para detener el supervisor de recursos en los nodos locales antes de detener el recurso en sí. Cuando se pone un recurso en línea, RGM invoca el método `Monitor_start` después de que se haya iniciado el recurso en sí.

Las funciones `scha_control()` de RMAPI y `scds_fm_action()` de DSDL (que invoca `scha_control()`) permiten que los supervisores de recursos soliciten una operación de recuperación de fallos de un grupo de recursos a un nodo diferente. Dentro de las comprobaciones de validez, `scha_control()` invoca `Monitor_check` (si se ha definido), para determinar si el nodo solicitado es lo suficientemente fiable para controlar el grupo de recursos que contiene el recurso. Si `Monitor_check` informa que el nodo no es fiable, o que el método agota el tiempo de espera, RGM busca otro nodo para poder realizar la solicitud de recuperación de fallos. Si `Monitor_check` falla en todos los nodos, se cancela la recuperación de fallos.

El supervisor de recursos puede establecer las propiedades `Status` y `Status_msg` para reflejar la vista del supervisor del estado del recurso. Utilice la función `scha_resource_setstatus()` o el comando `scha_resource_setstatus` de RMAPI o la función `scds_fm_action()` de DSDL para establecer estas propiedades.

Nota – Aunque `Status` y `Status_msg` son especialmente útiles para un supervisor de recursos, cualquier programa puede establecer estas propiedades.

Consulte «Definición de un supervisor de fallos» en la página 107 para ver un ejemplo de un supervisor de fallos implementado con RMAPI. Consulte «Supervisor de fallos `SUNW.xfnts`» en la página 153 para ver un ejemplo de un supervisor de fallos implementado con DSDL. Consulte *Sun Cluster 3.1 Data Service Planning and Administration Guide* para obtener información sobre los supervisores de fallos que se integran en los servicios de datos que suministra Sun.

Adición del registro de mensajes a un recurso

Si desea registrar los mensajes de estado en el mismo archivo de registro que los demás mensajes del clúster, utilice la función `scha_cluster_getlogfacility()` para recuperar el número del recurso que se utiliza para registrar los mensajes de clúster.

Use este número de recurso con la función `syslog()` normal de Solaris para escribir los mensajes en el registro del clúster. También puede acceder a la información del recurso de registro del clúster a través de la interfaz genérica `scha_cluster_get()`.

Provisión de la gestión de los procesos

RMAPI y DSDL proporcionan recursos de gestión de procesos para implementar supervisores de recursos y rellamadas de control de recursos. RMAPI define los siguientes recursos (consulte las páginas de comando man para obtener detalles sobre cada uno de estos comandos y programas):

Prestación del supervisor de procesos: `pmfadm` y `rpc.pmfd`
: Prestación del supervisor de procesos (PMF) permite supervisar los procesos y sus descendientes, y reiniciarlos si se terminan; incluye el comando `pmfadm` para iniciar y controlar los procesos supervisados y el daemon `rpc.pmfd`.

`halockrun`
: Un programa para ejecutar un programa subordinado mientras se mantiene bloqueado el archivo. Este comando es práctico en las secuencias de shell.

`hatimerun`
: Un programa para ejecutar un programa subordinado con control de tiempo de espera. Es un comando práctico para las secuencias de shell.

DSDL proporciona la función `scds_hatimerun` para implementar `hatimerun`.

DSDL proporciona un conjunto de funciones (`scds_pmf_*`) para implementar la función PMF. Consulte «Funciones de PMF» en la página 206 si desea obtener información general sobre la función PMF de DSDL y una lista de las funciones individuales.

Provisión de soporte administrativo de un recurso

Las acciones administrativas aplicables a los recursos incluyen el establecimiento y la modificación de las propiedades de los recursos. La API define los métodos de rellamada `Validate` y `Update` para que se pueda enlazar con estas acciones administrativas.

RGM invoca el método opcional `Validate` cuando se crea un recurso y cuando una acción administrativa actualiza las propiedades del recurso o del grupo que lo contiene. RGM pasa los valores de propiedad del recurso y su grupo de recursos al método `Validate`. RGM invoca `Validate` en el conjunto de nodos del clúster que indica la propiedad `Init_nodes` del tipo de recurso (consulte «Propiedades del tipo de recurso» en la página 241 o la página de comando man `rt_properties`(5)) para obtener información sobre `Init_nodes`. RGM invoca `Validate` antes de que se aplique la creación o la actualización, y un código de salida de fallo desde el método en cualquier nodo provoca un fallo en la creación o la actualización.

RGM invoca `Validate` sólo cuando se cambian las propiedades del recurso o del grupo mediante una acción administrativa, no cuando RGM establece propiedades ni cuando un supervisor establece las propiedades de recurso `Status` y `Status_msg`.

RGM invoca el método opcional `Update` para notificar a un recurso en ejecución que se han modificado las propiedades. RGM invoca `Update` después de que una acción administrativa establezca satisfactoriamente las propiedades de un recurso o del grupo al que pertenece. RGM invoca este método en los nodos en los que el recurso está en línea. Este método puede utilizar las funciones de acceso de la API para leer los valores de propiedad que pueden afectar a un recurso activo y ajustar el recurso en ejecución según corresponda.

Implementación de un recurso a prueba de fallos

Un grupo de recursos a prueba de fallos contiene direcciones de red, como la dirección compartida y el nombre lógico de servidor integrados de los tipos de recursos, y recursos a prueba de fallos como los recursos de aplicación de los servicios de datos para un servicio de datos a prueba de fallos. Los recursos de dirección de red, junto con sus recursos de los servicios de datos dependientes, se mueven de uno a otros nodos del clúster cuando se produce una operación de recuperación de fallos o conmutación en los servicios de datos. RGM proporciona diversas propiedades que respaldan la implementación de un recurso a prueba de fallos.

Establezca la propiedad de tipo de recurso booleano `Failover` en `TRUE` para que el recurso no se pueda configurar en un grupo de recursos que pueda estar en línea en más de un nodo simultáneamente. Esta propiedad tiene el valor predeterminado de `FALSE`, por lo que hay que declararla `TRUE` en el archivo RTR para un recurso a prueba de fallos.

La propiedad de recurso `Scalable` determina si el recurso utiliza el recurso de dirección compartida del clúster. Para un recurso a prueba de fallos, establezca `Scalable` en `FALSE`, porque los recursos a prueba de fallos no utilizan direcciones compartidas.

La propiedad de grupo de recursos `RG_mode` permite al administrador del clúster identificar un grupo de recursos como a prueba de fallos o escalable. Si `RG_mode` es `FAILOVER`, RGM fija la propiedad `Maximum_primaries` del grupo en 1 y limita el grupo de recursos a un único nodo maestro. RGM no permite que se cree un recurso con la propiedad `Failover` fijada en `TRUE` en un grupo de recursos cuyo `RG_mode` sea `SCALABLE`.

La propiedad de grupo de recursos `Implicit_network_dependencies` especifica que RGM debe imponer dependencias fuertes implícitas de recursos que no sean dirección de red a todos los recursos de dirección de red (dirección compartida y nombre lógico de servidor) del grupo. Esto significa que para los recursos que no sean de dirección de red (servicios de datos) del grupo no se invocarán los métodos `Start` hasta que se asignen las direcciones de red del grupo. La propiedad `Implicit_network_dependencies` tiene el valor predeterminado `TRUE`.

Implementación de un recurso escalable

Un recurso escalable puede estar en línea en varios nodos simultáneamente. Los recursos escalables incluyen servicios de datos, como Sun Cluster HA para Sun One Web Server y HA-Apache.

RGM proporciona diversas propiedades que admiten la implementación de un recurso escalable.

Establezca la propiedad del tipo de recurso booleano `Failover` en `FALSE` para permitir que el recurso se configure en un grupo de recursos que pueda estar en línea en varios nodos simultáneamente.

La propiedad de recurso `Scalable` determina si el recurso utiliza el recurso de dirección compartida del clúster. Establezca esta propiedad en `TRUE` porque los servicios escalables utilizan los recursos de dirección compartida para que las diversas instancias del servicio escalable aparezcan ante el cliente como un único servicio.

La propiedad de grupo de recursos `RG_mode` permite al administrador del clúster identificar un grupo de recursos como a prueba de fallos o escalable. Si `RG_mode` es `SCALABLE`, RGM permite que `Maximum_primaries` tenga un valor superior a 1; esto quiere decir que el grupo puede tener varios nodos maestros simultáneamente. RGM permite que un recurso cuya propiedad `Failover` sea `FALSE` se lance en un grupo de recursos cuyo `RG_mode` sea `SCALABLE`.

El administrador del clúster crea un grupo de recursos escalables para que contenga los recursos de servicio escalables y un grupo de recursos a prueba de fallos independiente para que contenga los recursos de dirección compartida de los que dependerán los recursos escalables.

El administrador del clúster utiliza la propiedad del grupo de recursos `RG_dependencies` para especificar el orden en el que los grupos de recursos se ponen en línea y fuera de línea, dentro del nodo. Este orden es importante para el servicio escalable, porque los recursos escalables y los de dirección compartida de los que dependen están en grupos de recursos diferentes. Un servicio de datos escalables requiere que los recursos de dirección de red (dirección compartida) se asignen antes de su inicio. Por tanto, el administrador debe establecer la propiedad `RG_dependencies` (del grupo de recursos que contiene el servicio escalable) para que incluya el grupo de recursos que contiene los recursos de dirección compartida.

Cuando se declara la propiedad escalable en el archivo RTR de un recurso, RGM crea automáticamente el siguiente conjunto de propiedades escalables para el recurso:

`Network_resources_used` Identifica los recursos de dirección compartida que utiliza este recurso. Esta propiedad acude de modo predeterminado a la secuencia vacía, por lo que el administrador del clúster debe aportar la lista real

de direcciones compartidas que usa el servicio escalable cuando cree el recurso. El comando `scsetup` y SunPlex Manager proporcionan funciones para configurar automáticamente los recursos y grupos necesarios para los servicios escalables.

`Load_balancing_policy` Especifica la política de equilibrio de cargas del recurso. Puede establecer explícitamente la política del archivo RTR (o permitir la predeterminada, LB_WEIGHTED). En ambos casos, el administrador del clúster puede modificar el valor al crear el recurso (salvo que se establezca `Tunable` para `Load_balancing_policy` en `NONE` o `FALSE` en el archivo RTR). Los valores legales son:

- `LB_WEIGHTED` La carga se distribuye entre varios nodos, de acuerdo con los pesos establecidos en la propiedad `Load_balancing_weights`.
- `LB_STICKY` Un cliente determinado (identificado por la dirección IP de cliente) del servicio escalable se envía siempre al mismo nodo del clúster.
- `LB_STICKY_WILD` Un cliente determinado (identificado por la dirección IP de cliente) que se conecta a una dirección IP de un servicio adherente con comodín, siempre se envía al mismo nodo de clúster, independientemente del número de puerto al que llegue.

Para un servicio escalable con `Load_balancing_policy` `LB_STICKY` o `LB_STICKY_WILD`, cambiar `Load_balancing_weights` con el servicio en línea puede provocar la puesta a cero de las afinidades existentes del cliente. En ese caso, un nodo diferente puede servir una solicitud posterior de un cliente, aunque antes éste haya sido atendido por otro nodo del clúster.

	Del mismo modo, iniciar una nueva instancia del servicio en un clúster puede poner a cero las afinidades existentes del cliente.
`Load_balancing_weights`	Especifica la carga que se enviará a cada nodo. El formato es *peso@nodo*,*peso@nodo*, donde *peso* es un número entero que refleja la parte relativa de carga distribuida al *nodo* especificado. La fracción de carga distribuida a un nodo es el peso de este nodo dividido entre la suma de todos los pesos de las instancias activas. Por ejemplo, `1@1,3@2` especifica que el nodo 1 recibe 1/4 de la carga y el nodo 2, 3/4.
`Port_list`	Identifica los puertos en los que recibe el servidor. Esta propiedad acude de modo predeterminado a la secuencia vacía. Puede indicarse una lista de puertos en el archivo RTR. En caso contrario, el administrador del clúster debe suministrar la lista real de puertos al crear el recurso.

Es posible crear un servicio de datos que el administrador pueda configurar para que sea de tipo escalable o a prueba de fallos. Para ello, declare la propiedad de tipo de recurso `Failover` y la propiedad de recurso `Scalable` `FALSE` en el archivo RTR del servicio de datos. Especifique la propiedad `Scalable` para que se pueda ajustar en el momento de la creación.

El valor de propiedad `Failover` (`FALSE`) permite que el recurso se configure como grupo de recursos escalables. El administrador puede habilitar direcciones compartidas, cambiando el valor de `Scalable` a `TRUE` al crear el recurso, creando así un servicio escalable.

Por otra parte, aunque `Failover` se establezca en `FALSE`, el administrador puede configurar el recurso en un grupo de recursos a prueba de fallos para implementar un servicio a prueba de fallos. El administrador no cambia el valor de `Scalable`, que es `FALSE`. Para admitir esta contingencia, debe proporcionar una comprobación en el método `Validate` de la propiedad `Scalable`. Si `Scalable` es `FALSE`, compruebe que el recurso esté configurado como grupo de recursos a prueba de fallos.

Sun Cluster 3.1: Guía de conceptos contiene información adicional sobre los recursos escalables.

Comprobaciones de validación de los servicios escalables

Siempre que se crea o actualiza un recurso con la propiedad escalable fijada en `TRUE`, RGM valida diversas propiedades de recursos. Si éstas no están debidamente configuradas, RGM rechaza el intento de creación o actualización. RGM realiza las comprobaciones siguientes:

- La propiedad `Network_resources_used` no debe estar vacía y debe contener los nombres de los recursos de direcciones compartidas. Cada uno de los nodos de `Nodelist` del grupo de recursos que contenga el recurso escalable debe aparecer en las propiedades `NetIfList` o `AuxNodeList` de cada uno de los recursos de dirección compartida con nombre.
- La propiedad RG_dependencies del grupo de recursos que contiene el recurso escalable debe incluir los grupos de recursos de todos los recursos de dirección compartida enumerados en la propiedad de recurso escalable `Network_resources_used`.
- La propiedad Port_list no debe estar vacía y debe contener una lista de pares puerto-protocolo; el protocolo debe ser tcp o udp. Por ejemplo,

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

Escritura y comprobación de los servicios de datos

Esta sección proporciona información sobre la escritura y comprobación de los servicios de datos.

Utilización de keep-alives

En el lado del servidor, la utilización de keep-alives de TCP evita que el servidor malgaste recursos de sistema en un cliente inactivo (o con partición de red). Si estos recursos no se reorganizan (en un servidor que permanezca activo el tiempo suficiente), los recursos malgastados aumentan sin límite, a medida que los sistemas de los clientes se caen y rearrancan.

Si la comunicación cliente-servidor utiliza un canal TCP, tanto el cliente como el servidor deberían habilitar el mecanismo keep-alive de TCP. Esta disposición se aplica incluso a los casos en los que no hay una alta disponibilidad y sólo hay un servidor.

Otros protocolos orientados a la conexión pueden disponer también de mecanismos keep-alive.

En el lado del cliente, la utilización de mecanismos keep-alive de TCP permite que el cliente reciba un aviso cuando se haya producido una recuperación de fallos o una conmutación de un recurso de dirección de red, desde un sistema físico a otro. Esa transferencia del recurso de dirección de red interrumpe la conexión de TCP. Sin embargo, salvo que el cliente haya habilitado el mecanismo keep-alive, no siempre recibirá información de la interrupción de la conexión, si ésta parece inactiva en ese momento.

Por ejemplo, suponga que el cliente está esperando una respuesta del servidor a una solicitud que requiere mucho tiempo y que el mensaje de solicitud del cliente ha llegado ya al servidor y se ha reconocido en la capa de TCP. En esta situación, el módulo de TCP del cliente no necesita seguir transmitiendo la solicitud y se bloquea la aplicación del cliente, mientras se espera la respuesta.

Siempre que sea posible, además de utilizar el mecanismo keep-alive TCP, la aplicación del cliente debe realizar su propia acción periódica keep-alive en este nivel, porque el mecanismo keep-alive de TCP no es perfecto en todas las situaciones de limitación posibles. La utilización de un mecanismo keep-alive en una aplicación suele requerir que el protocolo cliente-servidor admita una operación nula, o al menos una operación eficiente de sólo lectura, como una operación de estado.

Comprobación de los servicios de datos de alta disponibilidad

Esta sección aporta sugerencias sobre cómo comprobar una implementación de un servicio de datos en un entorno de alta disponibilidad. Los ejemplos de comprobación son sólo sugerencias y no cubren todas las posibilidades. Necesita acceso a una configuración de prueba de Sun Cluster para que el trabajo de comprobación no afecte a las máquinas de producción.

Compruebe que el servicio de datos de alta disponibilidad se comporte adecuadamente en todos los casos en los que un grupo de recursos se desplaza entre los sistemas físicos. Estos ejemplos incluyen caídas de sistemas y la utilización del comando `scswitch`. Compruebe que las máquinas clientes sigan recibiendo servicio tras estos acontecimientos.

Compruebe la idempotencia de los métodos. Por ejemplo, sustituya los métodos temporalmente con una secuencia breve de shell que llame al método original dos o más veces.

Coordinación de dependencias entre los recursos

En ocasiones, el servicio de datos cliente-servidor realiza solicitudes a otro servicio de datos cliente-servidor mientras responde a una solicitud de un cliente. Informalmente, un servicio de datos A depende de un servicio de datos B si, para que A proporcione su servicio, B debe proporcionar el suyo. Sun Cluster facilita que esto se realice permitiendo que las dependencias de recursos se configuren dentro de un grupo de recursos. Las dependencias afectan al orden en el que Sun Cluster inicia y detiene los servicios de datos. Consulte la página de comando man `scrgadm`(1M) para obtener más detalles.

Si los recursos de su tipo de recurso dependen de los recursos de otro tipo, tendrá que indicar al usuario que configure los recursos y grupos de recursos adecuadamente o proporcionar las secuencias o herramientas para configurarlos correctamente. Si el recurso dependiente debe ejecutarse en el mismo nodo que el recurso del que depende, ambos deben estar configurados en el mismo grupo de recursos.

Decida si se van a usar dependencias explícitas de recursos o si prefiere omitirlas y consultar la disponibilidad de otro(s) servicio(s) de datos en el código del servicio de datos de alta disponibilidad. En caso de que el recurso dependiente y aquél del que depende se puedan ejecutar en nodos diferentes, configúrelos en grupos de recursos separados. En este caso, la consulta es necesaria, porque no es posible configurar dependencias de recursos entre varios grupos.

Algunos servicios de datos no almacenan ningún dato directamente, sino que dependen de otro servicio de datos de componente trasero para almacenar datos. Este servicio de datos traduce todas las solicitudes de lectura y actualización a llamadas en el servicio de datos de componente trasero. Por ejemplo, supongamos un servicio de agenda de citas cliente-servidor hipotético que mantenga todos sus datos en una base de datos SQL, como Oracle. El servicio de agenda de citas tiene su propio protocolo de red cliente-servidor. Por ejemplo, su protocolo se puede haber definido con un idioma de especificación RPC, como ONC RPC.

En el entorno de Sun Cluster, puede utilizar HA-ORACLE para que la base de datos Oracle de componente trasero tenga una alta disponibilidad. Después, puede escribir métodos simples para iniciar y detener el daemon de la agenda de citas. Su usuario final registra el tipo de recurso de agenda de citas en Sun Cluster.

Si la aplicación de agenda de citas se debe ejecutar en el mismo nodo que la base de datos Oracle, el usuario final configura el recurso de agenda de citas en el mismo grupo de recursos que el recurso HA-ORACLE y hace que el recurso de agenda de citas dependa de éste. Esta dependencia se especifica con la etiqueta de propiedad `Resource_dependencies` en `scrgadm`.

Si el recurso HA-ORACLE puede ejecutarse en un nodo diferente que el recurso de agenda de citas, el usuario final los configura en grupos de recursos separados. El usuario final puede configurar una dependencia de grupo de recursos del grupo de recursos de agenda en el grupo de recursos Oracle. Sin embargo, las dependencias de

grupo de recursos sólo son eficaces cuando se inician o detienen ambos grupos de recursos en el mismo nodo simultáneamente. Por tanto, el daemon del servicio de datos de agenda, una vez iniciado, puede interrogar, esperando que la base de datos Oracle esté disponible. El método `Start` del tipo de recurso de agenda normalmente devolverá un resultado satisfactorio en este caso, porque si el método `Start` estuviera bloqueado indefinidamente, pondría su grupo de recursos en un estado ocupado, que impediría que hubiera otros cambios de estado (como ediciones, recuperaciones de fallos o conmutaciones) en el grupo. Sin embargo, si el método `Start` del recurso de agenda agotara su tiempo de espera o devolviera un valor diferente de cero, podría hacer que el grupo de recursos pasara de uno a otro nodo, mientras la base de datos Oracle no estuviera disponible.

CAPÍTULO 3

Actualización de un tipo de recurso

Este capítulo trata las cuestiones que se deben comprender para actualizar un tipo de recurso y migrar un recurso.

Visión general

Los administradores de sistemas tienen que poder instalar y registrar una nueva versión de un tipo de recurso existente para permitir el registro de múltiples versiones de un tipo de recurso determinado y para migrar un recurso a una nueva versión del tipo de recurso, sin tener que eliminar y volver a crear el recurso. Los desarrolladores de recursos tienen que comprender los requisitos para proporcionar una actualización del tipo de recurso y una migración del recurso.

Los tipos de recurso desarrollados de manera que admitan una modernización se denominan *habilitados para modernización*.

Una nueva versión de un tipo de recurso puede diferir de una versión anterior en muchas formas:

- Los atributos de las propiedades del tipo de recurso pueden cambiar
- El conjunto declarado de propiedades de recursos, incluidas propiedades estándar y de extensión, puede cambiar
- Los atributos de las propiedades de recursos, como `default`, `min`, `max`, `arraymin`, `arraymax` o la capacidad de ajuste pueden cambiar
- El conjunto de métodos declarados puede variar
- La implementación de métodos o supervisores puede cambiar.

El desarrollador del tipo de recursos decide cuándo se puede migrar un recurso a una nueva versión, entre las siguientes opciones de ajuste. Las opciones aparecen ordenadas de la menos a la más restrictiva:

- En cualquier momento (`Anytime`)
- Cuando el recurso no está bajo supervisión (`When_unmonitored`)
- Cuando el recurso está fuera de línea (`When_offline`)
- Cuando el recurso está inhabilitado (`When_disabled`)
- Cuando el grupo de recursos no está siendo gestionado (`When_unmanaged`)
- En el momento de la creación (`At_creation`)

Consulte «Propiedad `Type_version` del recurso» en la página 62 si desea ver una explicación de cada opción.

Nota – En este capítulo se usa el comando `scrgadm` cuando se trata de las modernizaciones. Sin embargo, el administrador no está obligado a usar el comando `scrgadm`, sino que también puede utilizar la GUI o el comando `scsetup` para realizar la modernización.

Archivo de registro del tipo de recurso

Nombre del tipo de recurso

Los tres componentes del nombre del tipo de recurso son propiedades especificadas en el archivo RTR, como *ID_fabricante*, *tipo_recurso* y *versión_TR*. El comando `scrgadm` introduce los delimitadores de punto y de punto y coma para crear el nombre del tipo de recurso:

```
ID_fabricante.tipo_recurso:versión_rt
```

El prefijo *ID_fabricante* sirve para diferenciar dos archivos de registro con el mismo nombre, suministrados por fabricantes diferentes. El sufijo *versión_TR* establece una distinción entre las diferentes versiones registradas (modernizaciones) del mismo tipo de recurso básico. Para garantizar que el *ID_fabricante* sea único, se recomienda utilizar el símbolo bursátil de la empresa que crea el tipo de recurso.

El registro del tipo de recurso no será satisfactorio si la secuencia de *versión_TR* incluye un espacio en blanco, una tabulación, barra inclinada (/), barra inclinada inversa (\), asterisco (*), interrogación (?), coma (,), punto y coma (;), corchete izquierdo ([) o corchete derecho (]).

La propiedad `RT_Version`, que era opcional en Sun Cluster 3.0, es obligatoria desde Sun Cluster 3.1.

El nombre completo es el nombre que devuelve el comando siguiente:

`scha_resource_get -O Type -R` *nombre_recurso* `-G` *nombre_grupo_recurso*

Los nombres del tipo de recursos registrados antes de Sun Cluster 3.1 siguen usando el formato:

`ID_fabricante.tipo_recurso`

Directivas

Los archivos RTR de los tipos de recursos habilitados para la modernización deben incluir una directiva `#$upgrade`, seguida de cero o más directivas con el formato:

`#$upgrade_from` *versión capacidad de ajuste*

La directiva `upgrade_from` consiste en la secuencia `#$upgrade_from`, seguida de `RT_Version`, seguida a su vez de la limitación de la capacidad de ajuste en el recurso. Si el tipo de recurso desde el que se está realizando la modernización no tiene una versión, `RT_Version` se especifica como la secuencia vacía, como muestra el último de los siguientes ejemplos:

```
#$upgrade_from   "1.1"   when_offline
#$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 impone estas limitaciones a un recurso cuando el administrador intenta cambiar el valor `Type_version` del recurso. Si la versión actual del tipo de recurso no aparece en la lista, RGM impone la capacidad de ajuste de `When_unmanaged`.

Estas directivas deben aparecer entre la sección de declaraciones de la propiedad del tipo de recurso del archivo RTR y la sección de declaraciones del recurso del archivo RTR. Consulte `rt_reg`(4).

Cambio de `RT_Version` en un archivo RTR

Cambie la secuencia `RT_Version` de un archivo RTR siempre que cambie el contenido de éste. El valor de esta propiedad debe dejar claro cuál es la versión más reciente del tipo de recurso y cuál es la más antigua. No es necesario cambiar la secuencia `RT_Version` si no hay cambios en el archivo RTR.

Nombres de los tipos de recursos en versiones anteriores de Sun Cluster

Los nombres de los tipos de recursos de Sun Cluster 3.0 no contenían el sufijo de la versión:

```
ID_fabricante.nombre_recurso
```

Un tipo de recurso registrado originalmente en Sun Cluster 3.0 sigue teniendo un nombre con este formato aunque se modernice el software del clúster a Sun Cluster 3.1. Del mismo modo, un tipo de recurso cuyo archivo RTR carezca de directiva `#$upgrade` recibe un nombre con formato de Sun Cluster 3.0, sin el sufijo de versión, si el archivo RTR está registrado en un clúster que funcione con el software Sun Cluster 3.1.

Puede registrar archivos RTR con las directivas `#$upgrade` o `#$upgrade_from` de Sun Cluster 3.0, pero la migración de recursos a nuevos tipos de recursos de Sun Cluster 3.0 no se admitirá.

Propiedad `Type_version` del recurso

La propiedad del recurso estándar `Type_version` almacena la propiedad `RT_Version` de un tipo de recurso. Esta propiedad no aparece en el archivo RTR. El administrador del sistema edita este valor de propiedad con el siguiente comando:

```
scrgadm -c -j recurso -y Type_version=versión_nueva
```

La capacidad de ajuste de esta propiedad se deriva de:

- La versión actual del tipo de recurso
- Las directivas `#$upgrade_from` del archivo RTR

Utilice los siguientes valores de ajuste en las directivas `#$upgrade_from`:

`Anytime`
: Si no hay restricciones sobre el momento de modernización del recurso. El recurso puede estar totalmente en línea.

`When_unmonitored`
Si se sabe que los métodos `Update`, `Stop`, `Monitor_check` y `Postnet_stop` de la nueva versión del tipo de recurso son compatibles con métodos de inicio de versiones anteriores del tipo de recurso, (`Prenet_stop` y `Start`), y que el método `Fini` de la nueva versión del tipo de recurso es compatible con el método `Init` de las versiones anteriores. En esta situación sólo será necesario cerrar el programa de supervisión del recurso antes de la modernización

`When_offline`
Si los métodos `Update`, `Stop`, `Monitor_check` o `Postnet_stop` de la nueva versión del tipo de recurso no son compatibles con métodos de inicio de versiones anteriores del tipo de recurso (`Prenet_stop` y `Start`), pero se sabe que son compatibles con el método `Init` de versiones anteriores, el recurso debe estar en línea cuando se le aplique la modernización.

`When_disabled`
Similar a `When_offline`. Sin embargo, este valor de capacidad de ajuste impone la condición más firme de que se inhabilite el recurso.

`When_unmanaged`
Si el método `Fini` de la nueva versión del tipo de recurso no es compatible con el método `Init` de versiones anteriores. Este valor de capacidad de ajuste requiere que el grupo de recursos se ponga en estado no gestionado antes de modernizar el recurso.

`At_creation`
Si los recursos no se pueden actualizar a la nueva versión del tipo de recurso. Sólo se pueden crear recursos nuevos de la nueva versión.

La capacidad de ajuste `At_creation` significa que el desarrollador del tipo de recurso puede impedir la migración de un recurso al tipo nuevo. En este caso, el administrador del sistema debe borrar y volver a crear el recurso. Equivale a declarar que la versión del recurso sólo se puede establecer en el momento de la creación.

Migración de un recurso a una versión diferente

Un recurso absorbe la nueva versión del tipo de recurso cuando el administrador del sistema edita la propiedad `Type_version` del recurso. Este proceso sigue las mismas convenciones empleadas para editar otras propiedades del recurso, salvo que cierta información se derivará u obtendrá de la nueva versión del tipo de recurso, y no de la versión actual:

- Los atributos de la propiedad del recurso de todas las propiedades, como `min`, `max`, `arraymin`, `arraymax`, valor predeterminado y capacidad de ajuste, se obtienen de la nueva versión del tipo de recurso
- La capacidad de ajuste aplicable a la propiedad `Type_version` se obtiene de las directivas `#$upgrade_from` del archivo RTR y la propiedad `RT_version` del tipo de recurso del recurso existente. Esta capacidad de ajuste es diferente de la descrita en `property_attributes`(5).
- Se aplicará el método `Validate` de la nueva versión del tipo de recurso lo que garantiza que los atributos de propiedad sean válidos para el nuevo tipo de recurso. Si los atributos existentes de la propiedad del recurso no cumplen las condiciones de validación de la nueva versión del tipo de recurso, el administrador del sistema debe proporcionar valores válidos para dichas propiedades en la línea de comandos `scrgadm`. Esto puede ocurrir si la versión más reciente del tipo de recurso comienza a usar una propiedad que no se declaró en la versión anterior y que no tiene una opción predeterminada. También puede ocurrir si el recurso ya tiene una propiedad a la que se le ha asignado un valor no válido para la nueva versión del tipo de recurso.
- Las propiedades del recurso declaradas en una versión anterior del tipo de recurso se pueden no declarar en la versión más actual. Cuando se migra el recurso a una versión más moderna, la propiedad se elimina de éste.

Nota – El método `Validate` puede consultar el `Type_version` actual del recurso (con `scha_resource_get`) y el nuevo `Type_version` (que se pasa a la línea de comandos `Validate`). Por tanto, `Validate` puede descartar las modernizaciones a partir de versiones no admitidas.

Modernización de un tipo de recurso y adaptación a una versión anterior

La sección "Upgrading a Resource Type" in *Sun Cluster 3.1 Data Service Planning and Administration Guide* contiene información adicional sobre la modernización o migración de un tipo de recurso.

▼ Cómo se moderniza un tipo de recurso

1. **Lea la documentación sobre la modernización de un nuevo tipo de recurso para ver los cambios de los tipos de recurso y las limitaciones a los ajustes del recurso.**

2. **Instale el paquete de modernización del tipo de recurso en todos los nodos de clúster.**

   La práctica recomendada para instalar paquetes de nuevos tipos de recursos es la misma que para el despliegue de modernizaciones: `pkgadd` se produce cuando el nodo se arranca en modo sin clúster.

   Hay situaciones en las que sería posible instalar paquetes de nuevos tipos de recursos en un nodo en modo clúster:

   - Si la instalación del paquete del tipo de recurso no cambia el código del método y sólo actualiza el supervisor, será necesario detener la supervisión de todos los recursos de ese tipo durante la instalación.
   - Si la instalación del paquete del tipo de recurso no cambia el código del supervisor ni el método, no será necesario detener la supervisión del recurso durante la instalación, porque ésta sólo está poniendo un nuevo archivo RTR en el disco.

3. **Registre la nueva versión del tipo de recurso con el comando `scrgadm` (o equivalente), que debe hacer referencia al archivo RTR de la modernización.**

   RGM crea un nuevo tipo de recurso, cuyo nombre tiene el formato

   `ID_fabricante.tipo_recurso:versión`

4. **Si la modernización del tipo de recurso se instala sólo en un subgrupo de nodos, debe establecer la propiedad `Installed_nodes` del nuevo tipo de recurso en los nodos en los que se instala efectivamente.**

   Cuando un recurso incorpora el nuevo tipo (porque se cree de cero o se actualice), RGM requiere que el grupo de recursos `nodelist` sea un subconjunto de la lista de `Installed_nodes` del tipo de recurso.

   `scrgadm -c -t` *tipo_recurso* `-h` *lista_nodos_instalados*

5. **Para cada recurso del tipo premodernizado que se vaya a migrar al tipo modernizado, invoque `scswitch` para cambiar el estado del recurso o su grupo de recursos al estado adecuado, como indica la documentación de modernización.**

6. **Para cada recurso del grupo premodernizado que se vaya a migrar al tipo modernizado, edite el recurso, cambiando su propiedad `Type_version` a la nueva versión.**

   `scrgadm -c -j` *recurso* `-y Type_version=`*versión_nueva*

   Si fuera necesario, cambie otras propiedades del mismo recurso a los valores apropiados del mismo comando.

7. **Restaure el estado anterior del recurso o grupo de recursos, invirtiendo el comando que se invoca en el Paso 5.**

▼ Cómo adaptar un recurso a una versión anterior de su tipo de recurso

Es posible adaptar un recurso a una versión anterior de su tipo de recurso. Las condiciones en las que esto es posible son más restrictivas que aquéllas en las que es posible modernizar el tipo de recurso a una nueva versión. Primero debe dejar el grupo de recursos sin gestionar. Además, sólo será posible recuperar versiones modernizables del tipo de recurso. Las versiones modernizables se pueden identificar con el comando `scrgadm -p`. En la salida, las versiones habilitadas para la modernización, contienen el sufijo `:`*versión*.

1. **Vaya al grupo de recursos que contenga el recurso que desee adaptar una versión anterior.**

   `scswitch -F -g` *grupo_recurso*

2. **Inhabilite el recurso y todos los recursos del grupo.**

   `scswitch -n -j` *recurso_que_adaptar*
   `scswitch -n -j` *recurso1*
   `scswitch -n -j` *recurso2*
   `scswitch -n -j` *recurso3*
   `...`

   Nota – Inhabilite los recursos en orden de dependencia, empezando con el más dependiente (recursos de aplicación) hasta llegar al menos dependiente (recursos de dirección de red).

3. **Deje el grupo de recursos sin gestión.**

   `scswitch -u -g` *grupo_recurso*

4. **¿La versión anterior del tipo de recurso que desea adaptar sigue registrada en el clúster?**

   - Si es así, vaya al paso siguiente.
   - En caso contrario, vuelva a registrar la versión anterior que desee.

     `scrgadm -a -t` *nombre_tipo_recurso*

5. **Adapte el recurso; especifique cuál es la versión anterior que desea para `Type_version`.**

   `scrgadm -c -j` *recurso_que_adaptar* **`-y Type_version=`***versión_anterior*

   Si fuera necesario, cambie otras propiedades del mismo recurso a los valores apropiados del mismo comando.

6. **Ponga el grupo de recursos que contiene el recurso que ha adaptado en un estado gestionado, habilite todos los recursos y ponga el grupo en línea.**

```
scswitch -Z -g grupo_recurso
```

Valores predeterminados de la propiedad

RGM almacena todos los recursos, de forma que cualquier propiedad que el administrador del sistema no haya establecido explícitamente (y que había sido predeterminada) no quede almacenada en la entrada de recurso del CCR (almacén de configuración de clúster). RGM obtiene el valor predeterminado de una propiedad de recurso ausente del tipo de recurso (o, si no se definiera aquí, de un valor predeterminado definido por el sistema) cuando se lee un recurso desde el CCR. Este método de almacenamiento de las propiedades permite que un tipo de recurso modernizado defina nuevas propiedades o nuevos valores predeterminados para propiedades existentes.

Cuando se editan las propiedades de recurso, RGM almacena en el CCR las especificadas en el comando de edición.

Si una versión modernizada del tipo de recurso declara un valor predeterminado nuevo para una propiedad cuyo valor seleccionado es el predeterminado, el nuevo valor predeterminado es heredado por los recursos existentes, aunque la propiedad sólo se declare ajustable en el momento `At_creation` o `When_disabled`. Si la aplicación del nuevo valor predeterminado hace que falle un método como `Stop`, `Postnet_stop` o `Fini`, el implantador del tipo debe restringir el estado del recurso en el momento de su modernización, según corresponda. Esto se hace limitando la capacidad de ajuste de la propiedad `Type_version`.

El método `Validate` de la nueva versión del tipo de recurso puede comprobar si los atributos existentes de la propiedad son los adecuados. Si no lo son, el administrador del sistema puede editar las propiedades de un recurso para indicar los valores correctos en el mismo comando que edita la propiedad `Type_version` para modernizar el recurso a la nueva versión del tipo de recurso.

Nota – Los recursos que se crearon en Sun Cluster 3.0 no heredan los nuevos atributos predeterminados de la propiedad del tipo de recurso cuando se migran a una versión posterior, porque sus propiedades predeterminadas se guardan en el CCR.

Documentación del desarrollador del tipo de recurso

El desarrollador del tipo de recurso debe proporcionar la documentación con el nuevo recurso en la que:

- Describa cualquier adición, cambio o eliminación en la propiedad
- Describa cómo hacer que las propiedades cumplan los nuevos requisitos
- Establezca las limitaciones de las capacidades de ajustes de los recursos
- Indique cualquier atributo predeterminado nuevo de la propiedad
- Informe al administrador del sistema de que es posible editar las propiedades de los recursos para que se ajusten a los valores apropiados con el mismo comando empleado para editar la propiedad `Type_version` para modernizar el recurso a la nueva versión del tipo de recurso

Implementaciones del nombre y del supervisor del tipo de recurso

Puede registrar un tipo de recurso habilitado para modernización en Sun Cluster 3.0, pero su nombre se registra en el CCR sin el sufijo de la versión. Para que se ejecute correctamente en Sun Cluster 3.0 y Sun Cluster 3.1, el supervisor de este tipo de recurso debe ser capaz de manejar las dos convenciones de asignación de nombre:

```
ID_fabricante.nombre_recurso:versión
ID_fabricante.nombre_recurso
```

El código del supervisor puede determinar el nombre adecuado que se debe utilizar, ejecutando el equivalente de:

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

Después, compare los valores de salida con `vers`. Sólo uno de estos comandos servirá para un valor concreto de `vers`, porque no es posible registrar la misma versión del tipo de recurso dos veces, con dos nombres diferentes.

Modernizaciones de las aplicaciones

La modernización del código de aplicación es diferente de la modernización del código del agente, aunque algunas de las cuestiones que surgen son similares. Una modernización de una aplicación puede estar o no acompañada de una modernización del tipo de recurso.

Ejemplos de la modernización del tipo de recurso

Estos ejemplos muestran diferentes situaciones de modernización e instalación de los tipos de recursos. La información de los paquetes y la capacidad de ajuste se han seleccionado en función de los tipos de cambios que se hayan realizado en la implementación del tipo de recurso. La capacidad de ajuste se aplica a la migración del recurso al nuevo tipo de recurso.

Todos los ejemplos presuponen que:

- El tipo de recurso se incluye en un paquete de tipo Solaris. Consulte `pkgadd`(1M) y `pkgrm`(1M)
- Sólo hay una versión anterior del tipo de recurso y, por tanto, sólo hay una directiva `#$upgrade_from` en el nuevo archivo RTR
- El procedimiento de instalación no eliminará ni sobrescribirá los métodos si es posible que RGM invoque los métodos mientras se eliminan del disco
- Los nuevos métodos son compatibles con los antiguos, salvo que se indique lo contrario
- Los recursos y grupos de recursos se ponen en el estado necesario antes de la instalación o migración, con el comando `scswitch`(1M) correcto, o equivalente. El ejemplo siguiente muestra cómo poner el grupo de recursos en un estado no gestionado:

  ```
  scswitch -M -n -j recurso
  scswitch -n -j recurso
  scswitch -F -g grupo_recursos
  scswitch -u -g grupo_recursos
  ```

- Los tipos de recursos se registran con el comando siguiente:

  ```
  scrgadm -a -t tipo_recurso -f ruta_a_archivo_RTR
  ```

- Los tipos de recursos se migran con el comando siguiente:

```
scrgadm -c -j recurso -y Type_version=versión \
           -y propiedad=valor \
           -x propiedad=valor ...
```

- Los recursos y grupos de recursos se devuelven a su estado anterior tras la migración, con el comando `scswitch`(1M) adecuado, o equivalente:

```
scswitch -M -e -j recurso
scswitch -e -j recurso
scswitch -o -g grupo_recursos
scswitch -Z -g grupo_recursos
```

Es posible que el desarrollador de tipos de recursos tenga que especificar valores más restrictivos para la capacidad de ajuste que los que se indican en estos ejemplos. Los valores de la capacidad de ajuste dependen de los cambios precisos que se realicen en la implementación del tipo de recurso. Asimismo, el desarrollador del tipo de recurso puede decidir utilizar un esquema de paquetes diferente, en lugar del de Solaris que se emplea en estos ejemplos.

TABLA 3–1 Ejemplos de la modernización de un tipo de recurso

Tipo de cambio	Capacidad de ajuste	Empaquetado	Procedimiento
Los cambios de propiedad sólo se realizan en el archivo RTR.	`Anytime`	Incluir sólo el archivo RTR nuevo.	Ejecutar `pkgadd` con el archivo RTR nuevo en todos los nodos. Registrar el nuevo tipo de recurso. Migrar el recurso.
Los métodos se actualizan.	`Anytime`	Colocar los métodos actualizados en una ruta que sea diferente de la de los métodos anteriores.	Ejecutar `pkgadd` con los métodos actualizados en todos los nodos. Registrar el nuevo tipo de recurso. Migrar el recurso.
Nuevo programa supervisor.	`When_unmonitored`	Sobrescribir únicamente la versión anterior del supervisor.	Inhabilitar la supervisión. Ejecutar `pkgadd` con el nuevo programa supervisor en todos los nodos. Registrar el nuevo tipo de recurso. Migrar el recurso. Habilitar la supervisión.

TABLA 3–1 Ejemplos de la modernización de un tipo de recurso *(Continuación)*

Tipo de cambio	Capacidad de ajuste	Empaquetado	Procedimiento
Los métodos se actualizan. Los nuevos métodos `Update/Stop` son incompatibles con los métodos `Start` antiguos.	`When_offline`	Colocar los métodos actualizados en una ruta que sea diferente de la de los métodos anteriores.	Ejecutar `pkgadd` con los métodos actualizados en todos los nodos. Registrar el nuevo tipo de recurso. Poner el recurso fuera de línea. Migrar el recurso. Poner el recurso en línea.
Los métodos se actualizan y se agregan propiedades nuevas al archivo RTR. Los nuevos métodos requieren nuevas propiedades. El objetivo es permitir que el grupo de recursos que lo contiene permanezca en línea, pero evitando que el recurso se ponga en línea si el grupo de recursos pasa del estado en línea al estado fuera de línea en algún nodo.	`When_disabled`	Sobrescribir las versiones anteriores de los métodos.	Inhabilitar el recurso. Para cada nodo: ■ Sacar el nodo del clúster ■ Ejecutar `pkgrm/pkgadd` con los métodos que se están actualizando ■ Restaurar el nodo al clúster Registrar el nuevo tipo de recurso. Migrar el recurso. Habilitar el recurso.

TABLA 3–1 Ejemplos de la modernización de un tipo de recurso *(Continuación)*

Tipo de cambio	Capacidad de ajuste	Empaquetado	Procedimiento
Los métodos se actualizan y se agregan propiedades nuevas al archivo RTR. Los nuevos métodos no requieren nuevas propiedades.	`Anytime`	Sobrescribir las versiones anteriores de los métodos.	Para cada nodo: ■ Sacar el nodo del clúster ■ Ejecutar `pkgrm`/`pkgadd` con los métodos que se están actualizando ■ Restaurar el nodo al clúster Durante este procedimiento, RGM invocará los nuevos métodos, aunque la migración (que configuraría las propiedades nuevas) no se haya realizado todavía. Es importante que los métodos nuevos puedan trabajar sin las propiedades nuevas. Registrar el nuevo tipo de recurso. Migrar el recurso.
Los métodos se actualizan. El nuevo método `Fini` es incompatible con el método `Init` antiguo.	`When_unmanaged`	Colocar los métodos actualizados en una ruta que sea diferente de la de los métodos anteriores.	Dejar sin gestionar el grupo de recursos que contiene el recurso en cuestión. Ejecutar `pkgadd` con los métodos actualizados en todos los nodos. Registrar el nuevo tipo de recurso. Migrar el recurso. Poner el grupo de recursos que contiene el recurso en cuestión bajo gestión.
Los métodos se actualizan. No se realizan cambios en el archivo RTR.	No se aplica. No se realizan cambios en el archivo RTR.	Sobrescribir las versiones anteriores de los métodos.	Para cada nodo: ■ Sacar el nodo del clúster ■ Ejecutar `pkgadd` con los métodos actualizados ■ Restaurar el nodo al clúster Dado que no se han realizado cambios en el archivo RTR, no es necesario registrar ni migrar el recurso.

Requisitos de instalación para los paquetes de tipos de recursos

Hay dos requisitos relacionados con la instalación de los nuevos paquetes de tipos de recursos:

- Cuando se registra un nuevo tipo de recurso, su archivo RTR debe estar accesible en disco
- Cuando se crea un recurso del nuevo tipo, todos los nombres de rutas de los métodos declarados y el programa supervisor del nuevo tipo deben estar en disco y deben ser ejecutables. El método y los programas supervisores antiguos deben permanecer en su lugar mientras el recurso se esté utilizando.

  Para decidir cuál es el paquete más adecuado, el implantador del tipo de recurso debe tener presente:

  - ¿Cambia el archivo RTR?
  - ¿Cambian el valor predeterminado o la capacidad de ajuste de una propiedad?
  - ¿Cambian los valores `min` o `max` de una propiedad?
  - ¿Agrega o elimina propiedades esta modernización?
  - ¿Cambia el código del método?
  - ¿Cambia el código del supervisor?
  - ¿Son compatibles los nuevos métodos o el código del supervisor con la versión anterior?

Información que se tiene que conocer antes de cambiar el archivo RTR

Algunas modernizaciones del tipo de recurso no implican la utilización de nuevos métodos o código del supervisor. Por ejemplo, una modernización de tipo de recurso puede cambiar sólo el valor predeterminado o la capacidad de ajuste de una propiedad del recurso. Dado que no cambia el código de método, el único requisito para instalar la modernización es disponer de un nombre de ruta válido a un archivo RTR legible.

Si no es necesario volver a registrar el antiguo tipo de recurso, el nuevo archivo RTR puede sobrescribir la versión anterior. En caso contrario, se puede situar el archivo RTR en un nuevo nombre de ruta.

Si la modernización cambia el valor predeterminado o la capacidad de ajuste de una propiedad, el método `Validate` de la nueva versión puede verificar en el momento de la migración que los atributos de la propiedad sean válidos para el nuevo tipo de recurso. Si la modernización cambia los atributos `min`, `max` o `type` de una propiedad, el comando `scrgadm` valida automáticamente estas limitaciones en el momento de la migración.

La documentación de modernización debe indicar cualquier atributo nuevo predeterminado de las propiedades. La documentación debe informar al administrador del sistema de que las propiedades de recurso se pueden modificar según los valores adecuados con el mismo comando que edita la propiedad `Type_version` para modernizar el recurso a la nueva versión del tipo de recurso.

Si la modernización agrega o elimina propiedades, es probable que haya que modificar también algunos métodos de rellamada o código del supervisor.

Cambio del código del supervisor

Si el código del supervisor es el único cambio del tipo de recurso actualizado, la instalación del paquete podrá sobrescribir los binarios del supervisor. La documentación debe indicarle al administrador del sistema que suspenda la supervisión antes de instalar el paquete nuevo.

Cambio del código del método

Si el código del método es el único cambio del tipo de recurso actualizado, es importante determinar si el nuevo código del método es compatible con la versión anterior. Esto determina si el nuevo código del método se debe almacenar en un nuevo nombre de ruta o si se pueden sobrescribir los métodos antiguos.

Si los nuevos métodos `Stop`, `Postnet_stop` y `Fini` (si se declaran) se pueden aplicar a los recursos que se inicializaron o iniciaron por las versiones anteriores de los métodos `Start`, `Prenet_stop` o `Init` será posible sobrescribir los métodos antiguos con los nuevos.

Si el nuevo código del método no es compatible con la versión anterior, será necesario detener o desconfigurar un recurso con las versiones anteriores de los métodos antes de poder migrarlo al tipo de recurso modernizado. Si los nuevos métodos sobrescriben los antiguos, es posible que sea necesario apagar (y probablemente dejar sin gestión) todos los recursos del tipo antes de realizar la modernización del tipo de recurso. Si los nuevos métodos se guardan aparte de los antiguos (y ambos resultan accesibles inmediatamente), aún sin la compatibilidad con versiones anteriores será posible instalar la nueva versión del tipo de recurso y modernizar los recursos uno a uno.

Aunque los métodos nuevos sean compatibles con versiones anteriores, es posible que haya que modernizar los recursos de uno en uno para utilizar los nuevos métodos, mientras otros recursos siguen usando los métodos antiguos. Sigue siendo necesario guardar los métodos nuevos en un directorio aparte, en lugar de sobrescribir los antiguos.

Ello facilita el regreso a la versión anterior del tipo de recurso si surgiera un problema con la versión nueva.

Una forma de empaquetado consiste en incluir todas las versiones anteriores que siguen estando admitidas en el nuevo paquete. Esto permite que la nueva versión del paquete sustituya la versión anterior, sin sobrescribir ni suprimir las rutas antiguas a los métodos. El desarrollador del tipo de recurso deberá decidir cuántas versiones anteriores deben estar admitidas.

Nota – No se recomienda sobrescribir métodos `pkgrm`/`pkgadd` en un nodo que esté actualmente en el clúster, ya que si RGM llama a un método cuando éste no está accesible en disco, se pueden producir resultados inesperados. La eliminación o sustitución del binario de un método en ejecución puede provocar también resultados inesperados.

CAPÍTULO 4

Referencia de la API de gestión de recursos

Este capítulo proporciona una referencia a las funciones de acceso y los métodos de rellamada que forman la API de gestión de recursos (RMAPI). Enumera y describe brevemente cada función y método. Sin embargo, la referencia definitiva para estas funciones y métodos son las páginas de comando man de RMAPI.

La información de este capítulo incluye:

- «Métodos de acceso a RMAPI» en la página 78: en forma de comandos de secuencias de shell y funciones de C
  - `scha_resource_get`(1HA), `scha_resource_close`(3HA), `scha_resource_get`(3HA), `scha_resource_open`(3HA)
  - `scha_resource_setstatus`(1HA), `scha_resource_setstatus`(3HA)
  - `scha_resourcetype_get`(1HA), `scha_resourcetype_close`(3HA), `scha_resourcetype_get`(3HA), `scha_resourcetype_open`(3HA)
  - `scha_resourcegroup_get`(1HA), `scha_resourcegroup_get`(3HA), `scha_resourcegroup_close`(3HA), `scha_resourcegroup_open`(3HA)
  - `scha_control`(1HA), `scha_control`(3HA)
  - `scha_cluster_get`(1HA), `scha_cluster_close`(3HA), `scha_cluster_get`(3HA), `scha_cluster_open`(3HA)
  - `scha_cluster_getlogfacility`(3HA)
  - `scha_cluster_getnodename`(3HA)
  - `scha_strerror`(3HA)
- «Métodos de rellamada de RMAPI» en la página 83: se describen en la página de comando man `rt_callbacks`(1HA).
  - `Start`
  - `Stop`
  - `Init`
  - `Fini`
  - `Boot`

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

Métodos de acceso a RMAPI

La API proporciona funciones para acceder a las propiedades tanto de los recursos como del grupo y tipo de recursos, además de otra información del clúster. Estas funciones se suministran en forma de comandos del shell y funciones de C, lo que permite que los suministradores de tipos de recursos implementen programas de control en forma de secuencias del shell o programas de C.

Comandos del shell de RMAPI

Los comandos del shell se usan en las implementaciones de las secuencias del shell de los métodos de rellamada para los tipos de recursos que representan los servicios controlados por RGM del clúster. Puede utilizar estos comandos para:

- Acceder a información sobre recursos, tipos de recursos, grupos de recursos y clústers
- Utilizarlos con un supervisor para establecer las propiedades `Status` y `Status_msg` de un recurso
- Solicitar el reinicio o la reubicación de un grupo de recursos

---

Nota – Esta sección proporciona descripciones breves de los comandos del shell; las páginas de comando man individuales de la sección 1HA sirven de referencia definitiva para los comandos del shell. Cada comando dispone de una página de comando man del mismo nombre salvo que se indique lo contrario.

Comandos de los recursos de RMAPI

Estos comandos permiten acceder a información sobre un recurso o establecer las propiedades `Status` y `Status_msg` de un recurso.

`scha_resource_get`
: Accede a información sobre un recurso o tipo de recurso controlado por RGM. Proporciona la misma información que la función `scha_resource_get()`.

`scha_resource_setstatus`
Establece las propiedades `Status` y `Status_msg` de un recurso controlado por RGM. El supervisor del recurso lo utiliza para indicar el estado del recurso, según lo percibe. Realiza la misma tarea que la función de C `scha_resource_setstatus()`.

Nota – `scha_resource_setstatus()` es especialmente útil para un supervisor de recursos, cualquier programa puede invocarlo.

Comando del tipo de recurso

Este comando accede a información sobre un tipo de recurso registrado con RGM.

`scha_resourcetype_get`
Realiza la misma tarea que la función de C `scha_resource_setstatus()`.

Comandos del grupo de recursos

Estos comandos permiten acceder a información acerca de un grupo de recursos o reiniciarlo.

`scha_resourcegroup_get`
Accede a información sobre un grupo de recursos controlado por RGM. Realiza la misma tarea que la función de C `scha_resourcetype_get()`.

`scha_control`
Solicita el reinicio de un grupo de recursos controlado por RGM o su reubicación en un nodo diferente. Realiza la misma tarea que la función de C `scha_control()`.

Comando del clúster

Este comando permite acceder a información acerca de un clúster, como nombres de nodo, ID, estados, el nombre de clúster, grupos de recursos, etc.

`scha_cluster_get`
Este comando proporciona la misma información que la función de C `scha_cluster_get()`.

Funciones de C

Las funciones de C se deben usar en las implementaciones del programa de C de los métodos de rellamada para los tipos de recursos que representan servicios controlados por RGM del clúster. Puede utilizar estas funciones para:

- Acceder a información sobre recursos, tipos de recursos, grupos de recursos y clústers
- Utilizarlos con un supervisor para establecer las propiedades `Status` y `Status_msg` de un recurso
- Solicitar el reinicio o la reubicación de un grupo de recursos
- Convertir un código de error en un mensaje de error adecuado

---

Nota – Esta sección proporciona descripciones breves de las funciones de C; las páginas de comando man individuales de `(3HA)` sirven de referencia definitiva para las funciones de C. Cada función tiene una página de comando man del mismo nombre, salvo que se indique lo contrario. Consulte la página de comando man de `scha_calls`(3HA) para obtener información sobre los argumentos de salida y códigos de retorno de las funciones de C.

Funciones de los recursos

Estas funciones permiten acceder a información sobre un recurso gestionado por RGM o indicar el estado del recurso, como lo percibe el supervisor.

`scha_resource_open()`, `scha_resource_get()` y `scha_resource_close()`
Estas funciones, juntas, permiten acceder a información sobre un recurso gestionado por RGM. La función `scha_resource_open()` inicializa el acceso a un recurso y devuelve un manejador para `scha_resource_get()`, que accede a la información del recurso. La función `scha_resource_close()` anula el manejador y libera la memoria asignada para los valores de retorno de `scha_resource_get()`.

Un recurso puede cambiar, mediante una reconfiguración del clúster o una acción administrativa, después de que `scha_resource_open()` devuelva el manejador del recurso, en cuyo caso la información que obtiene `scha_resource_get()` a través del manejador puede no ser precisa. En el caso de una reconfiguración del clúster o una acción administrativa en un recurso, RGM devuelve el código de error `scha_err_seqid` a `scha_resource_get()` para indicar que la información del recurso puede haber cambiado. Se trata de un mensaje de error de tipo no fatal; la función devuelve información de forma satisfactoria. En consecuencia, puede ignorar el mensaje y aceptar la información que ha devuelto o cerrar el manejador actual y abrir uno nuevo para acceder a la información del recurso.

Una sola página de comando man describe estas tres funciones; es accesible a través de cualquiera de las tres funciones, `scha_resource_open`(3HA), `scha_resource_get`(3HA) o `scha_resource_close`(3HA).

`scha_resource_setstatus()`
Establece las propiedades `Status` y `Status_msg` de un recurso controlado por RGM. El supervisor del recurso utiliza esta función para indicar el estado del recurso.

Nota – `scha_resource_setstatus()` es especialmente útil para un supervisor de recursos, cualquier programa puede invocarlo.

Funciones del tipo de recurso

Estas funciones, juntas, permite acceder a información sobre un tipo de recurso registrado con RGM.

`scha_resourcetype_open()`, `scha_resourcetype_get()`, `scha_resourcetype_close()`

La función `scha_resourcetype_open()` inicializa el acceso a un recurso y devuelve un manejador para `scha_resourcetype_get()` que accede a la información de tipo de recurso. La función `scha_resourcetype_close()` invalida el manejador y libera la memoria asignada para los valores de retorno de `scha_resourcetype_get()`.

Un tipo de recurso puede cambiar, mediante una reconfiguración de clúster o un acción administrativa, después de que `scha_resourcetype_open()` devuelva el manejador del tipo de recurso, en cuyo caso la información que obtiene `scha_resourcetype_get()` a través del manejador puede no ser precisa. En el caso de una reconfiguración del clúster o una acción administrativa en un tipo de recurso, RGM devuelve el código de error `scha_err_seqid` a `scha_resourcetype_get()` para indicar que la información acerca del tipo de recurso puede haber cambiado. Se trata de un mensaje de error de tipo no fatal; la función devuelve información de forma satisfactoria. En consecuencia, puede ignorar el mensaje y aceptar la información que ha devuelto o cerrar el manejador actual y abrir uno nuevo para acceder a la información del tipo de recurso.

Una sola página de comando man describe estas tres funciones; es accesible a través de cualquiera de las tres funciones individuales, `scha_resourcetype_open`(3HA), `scha_resourcetype_get`(3HA) o `scha_resourcetype_close`(3HA).

Funciones del grupo de recursos

Estas funciones permiten acceder a información acerca del grupo de recursos o reiniciarlo.

`scha_resourcegroup_open`(3HA), `scha_resourcegroup_get`(3HA) y `scha_resourcegroup_close`(3HA)

Estas funciones, juntas, permiten acceder a información sobre un grupo de recursos gestionado por RGM. La función `scha_resourcegroup_open()` inicializa el acceso a un grupo de recursos y devuelve un manejador para `scha_resourcegroup_get()`, que accede a la información del grupo de

recursos. La función `scha_resourcegroup_close()` invalida el manejador y libera la memoria asignada para los valores de retorno de `scha_resourcegroup_get()`.

Un grupo de recursos puede cambiar, mediante una reconfiguración del clúster o una acción administrativa, después de que `scha_resourcegroup_open()` devuelva el manejador del grupo de recursos, en cuyo caso la información que obtiene `scha_resourcegroup_get()` a través del manejador puede no ser precisa. En el caso de una reconfiguración del clúster o una acción administrativa en un grupo de recursos, RGM devuelve el código de error `scha_err_seqid` a `scha_resourcegroup_get()` para indicar que la información acerca del grupo de recursos puede haber cambiado. Se trata de un mensaje de error de tipo no fatal; la función devuelve información de forma satisfactoria. En consecuencia, puede ignorar el mensaje y aceptar la información que ha devuelto o cerrar el manejador actual y abrir uno nuevo para acceder a la información del grupo de recursos.

`scha_control`(3HA)
Solicita el reinicio de un grupo de recursos controlado por RGM o su reubicación en un nodo diferente.

Funciones del clúster

Estas funciones permiten acceder a información sobre un clúster o la devuelven.

`scha_cluster_open`(3HA), `scha_cluster_get`(3HA), `scha_cluster_close`(3HA)
Estas funciones, juntas, permiten acceder a información sobre un clúster, como los nombres de nodos, ID, estados, nombre del clúster, grupos de recursos, etc.

Un clúster puede cambiar, mediante una reconfiguración o una acción administrativa, después de que `scha_cluster_open()` devuelva el manejador del clúster, en cuyo caso la información que obtiene `scha_cluster_get()` a través del manejador puede no ser precisa. En el caso de una reconfiguración del clúster o una acción administrativa, RGM devuelve el código de error `scha_err_seqid` a `scha_cluster_get()` para indicar que la información del clúster puede haber cambiado. Se trata de un mensaje de error de tipo no fatal; la función devuelve información de forma satisfactoria. En consecuencia, puede ignorar el mensaje y aceptar la información que ha devuelto o cerrar el manejador actual y abrir uno nuevo para acceder a la información del clúster.

`scha_cluster_getlogfacility`(3HA)
Devuelve el número del recurso de registro del sistema que se está usando como registro del clúster. Utiliza el valor devuelto con la función `syslog()` de Solaris para registrar mensajes de estado y eventos en el registro del clúster.

`scha_cluster_getnodename`(3HA)
Devuelve el nombre del nodo del clúster en el que se invoca la función.

Función de utilidad

Esta función convierte un código de error en un mensaje de error.

`scha_strerror`(3HA)
: Traduce un código de error, devuelto por una de las funciones `scha_`, al mensaje de error adecuado. Utilice esta función con `logger` para introducir los mensajes en el registro del sistema (`syslog`).

Métodos de rellamada de RMAPI

Los métodos de rellamada son los elementos clave que proporciona la API para implementar un tipo de recurso, ya que permiten que RGM controle los recursos del clúster en caso de un cambio en los miembros del clúster, como un arranque o una caída del nodo.

Nota – RGM ejecuta los métodos de rellamada con permisos de root, porque los programas cliente controlan los servicios de alta disponibilidad (HA) del sistema del clúster. Instale y administre estos métodos con permisos y propiedad de archivos restrictivos. Específicamente, déles un propietario con privilegios, como `bin` o `root`, y no los haga grabables.

Esta sección describe los argumentos del método de rellamada y los códigos de salida y enumera y describe los métodos de rellamada de las siguientes categorías:

- Métodos de control e inicialización
- Métodos de soporte administrativo
- Métodos relacionados con la red
- Métodos de control del supervisor

Nota – Esta sección proporciona descripciones breves de los métodos de rellamada, incluidos el punto en el que se invoca el método y el efecto esperado en el recurso; la página de comando man `rt_callbacks`(1HA) es la referencia definitiva para los métodos de rellamada.

Argumentos del método

La sintaxis de los métodos de rellamada de RGM es:

método `-R` *nombre_recurso* `-T` *nombre_tipo* `-G` *nombre_grupo*

El método es el nombre de ruta del programa que se registra como `Start`, `Stop` u otra rellamada. Los métodos de rellamada de un tipo de recurso se declaran en su archivo de registro.

Todos los argumentos del método de rellamada se pasan como valores con indicadores, donde `-R` indica el nombre de la instancia de recursos, `-T` indica el tipo del recurso y `-G` indica el grupo en el que se configura el recurso. Utilice los argumentos con funciones de acceso para recuperar información sobre el recurso.

El método `Validate` se invoca con argumentos adicionales (los valores de la propiedad del recurso y del grupo de recursos en el que se invoca).

Consulte `scha_calls`(3HA) para obtener más información.

Códigos de salida

Todos los métodos de rellamada tienen los mismos códigos de salida para especificar el efecto de la invocación del método en el estado del recurso. La página de comando man `scha_calls`(3HA) describe todos estos códigos de salida. Éstos son:

- 0: el método ha sido satisfactorio
- Cualquier valor diferente de cero: el método no ha sido satisfactorio

RGM también maneja los fallos anómalos en la ejecución del método de rellamada, como volcados de núcleo central y la finalización del tiempo de espera.

Las implementaciones del método deben emitir la información sobre fallos utilizando `syslog` en cada nodo. La salida que se graba en `stdout` o `stderr` no está garantizado que se entregue al usuario (aunque aparezca actualmente en la consola del nodo local).

Métodos de rellamada de control e inicialización

Los métodos primarios de rellamada de control e inicialización inician y detienen un recurso. Otros métodos ejecutan un código de inicialización y terminación en un recurso.

`Start`
: Este método obligatorio se invoca en un nodo del clúster cuando el grupo de recursos que contiene el recurso se pone en línea en ese nodo. Este método activa el recurso en ese nodo.

  Un método `Start` no debe salir hasta que el recurso que activa se haya iniciado y esté disponible en el nodo local. Por tanto, antes de salir, el método `Start` debe interrogar al recurso para determinar si ya se ha iniciado. Se debe establecer un valor de tiempo de espera suficientemente largo para este método. Por ejemplo, algunos recursos, como daemons de base de datos, tardan más tiempo en empezar, por lo que el valor del tiempo de espera de estos métodos debe ser mayor.

La forma en que RGM responde a un fallo del método `Start` depende de la configuración de la propiedad `Failover_mode`.

La propiedad `START_TIMEOUT` del archivo de registro del tipo de recurso establece el valor de tiempo de espera para un método `Start` de un recurso.

`Stop`
Este método obligatorio se invoca en un nodo del clúster cuando el grupo de recursos que contiene el recurso se pone fuera de línea en ese nodo. Este método desactiva el recurso si está activo.

Un método `Stop` no debe salir hasta que el recurso que controla haya detenido completamente toda su actividad en el nodo local y después de que haya cerrado todos los descriptores de archivos. En caso contrario, dado que RGM asume que el recurso se ha detenido, cuando en realidad sigue activo, se puede producir una corrupción de los datos. La forma más segura de evitar que los datos se corrompan es terminar todos los procesos en el nodo local relacionado con el recurso.

Antes de salir, el método `Stop` deberá interrogar al recurso para determinar si ya se ha detenido. Se debe establecer un valor de tiempo de espera suficientemente largo para este método. Por ejemplo, algunos recursos, como daemons de base de datos, tardan más tiempo en detenerse, por lo que el valor del tiempo de espera de estos métodos debe ser mayor.

La forma en que RGM responde a un fallo del método `Stop` depende de la configuración de la propiedad `Failover_mode` (consulte la Tabla A–2).

La propiedad `STOP_TIMEOUT` del archivo de registro del tipo de recurso establece el valor del tiempo de espera para un método `Stop` del recurso.

`Init`
Este método opcional se invoca para realizar una inicialización del recurso una sola vez, cuando éste pase a ser gestionado, tanto cuando el grupo de recursos al que pertenece pasa de un estado no gestionado a un estado gestionado como cuando se crea en un grupo de recursos que ya está gestionado. El método lo invoca en los nodos determinados la propiedad de recurso `Init_nodes`.

`Fini`
Este método opcional se invoca para realizar una reorganización detrás del recurso, cuando éste deja de estar gestionado, tanto cuando el grupo de recursos al que pertenece pasa a un estado no gestionado como cuando el recurso se elimina de un grupo de recursos gestionado. El método lo invoca en los nodos determinados la propiedad de recurso `Init_nodes`.

`Boot`
Este método opcional, similar a Init, se invoca para inicializar el recurso en los nodos que se unen al clúster después de que el grupo de recursos al que pertenece dicho recurso se haya puesto bajo la gestión de RGM. El método lo invoca en los nodos determinados la propiedad de recurso `Init_nodes`. El método `Boot` se invoca cuando el nodo se une o se vuelve a unir al clúster, tras un arranque o un rearranque.

Nota – Un fallo en los métodos `Init`, `Fini` o `Boot` hace que la función `syslog()` genere un mensaje de error, pero no afecta de ningún otro modo a la gestión del recurso por parte de RGM.

Métodos de soporte administrativo

Las acciones administrativas en los recursos incluyen la configuración y modificación de las propiedades de los recursos. Los métodos de rellamada `Validate` y `Update` permiten que una implementación de tipo de recurso enlace con estas acciones administrativas.

`Validate`
: Este método opcional se invoca cuando se crea un recurso y cuando una acción administrativa actualiza las propiedades del recurso o el grupo de recursos que lo contiene. Esta invocación se realiza en el conjunto de nodos del clúster señalado por la propiedad Init_nodes del tipo de recurso antes de que se aplique la creación o actualización; un código de salida no satisfactorio desde el método en cualquier nodo hace que se cancele la creación o actualización.

  `Validate` se invoca sólo cuando las propiedades del recurso o grupo de recursos se modifican con una acción administrativa, no cuando RGM establece las propiedades ni cuando un supervisor establece las propiedades del recurso `Status` y `Status_msg`.

`Update`
: Este método opcional se invoca para notificar un recurso en ejecución cuyas propiedades se hayan cambiado después de que una acción de administración haya configurado satisfactoriamente las propiedades de un recurso o su grupo; se invoca en los nodos en los que el recurso está en línea. El método utiliza las funciones de acceso de API para leer los valores de propiedad que puedan afectar a un recurso activo y ajusta el recurso en ejecución según corresponda.

  El fallo del método `Update` hace que la función `syslog()` genere un mensaje de error, pero no afecta de ningún otro modo a la gestión del recurso por parte del RGM.

Métodos de rellamada relacionados con la red

Los servicios que usan los recursos de dirección de red pueden requerir que los inicios y las paradas se realicen en un orden determinado, dependiendo de la configuración de la dirección de red. Los siguientes métodos de rellamada opcionales, `Prenet_start` y `Postnet_stop`, permiten que una implementación de un tipo de recurso realice acciones especiales de arranque y apagado, antes y después de que se configure o desconfigure una dirección de red relacionada.

`Prenet_start`
Este método opcional se invoca para que realice acciones de inicio especiales antes de que se configuren las direcciones de red del mismo grupo de recursos.

`Postnet_stop`
Este método opcional se invoca para que realice acciones de apagado especiales después de que se desconfiguren las direcciones de red del mismo grupo de recursos.

Métodos de rellamada del control del supervisor

Una implementación del tipo de recurso puede incluir, opcionalmente, un programa para supervisar el rendimiento de un recurso, informar de su estado o tomar medidas cuando se produce un fallo de un recurso. Los métodos `Monitor_start`, `Monitor_stop` y `Monitor_check` admiten la implementación de un supervisor de recursos en una implementación del tipo de recurso.

`Monitor_start`
Este método opcional se invoca para iniciar un supervisor para el recurso, una vez que se haya iniciado éste.

`Monitor_stop`
Este método opcional se invoca para detener un supervisor de un recurso antes de que éste se detenga.

`Monitor_check`
Este método opcional se invoca para evaluar la fiabilidad de un nodo antes de que se reubique un grupo de recursos en el nodo. El método `Monitor_check` se debe implementar de forma que no entre en conflicto con la ejecución simultánea de otro método.

CAPÍTULO 5

Servicio de datos de ejemplo

Este capítulo describe un ejemplo del servicio de datos de Sun Cluster, HA-DNS, para la aplicación `in.named`. El daemon `in.named` es la implementación de Solaris del servicio de nombres de dominio (DNS). El servicio de datos de ejemplo demuestra cómo hacer que una aplicación tenga una alta disponibilidad, con la API de gestión de recursos.

Ésta admite una interfaz de secuencias de comandos del shell y una de programa C. La aplicación de ejemplo de este capítulo se escribe con aquélla.

La información de este capítulo incluye:

Información general del servicio de datos de ejemplo

El servicio de datos de ejemplo inicia, detiene, reinicia y conmuta la aplicación de DNS entre los nodos del clúster, en respuesta a eventos del clúster, como una acción administrativa o fallos de la aplicación o del nodo.

El reinicio de la aplicación lo gestiona la Prestación del supervisor de procesos (PMF). Si las terminaciones de la aplicación superan el recuento de fallos de la ventana de tiempo de fallos, el supervisor de fallos realiza una operación de recuperación de fallos del grupo de recursos que contiene el recurso de la aplicación a otro nodo.

El servicio de datos de ejemplo proporciona una supervisión de fallos en forma de un método `PROBE` que utiliza el comando `nslookup` para garantizar que la aplicación esté en buen estado. Si el análisis detecta un servicio de DNS colgado, intentará corregir la situación reiniciando localmente la aplicación de DNS. Si esto no mejora la situación y el análisis sigue detectando problemas en el servicio, intentará realizar una operación de recuperación de fallos del servicio a otro nodo del clúster.

El servicio de datos de ejemplo incluye, concretamente:

- Un archivo de registro del tipo de recurso que define las propiedades estáticas del servicio de datos.
- Un método de rellamada `Start` invocado por RGM para que inicie el daemon `in.named` cuando el grupo de recursos que contiene el servicio de datos HA-DNS se ponga en línea.
- Un método de rellamada `Stop` que invoca RGM para detener el daemon `in.named` cuando el grupo de recursos que contiene el HA-DNS pase a estar fuera de línea.
- Un supervisor de fallos para comprobar la disponibilidad del servicio, verificando que el servidor de DNS esté en ejecución. Un método `PROBE` definido por el usuario implementa el supervisor; los métodos de rellamada `Monitor_start` y `Monitor_stop` lo inician y detienen.
- Un método de rellamada `Validate` que invoca RGM para confirmar que el directorio de configuración del servicio está accesible.
- Un método de rellamada `Update` invocado por RGM para reiniciar el supervisor de fallos cuando el administrador del sistema cambie el valor de una propiedad del recurso.

Definición del archivo de registro del tipo de recurso

El archivo de registro del tipo de recurso (RTR) de este ejemplo define la configuración estática del tipo de recurso de DNS. Los recursos de este tipo heredan las propiedades definidas en el archivo RTR.

RGM lee la información del archivo RTR cuando el administrador de clúster registra el servicio de datos de HA-DNS.

Información general del archivo RTR

El archivo RTR sigue un formato bien definido. Las propiedades del recurso se definen primero en el archivo; después se definen las propiedades del recurso definidas por el sistema; finalmente, se definen las propiedades de extensión. Consulte la página de comando man `drt_reg`(4) y «Establecimiento del recurso y las propiedades del tipo de recurso» en la página 34 para obtener más información.

Esta sección describe las propiedades especificas del archivo RTR de ejemplo. Proporciona listados de partes diferentes del archivo. Para un listado completo del contenido del archivo RTR de ejemplo, consulte «Listado del archivo de registro del tipo de recurso» en la página 265.

Propiedades de tipo en el archivo RTR de ejemplo

El archivo RTR de ejemplo empieza con comentarios, seguidos de las propiedades del tipo de recurso que definen la configuración de HA-DNS, como se muestra en la lista siguiente.

```
#
# Copyright (c) 1998-2003 de Sun Microsystems, Inc.
# Reservados todos los derechos.
#
# Información de registro del servicio de nombres de dominio (DNS)
#

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

RESOURCE_TYPE = "ejemplo";
VENDOR_ID = SUNW;
RT_DESCRIPTION = "Servicio de nombres de dominio en 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;
```

Consejo – Debe declarar la propiedad `Resource_type` como la primera entrada del archivo RTR, de lo contrario, el registro del tipo de recurso no será satisfactorio.

Nota – RGM no diferencia entre mayúsculas y minúsculas en los nombres de las propiedades. La convención para las propiedades en los archivos RTR suministrados por Sun, salvo los nombres de métodos, es la utilización de mayúscula en el inicio del nombre y minúscula en el resto. Los nombres de métodos (y los atributos de las propiedades) contienen sólo mayúsculas.

A continuación se incluye alguna información sobre estas propiedades.

- El nombre del tipo de recurso se puede especificar sólo con la propiedad `Resource_type` (`sample`) o con `Vendor_id` como un prefijo con "." que lo separe del tipo de recurso (`SUNW.sample`).

  Si utiliza `Vendor_id`, use el símbolo bursátil de la empresa que define el tipo de recurso. El nombre de éste debe ser exclusivo, dentro del clúster.
- La propiedad `Rt_version` identifica la versión del servicio de datos de ejemplo según especifique el fabricante.
- La propiedad `API_version` identifica la versión de Sun Cluster. Por ejemplo, `API_version = 2`, indica que el servicio de datos se ejecuta en Sun Cluster, versión 3.0.
- `Failover = TRUE` indica que el servicio de datos no se puede ejecutar en un grupo de recursos que pueda estar en línea en varios nodos simultáneamente.
- `RT_basedir` apunta a `/opt/SUNWsample/bin` como la ruta del directorio para completar las rutas relativas, como las rutas de los métodos de rellamada.
- `Start`, `Stop`, `Validate`, etc., proporcionan las rutas a los programas de método de rellamada respectivos, invocados por RGM. Estas rutas son relativas al directorio que especifica `RT_basedir`.
- `Pkglist` identifica `SUNWsample` como el paquete que contiene la instalación del servicio de datos de ejemplo.

Las propiedades del tipo de recurso que no se especifican en este archivo RTR, como `Single_instance`, `Init_nodes` y `Installed_nodes` tienen su valor predeterminado. Consulte la Tabla A–1 para ver una lista completa de las propiedades del tipo de recurso, incluidos sus valores predeterminados.

El administrador del clúster no puede cambiar los valores especificados para las propiedades del tipo de recurso en el archivo RTR.

Propiedades del recurso en el archivo RTR de ejemplo

Por convención, las propiedades del recurso se declaran de acuerdo con las del tipo de recurso del archivo RTR; incluyen las propiedades definidas por el sistema, proporcionadas por Sun Cluster, y las propiedades de extensión definidas por el usuario. Para ambos tipo se pueden especificar varios atributos de propiedad proporcionados por Sun Cluster, como valores mínimo, máximo y predeterminado.

Propiedades definidas por el sistema en el archivo RTR

La lista siguiente muestra las propiedades definidas por el sistema en el archivo RTR de ejemplo.

```
# Una lista de declaraciones de propiedades del recurso entre llaves
# sigue a las declaraciones del tipo de recurso. La declaración del nombre de propiedad
# debe ser el primer atributo tras la llave abierta de cada entrada.

# Las propiedades <method>_timeout establecen el valor en segundos tras el cual
# RGM determina que la llamada del método no ha sido satisfactoria.

# El valor MIN para todos los tiempos de espera de los métodos es de 60 segundos.
# Así se impide que los administradores definan tiempos de espera menores, que no mejoran
# el rendimiento de las operaciones de conmutación/recuperación de fallos y pueden provocar
# acciones indeseadas de RGM (recuperaciones de fallos falsas, rearranque de nodo o
# desplazamiento del grupo de recursos a un estado ERROR_STOP_FAILED que requiera la
# intervención de un operador). Definir tiempos de espera demasiado cortos provoca una
# *disminución* en la disponibilidad global del servicio de datos.
{
   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;
}

# El número de reintentos que se va a realizar en un determinado periodo antes de
# determinar que la aplicación no se puede iniciar satisfactoriamente en este nodo.
{
        PROPERTY = Retry_Count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Definir Retry_Interval como múltiplo de 60, porque se convierte de segundos
# a minutos, en un redondeo. Por ejemplo, un valor de 50 (segundos)
# se convierte en 1 minuto. Utilizar esta propiedad para cronometrar el número
# de reintentos (Retry_Count).
{
        PROPERTY = Retry_Interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}

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

Aunque Sun Cluster proporciona las propiedades definidas por el sistema, es posible definir diferentes valores predeterminados con los atributos de propiedades del recurso. Consulte «Atributos de las propiedades de recursos» en la página 262 para ver una lista completa de los atributos disponibles para aplicarlos a las propiedades del recurso.

Tenga presente lo siguiente sobre las propiedades de los recursos definidas por el sistema en el archivo RTR de ejemplo:

- Sun Cluster proporciona un valor mínimo (1 segundo) y un valor predeterminado (3600 segundos) para todos los tiempos de espera. El archivo RTR de ejemplo cambia el mínimo a 60 y establece el valor predeterminado en 300 segundos. Un administrador del clúster puede aceptar este valor predeterminado o cambiar el valor del tiempo de espera (60 o superior). Sun Cluster no tiene un valor máximo permitido.
- Las propiedades `Thorough_Probe_Interval`, `Retry_count` y `Retry_interval` tienen el atributo `TUNABLE` definido en `ANYTIME`. Esto significa que el administrador del clúster puede cambiar el valor de estas propiedades, aunque el servicio de datos esté en ejecución. Estas propiedades las utiliza el supervisor de fallos implementado por el servicio de datos de ejemplo que implementa un método `Update` para detener y reiniciar el supervisor de fallos cuando éstas u otras propiedades de recurso se modifican mediante una acción administrativa. Consulte «Método `Update`» en la página 121.
- Las propiedades de recurso se clasifican como
  - *necesarias*: el administrador del clúster debe especificar un valor cuando cree un recurso;.
  - *opcionales*: si el administrador no especifica ningún valor, el sistema proporciona un valor predeterminado.
  - *condicionales*: RGM crea la propiedad sólo si está declarada en el archivo RTR.

  El supervisor de fallos del servicio de datos de ejemplo utiliza las propiedades condicionales `Thorough_probe_interval`, `Retry_count`, `Retry_interval` y `Network_resources_used` por lo que el desarrollador debe declararlas en el archivo RTR. Consulte la página de comando man `r_properties`(5) o «Propiedades de recurso» en la página 248 para obtener información sobre cómo se clasifican las propiedades.

Propiedades de extensión en el archivo RTR

Al final del archivo RTR de ejemplo se encuentran las propiedades de extensión:

```
# Propiedades de extensión

# El administrador del clúster debe definir el valor de esta propiedad para que apunte al
# directorio que contiene los archivos de configuración que utiliza la aplicación.
# Para esta aplicación, DNS, la ruta del archivo de configuración de DNS se especifica en
# PXFS (normalmente, named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "La ruta al directorio de configuración";
}

# Valor de tiempo de espera en segundos, antes de declarar que el análisis no ha sido
```

```
# satisfactorio.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 120;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Valor de tiempo de espera del análisis (segundos)";
}
```

El archivo RTR de ejemplo define dos propiedades de extensión, `Confdir` y `Probe_timeout`. La primera especifica la ruta al directorio de configuración de DNS que contiene el archivo `in.named` necesario para que el DNS funcione satisfactoriamente. Los métodos `Start` y `Validate` del servicio de datos de ejemplo utilizan esta propiedad para verificar que el directorio de configuración y el archivo `in.named` sean accesibles antes de iniciar el DNS.

Cuando el servicio de datos está configurado, el método `Validate` verifica si el nuevo directorio es accesible.

El método `PROBE` del servicio de datos de ejemplo no es un método de rellamada de Sun Cluster, sino un método definido por el usuario. Por tanto, Sun Cluster no proporciona una propiedad `Probe_timeout` para él. El desarrollador ha definido una propiedad de extensión en el archivo RTR para permitir que un administrador del clúster configure un valor Probe_timeout.

Funciones comunes para todos los métodos

Esta sección describe las siguientes funciones que se emplean en todos los métodos de rellamada del servicio de datos de ejemplo:

- «Identificación del intérprete de comandos y exportación de la ruta» en la página 97.
- «Declaración de las variables `PMF_TAG` y `SYSLOG_TAG`» en la página 97.
- «Análisis de los argumentos de función» en la página 98.
- «Generación de mensajes de error» en la página 100.
- «Obtención de la información de la propiedad» en la página 100.

Identificación del intérprete de comandos y exportación de la ruta

La primera línea de una secuencia de comandos del shell debe identificar el intérprete de éstos. Cada una de las secuencias de comandos de los métodos del servicio de datos de ejemplo identifica el intérprete de comandos como se indica a continuación:

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

Todas las secuencias de comandos del método de la aplicación de ejemplo exportan la ruta a los binarios y bibliotecas de Sun Cluster, en lugar de tener en cuenta la configuración de `PATH` del usuario.

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

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

Declaración de las variables `PMF_TAG` y `SYSLOG_TAG`

Todas las secuencias de comandos del método (salvo `Validate`) utilizan `pmfadm` para iniciar (o detener) el servicio de datos o el supervisor, pasando el nombre del recurso. Cada secuencia de comandos define una variable, `PMF_TAG`, que puede pasarse a `pmfadm` para identificar el servicio de datos o el supervisor.

Del mismo modo, cada secuencia de comandos del método utiliza el comando `logger` para guardar los mensajes en el registro del sistema. Cada secuencia de comandos define una variable, `SYSLOG_TAG`, que se puede pasar a `logger` con la opción `-t` para identificar el tipo de recurso, el grupo de recursos y el nombre de recurso del recurso para el que se está registrando el mensaje.

Todos los métodos definen `SYSLOG_TAG` del mismo modo, como muestra el ejemplo siguiente. Los métodos `dns_probe`, `dns_svc_start`, `dns_svc_stop` y `dns_monitor_check` definen `PMF_TAG` como se indica a continuación (la utilización de `pmfadm` y `logger` se produce desde el método `dns_svc_stop`):

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

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # Enviar una señal SIGTERM al servicio de datos y esperar el 80% del
   # valor total del tiempo de espera.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
```

```
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.info \
           -t [$SYSLOG_TAG] \
           "${ARGV0} No se ha podido detener HA-DNS con SIGTERM; Reitentar \
            con SIGKILL"
```

Los métodos `dns_monitor_start`, `dns_monitor_stop` y `dns_update` definen `PMF_TAG` como se indica a continuación (la utilización de `pmfadm` se realiza desde el método `dns_monitor_stop`):

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

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

# Verificar si el supervisor está en ejecución; si así fuera, detenerlo.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

Análisis de los argumentos de función

RGM invoca todos los métodos de rellamada, salvo `Validate`, como se indica a continuación.

nombre_método `-R` *nombre_recurso* `-T` *nombre_tipo_recurso* `-G` *nombre_grupo_recurso*

El nombre del método es el de la ruta del programa que implementa el método de rellamada. Un servicio de datos especifica el nombre de la ruta de cada método en el archivo RTR. Estos nombres de ruta son relativos al directorio especificado por la propiedad `Rt_basedir` también en el archivo RTR. Por ejemplo, en el archivo RTR del servicio de datos de ejemplo, los nombres del directorio básico y del método se especifican como se indica a continuación.

```
RT_BASEDIR=/opt/SUNWsample/bin;
START = dns_svc_start;
STOP =  dns_svc_stop;
...
```

Todos los argumentos del método de rellamada se pasan como valores con indicadores, donde `-R` indica el nombre de la instancia de recursos, `-T`, el tipo del recurso, y `-G`, el grupo en el que se configura el recurso. Consulte la página de comando man `rt_callbacks`(1HA) para obtener más información sobre los métodos de rellamada.

Nota – El método `Validate` se invoca con argumentos adicionales (los valores de la propiedad del recurso y del grupo de recursos en el que se invoca). Consulte «Manejo de las actualizaciones de propiedades» en la página 116 para obtener más información.

Cada método de rellamada necesita una función para analizar los argumentos que se le pasan. Dado que las rellamadas pasan siempre por los mismos argumentos, el servicio de datos proporciona una única función de análisis para todas las rellamadas de la aplicación.

A continuación se muestra la función `parse_args()` empleada para los métodos de rellamada en la aplicación de ejemplo.

```
##############################################################################
# Analizar argumentos del programa.
#
function parse_args # [args ...]
{
      typeset opt

      while getopts 'R:G:T:' opt
      do
             case "$opt" in
             R)
                  # Nombre del recurso de DNS.
                  RESOURCE_NAME=$OPTARG
                  ;;
             G)
                  # Nombre del grupo de recursos en el que se ha configurado
                  # el recurso.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
             T)
                  # Nombre del tipo de recurso.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
             *)
                  logger -p ${SYSLOG_FACILITY}.err \
                  -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                  "ERROR: Opción $OPTARG desconocida"
                  exit 1
                      ;;
             esac
    done
}
```

Nota – Aunque el método `PROBE` de la aplicación de ejemplo está definido por el usuario (no es un método de rellamada de Sun Cluster), se invoca con los mismos argumentos que los métodos de rellamada. Por tanto, este método contiene una función de análisis idéntica a la que utilizan otros métodos de rellamada.

La función de análisis se invoca en `MAIN` como:

```
parse_args "$@"
```

Generación de mensajes de error

Se recomienda que los métodos de rellamada utilicen la función `syslog` para enviar mensajes de error a los usuarios finales. Todos los métodos de rellamada del servicio de datos de ejemplo utilizan la función `scha_cluster_get()` para recuperar el número del recurso `syslog` utilizado para el registro del clúster, como se indica a continuación:

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

El valor se almacena en una variable del shell, `SYSLOG_FACILITY`, y se puede utilizar como el recurso del comando `logger` para guardar mensajes en un registro del clúster. Por ejemplo, el método `Start` del servicio de datos de ejemplo recupera el recurso syslog y registra un mensaje que indica que el servicio de datos se ha iniciado, como se muestra a continuación:

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

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

Consulte la página de comando man `scha_cluster_get`(1HA) para obtener más información.

Obtención de la información de la propiedad

La mayoría de los métodos de rellamada necesitan obtener información sobre las propiedades de recurso y el tipo de recurso del servicio de datos. La API proporciona la función `scha_resource_get()` con este fin.

Hay dos tipos disponibles de propiedades de recurso: propiedades definidas por el sistema y propiedades de extensión; aquellas están predefinidas; éstas se definen en el archivo RTR.

Cuando se usa `scha_resource_get()` para obtener el valor de una propiedad definida por el sistema, se especifica el nombre de la propiedad con el parámetro `-O`. El comando sólo devuelve el *valor* de la propiedad. Por ejemplo, en el servicio de datos de ejemplo, el método `Monitor_start` necesita ubicar el programa de análisis para poder ejecutarlo; éste reside en el directorio básico del servicio de datos, señalado por la propiedad `RT_BASEDIR`. De esta manera el método `Monitor_start` recupera el valor de `RT_BASEDIR` y lo sitúa en la variable `RT_BASEDIR` como se muestra a continuación.

```
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAMÈ
```

Las propiedades de extension se deben especificar con la opción `-O` e indicar el nombre de la propiedad como el último parámetro. Para estas propiedades, el comando devuelve el *tipo* y el *valor* de la propiedad. Por ejemplo, en el servicio de datos de ejemplo, el programa de análisis recupera el tipo y el valor de la propiedad de extensión `probe_timeout` y después usa `awk` para poner el valor sólo en la variable del shell `PROBE_TIMEOUT` como se muestra a continuación.

```
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}'`
```

Control del servicio de datos

Un servicio de datos debe proporcionar un método `Start` o `Prenet_start` para activar el daemon de la aplicación en el clúster y un método `Stop` o `Postnet_stop` para detener el daemon de la aplicación en el clúster. El servicio de datos de ejemplo implementa los métodos `Start` y `Stop`. Consulte «Elección de los métodos `Start` y `Stop` que se van a utilizar» en la página 46 para obtener información sobre cuándo puede ser recomendable utilizar `Prenet_start` y `Postnet_stop` en su lugar.

Método `Start`

RGM invoca el método `Start` en un nodo de clúster cuando el grupo de recursos que contiene el recurso de servicio de datos se pone en línea en ese nodo o cuando el grupo de recursos ya está en línea y se activa el recurso. En la aplicación de ejemplo, el método `Start` activa el daemon `in.named` (DNS) en ese nodo.

En esta sección se describe los principales fragmentos del método `Start` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()`, ni la obtención del recurso syslog; éstos se describen en «Funciones comunes para todos los métodos» en la página 96.

Para obtener una lista completa del método `Start`, consulte «Método `Start`» en la página 268.

Información general sobre `Start`

Antes de intentar ejecutar el DNS, el método `Start` del servicio de datos de ejemplo comprueba si el directorio y el archivo de configuración (`named.conf`) están accesibles y disponibles. La información que contiene `named.conf` es esencial para que el DNS funcione correctamente.

Este método de rellamada utiliza la función de supervisión de procesos (`pmfadm`) para iniciar el daemon de DNS (`in.named`). Si éste sufriera una caída o no pudiera iniciarse, PMF intentaría iniciarlo un número de veces predeterminado durante un intervalo de tiempo definido, ambos especificados por las propiedades del archivo RTR del servicio de datos.

Comprobación de la configuración

Para funcionar, el DNS necesita información del archivo `named.conf`, en el directorio de configuración. En consecuencia, el método `Start` realiza varias comprobaciones de estado para verificar que el directorio y el archivo estén accesibles antes de intentar ejecutar el DNS.

La propiedad de extensión `Confdir` proporciona la ruta al directorio de configuración. La propiedad en sí se define en el archivo RTR. Sin embargo, el administrador del clúster especifica la ubicación real al configurar el servicio de datos.

En el servicio de datos del ejemplo, el método `Start` recupera la ubicación del directorio de configuración con la función `scha_resource_get()`.

Nota – Dado que `Confdir` es una propiedad de extensión, `scha_resource_get()` devuelve tanto el tipo como el valor. El comando `awk` recupera sólo el valor y lo pone en una variable del shell, `CONFIG_DIR`.

```
# Buscar el valor de Confdir definido por el administrador del sistema al
# agregar el recurso.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get devuelve el "tipo" y el "valor" de las
# propiedades de extensión. Obtener sólo el valor de la propiedad de extensión
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

El método `Start` utiliza a continuación el valor de `CONFIG_DIR` para verificar que el directorio esté accesible. Si no lo está, `Start` registra un mensaje de error y cierra con un estado de error. Consulte «Estado de salida de `Start`» en la página 104.

```
# Comprobar si $CONFIG_DIR está accesible.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} El directorio $CONFIG_DIR falta o no está montado"
   exit 1
fi
```

Antes de iniciar el daemon de la aplicación, este método realiza una comprobación final para verificar que el archivo `named.conf` esté presente. Si no lo esta, `Start` registra un mensaje de error y sale con un estado de error.

```
# Cambiar al directorio $CONFIG_DIR si hay nombres
# de ruta relativas en los archivos de datos.
cd $CONFIG_DIR

# Comprobar que el archivo named.conf esté presente en el directorio $CONFIG_DIR
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} El archivo $CONFIG_DIR/named.conf falta o está vacío"
   exit 1
fi
```

Inicio de la aplicación

Este método utiliza el recurso de gestor de procesos (`pmfadm`) para ejecutar la aplicación, ya que permite establecer el número de veces que se reiniciará la aplicación durante un marco temporal definido. El archivo RTR contiene dos propiedades, `Retry_count`, que especifica el número de veces que se intentará reiniciar una aplicación, y `Retry_interval`, que especifica el periodo de tiempo durante el cual se harán los intentos.

El método `Start` recupera los valores de `Retry_count` y `Retry_interval` con la función `scha_resource_get()` y guarda sus valores en variables del shell y después los pasa a `pmfadm` con las opciones `-n` y `-t`.

```
# Obtener el valor para el recuento de reintentos del archivo RTR.
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ
# Obtener el valor para el intervalo de reintentos del archivo RTR. Este valor se indica
# en segundos y se debe convertir a minutos para pasarlos a pmfadm. Observe que la conversión
# se redondea hacia arriba: 50 segundos se convierte en 1 minuto.
((RETRY_INTRVAL=`scha_resource_get -O Retry_Interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAMÈ / 60))

# Iniciar el daemon in.named bajo el control de PMF. Deje que se produzca una caída y que
# se reinicie $RETRY_COUNT veces en un periodo de $RETRY_INTERVAL; si se cae más
# a menudo, PMF dejará de intentar reiniciarlo.
# Si hay algún proceso ya registrado con la etiqueta
# <$PMF_TAG>, PMF enviará un mensaje de alerta indicando que el
# proceso ya está en ejecución.
```

```
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Registrar un mensaje que indique que se ha iniciado HA-DNS.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS iniciado satisfactoriamente"
fi
exit 0
```

Estado de salida de `Start`

Un método `Start` no debería salir satisfactoriamente hasta que la aplicación subyacente estuviera en ejecución y disponible, especialmente si hay otros servicios de datos que dependen de él. Una forma de comprobar si la salida será la correcta es consultar la aplicación para verificar si está en ejecución antes de salir del método `Start`. En el caso de una aplicación compleja, como una base de datos, establezca una valor suficientemente alto para la propiedad `Start_timeout` en el archivo RTR para permitir que la aplicación tenga tiempo de inicializarse y realizar una recuperación de una caída.

Nota – Dado que el recurso de aplicación, DNS, en el servicio de datos de ejemplo se ejecuta con rapidez, el servicio de datos de ejemplo no lo interroga para verificar que se está ejecutando antes de salir de forma satisfactoria.

Si este método no logra iniciar el DNS y sale con un estado de fallo, RGM comprueba la propiedad `Failover_mode`, que determina el modo de reaccionar. El servicio de datos de ejemplo no establece explícitamente la propiedad `Failover_mode`, por lo que esta propiedad tiene el valor predeterminado `NONE` (salvo que el administrador del sistema haya anulado el valor predeterminado y haya especificado otro). En ese caso, RGM no realiza ninguna acción, salvo establecer el estado del servicio de datos. La intervención del usuario es necesaria para que se produzca un reinicio en el mismo nodo o una operación de recuperación de fallos en un nodo diferente.

Método `Stop`

El método `Stop` se invoca en un nodo del clúster cuando el grupo de recursos que contiene el recurso HA-DNS se pone fuera de línea en ese nodo o si el grupo de recursos está en línea y se inhabilita el recurso. Este método detiene el daemon `in.named` (DNS) en ese nodo.

Esta sección describe los principales fragmentos del método `Stop` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()`, ni la obtención del recurso syslog; éstos se describen en «Funciones comunes para todos los métodos» en la página 96.

Para obtener una lista completa del método `Stop`, consulte «Método `Stop`» en la página 271.

Información general sobre `Stop`

Hay que tener presentes dos consideraciones fundamentales a la hora de intentar detener el servicio de datos. La primera es proporcionar un apagado ordenado. Enviar una señal de `SIGTERM` mediante `pmfadm` es la mejor forma de lograrlo.

La segunda consideración es garantizar que el servicio de datos se haya detenido realmente para evitar ponerlo en el estado `Stop_failed`. La mejor forma de hacerlo es enviar una señal de `SIGKILL` mediante `pmfadm`.

El método `Stop` del servicio de datos de ejemplo tiene presentes ambas consideraciones. Primero envía una señal de `SIGTERM`. Si ésta no logra detener el servicio de datos, envía una señal `SIGKILL`.

Antes de intentar detener el DNS, este método `Stop` verifica que el proceso esté realmente en ejecución. Si el proceso está en ejecución, `Stop` utiliza el recurso del supervisor de procesos (`pmfadm`) para detenerlo.

Está garantizado que el método `Stop` sea idempotente. Aunque RGM no debería invocar un método `Stop` por segunda vez sin haber iniciado antes el servicio de datos mediante una llamada a su método `Start`, sí puede invocar un método `Stop` en un recurso aunque éste no se haya iniciado nunca o se haya terminado solo. Por tanto, el método `Stop` sale de forma satisfactoria aunque el DNS no esté en ejecución.

Parada de la aplicación

El método `Stop` proporciona un enfoque en dos hileras para detener el servicio de datos: un enfoque ordenado, o suave, que utiliza una señal de `SIGTERM` a través de `pmfadm` y un enfoque duro o brusco, con la señal de `SIGKILL`. El método `Stop` obtiene el valor `Stop_timeout` (la cantidad de tiempo en el que el método `Stop` debe volver). `Stop` asigna entonces el 80% de este tiempo a una parada suave y el 15% a una parada brusca (el 5% queda reservado), como se muestra a continuación.

```
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))
```

El método `Stop` utiliza `pmfadm -q` para verificar que el daemon de DNS está en ejecución. Si es así, `Stop` utiliza primero `pmfadm -s` para enviar una señal `TERM` para terminar el proceso de DNS. Si esta señal no logra terminar el proceso pasado un 80%

del valor de tiempo de espera, `Stop` envía una señal `SIGKILL`. Si esta señal tampoco puede finalizar el proceso en un 15% del valor de tiempo de espera, el método registra un mensaje de error y sale con estado de error.

Si `pmfadm` termina el proceso, el método registra un mensaje que indica que el proceso se ha detenido y sale de forma satisfactoria.

Si el proceso de DNS no está en ejecución, el método registra un mensaje de que no está en ejecución y sale de forma satisfactoria de todas formas. El siguiente ejemplo de código muestra cómo `Stop` utiliza `pmfadm` para detener el proceso de DNS.

```
# Ver si in.named está en ejecución y, en caso afirmativo, terminarlo.
if pmfadm -q $PMF_TAG; then
   # Enviar una señal SIGTERM al servicio de datos y esperar un 80%
   # del valor total del tiempo de espera.
   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} No se ha podido detener HA-DNS con SIGTERM; Reintentarlo con \
           SIGKILL"

      # Dado que el servicio de datos no se ha detenido con una señal SIGTERM
      # usar SIGKILL ahora y esperar otro 15% del valor de tiempo de espera total.
      pmfadm -s $PMF_TAG -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG]
          "${ARGV0} No se ha podido detener HA-DNS; Salida NO SATISFACTORIA"

          exit 1
      fi
fi
else
   # El servicio de datos no se está ejecutando actualmente. Registrar un mensaje y
   # salir con resultado satisfactorio.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS no se ha iniciado"

   # Aunque HA-DNS no esté en ejecución, salir con resultado satisfactorio para
   # no poner el recurso del servicio de datos en el estado STOP_FAILED.

   exit 0

fi

# Se ha podido detener satisfactoriamente DNS. Registrar un mensaje y salir con
# resultado satisfactorio.
logger -p ${SYSLOG_FACILITY}.err \
    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
    "HA-DNS se ha detenido satisfactoriamente"
exit 0
```

Estado de salida de `Stop`

Un método `Stop` no debería salir satisfactoriamente hasta que la aplicación subyacente se hubiera detenido realmente, en especial cuando otros servicios de datos dependan de ella. De lo contrario se podrían corromper los datos.

En el caso de una aplicación compleja, como una base de datos, establezca un valor suficientemente alto para la propiedad `Stop_timeout` en el archivo RTR para permitir que la aplicación tenga tiempo de reorganizarse mientras se detiene.

Si este método no logra detener el DNS y sale con estado de fallo, RGM comprueba la propiedad `Failover_mode`, que determina el modo de reaccionar. El servicio de datos de ejemplo no establece explícitamente la propiedad `Failover_mode`, por lo que tiene el valor predeterminado `NONE` (salvo que el administrador del sistema haya anulado el valor predeterminado y haya especificado otro). En este caso, RGM no realiza ninguna acción, salvo establecer el estado del servicio de datos en `Stop_failed`. Se requiere la intervención del usuario para forzar la parada de la aplicación y borrar el estado `Stop_failed`.

Definición de un supervisor de fallos

La aplicación de ejemplo implementa un supervisor básico de fallos para supervisar la fiabilidad del recurso de DNS (`in.named`). El supervisor de fallos se compone de:

- `dns_probe`, un programa definido por el usuario que utiliza `nslookup` para comprobar que el recurso de DNS controlado por el servicio de datos de ejemplo esté en ejecución. Si no lo está, este método intenta reiniciarlo localmente o, dependiendo del número de intentos de reinicio, solicita que RGM reubique el servicio de datos en otro nodo.
- `dns_monitor_start`, un método de rellamada que ejecuta `dns_probe`. RGM lo invoca automáticamente después de que el servicio de datos de ejemplo se ponga en línea, cuando la supervisión está habilitada.
- `dns_monitor_stop`, un método de rellamada que detiene `dns_probe`. RGM lo invoca automáticamente antes de poner fuera de línea el servicio de datos de ejemplo.
- `dns_monitor_check`, un método de rellamada que invoca el método `Validate` para verificar que el directorio de configuración esté disponible cuando el programa `PROBE` no pueda realizar una operación de recuperación de fallos del servicio de datos a otro nodo.

Programa de análisis

El programa `dns_probe` implementa un proceso en ejecución permanente que comprueba que el recurso de DNS controlado por el servicio de datos de ejemplo esté en ejecución. `dns_probe` lo ejecuta el método `dns_monitor_start`, invocado automáticamente por RGM una vez el servicio de datos de ejemplo se ha puesto en línea. El servicio de datos lo detiene el método `dns_monitor_stop`, invocado por RGM antes de que el servicio de datos de ejemplo esté fuera de línea.

Esta sección describe los principales fragmentos del método `PROBE` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()` ni la obtención del recurso syslog; éstos describen en «Funciones comunes para todos los métodos» en la página 96.

Para obtener una lista completa del método `PROBE`, consulte «Programa `PROBE`» en la página 274.

Información general de análisis

El análisis se ejecuta en un bucle infinito. Utiliza `nslookup` para comprobar que se está ejecutando el recurso de DNS correcto. Si éste está en ejecución, el análisis reposa durante un intervalo establecido (que establece la propiedad definida por el sistema `Thorough_probe_interval`) y luego realiza otra comprobación. Si DNS no está en ejecución, este programa intenta reiniciarlo localmente o, dependiendo del número de intentos de reinicio, solicita que RGM reubique el servicio de datos en otro nodo.

Obtención de los valores de propiedad

Este programa necesita los valores de las propiedades siguientes:

- `Thorough_probe_interval`, para establecer el periodo de reposo del análisis
- `Probe_timeout`, para aplicar el valor de tiempo de espera del análisis en el comando `nslookup` que realiza el análisis
- `Network_resources_used`, para obtener la dirección IP en la que se está ejecutando DNS
- `Retry_count` y `Retry_interval`, para determinar el número de intentos de reinicio y el periodo durante el cual se deben contar
- `Rt_basedir`, para obtener el directorio que contiene el programa `PROBE` y la utilidad `gettime`

La función `scha_resource_get()` obtiene los valores de estas propiedades y los guarda en variables del shell, como se indica a continuación.

```
PROBE_INTERVAL=`scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ
```

```
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_NAMÈ

RETRY_COUNT=`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME
-G\
 $RESOURCEGROUP_NAMÈ

RETRY_INTERVAL=`scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME
-G\
 $RESOURCEGROUP_NAMÈ

RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G\
 $RESOURCEGROUP_NAMÈ
```

Nota – En el caso de propiedades definidas por el sistema, como `Thorough_probe_interval`, `scha_resource_get()` sólo devuelve el valor. En el caso de propiedades de extensión, como `Probe_timeout`, `scha_resource_get()` devuelve tanto el tipo como el valor. Utilice el comando `awk` para obtener sólo el valor.

Comprobación de la fiabilidad del servicio

El análisis en sí es un bucle `while` infinito de comandos `nslookup`. Antes del bucle while, se configura un archivo temporal para contener las respuestas de `nslookup`. Las variables *probefail* y *retries* se inicializan a 0.

```
# Configurar un archivo temporal para las respuestas de nslookup.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

El propio bucle `while`:

- Establece el intervalo de reposo para el análisis
- Utiliza `hatimerun` para ejecutar `nslookup`, pasando el valor `Probe_timeout` e identificando el sistema de destino
- Establece la variable *probefail* en función del éxito o fracaso del código de retorno de `nslookup`
- Si se establece *probefail* en 1 (fallo), verifica que la respuesta a `nslookup` haya provenido del servicio de datos de ejemplo y no de algún otro servidor de DNS

Éste es el código bucle `while`.

```
while :
do
```

```
   # El intervalo al que se debe ejecutar el análisis se especifica en la
   # propiedad THOROUGH_PROBE_INTERVAL. Por tanto, establecer el análisis en
   # reposo durante el tiempo que indica THOROUGH_PROBE_INTERVAL.
   sleep $PROBE_INTERVAL

   # Ejecutar un comando nslookup de la dirección IP en la que sirve DNS.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
> $DNSPROBEFILE 2>&1

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

   # Asegurarse de que la respuesta a nslookup provenga del servidor
   # de HA-DNS y no de otro servidor de nombres mencionado en el
   # archivo /etc/resolv.conf.
   if [ $probefail -eq 0 ]; then
# Obtener el nombre del servidor que ha respondido a la consulta de nslookup.
   SERVER=` awk ' $1=="Server:" { print $2 }' \
   $DNSPROBEFILE | awk -F. ' { print $1 } ' `
   if [ -z "$SERVER" ]; then
      probefail=1
      else
         if [ $SERVER != $DNS_HOST ]; then
            probefail=1
         fi
   fi
fi
```

Evaluación de reinicio frente a recuperación de fallos

Si la variable *probefail* es distinta de 0 (satisfactorio), significa que el comando `nslookup` agotó el tiempo de espera o que la respuesta ha provenido de un servidor diferente de DNS del servicio de ejemplo. En ambos casos, el servidor de DNS no funciona como debería y el supervisor de fallos invoca la función `decide_restart_or_failover()` para determinar si se debe reiniciar el servicio de datos localmente o solicitar que RGM reubique el servicio de datos en otro nodo. Si la variable *probefail* es 0, se genera un mensaje que indica que el análisis ha sido satisfactorio.

```
   if [ $probefail -ne 0 ]; then
         decide_restart_or_failover
   else
         logger -p ${SYSLOG_FACILITY}.err\
         -t [$SYSLOG_TAG]\
         "${ARGV0} Análisis del recurso HA-DNS satisfactorio"
   fi
```

La función `decide_restart_or_failover()` utiliza una ventana de tiempo (`Retry_interval`) y un recuento de fallos (`Retry_count`) para determinar si debe reiniciar el DNS localmente o solicitar que RGM reubique el servicio de datos en otro nodo. Implementa el siguiente código condicional (consulte la lista de códigos para `decide_restart_or_failover()` en «Programa `PROBE`» en la página 274).

- Si es el primer fallo, se reinicia el servicio de datos. Se registra un mensaje de error y se detiene el contador de la variable `retries`.
- Si no es el primer fallo, pero se ha superado la ventana de tiempo, se reinicia el servicio de datos. Se registra un mensaje de error, se pone a cero el contador y se desliza la ventana.
- Si el tiempo sigue dentro de la ventana y el contador de reintentos se ha superado, se produce una recuperación de fallos a otro nodo. Si la operación de recuperación de fallos no es satisfactoria, se registra un error y se sale del programa de análisis con el estado 1 (fallo).
- Si el tiempo sigue dentro de la ventana, pero no se ha superado el contador de reintentos, se reinicia el servicio de datos. Se registra un mensaje de error y se detiene el contador de la variable `retries`.

Si el número de reinicios alcanza el límite durante el intervalo de tiempo, la función solicita que RGM reubique el servicio de datos en otro nodo. Si el número de reinicios está dentro del límite o se ha superado el intervalo, por lo que el recuento vuelve a empezar, la función intenta reiniciar DNS en el mismo nodo. Sobre esta función debe tener en cuenta lo siguiente:

- La utilidad `gettime` se utiliza para realizar un seguimiento del tiempo entre los reinicios. Se trata de un programa de C que reside en el directorio (`Rt_basedir`).
- Las propiedades definidas por el sistema `Retry_count` y `Retry_interval` determinan el número de intentos de reinicio y el intervalo en el que se deben realizar. Estas propiedades toman el valor predeterminado de 2 intentos en un periodo de 5 minutos (300 segundos) en el archivo RTR, aunque el administrador del clúster puede cambiarlo.
- La función `restart_service()` se invoca para reiniciar el servicio de datos en el mismo nodo. Consulte la sección siguiente, «Reinicio del servicio de datos» en la página 111, para obtener información sobre esta función.
- La función de API `scha_control()`, con la opción `GIVEOVER`, pone el grupo de recursos que contiene el servicio de datos de ejemplo fuera de línea y lo vuelve a poner en línea en otro nodo.

Reinicio del servicio de datos

La función `restart_service()` la invoca `decide_restart_or_failover()` para intentar reiniciar el servicio de datos en el mismo nodo. Esta función realiza lo siguiente:

- Determina si el servicio de datos sigue registrado bajo PMF. Si el servicio sigue registrado, la función:

- Obtiene el nombre del método `Stop` y el valor `Stop_timeout` del servicio de datos.
- Utiliza `hatimerun` para ejecutar el método `Stop` para el servicio de datos, pasando el valor `Stop_timeout`.
- (Si el servicio de datos se detiene satisfactoriamente) obtiene el nombre del método `Start` y el valor `Start_timeout` para el servicio de datos.
- Utiliza `hatimerun` para ejecutar el método `Start` para el servicio de datos, pasando el valor `Start_timeout`.

- Si el servicio de datos ya no está registrado bajo PMF, ello quiere decir que el servicio de datos ha superado el número máximo de reintentos permitidos según PMF, por lo que la función `scha_control()` se invoca con la opción `GIVEOVER` para realizar una operación de recuperación de fallos del servicio de datos a otro nodo.

```
function restart_service
{

        # Para reiniciar el servicio de datos, comprobar primero que
        # éste siga registrado en PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Dado que TAG para el servicio de datos sigue registrado
                # en PMF, detener el servicio de datos y volver a iniciarlo.

                # Obtener el nombre del método Stop y el valor de STOP_TIMEOUT
                # para este recurso.
                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} Método de parada no satisfactorio."
                        return 1
                fi

                # Obtener el nombre del método START y el valor
                # de START_TIMEOUT para este recurso.
                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} Método de inicio no satisfactorio."
                return 1
        fi


else
        # La ausencia de TAG para el servicio de datos
        # implica que el servicio de datos ya ha superado el
        # número máximo de reintentos permitidos en PMF.
        # No intentar reiniciar el servicio de datos otra vez;
        # intentar realizar una operación de recuperación de
        # fallos a otro nodo del clúster.
        scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                -R $RESOURCE_NAME
fi

return 0
}
```

Consultar el estado de salida

El programa `PROBE` del servicio de datos de ejemplo sale con un fallo si los intentos de reinicio local han fallado y el intento de realizar una recuperación de fallos a otro nodo también. Registra el mensaje “Intento de recuperación de fallos no satisfactorio”.

Método `Monitor_start`

RGM invoca el método `Monitor_start` para ejecutar el método `dns_probe` después de que el servicio de datos de ejemplo se ponga en línea.

Esta sección describe los principales fragmentos del método `Monitor_start` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()` ni la obtención del recurso syslog; éstos describen en «Funciones comunes para todos los métodos» en la página 96.

Para una lista completa del método `Monitor_start`, consulte «Método `Monitor_start`» en la página 280.

Información general de `Monitor_start`

Este método utiliza el recurso del supervisor de procesos (`pmfadm`) para ejecutar el análisis.

Inicio del análisis

El método `Monitor_start` obtiene el valor de la propiedad `Rt_basedir` para construir el nombre completo de la ruta del programa `PROBE`. Este método ejecuta el análisis con la opción de reintentos infinitos de `pmfadm` (-n -1, -t -1), lo que significa que si el análisis no se inicia, PMF intenta iniciarlo un número infinito de veces durante un periodo de tiempo infinito.

```
# Buscar dónde reside el programa de análisis mediante la obtención
# del valor de la propiedad RT_BASEDIR del recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAMÈ

# Iniciar el análisis del servicio de datos en PMF. Utilizar la opción de reintentos
# infinitos para iniciar el análisis. Pasar el nombre de recurso, tipo y grupo al
# programa de análisis.
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
   $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME
```

Método `Monitor_stop`

RGM invoca el método `Monitor_stop` para detener la ejecución de `dns_probe` cuando el servicio de datos de ejemplo se pone fuera de línea.

Esta sección describe los principales fragmentos del método `Monitor_stop` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()` ni la obtención del recurso syslog; éstos describen en «Funciones comunes para todos los métodos» en la página 96.

Para una lista completa del método `Monitor_stop`, consulte «Método `Monitor_stop`» en la página 282.

Información general de `Monitor_stop`

Este método utiliza el recurso del supervisor de procesos (`pmfadm`) para ver si el análisis está en ejecución y, en caso de que lo esté, detenerlo.

Parada del supervisor

El método `Monitor_stop` utiliza `pmfadm -q` para ver si el análisis está en ejecución y, si así fuera, utiliza `pmfadm -s` para detenerlo. Si el análisis ya está detenido, el método sigue saliendo de forma satisfactoria, lo que garantiza la idempotencia del método.

```
# Consultar si el supervisor está en ejecución. Si es así, terminarlo.
if pmfadm -q $PMF_TAG; then
   pmfadm -s $PMF_TAG KILL
```

```
   if [ $? -ne 0 ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} No se puede detener el supervisor del recurso " \
            $RESOURCE_NAME
           exit 1
   else
         # se ha detenido satisfactoriamente el supervisor. Registrar un mensaje.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Supervisor del recurso " $RESOURCE_NAME \
            " detenido satisfactoriamente"
   fi
fi
exit 0
```

Precaución – Asegúrese de usar la señal de `KILL` con `pmfadm` para detener el análisis y no una señal enmascarable, como `TERM`. En caso contrario, el método `Monitor_stop` puede quedar indefinidamente bloqueado hasta agotar el tiempo de espera. El motivo de este problema es que el método `PROBE` invoca `scha_control()` cuando hay que reiniciar o realizar una operación de recuperación de fallos del servicio de datos. Cuando `scha_control()` invoca `Monitor_stop` dentro del proceso de poner el servicio de datos fuera de línea, si `Monitor_stop` utiliza una señal enmascarable, se queda bloqueado esperando que `scha_control()` finalice y `scha_control()` se bloquea, esperando que `Monitor_stop` termine.

Estado de salida de `Monitor_stop`

El método `Monitor_stop` registra un mensaje de error si no puede detener el método `PROBE`. RGM pone el servicio de datos de ejemplo en el estado `MONITOR_FAILED` en el nodo primario, lo que puede enviar un aviso grave al nodo.

`Monitor_stop` no debería salir antes de que se haya detenido el análisis.

Método `Monitor_check`

RGM invoca el método `Monitor_check` cuando el método `PROBE` intenta realizar una recuperación de fallos del grupo de recursos que contiene el servicio de datos a otro nodo.

Esta sección describe los fragmentos principales del método `Monitor_check` para la aplicación de ejemplo, pero no las funciones comunes a todos los métodos de rellamada, como la función `parse_args()` ni la obtención del recurso syslog; éstos describen en «Funciones comunes para todos los métodos» en la página 96.

Para una lista completa del método `Monitor_check`, consulte «Método `Monitor_check`» en la página 284.

El método `Monitor_check` se debe implementar de forma que no entre en conflicto con la ejecución simultánea de otro método.

El método `Monitor_check` invoca el método `Validate` para verificar que el directorio de configuración de DNS está disponible en el nuevo nodo. La propiedad de extensión `Confdir` apunta al directorio de configuración de DNS. Por tanto, `Monitor_check` obtiene la ruta y el nombre del método `Validate` y el valor de `Confdir`. Pasa este valor a `Validate`, como muestra la lista siguiente.

```
# Obtener la ruta completa del método Validate desde
# la propiedad RT_BASEDIR del tipo de recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAMÈ

# Obtener el nombre del método Validate para este recurso.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtener el valor de la propiedad Confdir para iniciar el
# servicio de datos. Utilizar el nombre de recurso y grupo de recursos introducidos
# para obtener el valor Confdir establecido al agregar el recurso.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
 -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get devuelve el tipo y el valor para las propiedades de
# extensión. Utilizar awk para obtener sólo el valor de la propiedad de extensión.
CONFIG_DIR=`echo $config_info | awk `{print $2}'`

# Invocar el método validate para que el servicio de datos se pueda
# recuperar de un fallo a otro nodo satisfactoriamente.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

Consulte «Método `Validate`» en la página 117 para ver cómo verifica la aplicación de ejemplo la adecuación de un nodo para alojar el servicio de datos.

Manejo de las actualizaciones de propiedades

El servicio de datos de ejemplo implementa los métodos `Validate` y `Update` para permitir que un administrador del clúster maneje la actualización de las propiedades.

Método `Validate`

RGM invoca el método `Validate` cuando se crea un recurso y cuando una acción administrativa actualiza las propiedades del recurso o el grupo que lo contiene. RGM invoca `Validate` antes de que se apliquen la creación o la actualización y un código de salida fallido del método en cualquier nodo provoque la cancelación de la creación o actualización.

RGM invoca `Validate` sólo cuando se cambian las propiedades del recurso o del grupo mediante una acción administrativa, no cuando RGM establece propiedades ni cuando un supervisor establece las propiedades de recurso `Status` y `Status_msg`.

Nota – El método `Monitor_check` invoca también explícitamente el método `Validate` cuando el método `PROBE` intenta realizar una operación de recuperación de fallos del servicio de datos a otro nodo.

Información general de `Validate`

RGM invoca `Validate` con argumentos adicionales a los que se pasan a otros métodos, incluidos las propiedades y los valores que se están actualizando. Por tanto, este método en el servicio de datos de ejemplo debe implementar una función `parse_args()` diferente para manejar argumentos adicionales.

El método `Validate` del servicio de datos de ejemplo verifica una única propiedad, la propiedad de extensión `Confdir`. Esta propiedad apunta al directorio de configuración de DNS, que es fundamental para una operación satisfactoria del DNS.

Nota – Dado que no se puede modificar el directorio de configuración mientras DNS está en ejecución, la propiedad `Confdir` se declara en el archivo RTR como `TUNABLE = AT_CREATION`. Por tanto, el método `Validate` no se invoca nunca para verificar la propiedad `Confdir` tras una actualización; sólo se invoca durante la creación del recurso de servicio de datos.

Si `Confdir` es una de las propiedades que RGM pasa a `Validate`, la función `parse_args()` recupera y guarda su valor. Después, `Validate` verifica que el directorio al que señala el nuevo valor de `Confdir` esté accesible y que el archivo `named.conf` exista en el directorio y contenga datos.

Si la función `parse_args()` no puede recuperar el valor de `Confdir` desde los argumentos de línea de comandos que pasa RGM, `Validate` seguirá intentando validar la propiedad `Confdir`. `Validate` utiliza `scha_resource_get()` para obtener el valor de `Confdir` desde la configuración estática. Después realiza las mismas comprobaciones para comprobar que el directorio de configuración esté accesible y contenga un archivo `named.conf` que no esté vacío.

Si `Validate` sale con un fallo, fallarán la actualización o la creación de todas las propiedades, no sólo de `Confdir`.

Función de análisis del método `Validate`

RGM pasa al método `Validate` un conjunto diferente de parámetros que los otros métodos de rellamada, por lo que `Validate` necesita una función para analizar argumentos diferente de la de los demás métodos. Consulte la página de comando man `rt_callbacks`(1HA) para obtener más información sobre los parámetros que se pasan a `Validate` y los demás métodos de rellamada. A continuación se muestra la función `parse_args()` de `Validate`.

```
#############################################################################
# Analizar argumentos de Validate.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Nombre del recurso de DNS.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Nombre del grupo de recursos en el que se ha configurado
                  # el recurso.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
         T)
                  # Nombre del tipo de recurso.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
         r)
                  # El método no está accediendo a ninguna propiedad
                  # definida por el sistema, por lo que no tiene ninguna
                  # operación
                  ;;
         g)
                  # El método no está accediendo a ninguna propiedad de
                  # grupo de recursos, por lo que no tiene ninguna operación
                  ;;
         c)
                  # Indica que se está invocando el método Validate mientras
                  # se crea el recurso, por lo que el indicador no tiene ninguna
                  # operación
                  ;;
         u)
                  # Indica la actualización de una propiedad cuando el recurso
                  # ya existe. Si la actualización es de la propiedad
```

```
                # Confdir, Confdir debería aparecer en los argumentos
                # de línea de comandos. En caso contrario, el método debe
                # buscarlo específicamente con scha_resource_get.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Lista de propiedades de extensión. Separar los pares de
                # propiedad y valor con "=" como separador.
                PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                VAL=`echo $OPTARG | awk -F= '{print $2}'`
                # Si la propiedad de extensión Confdir se encuentra en la
                # línea de comandos, anotar su valor.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "ERROR: Option $OPTARG desconocida"
                exit 1
                ;;
        esac
    done
}
```

Al igual que con la función `parse_args()` para otros métodos, esta función proporciona un indicador (`R`) para capturar el nombre de recurso, (`G`) para capturar el nombre de grupo de recurso y (`T`) para capturar el tipo de recurso que ha pasado RGM.

Los indicadores `r` (que indica una propiedad definida por el sistema), `g` (que indica una propiedad de grupo de recursos) y `c` (que indica que la validación se produce durante la creación del recurso) se ignoran porque este método se invoca para validar una propiedad de extensión cuando el recurso se está actualizando.

El indicador `u` establece el valor de la variable del shell `UPDATE_PROPERTY` en `1` (`TRUE`). El indicador `x` captura los nombres y valores de las propiedades que se están actualizando. Si `Confdir` es una de las propiedades que se está actualizando, su valor se sitúa en la variable del shell *CONFDIR* y *CONFDIR_FOUND* se establece en `1` (`TRUE`).

Validación de `Confdir`

En su función `MAIN`, `Validate` establece primero la variable *CONFDIR* en la secuencia vacía y *UPDATE_PROPERTY* y *CONFDIR_FOUND* en 0.

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

Después, `Validate` invoca `parse_args()` para analizar los argumentos que pasa RGM.

```
parse_args "$@"
```

`Validate` comprueba entonces si se está invocando `Validate` por una actualización de propiedades y si la propiedad de extensión `Confdir` estaba en la línea de comandos `Validate` verifica si la propiedad `Confdir` tiene un valor y, en caso contrario, sale con un estado de fallo y un mensaje de error.

```
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

# Verificar que la propiedad Confdir tiene un valor. En caso contrario,
# hay un fallo y sale con estado 1
if [[ -z $CONFDIR ]]; then
         logger -p ${SYSLOG_FACILITY}.err \
            "${ARGV0} Método de validar el recurso "$RESOURCE_NAME " no satisfactorio"
         exit 1
fi
```

Nota – Específicamente, el código anterior comprueba si una actualización (`$UPDATE_PROPERTY == 1`) está invocando `Validate` y si la propiedad *no* se ha encontrado en la línea de comandos (`CONFDIR_FOUND == 0`), en cuyo caso recupera el valor existente de `Confdir` con `scha_resource_get()`. Si se encontró `Confdir` en la línea de comandos (`CONFDIR_FOUND == 1`), el valor de *CONFDIR* proviene de la función `parse_args()`, no de `scha_resource_get()`.

El método `Validate` usa el valor de *CONFDIR* para verificar que el directorio es accesible. Si no lo fuera, `Validate` registraría un mensaje de error y cerraría con un estado de error.

```
# Comprobar si $CONFDIR es accesible.
if [ ! -d $CONFDIR ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} Directorio $CONFDIR falta o no está montado"
   exit 1
fi
```

Antes de validar la actualización de la propiedad `Confdir`, `Validate` realiza una comprobación final para verificar si se encuentra el archivo `named.conf`. En caso de que no se encuentre, el método registra un mensaje de error y sale con un estado de error.

```
# Comprobar que el archivo named.conf esté presente en el directorio Confdir
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
```

```
            -t [$SYSLOG_TAG] \
            "${ARGV0} El archivo $CONFDIR/named.conf falta o está vacío"
          exit 1
fi
```

Si se supera la comprobación final, `Validate` registra un mensaje que indica que la operación ha sido satisfactoria y sale con estado satisfactorio.

```
# Registrar un mensaje que indique que el método Validate ha sido satisfactorio.
logger -p ${SYSLOG_FACILITY}.err \
   -t [$SYSLOG_TAG] \
   "${ARGV0} Método de validar para el recurso "$RESOURCE_NAME \
   " se ha completado satisfactoriamente"

exit 0
```

Estado de salida de `Validate`

Si `Validate` sale con éxito (0), se crea `Confdir` con el nuevo valor. Si `Validate` sale con fallo (1), no se crea `Confdir` ni ninguna otra propiedad y se envía un mensaje al administrador del clúster.

Método `Update`

RGM invoca el método `Update` para notificar a un recurso en ejecución que se han modificado sus propiedades. RGM invoca `Update` después de que una acción administrativa logre establecer satisfactoriamente las propiedades de un recurso o su grupo. Este método se invoca en los nodos en los que el recurso está en línea.

Información general de `Update`

El método `Update` no actualiza las propiedade, de eso se encarga RGM. En su lugar, notifica a los procesos en ejecución que se ha producido una actualización. El único proceso del servicio de datos de ejemplo que está afectado por una actualización de una propiedad es el supervisor de fallos, por lo que es este proceso el que detiene y reinicia el método `Update`.

El método `Update` debe comprobar que el supervisor de fallos esté en ejecución y terminarlo con `pmfadm`. El método obtiene la ubicación del programa de análisis que implementa el supervisor de fallos y lo reinicia otra vez con `pmfadm`.

Parada del supervisor con `Update`

El método `Update` utiliza `pmfadm -q` para verificar que el supervisor se esté ejecutando y, en caso de que sea así, lo termina con `pmfadm -s TERM`. Si el supervisor se termina satisfactoriamente, se envía un mensaje para indicárselo al usuario administrativo. Si el supervisor no se puede detener, `Update` sale con un estado de fallo y envía un mensaje de error al usuario administrativo.

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

# Terminar el supervisor que se está ejecutando
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
                 "${ARGV0} No se puede detener el supervisor"
       exit 1
    else
    # se ha podido detener satisfactoriamente el DNS. Registrar un
    # mensaje.
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                 "Supervisor para HA-DNS detenido satisfactoriamente"
    fi
```

Reinicio del supervisor

Para reiniciar el supervisor, el método `Update` debe encontrar la secuencia que implementa el programa de análisis. Éste reside en el directorio básico del servicio de datos, al que señala la propiedad `Rt_basedir`. `Update` recupera el valor de `Rt_basedir` y lo guarda en la variable *RT_BASEDIR* como se muestra a continuación.

```
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAMÈ
```

`Update` usa el valor de *RT_BASEDIR* con `pmfadm` para reiniciar el programa dns_probe. Si la operación es satisfactoria, `Update` sale con éxito y envía un mensaje para indicárselo al usuario administrativo. Si `pmfadm` no puede ejecutar el programa de análisis, `Update` sale con estado de fallo y registra un mensaje de error.

Estado de salida de `Update`

Un fallo del método `Update` hace que el recurso se ponga en estado de “actualización no satisfactoria”. Este estado no afecta a la gestión de RGM del recurso, pero indica el fallo de la acción de actualización a las herramientas de administración, a través del recurso syslog.

CAPÍTULO 6

Biblioteca de desarrollo del servicio de datos (DSDL)

Este capítulo proporciona información general sobre las interfaces de programación de aplicaciones que componen la Biblioteca de desarrollo del servicio de datos o DSDL. Ésta se implementa en la biblioteca `libdsdev.so` y está incluida en el paquete de Sun Cluster.

En este capítulo se tratan los temas siguientes:

Información general sobre DSDL

La API de DSDL está estratificada sobre RMAPI. De este modo no anula ésta, sino que encapsula y amplía sus funciones. DSDL simplifica el desarrollo de servicios de datos aportando soluciones predeterminadas a cuestiones concretas de integración de Sun Cluster. Así, es posible dedicar la mayor parte del tiempo de desarrollo a las cuestiones de alta disponibilidad y escalabilidad intrínsecas a la aplicación y evitar malgastar mucho tiempo en la integración de los procedimientos de encendido, apagado y supervisión en Sun Cluster.

Gestión de las propiedades de configuración

Todos los métodos de rellamada requieren acceso a las propiedades de configuración. DSDL admite el acceso a las propiedades mediante:

- La inicialización del entorno
- El suministro de un conjunto de funciones prácticas para recuperar los valores de las propiedades

La función `scds_initialize`, que se debe invocar al principio de todos los métodos de rellamada:

- Comprueba y procesa los argumentos de la línea de comandos (`argc` y `argv[]`) que RGM pasa al método de rellamada, para evitar la necesidad de escribir una función de análisis de la línea de comandos.
- Configura las estructuras de datos internos para que las utilicen otras funciones de DSDL. Por ejemplo, las funciones prácticas que recuperan los valores de propiedad de RGM guardan los valores en estas estructuras. También se almacenan en ellas los valores de la línea de comandos, que priman sobre otros valores recuperados de RGM.

Nota – Para el método `Validate`, scds_initialize analiza los valores de propiedad que se pasan en la línea de comandos, lo que evita tener que escribir una función de análisis para `Validate`.

La función `scds_initialize` inicializa también el entorno del registro y valida la configuración del análisis del supervisor de fallos.

DSDL proporciona conjuntos de funciones para recuperar las propiedades de recursos, tipos y grupos de recursos, además de las propiedades de extensión de uso común. Estas funciones estandarizan el acceso a las propiedades, con las convenciones siguientes.

- Cada función toma sólo un argumento de manejo (devuelto por `scds_initialize`).
- Cada función corresponde a una propiedad determinada. El tipo de valor de retorno de la función concuerda con el de la propiedad que recupera.
- Las funciones no devuelven errores, porque `scds_initialize` ha calculado previamente los valores. Las funciones recuperan valores de RGM salvo que se pase un nuevo valor a la línea de comandos.

Inicio y parada de un servicio de datos

Se espera que un método `Start` realice las acciones necesarias para iniciar un servicio de datos en un nodo del clúster. Generalmente, esto incluye la recuperación de las propiedades de recurso, la localización de los archivos de configuración y ejecutables específicos de la aplicación y la ejecución de la aplicación con los argumentos de línea de comandos correctos.

La función `scds_initialize` recupera la configuración de recursos. El método `Start` puede utilizar funciones convenientes de la propiedad para recuperar valores para propiedades específicas, como `Confdir_list`, que identifican los archivos y directorios de configuración para que se ejecute la aplicación.

Un método `Start` puede invocar `scds_pmf_start` para ejecutar una aplicación controlada por la Prestación del supervisor de procesos (PMF) que permite especificar el nivel de supervisión aplicable a los procesos y reiniciar éstos en caso de un fallo. Consulte «Método `xfnts_start`» en la página 144 para ver un ejemplo de un método `Start` implementado con DSDL.

Un método `Stop` debe ser idempotente para que salga de forma satisfactoria, aunque se invoque en un nodo cuando la aplicación no esté en ejecución. Si el método `Stop` falla, el recurso que se está deteniendo se pone en el estado `STOP_FAILED`, lo que puede provocar un rearranque por hardware del clúster.

Para evitar poner el recurso en el estado `STOP_FAILED`, el método `Stop` debe intentar detener el recurso como sea posible. La función `scds_pmf_stop` permite intentar detener el recurso gradualmente. En primer lugar intenta detenerlo con la señal de `SIGTERM`; si eso falla, utiliza una señal de `SIGKILL`. Consulte `scds_pmf_stop`(3HA) para obtener más detalles.

Implementación de un supervisor de fallos

DSDL absorbe una buena parte de la complejidad de implementar un supervisor de fallos, gracias a un modelo predeterminado. Un método `Monitor_start` ejecuta el supervisor de fallos, controlado por PMF, cuando el recurso se inicia en un nodo. El supervisor de fallos funciona en bucle mientras el recurso esté ejecutándose en el nodo. La lógica de alto nivel de un supervisor de fallos de DSDL es la siguiente:

- La función `scds_fm_sleep` utiliza la propiedad `Thorough_probe_interval` para determinar el tiempo entre análisis. Cualquier fallo de proceso de una aplicación que detecte PMF en ese intervalo provoca un reinicio del recurso.

- El análisis mismo devuelve un valor que indica la gravedad del fallo, de `0` (sin fallos) a `100` fallo total.
- El valor de retorno del análisis se envía a la función `scds_action`, que mantiene un historial de fallos acumulativo en el intervalo de la propiedad `Retry_interval`.
- La función `scds_action` determina lo que hay que hacer en caso de fallo, como se indica a continuación.
  - Si el fallo acumulativo está por debajo de `100`, no hay que hacer nada.
  - Si el fallo acumulativo alcanza el valor de `100` (fallo total), se debe reiniciar el servicio de datos. Si se supera `Retry_interval`, hay que poner a cero el historial.
  - Si el número de reinicios supera el valor de la propiedad `Retry_count`, en el periodo especificado por `Retry_interval`, se debe realizar una operación de recuperación de fallos del servicio de datos.

Acceso a la información de dirección de la red

DSDL proporciona funciones de conveniencia para devolver información de dirección de la red de recursos y grupos de recursos. Por ejemplo, `scds_get_netaddr_list` recupera los recursos de dirección de la red que utiliza un recurso, lo que permite que un supervisor de fallos analice la aplicación.

DSDL proporciona también un conjunto de funciones para la supervisión basada en TCP. Generalmente, estas funciones establecen una conexión de zócalo simple al servicio, leen y escriben datos en éste y después se desconectan. El resultado del análisis se puede enviar a la función `scds_fm_action` de DSDL para determinar qué acción hay que realizar.

Consulte «Método `xfnts_validate`» en la página 159 para ver un ejemplo de supervisión de fallos basada en TCP.

Depuración de la implementación del tipo de recurso

DSDL incorpora funciones integradas para ayudar a depurar el servicio de datos.

La utilidad `scds_syslog_debug()` de DSDL proporciona un marco básico para agregar instrucciones de depuración a la implementación del tipo de recurso. El *nivel* de depuración (un número entre 1-9) se puede establecer dinámicamente por implementación del tipo de recurso y el nodo del clúster. Un archivo con el nombre `/var/cluster/rgm/rt/`*nombre_tipo_recurso*`/loglevel`, que sólo contiene un número entero entre 1 y 9, lo leen todos los métodos de rellamada del tipo de recurso. La rutina DSDL `scds_initialize()` lee este archivo y establece internamente la depuración en el nivel especificado. El nivel predeterminado de depuración es 0: el servicio de datos no debe registrar ningún mensaje de depuración.

La función `scds_syslog_debug()` utiliza el recurso devuelto por la función `scha_cluster_getlogfacility()` con una prioridad de `LOG_DEBUG`. Estos mensajes de depuración se pueden configurar en `/etc/syslog.conf`.

Es posible convertir algunos mensajes de depuración en mensajes informativos para un funcionamiento habitual del tipo de recurso (tal vez en la prioridad `LOG_INFO`), mediante la utilidad `scds_syslog`. Si se observa la aplicación DSDL de ejemplo del Capítulo 8, se ve que utiliza de forma libre las funciones `scds_syslog_debug` y `scds_syslog`.

Habilitación de sistemas de archivos locales de alta disponibilidad

Se puede usar el tipo de recurso `HAStoragePlus` para hacer que un sistema de archivos local tenga una alta disponibilidad, dentro de un entorno Sun Cluster. Las particiones del sistema de archivos local deben ubicarse en grupos globales de disco. Se deben habilitar los conmutadores por afinidad y el entorno Sun Cluster debe configurarse para que admita la recuperación de fallos. Esta configuración permite que el usuario haga que todos los sistemas de archivos situados en discos con varios sistemas principales estén accesibles desde todos los sistemas conectados directamente a dichos discos. Se recomienda la utilización de un sistema de archivos local de alta disponibilidad para algunos servicios de datos de E/S intensiva. “Enabling Highly Available Local File Systems” in *Sun Cluster 3.1 Data Service Planning and Administration Guide* contiene información sobre cómo configurar el recurso `HAStoragePlus`.

CAPÍTULO 7

Diseño de tipos de recurso

En este capítulo se explica la utilización habitual de DSDL para diseñar e implementar tipos de recursos. Se presta especial atención al diseño de los tipos de recursos para validar la configuración del recurso, así como iniciar, detener y supervisar éste. Finalmente, se describe cómo utilizar DSDL para implementar los métodos de rellamada del tipo de recurso.

Consulte la página de comando man `rt_callbacks`(1HA) para obtener información adicional.

Para realizar estas tareas deberá acceder a la configuración de la propiedad del recurso. La utilidad `scds_initialize()` de DSDL permite acceder de modo uniforme a las propiedades de los recursos. Esta función esta diseñada para invocarse al principio de los métodos de rellamada; permite recuperar las propiedades de un recurso desde la estructura del clúster y lo pone a disposición de la familia de funciones `scds_get`*nombre*`()`.

En este capítulo se tratan los temas siguientes:

El archivo RTR

El archivo de registro del tipo de recurso (RTR) es un componente importante de un tipo de recurso. Este archivo especifica los detalles del tipo de recurso para Sun Cluster que incluyen información, como las propiedades necesarias para la implementación, los tipos de datos y los valores predeterminados de esas propiedades, la ruta del sistema de archivos para los métodos de rellamada de la implementación del tipo de recurso y varios valores de las propiedades definidas por el sistema.

El archivo RTR de ejemplo que se incluye con DSDL debería bastar para la mayoría de las implementaciones del tipo de recurso. Sólo hay que editar algunos elementos básicos, como los nombres del tipo de recurso y de la ruta de los métodos de rellamada del tipo de recurso. Si se necesita una nueva propiedad para implementar el tipo de recurso, puede declararla como propiedad de extensión en el archivo de registro del tipo de recurso (RTR) de la implementación del tipo de recurso y acceder después a la nueva propiedad con la utilidad `scds_get_ext_property()` de DSDL.

El método `Validate`

RGM invoca el método `Validate` de una implementación de tipo de recurso en dos casos: 1) cuando se crea un nuevo recurso del tipo de recurso y 2) cuando se actualiza una propiedad del recurso o grupo de recursos. Estas dos situaciones se pueden distinguir por la presencia de la opción de la línea de comandos `-c` (creación) o `-u` (actualización) que se pasa al método `Validate` del recurso.

El método `Validate` se invoca en todos los nodos de un conjunto definido por el valor de la propiedad del tipo de recurso `INIT_NODES`. Si se establece `INIT_NODES` en `RG_PRIMARIES`, `Validate` se invoca en todos los nodos que puedan alojar (ser principales de) el grupo de recursos que contiene el recurso en cuestión. Si se establece `INIT_NODES` en `RT_INSTALLED_NODES`, `Validate` se invoca en todos los nodos en los que está instalado el software del tipo de recurso, que suelen ser todos los nodos del clúster. El valor predeterminado de `INIT_NODES` es `RG_PRIMARIES` (consulte `rt_reg`(4). En el momento en que se invoca el método `Validate`, RGM no ha creado aún el recurso (en el caso de una rellamada de creación) o todavía no ha aplicado el o los valor(es) actualizado(s) de las propiedades que se van a actualizar (en el caso de una rellamada de actualización). El objetivo del método `Validate` de una implementación de tipo de recurso es comprobar que los valores propuestos para el recurso (tal como indica la configuración de la propiedad propuesta en el recurso) son aceptables para el tipo de recurso.

Nota – Si está usando sistemas de archivos locales gestionados por `HAStoragePlus`, debe utilizar `scds_hasp_check` para comprobar el estado del recurso `HAStoragePlus`. Esta información se obtiene del estado (en línea u otro) de todos los recursos `SUNW.HAStoragePlus`(5) del que depende el recurso, utilizando las propiedades de sistema `Resource_dependencies` o `Resource_dependencies_weak` definidas para el recurso. Consulte `scds_hasp_check`(3HA) para ver una lista completa de códigos de estado que devuelve la llamada `scds_hasp_check`.

La función `scds_initialize()` de DSDL se encarga de estas situaciones como sigue:

- En el caso de creación de recursos, analiza las propiedades propuestas para el recurso, como se han pasado a la línea de comandos. Los valores propuestos para las propiedades del recurso quedan disponibles para el desarrollador del tipo de recurso como si éste ya se hubiera creado en el sistema.
- En el caso de una actualización del recurso o el grupo de recursos, los valores propuestos de las propiedades que está actualizando el administrador se leen en la línea de comandos y el resto de las propiedades (cuyos valores no se están actualizando) se leen de Sun Cluster con la API de gestión de recursos (RMAPI). Un desarrollador de tipo de recursos que utilice DSDL no tendrá que preocuparse de todas estas tareas domésticas. La validación de un recurso se puede realizar como si todas las propiedades del recurso estuvieran disponibles para el desarrollador.

Supongamos que la función que implementa la validación de las propiedades de un recurso se llama `svc_validate()` y utiliza la familia de funciones `scds_get_`*nombre*`()` para ver la propiedad que interesa validar. Suponiendo que un valor de recurso aceptable se representa con un código de retorno 0 de esta función, el método `Validate` del tipo de recurso se puede representar con el siguiente fragmento de código:

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

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Error de inicialización */
   }
   rc = svc_validate(handle);
   scds_close(&handle);
   return (rc);
}
```

La función de validación debe también registrar el motivo del fallo en la validación del recurso. Dejando a un lado los detalles (si desea ver un tratamiento más realista de una función de validación, consulte el capítulo siguiente), un ejemplo sencillo de función `svc_validate()` se puede implementar como:

```
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);   /* Configuración de propiedad de recurso no válida */
   }
   return (0);   /* Configuración aceptable*/
}
```

El desarrollador de tipos de recursos se debe encargar sólo de la implementación de la función `svc_validate()`. Un ejemplo típico de implementación de un tipo de recurso podría ser garantizar que un archivo de configuración de una aplicación denominado app.conf existiera dependiente de la propiedad `Confdir_list`. Esto se puede implementar convenientemente con una llamada de sistema `stat()` en el nombre de ruta adecuado, derivado de la propiedad `Confdir_list`.

El método `Start`

RGM invoca el método de rellamada `Start` de una implementación de tipo de recursos en un nodo del clúster seleccionado para iniciar el recurso. Los nombres del grupo de recursos, del recurso y del tipo de recurso se pasan a la línea de comandos. Se espera que el método `Start` realice las acciones necesarias para iniciar un recurso de servicio de datos en el nodo del clúster. Generalmente, esto implica recuperar las propiedades del recurso, ubicar los ejecutables específicos de la aplicación y los archivos de configuración y ejecutar la aplicación con los argumentos de línea de comando adecuados.

Con DSDL, la configuración del recurso se recupera con la utilidad `scds_initialize()`. La acción de inicio de la aplicación se puede contener en una función `svc_start()`. Otra función, `svc_wait()`, puede invocarse para verificar que la aplicación se inicie efectivamente. El código simplificado para el método `Start` se convierte en:

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

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

```
    return (1);   /* Error de inicialización */
    }
    if (svc_validate(handle) != 0) {
    return (1);   /* Valores no válidos */
    }
    if (svc_start(handle) != 0) {
    return (1);   /* Fallo de inicio */
    }
    return (svc_wait(handle));
}
```

Esta implementación del método de inicio invoca `svc_validate()` para validar la configuración del recurso. Si falla, o bien la configuración del recurso o la configuración de la aplicación no coinciden o bien hay actualmente un problema relacionado con el sistema en el nodo del clúster. Por ejemplo, es posible que un sistema de archivos global necesario para el recurso no esté disponible actualmente en este nodo del clúster. En tal caso, es inútil intentar siquiera iniciar el recurso en este nodo del clúster, es mejor que RGM intente iniciar el recurso en otro nodo. Sin embargo, tenga presente que el caso anterior presupone que `svc_validate()` es lo suficientemente conservador (por lo que sólo comprueba los recursos del nodo del clúster imprescindibles para la aplicación), porque en caso contrario es posible que el inicio del recurso fallara en todos los nodos del clúster y acabara con un estado `START_FAILED`. Consulte `scswitch`(1M) y *Sun Cluster 3.1 Data Service Planning and Administration Guide* para ver una explicación sobre este estado.

La función `svc_start()` debe devolver 0 para que el inicio del recurso en el nodo sea satisfactorio. Si la función de inicio ha encontrado algún problema, devolverá un valor distinto de cero. Si esta función falla, RGM intenta iniciar el recurso en otro nodo del clúster.

Para aprovechar DSDL en lo posible, la función `svc_start()` puede utilizar la utilidad `scds_pmf_start()` para iniciar la aplicación bajo la Prestación del supervisor de procesos (PMF). Esta utilidad aprovecha también la función de acción de rellamada en caso de fallo de PMF (consulte el indicador de acción `-a` en `pmfadm`(1M)) para implementar la detección de fallos de procesos.

El método `Stop`

RGM invoca el método de rellamada `Stop` de una implementación del tipo de recurso en un nodo del clúster para detener la aplicación. La semántica de la rellamada del método `Stop` requiere estas condiciones:

- El método `Stop`ha de ser *idempotente*, ya que RGM lo puede invocar aun cuando el método `Start` no se haya completado satisfactoriamente en el nodo. Por tanto, el método `Stop` debe ser satisfactorio (salida cero) aunque la aplicación no se esté ejecutando actualmente en el nodo del clúster y no tenga que realizar ninguna

tarea.

- Si el método `Stop` del tipo de recurso falla (la salida es diferente de cero) en un nodo del clúster, el recurso que se está deteniendo acabaría con el estado `STO_FAILED`. Según la configuración de `Failover_mode` del recurso, esto puede hacer que RGM realice un rearranque por hardware del nodo del clúster. Por eso es importante diseñar el método `Stop` de forma que intente realmente detener la aplicación, aunque sea de forma seca y brusca (por ejemplo, con `SIGKILL`) si la aplicación no se puede terminar de otra forma. También debería hacerlo en un tiempo determinado, porque cuando se cumple el tiempo de espera de `Stop_timeout` la estructura considera que se ha producido un fallo de parada y pone el recurso en el estado `STOP_FAILED`.

La utilidad `scds_pmf_stop()` de DSDL debería ser suficiente para la mayoría de las aplicaciones, porque intenta, primero, detener la aplicación suavemente (mediante `SIGTERM`): presupone que se inició en PMF con `scds_pmf_start()` y. después, envía un `SIGKILL` al proceso. Consulte «Funciones de PMF» en la página 206 para obtener más detalles sobre esta utilidad.

Siguiendo el modelo del código que hemos utilizado hasta ahora, presuponiendo que la función específica de la aplicación para detener la aplicación se denomina `svc_stop()` (que la implementación de `svc_stop()` utilice `scds_pmf_stop()` no es relevante en este momento y dependerá de que la aplicación se haya iniciado bajo PMF con el método `Start`), el método `Stop` se puede implementar de la manera siguiente:

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

El método `svc_validate()` no se utiliza en la implementación del método `Stop` porque, aunque el sistema tenga un problema en este momento, el método `Stop` debería intentar detener la aplicación en este nodo.

El método `Monitor_start`

RGM invoca el método `Monitor_start` para iniciar un supervisor de fallos para el recurso. Los supervisores de fallos controlan el estado de la aplicación que gestiona el recurso. Las implementaciones del tipo de recurso se suelen realizar en un supervisor de fallos como daemon separado, que se ejecuta en segundo plano. El método de rellamada `Monitor_start` se usa para ejecutar este daemon con los argumentos apropiados.

Dado que el daemon del supervisor está sujeto, también, a sufrir fallos (por ejemplo, podría terminarse inesperadamente, dejando la aplicación sin supervisión), se debería iniciar PMF para iniciar el daemon del supervisor. La utilidad `scds_pmf_start()` de DSDL incorpora soporte integrado para iniciar supervisores de fallos. Esta utilidad emplea el nombre de ruta relativo (relativo a `RT_basedir` para la ubicación de las implementaciones de métodos de rellamada de tipo de recurso) del programa daemon del supervisor. Emplea las propiedades de extensión `Monitor_retry_interval` y `Monitor_retry_count` gestionadas por DSDL para impedir que se produzca un número ilimitado de reinicios del daemon. Impone la misma sintaxis de línea de comandos definida para todos los métodos de rellamada (es decir, `-R` *recurso* `-G` *grupo_recursos*`-T` *tipo_recurso*) al daemon del supervisor, aunque RGM no lo invoca nunca directamente. Permite que la implementación del daemon del supervisor aproveche la utilidad `scds_initialize()` para configurar su propio entorno. El esfuerzo principal se destina a diseñar el propio daemon del supervisor.

El método `Monitor_stop`

RGM invoca el método `Monitor_stop` para detener el daemon del supervisor de fallos iniciado mediante el método `Monitor_start`. Un fallo de este método de rellamada se trata exactamente igual que un fallo del método `Stop`; por tanto, el método `Monitor_stop` debe ser idempotente y robusto, como el método `Stop`.

Si emplea `scds_pmf_start()` para iniciar el daemon del supervisor de fallos, utilice `scds_pmf_stop()` para detenerlo.

El método `Monitor_check`

El método de rellamada `Monitor_check` en un recurso se invoca en un nodo para que el recurso especificado pueda determinar si el nodo del clúster es capaz de controlar el recurso (es decir, ¿pueden ejecutarse satisfactoriamente en el nodo las aplicaciones que está gestionando el recurso?). Generalmente, esta situación implica asegurarse de que todos los recursos del sistema que necesita la aplicación estén de hecho disponibles en el nodo del clúster. Como se explica en «El método `Validate`» en la página 130, la función `svc_validate()` que implementa el desarrollador está destinada a realizar por lo menos esta comprobación.

Según la aplicación concreta que esté gestionando la implementación del tipo de recurso, el método `Monitor_check` se puede escribir para que realice otras tareas. El método `Monitor_check` debe implementarse de forma que no entre en conflicto con

otros métodos que funcionen simultáneamente. Para los desarrolladores que utilizan DSDL es recomendable que el método `Monitor_check` aproveche la función `svc_validate()` escrita para implementar una validación de las propiedades de recurso específicas de la aplicación.

El método `Update`

RGM invoca el método `Update` de una implementación del tipo de recurso para aplicar cualquier cambio realizado por el administrador del sistema en la configuración del recurso activo. El método `Update` sólo se invoca en los nodos en los que el recurso está en línea en este momento (si los hubiera).

Los cambios que se acaban de realizar en la configuración del recurso serán aceptables para la implementación del tipo de recurso porque RGM ejecuta el método `Validate` del tipo de recurso antes de ejecutar el método `Update`. El método `Validate` se invoca antes de que las propiedades del recurso o grupo de recursos sean modificadas y de que el método `Validate` pueda vetar los cambios propuestos. El método `Update` se invoca después de que los cambios se hayan aplicado para darle al recurso activo (en línea) la oportunidad de notar los nuevos valores.

Como desarrollador del tipo de recurso, tiene que decidir con mucho cuidado las propiedades que desea poder actualizar dinámicamente y marcarlas con el valor `TUNABLE = ANYTIME` en el archivo RTR. Generalmente se puede especificar si se desea poder actualizar dinámicamente cualquier propiedad de una implementación del tipo de recurso que utilice el daemon del supervisor de fallos, siempre que la implementación del método `Update` reinicie al menos el daemon del supervisor.

Las candidatas posibles son las siguientes:

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

Estas propiedades afectan al modo en que un daemon del supervisor de fallos realiza la comprobación del estado del servicio, la frecuencia de las comprobaciones, el intervalo de historial que utiliza para mantener un seguimiento de los errores y los umbrales de reinicio que ha establecido PMF. Para implementar las actualizaciones de estas propiedades se incluye la utilidad `scds_pmf_restart()` en DSDL.

Si necesita poder actualizar dinámicamente una propiedad de recurso, cuya modificación puede afectar a la aplicación en curso, deberá emplear las acciones necesarias para que las actualizaciones se apliquen correctamente a todas las instancias en ejecución de la aplicación. Actualmente no existe ninguna forma de facilitar esta tarea mediante DSDL. `Update` no recibe las propiedades modificadas en la línea de comandos (como `Validate`).

Los métodos `Init`, `Fini` y `Boot`

Son métodos *de acción única*, como se indica en las especificaciones de la API de gestión de recursos. La implementación de ejemplo que se incluye con DSDL no ilustra la utilización de estos métodos. Sin embargo, todos los recursos de DSDL están también disponibles para estos métodos por si un desarrollador de tipo de recursos los necesita. Generalmente, los métodos `Init` y `Boot` serán exactamente iguales que los de una implementación del tipo de recurso para implementar una *acción única*. El método `Fini` normalmente realiza una acción para *deshacer* la acción de los métodos `Init` o `Boot`.

Diseño de un daemon del supervisor de fallos

Las implementaciones del tipo de recurso con DSDL suelen tener un daemon del supervisor de fallos con las responsabilidades siguientes.

- Supervisar periódicamente el estado de la aplicación gestionada. Este aspecto concreto de un daemon de supervisor depende enormemente de cada aplicación y puede variar mucho de un tipo de recurso a otro. DSDL tiene ciertas funciones útiles integradas para realizar comprobaciones de estado para servicios sencillos basados en TCP. Las aplicaciones que implementan protocolos basados en ASCII, como HTTP, NNTP, IMAP y POP3, pueden implementarse con estas utilidades.
- Mantener un seguimiento de los problemas que encuentra la aplicación al utilizar las propiedades de recurso `Retry_interval` y `Retry_count`. Cuando se producen fallos completos de la aplicación, decidir si la secuencia de acción de PMF debería reiniciar el servicio o si los fallos de la aplicación se han acumulado con tanta rapidez que se debería plantear la posibilidad de una recuperación de fallos. Las utilidades `scds_fm_action()` y `scds_fm_sleep()` de DSDL están diseñadas para ayudar a implementar este mecanismo.

- Tomar las medidas adecuadas (generalmente, reiniciar la aplicación o intentar una operación de recuperación de fallos del grupo de recursos que lo contiene). La utilidad `scds_fm_action()` de DSDL implementa este algorritmo. Para ello, calcula la acumulación actual de fallos de análisis en los últimos Retry_interval segundos.
- Actualizar el estado del recurso para que el estado de la aplicación quede disponible para el comando `scstat` y la GUI de gestión del clúster.

Las utilidades de DSDL están diseñadas de modo que el bucle principal del daemon del supervisor de fallos se pueda representar con el siguiente pseudocódigo.

Para los supervisores de fallos implementados con DSDL:

- La detección de la terminación de los procesos de una aplicación con `scds_fm_sleep()` es bastante rápida, porque la notificación de la terminación del proceso a través de PFM es asíncrona. Contrasta esta situación con el caso en el que un supervisor de fallos se activa con una frecuencia determinada para comprobar el estado del servicio y se encuentra con que la aplicación se ha terminado. El tiempo de detección del fallo se reduce de manera notable y se aumenta, por tanto, la disponibilidad del servicio.
- Si RGM rechaza el intento de realizar una operación de recuperación de fallos del servicio con la API `scha_control(3HA)`, `scds_fm_action()` *pone a cero* (olvida) el historial actual de fallos. El motivo es que el historial de fallos ya está por encima de `Retry_count` y si el daemon del supervisor se activa en la siguiente iteración y no puede finalizar la comprobación del estado del daemon, volvería a intentar invocar la llamada `scha_control()`, lo que probablemente sería rechazado otra vez, ya que la situación que provocó el primer rechazo en la iteración anterior sigue vigente. Al poner a cero el historial se garantiza que el supervisor de fallos intentará al menos corregir localmente la situación (por ejemplo, mediante un reinicio de la aplicación) en la próxima iteración.
- `scds_fm_action()` *no* pone a cero el historial de fallos de la aplicación cuando se producen fallos de reinicio, porque normalmente se intentaría utilizar `scha_control()` muy pronto si la situación no se corrigiera sola.
- La utilidad `scds_fm_action()` actualiza el estado del recurso a `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_DEGRADED` o `SCHA_RSSTATUS_FAULTED`, según el historial de fallos. Este estado queda disponible para la gestión del sistema de clúster.

En la mayoría de los casos, la acción de comprobación del estado específico de la aplicación se puede implementar en una utilidad autónoma separada (por ejemplo, `svc_probe()`) y se puede integrar con este bucle principal genérico.

```
for (;;) {

   / * reposo durante un intervalo thorough_probe_interval entre
     * análisis sucesivos. */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));
```

```
/* Analizar ahora todas las direcciones ip que se usan. Recorrer
* 1. Todos los recursos de red que usamos.
* 2. Todas las direcciones ip de un recurso determinado.
* Para cada una de las direcciones ip analizadas,
* calcular el historial de fallos. */
probe_result = 0;
/* Iterar todos los recursos para obtener todas las
 * direcciones IP para invocar svc_probe() */
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
/* Anotar el nombre del sistema y puerto en los que
 * se debe supervisar el estado.
 */
hostname = netaddr->netaddrs[ip].hostname;
port = netaddr->netaddrs[ip].port_proto.port;
/*
 * HA-XFS sólo admite un puerto; por tanto,
 * obtener el valor de puerto de la primera
 * entrada de la matriz de puertos.
 */
ht1 = gethrtime(); /* Bloquear tiempo de inicio de análisis */
probe_result = svc_probe(scds_handle,

hostname, port, timeout);
/*
 * Actualizar historial de análisis,
 * tomar medidas si fuera necesario.
 * Bloquear tiempo de finalización de análisis.
 */
ht2 = gethrtime();
/* Convertir a milisegundos */
dt = (ulong_t)((ht2 - ht1) / 1e6);

/*
 * Calcular historial de fallos y tomar
 * medidas si fuera necesario
 */
(void) scds_fm_action(scds_handle,
probe_result, (long)dt);
}       /* Cada recurso de red */
}       /* Seguir analizando para siempre */
```

CAPÍTULO 8

Ejemplo de implementación del tipo de recurso con DSDL

En este capítulo se describe un tipo de recurso de ejemplo, `SUNW.xfnts`, implementado con DSDL. El servicio de datos se escribe en C. La aplicación subyacente es el servidor de fuentes X, un servicio basado en TCP/IP.

La información de este capítulo incluye.

Servidor de fuentes X

El servidor de fuentes X es un servicio sencillo basado en TCP/IP que sirve archivos de fuentes a sus clientes. Éstos se conectan al servidor para solicitar un conjunto de fuentes y el servidor lee los archivos de fuentes del disco y se los sirve a los clientes. El daemon del servidor de fuentes X está formado por un binario de servidor `/usr/openwin/bin/xfs` y suele iniciarse desde `inetd`; sin embargo, para el ejemplo actual, suponemos que la entrada adecuada del archivo `/etc/inetd.conf` se ha inhabilitado (por ejemplo, con el comando `fsadmin -d`) así que el daemon está controlado únicamente por Sun Cluster.

Archivo de configuración del servidor de fuentes X

De forma predeterminada, el servidor de fuentes X lee la información de configuración en el archivo `/usr/openwin/lib/X11/fontserver.cfg` cuya entrada de catálogo contiene una lista de directorios de fuentes disponibles para el daemon a fin que pueda realizar el servicio. El administrador del clúster puede ubicar los directorios de fuentes en el sistema global de archivos (para optimizar la utilización del servidor de fuentes X en Sun Cluster, manteniendo una única copia de la base de datos de las fuentes en el sistema). En ese caso, el administrador debe editar `fontserver.cfg` para reflejar las nuevas rutas de los directorios de fuentes.

Para facilitar la configuración, el administrador también puede colocar el archivo de configuración en el sistema global de archivos. El daemon `xfs` proporciona argumentos de línea de comandos para anular la ubicación integrada y predeterminada de este archivo. El tipo de recurso `SUNW.xfnts` utiliza el comando siguiente para iniciar el daemon, controlado por Sun Cluster.

```
/usr/openwin/bin/xfs -config <ubicación_del_archivo_de_configuración>/fontserver.cfg \
-port <número_puerto>
```

En la implementación del tipo de recurso `SUNW.xfnts` se puede usar la propiedad `Confdir_list` para gestionar la ubicación del archivo de configuración `fontserver.cfg`.

Número del puerto de TCP

El número de puerto TCP en el que el daemon del servidor `xfs` recibe suele ser el puerto “fs” (definido normalmente como `7100` en el archivo `/etc/services`). Sin embargo, la opción `-port` de la línea de comandos de `xfs` permite al administrador del sistema anular el valor predeterminado. Puede utilizar la propiedad `Port_list` del tipo de recurso `SUNW.xfnts` para establecer el valor predeterminado y admitir la utilización de la opción `-port` en la línea de comandos de `xfs`. Puede definir el valor predeterminado de esta propiedad como `7100/tcp` en el archivo RTR. En el método `SUNW.xfnts Start` se pasa `Port_list` a la opción `-port` de la línea de comandos de `xfs`. Por tanto, un usuario de este tipo de recurso no tiene que especificar un número de puerto (el puerto predeterminado es `7100/tcp`) pero sí puede especificar un puerto diferente si lo desea cuando configure el tipo de recurso, indicando un valor diferente para la propiedad `Port_list`.

Convenciones de asignación de nombres

Puede identificar los diferentes fragmentos del código de ejemplo si tiene presentes las convenciones siguientes:

- Las funciones de RMAPI empiezan con `scha_`.
- Las funciones de DSDL empiezan con `scds_`.
- Los métodos de rellamada empiezan con `xfnts_`.
- Las funciones escritas por los usuarios empiezan con `svc_`.

Archivo RTR de `SUNW.xfnts`

En esta sección se describen varias propiedades clave del archivo RTR de `SUNW.xfnts`, pero no se explica el objetivo de todas ellas. Para conocer estas descripciones, consulte «Establecimiento del recurso y las propiedades del tipo de recurso» en la página 34.

La propiedad de extensión `Confdir_list` identifica el directorio de configuración (o una lista de directorios), como se indica a continuación.

```
{
        PROPERTY = Confdir_list;
        EXTENSION;
        STRINGARRAY;
        TUNABLE = AT_CREATION;
        DESCRIPTION = "Ruta de directorio de configuración";
}
```

La propiedad `Confdir_list` no especifica un valor predeterminado. El administrador del clúster debe especificar un directorio en el momento de la creación del recurso. Este valor no se puede modificar más adelante, porque la posibilidad de ajuste está limitada al momento `AT_CREATION`.

La propiedad `Port_list` identifica el puerto en el que recibe el daemon del servidor, como sigue:

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

Dado que la propiedad declara un valor predeterminado, el administrador del clúster puede especificar un valor nuevo o aceptar el predeterminado en el momento de la creación del recurso. Este valor no se puede modificar más adelante, porque la posibilidad de ajuste está limitada al momento `AT_CREATION`.

Función `scds_initialize()`

DSDL requiere que todos los métodos de rellamada invoquen la función `scds_initialize`(3HA) al principio del método. Esta función realiza las operaciones siguientes:

- Comprueba y procesa los argumentos de la línea de comandos (`argc` y `argv`) que la estructura pasa al método del servicio de datos. El método no tiene que realizar ningún procesamiento adicional de los argumentos de la línea de comandos.
- Configura las estructuras de datos internos para que las utilicen otras funciones de DSDL.
- Inicializa el entorno de registro.
- Valida los valores de análisis del supervisor de fallos.

Utilice la función `scds_close()` para reclamar los recursos asignados por `scds_initialize()`.

Método `xfnts_start`

RGM invoca el método `Start` en un nodo del clúster cuando el grupo de recursos que contiene el recurso del servicio de datos se pone en línea en ese nodo o cuando se habilita el nodo. En el tipo de recurso de ejemplo, `SUNW.xfnts`, el método `xfnts_start` activa el daemon `xfs` en ese nodo.

El método `xfnts_start` invoca `scds_pmf_start()` para iniciar el daemon en PMF. PMF proporciona funciones automáticas de notificación de fallo y reinicio, además de integración con el supervisor de fallos.

Nota – La primera llamada de `xfnts_start` se realiza a `scds_initialize()`, que ejecuta ciertas *tareas domésticas* necesarias («Función `scds_initialize()`» en la página 144 y la página de comando man `scds_initialize`(3HA) ofrecen información más detallada).

Validación del servicio antes de empezar

Antes de intentar iniciar el servidor de fuentes X, el método `xfnts_start` invoca `svc_validate()` para verificar que haya una configuración adecuada para admitir el daemon `xfs` (consulte «Método `xfnts_validate`» en la página 159 para obtener más detalles), como se indica a continuación:

```
rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "No se ha podido validar la configuración.");
      return (rc);
   }
```

Inicio del servicio

El método `xfnts_start` invoca el método `svc_start()`, definido en `xfnts.c` para iniciar el daemon `xfs`. Esta sección describe `svc_start()`.

El comando para ejecutar el daemon `xfs` es el siguiente:

`xfs -config` *directorio_config*`/fontserver.cfg -port` *número_puerto*

La propiedad de extensión `Confdir_list` identifica el *directorio_config* y la propiedad de sistema `Port_list` identifica el *número_puerto*. Cuando el administrador del clúster configura el servicio de datos, proporciona valores específicos para estas propiedades.

El método `xfnts_start` declara estas propiedades como matrices de secuencias y obtiene los valores que fija el administrador con las funciones `scds_get_ext_confdir_list()` y `scds_get_port_list()` (descritas en `scds_property_functions`(3HA)), como sigue:

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

   /* obtener el directorio de configuración de la propiedad confdir_list */
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

   /* obtener el puerto que va a usar XFS de la propiedad Port_list */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se puede acceder a la propiedad Port_list.");
      return (1);
   }
```

Observe que la variable `confdirs` apunta al primer elemento (0) de la matriz.

El método `xfnts_start` utiliza `sprintf` para formar la línea de comandos para `xfs` como se muestra a continuación.

```
/* Construir el comando para que inicie el daemon xfs. */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);
```

Observe que la salida se redirige a `dev/null` para suprimir los mensajes que genera el daemon.

El método `xfnts_start` pasa la línea de comandos `xfs` a `scds_pmf_start()` para iniciar el servicio de datos controlado por PMF, como se indica a continuación.

```
scds_syslog(LOG_INFO, "Emitir un solicitud de inicio.");
   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,
          "Comando de inicio completado satisfactoriamente.");
   } else {
      scds_syslog(LOG_ERR,
          "No se ha podido iniciar HA-XFS ");
   }
```

Observe los siguientes aspectos de la llamada a `scds_pmf_start()`.

- El parámetro `SCDS_PMF_TYPE_SVC` identifica el programa que se va a iniciar como aplicación de servicio de datos; este método también puede iniciar un supervisor de fallos u otro tipo de aplicación.
- El parámetro `SCDS_PMF_SINGLE_INSTANCE` lo identifica como un recurso de una sola instancia.
- El parámetro `cmd` es la línea de comandos generada previamente.
- El parámetro final, `-1`, especifica el nivel de supervisión de secundarios. El -1 especifica que PMF supervisa todos los secundarios y el proceso original.

Antes de volver, `svc_pmf_start()` libera la memoria asignada para la estructura `portlist` como se muestra a continuación.

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

Retorno desde `svc_start()`

Aun cuando la función `svc_start()` dé un retorno satisfactorio, es posible que la aplicación subyacente no se haya iniciado correctamente. Por tanto, `svc_start()` debe analizar la aplicación para verificar si se está ejecutando antes de devolver un

mensaje satisfactorio. El análisis debe tener en cuenta también que es posible que la aplicación no esté disponible inmediatamente porque tarda un tiempo en iniciarse. El método `svc_start()` invoca `svc_wait()`, que se define en `xfnts.c`, para verificar que la aplicación está en ejecución, como se indica a continuación:

```
/* Esperar a que el servicio se inicie completamente */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Llamar a svc_wait para verificar que se haya iniciado el servicio.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Devuelto desde svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "El servicio se ha iniciado satisfactoriamente.");
   } else {
      scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
   }
```

La función `svc_wait()` invoca `scds_get_netaddr_list`(3HA) para obtener los recursos de dirección de red necesarios para analizar la aplicación, como se indica a continuación:

```
/* obtener el recurso de red para el análisis*/
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No se han encontrado recursos de dirección de red en el grupo
           de recursos.");
      return (1);
   }

   /* Devolver un error si no hay recursos de red */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No hay recursos de dirección de red en el grupo de recursos.");
      return (1);
   }
```

Después, `svc_wait()` obtiene los valores `start_timeout` y `stop_timeout` como se indica a continuación.

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

Para justificar el tiempo que puede tardar en iniciarse el servidor, `svc_wait()` invoca `scds_svc_wait()` y pasa un valor de tiempo de espera equivalente a un tres por ciento del valor `start_timeout`. A continuación, `svc_wait()` invoca `svc_probe()` para verificar si la aplicación se ha iniciado. El método `svc_probe()`

realiza una conexión de zócalo sencilla con el servidor en el puerto especificado. Si no puede conectarse con el puerto, `svc_probe()` devuelve un valor de 100, que indica un fallo total. Si la conexión pasa, pero la desconexión del puerto falla, `svc_probe()` devuelve un valor de 50.

Cuando se produce un fallo parcial o total de `svc_probe()`, `svc_wait()` invoca `scds_svc_wait()` con un valor de tiempo de espera de 5. El método `scds_svc_wait()` limita la frecuencia de los análisis a cada 5 segundos. Este método cuenta también el número de intentos hechos para iniciar el servicio. Si el número de intentos supera el valor de la propiedad `Retry_count` del recurso en el periodo especificado por la propiedad `Retry_interval` del recurso, la función `scds_svc_wait()` devuelve un fallo. En ese caso, la función `svc_start()` también devuelve un fallo.

```
# definir SVC_CONNECT_TIMEOUT_PCT    95
# definir   SVC_WAIT_PCT       3
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
      return (1);
   }

   do {
      /*
       * analizar el servicio de datos en la dirección IP del
       * recurso de red y el nombre de puerto
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Satisfactorio. Liberar recursos y retornar */
         scds_free_netaddr_list(netaddr);
         return (0);
      }

       /* Invocar scds_svc_wait() por si el servicio también falla
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
         return (1);
      }

   /* Confiar en el tiempo de espera de RGM y terminar el programa*/
   } while (1);
```

Nota – Antes de salir, el método `xfnts_start` invoca `scds_close()` para reclamar recursos asignados por `scds_initialize()`. Consulte «Función `scds_initialize()`» en la página 144 y la página de comando man `scds_close`(3HA) para obtener más detalles.

Método `xfnts_stop`

Dado que el método `xfnts_start` utiliza `scds_pmf_start()` para iniciar el servicio bajo PMF, `xfnts_stop` emplea `scds_pmf_stop()` para detenerlo.

Nota – La primera llamada en `xfnts_stop` es a `scds_initialize()`, que ejecuta ciertas *tareas domésticas* necesarias («Función `scds_initialize()`» en la página 144 y la página de comando man de `scds_initialize`(3HA) ofrecen información más detallada).

El método `xfnts_stop` invoca el método `svc_stop()`, definido en `xfnts.c` como sigue:

```
scds_syslog(LOG_ERR, "Emitir una solicitud de inicio.");
   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,
          "No se ha podido detener HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "HA-XFS detenido satisfactoriamente.");
   return (SCHA_ERR_NOERR); /* Detenido satisfactoriamente */
```

Observe lo siguiente sobre la llamada de `svc_stop()` a la función `scds_pmf_stop()`.

- El parámetro `SCDS_PMF_TYPE_SVC` identifica el programa que se va a detener como aplicación de servicio de datos; este método también puede detener un supervisor de fallos u otro tipo de aplicación.
- El parámetro `SCDS_PMF_SINGLE_INSTANCE` identifica la señal.

- El parámetro `SIGTERM` identifica la señal que se va a usar para detener la instancia del recurso. Si la señal no logra detener la instancia, `scds_pmf_stop()` envía `SIGKILL` para detenerla y, si esto también falla, retorna un error de tiempo de espera agotado. Consulte la página de comando man `scds_pmf_stop`(3HA) para obtener más información.
- El valor de tiempo de espera es el que indica la propiedad `Stop_timeout` del recurso.

Nota – Antes de salir, el método `xfnts_stop` invoca `scds_close()` para reclamar los recursos asignados por `scds_initialize()`. Consulte «Función `scds_initialize()`» en la página 144 y la página de comando man `scds_close`(3HA) para obtener más detalles.

Método `xfnts_monitor_start`

RGM invoca el método `Monitor_start` en un nodo para iniciar el supervisor de fallos después de que se inicie un recurso en el nodo. El método `xfnts_monitor_start` utiliza `scds_pmf_start()` para iniciar el daemon del supervisor bajo PMF.

Nota – La primera llamada de `xfnts_monitor_start` es a `scds_initialize()`, que ejecuta ciertas *tareas domésticas* necesarias («Función `scds_initialize()`» en la página 144 y la página de comando man `scds_initialize`(3HA) ofrecen información más detallada).

El método `xfnts_monitor_start` invoca el método `mon_start`, definido en `xfnts.c` como sigue:

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Llamar al método Monitor_start para el recurso <%s>.",
      scds_get_resource_name(scds_handle));

    /* Invocar scds_pmf_start y pasar el nombre del analizador. */
   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,
          "No se ha podido iniciar el supervisor de fallos.");
      return (1);
   }
```

```
    scds_syslog(LOG_INFO,
        "Se ha iniciado el supervisor de fallos.");

    return (SCHA_ERR_NOERR); /* Supervisor de fallos iniciado satisfactoriamente */
}
```

Observe lo siguiente sobre la llamada de `svc_mon_start()` a la función `scds_pmf_start()`.

- El parámetro `SCDS_PMF_TYPE_MON` identifica el programa que se va a iniciar como supervisor de fallos; este método también puede iniciar un servicio de datos u otro tipo de aplicación.
- El parámetro `SCDS_PMF_SINGLE_INSTANCE` lo identifica como un recurso de una sola instancia.
- El parámetro `xfnts_probe` identifica el daemon de supervisor que se debe iniciar. Se presupone que el daemon del supervisor está en el mismo directorio que los otros programas de rellamada.
- El parámetro final, `0`, especifica el nivel de supervisión de secundarios; en este caso, sólo se supervisa el daemon del supervisor.

Nota – Antes de salir, el método `xfnts_monitor_start` invoca `scds_close()` para reclamar recursos asignados por `scds_initialize()`. Consulte «Función `scds_initialize()`» en la página 144 y la página de comando man `scds_close`(3HA) para obtener más detalles.

Método `xfnts_monitor_stop`

Dado que el método `xfnts_monitor_start` utiliza `scds_pmf_start()` para iniciar el daemon del supervisor bajo PMF, `xfnts_monitor_stop` emplea `scds_pmf_stop()` para detener el daemon.

Nota – La primera llamada de `xfnts_monitor_stop` es a `scds_initialize()`, que ejecuta ciertas *tareas domésticas* necesarias («Función `scds_initialize()`» en la página 144 y la página de comando man `scds_initialize`(3HA) ofrecen información más detallada.

El método `xfnts_monitor_stop()` invoca el método `mon_stop`, que se define en `xfnts.c` como se indica a continuación.

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Llamar al método scds_pmf_stop");

   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,
          "No se ha podido detener el supervisor de fallos.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Se ha detenido el supervisor de fallos.");

   return (SCHA_ERR_NOERR); /* Supervisor de fallos detenido satisfactoriamente*/
}
```

Observe lo siguiente sobre la llamada de `svc_mon_stop()` a la función `scds_pmf_stop()`.

- El parámetro `SCDS_PMF_TYPE_MON` identifica el programa que se va a detener como supervisor de fallos; este método también puede detener un servicio de datos u otro tipo de aplicación.
- El parámetro `SCDS_PMF_SINGLE_INSTANCE` lo identifica como un recurso de una sola instancia.
- El parámetro `SIGKILL` identifica la señal que se va a usar para detener la instancia del recurso. Si esta señal no puede detener la instancia, `scds_pmf_stop()` devuelve un error de tiempo de espera agotado. Consulte la página de comando man `scds_pmf_stop`(3HA) para obtener más información.
- El valor de tiempo de espera es el que indica la propiedad `Monitor_stop_timeout` del recurso.

Nota – Antes de salir, el método `xfnts_monitor_stop` invoca `scds_close()` para reclamar recursos asignados por `scds_initialize()`. Consulte «Función `scds_initialize()`» en la página 144 y la página de comando man `scds_close`(3HA) para obtener más detalles.

Método `xfnts_monitor_check`

RGM invoca el método `Monitor_check` siempre que el supervisor de fallos intenta realizar una operación de recuperación de fallos a otro nodo con el grupo de recursos donde se encuentra el recurso en cuestión. El método `xfnts_monitor_check` invoca

el método `svc_validate()` para verificar que exista una configuración adecuada para admitir el daemon `xfs` (consulte «Método `xfnts_validate`» en la página 159 para obtener más información). El código de `xfnts_monitor_check` es el siguiente.

```
   /* Procesar los argumentos que pasa RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido iniciar el manejo.");
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "método monitor_check"
       "se ha llamado y ha devuelto <%d>.", rc);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado de la ejecución del método de validación dentro de la
    * comprobación del supervisor */
   return (rc);
}
```

Supervisor de fallos `SUNW.xfnts`

RGM no invoca directamente el método `PROBE`, sino que invoca el método `Monitor_start` para iniciar el supervisor después de que se inicie un recurso en un nodo. El método `xfnts_monitor_start` inicia el supervisor de fallos bajo el control de PMF. El método `xfnts_monitor_stop` detiene el supervisor de fallos.

El supervisor de fallos `SUNW.xfnts` realiza las operaciones siguientes:

- Supervisa periódicamente el estado del daemon del servidor `xfs` con utilidades diseñadas especialmente para comprobar servicios sencillos basados en TCP, como `xfs`.
- Realiza un seguimiento de los problemas que encuentra la aplicación en un marco temporal (con las propiedades `Retry_count` y `Retry_interval`) y opta por reiniciar o realizar una operación de recuperación de fallos del servicio de datos si la aplicación falla completamente. Las funciones `scds_fm_action()` y `scds_fm_sleep()` proporcionan soporte integrado para este mecanismo de rastreo y toma de decisiones.
- Implementa la decisión de reinicio o recuperación de fallos con `scds_fm_action()`.
- Actualiza el estado del recurso y lo pone a disposición de las herramientas administrativas y las interfaces gráficas de usuario.

Bucle principal de `xfonts_probe`

El método `xfonts_probe` implementa un bucle. Antes de hacerlo, `xfonts_probe` ejecuta estas acciones:

- Recupera los recursos de dirección de red para el recurso xfnts, como se indica a continuación.

```
/* Obtener la dirección ip disponible para este recurso */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No existe un recurso de dirección de la red en el
           grupo de recursos.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Devolver un error si no hay recursos de red */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
          "No existe un recurso de dirección de la red en el
           grupo de recursos.");
      return (1);
   }
```

- Llama a `scds_fm_sleep()` y pasa el valor de `Thorough_probe_interval` como valor de tiempo de espera. El análisis reposa durante el valor de `Thorough_probe_interval`.

```
timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {
      /*
       * reposo durante un tiempo de thorough_probe_interval entre
       * análisis sucesivos.
       */
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));
```

El método `xfnts_probe` implementa el bucle como sigue:

```
for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Tomar el nombre de sistema y puerto en el que se va a
          * supervisar el estado.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS admite un solo puerto y obtiene el
          * valor del puerto de la primera
          * entrada de la matriz de puertos.
          */
```

```
        ht1 = gethrtime(); /* Bloquear el tiempo de inicio del análisis */
        scds_syslog(LOG_INFO, "Analizar el servicio en el puerto: %d.", port);

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

        /*
         * Actualizar historial de análisis de servicio,
         * tomar medidas si fuera necesario.
         * Bloquear el tiempo de finalización del análisis.
         */
        ht2 = gethrtime();

        /* Convertir a milisegundos */
        dt = (ulong_t)((ht2 - ht1) / 1e6);

        /*
         * Calcular el historial de fallos y tomar
         * medidas si corresponde
         */
        (void) scds_fm_action(scds_handle,
            probe_result, (long)dt);
      }   /* Cada recurso de red */
   }    /* Seguir analizando indefinidamente */
```

La función `svc_probe()` implementa la lógica del análisis. El valor de retorno de `svc_probe()` se pasa a `scds_fm_action()`, que determina si debe reiniciar la aplicación, realizar una recuperación de fallos del grupo de recursos o si no debe hacer nada.

Función `svc_probe()`

La función `svc_probe()` establece una conexión de zócalo simple al puerto especificado, mediante una llamada a `scds_fm_tcp_connect()`. Si falla la conexión, `svc_probe()` retorna un valor de `100` que indica que el fallo es total. Si la conexión es satisfactoria, pero falla la desconexión, `svc_probe()` devuelve un valor de `50`, lo que indica un fallo parcial. Si la conexión y la desconexión son satisfactorias, `svc_probe()` devuelve un valor de `0`, que indica que la operación ha sido satisfactoria.

El código de `svc_probe()` es el siguiente.

```
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;


/*
 * analizar el servicio de datos mediante una conexión de zócalo al
 * puerto especificado en la propiedad port_list property del sistema
 * que sirve al servicio de datos de XFS. Si el servicio XFS que se ha
 * configurado para recibir en el puerto especificado responde a la
 * conexión, el análisis es satisfactorio. En caso contrario, esperar
 * durante un tiempo definido en la propiedad probe_timeout antes de
 * determinar que el análisis ha fallado.
 */

/*
 * Utilizar el porcentaje SVC_CONNECT_TIMEOUT_PCT de tiempo de
 * espera para conectarse al puerto
 */
connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
t1 = (hrtime_t)(gethrtime()/1E9);

/*
 * el análisis realiza una conexión al nombre de sistema y puerto
 * especificados.
 * La conexión tiene un 95% del valor real de probe_timeout.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "No se ha podido conectar con el puero <%d> del
        recurso<%s>.",
       port, scds_get_resource_name(scds_handle));
   /* es un fallo total */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

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

/*
 * Calcular el tiempo real que tardó en conectarse. Debe ser
 * menor o igual que connect_timeout, el tiempo asignado
 * para la conexión. Si la conexión utiliza todo el tiempo que se
 * le asigna, el valor restante de probe_timeout que se pasa a
 * esta función se utilizará como tiempo de desconexión. En
 * caso contrario, el tiempo restante de la llamada de conexión
 * se agregará también al tiempo de espera de desconexión.
 *
 */

time_used = (int)(t2 - t1);

/*
 * Usar el tiempo restante (timeout - time_took_to_connect)
   para la desconexión
 */
```

```
time_remaining = timeout - (int)time_used;

/*
 * Si se ha usado todo el tiempo, utilice un tiempo de espera
 * reducido y no modificable para seguir intentando la desconexión.
 * Así se evitará la fuga de fd.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe ha usado el tiempo total de espera de "
       "%d segundos durante la operación de conexión y ha"
       "sobrepasado el tiempo de espera en %d segundos."
       "Intentado desconectar con el tiempo de espera"
       "%d",
       connect_timeout,
       abs(time_used),
       SVC_DISCONNECT_TIMEOUT_SECONDS);

   time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Devolver fallo parcial en caso de fallo de desconexión.
 * Motivo: la llamada de conexión es satisfactoria, lo que
 * significa que la aplicación está activa. Un fallo de
 * desconexión se puede producir debido a una aplicación
 * bloqueada o a una carga excesiva.
 * Si fuera este último caso, no declarar la aplicación
 * terminada devolviendo un fallo completo. En su lugar,
 * declararlo fallo parcial. Si la situación persiste, la llamada
 * de desconexión volverá a fallar y se reiniciará la aplicación.
 */
rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
if (rc != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "No se ha podido desconectar el puerto %d del recurso %s.",
       port, scds_get_resource_name(scds_handle));
   /* es un fallo parcial */
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

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

/*
 * Si no queda tiempo, no realizar la prueba completa con
 * fsinfo. Devolver SCDS_PROBE_COMPLETE_FAILURE/2
 * en su lugar. Así se asegurará que si el tiempo de espera
 * agotado persiste, se reiniciará el servidor.
 */
if (time_remaining <= 0) {
   scds_syslog(LOG_ERR, "Probe timed out.");
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
```

```
    }

    /*
     * La conexión y desconexión del puerto han sido satisfactorias.
     * Ejecutar el comando fsinfo para realizar una comprobación
     * completa del estado del servidor.
     * Redirigir stdout o la salida de fsinfo
     * terminará en la consola.
     */
    (void) sprintf(testcmd,
        "/usr/openwin/bin/fsinfo -server %s:%d> /dev/null",
        hostname, port);
    scds_syslog_debug(DBG_LEVEL_HIGH,
        "Comprobar el estado del servidor con %s.", testcmd);
    if (scds_timerun(scds_handle, testcmd, time_remaining,
       SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

       scds_syslog(LOG_ERR,
          "No se ha podido comprobar el estado del servidor"
          "con el comando <%s>",
          testcmd);
       return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }
    return (0);
}
```

Al terminar, `svc_probe()` devuelve un valor de éxito (`0`), fallo parcial (`50`) o fallo total (`100`). El método `xfnts_probe` pasa este valor a `scds_fm_action()`.

Selección de la acción del supervisor de fallos

El método `xfnts_probe` invoca `scds_fm_action()` para determinar qué acción debe tomar. La lógica de `scds_fm_action()` es la siguiente:

- Mantener un historial acumulativo de fallos dentro del valor de la propiedad `Retry_interval`.
- Si el número acumulado de fallos alcanza el valor de 100 (fallo total) se debe reiniciar el servicio de datos. Si se supera `Retry_interval`, hay que poner a cero el historial.
- Si el número de reinicios supera el valor de la propiedad `Retry_count`, en el periodo que indica `Retry_interval`, se debe realizar una operación de recuperación de fallos del servicio de datos.

Por ejemplo, supongamos que el análisis establece una conexión con el servidor xfs, pero no logra desconectarse satisfactoriamente. Esto indica que el servidor está en ejecución, pero que es posible que esté bloqueado o bajo una carga temporal. El fallo de desconexión envía un fallo parcial (`50`) a `scds_fm_action()`. Este valor está por debajo del umbral para reiniciar el servicio de datos, pero se mantiene el valor en el historial de fallos.

Si durante el análisis siguiente, el servidor vuelve a fallar en la desconexión, se añadirá un valor de `50` al historial de fallos que mantiene `scds_fm_action()`. El valor acumulado de fallos es de `100`, por lo que `scds_fm_action()` reiniciará el servicio de datos.

Método `xfnts_validate`

RGM invoca el método `Validate` cuando se crea un recurso y cuando una acción administrativa actualiza las propiedades del recurso o del grupo que lo contiene. RGM invoca `Validate` antes de que se apliquen la creación o la actualización y un código de salida fallido del método en cualquier nodo provoque la cancelación de la creación o actualización.

RGM invoca `Validate` sólo cuando se cambian las propiedades del recurso o del grupo mediante una acción administrativa, no cuando RGM establece propiedades ni cuando un supervisor establece las propiedades de recurso `Status` y `Status_msg`.

Nota – El método `Monitor_check` invoca también explícitamente el método `Validate` cuando el método `PROBE` intenta realizar una operación de recuperación de fallos del servicio de datos a otro nodo.

RGM invoca `Validate` con argumentos adicionales a los que se pasan a otros métodos, incluidos las propiedades y los valores que se están actualizando. La llamada a `scds_initialize()` al principio de `xfnts_validate` analiza todos los argumentos que RGM pasa a `xfnts_validate` y guarda la información en el parámetro `scds_handle`. Las subrutinas que invoca `xfnts_validate` utilizan esta información.

El método `xfnts_validate` invoca `svc_validate()`, que verifica lo siguiente.

- La propiedad `Confdir_list` del recurso se ha configurado y define un único directorio.

```
scha_str_array_t *confdirs;
   confdirs = scds_get_ext_confdir_list(scds_handle);

/* Devolver error si no hay una propiedad de extensión confdir_list */
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "La propiedad Confdir_list no se ha establecido adecuadamente.");
      return (1); /* Fallo de validación */
   }
```

- El directorio que especifica `Confdir_list` contiene el archivo `fontserver.cfg`.

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

   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suprimir el error lint porque el prototipo de errno.h
       * no tiene un argumento vacío
       */
      scds_syslog(LOG_ERR,
          "No se ha podido acceder al archivo <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }
```

- El binario del daemon de servidor está accesible en el nodo del clúster.

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
      scds_syslog(LOG_ERR,
          "No se puede acceder al binario XFS: <%s> ", strerror(errno));
      return (1);
   }
```

- La propiedad Port_list especifica un solo puerto.

```
scds_port_list_t   *portlist;
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se puede acceder a la propiedad Port_list: %s.",
         scds_error_string(err));
      return (1); /* Fallo de validación */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
      scds_syslog(LOG_ERR,
          "La propiedad Port_list sólo debe tener un valor.");
      scds_free_port_list(portlist);
      return (1); /* Fallo de validación */
   }
#endif
```

- El grupo de recursos que contiene el servicio de datos incluye también al menos un recurso de dirección de red.

```
scds_net_resource_list_t *snrlp;
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
      != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No existe un recurso de dirección de la red en el grupo"
          "de recursos: %s.",
         scds_error_string(err));
      return (1); /* Fallo de validación */
   }
```

```
/* Devolver un error si no hay recursos de dirección de red */
if (snrlp == NULL || snrlp->num_netresources == 0) {
   scds_syslog(LOG_ERR,
       "No existe un recurso de dirección de la red en el grupo"
       "de recursos:.");
   rc = 1;
   goto finished;
}
```

Antes de retornar, `svc_validate()` libera todos los recursos asignados.

```
finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* devolver el resultado de la validación */
```

Nota – Antes de salir, el método `xfnts_validate` invoca `scds_close()` para reclamar los recursos asignados por `scds_initialize()`. Consulte «Función `scds_initialize()`» en la página 144 y la página de comando man `scds_close`(3HA) para obtener más detalles.

Método `xfnts_update`

RGM invoca el método `Update` para notificar a un recurso en ejecución que sus propiedades han cambiado. Las únicas propiedades que se pueden cambiar para el servicio de datos de `xfnts` corresponden al supervisor de fallos. Por tanto, siempre que se actualiza una propiedad, el método `xfnts_update` invoca `scds_pmf_restart_fm()` para reiniciar el supervisor de fallos.

```
   * comprobar si el supervisor de fallos está ya en ejecución y, en caso
   * afirmativo, detenerlo y reiniciarlo. El segundo parámetro de
   * scds_pmf_restart_fm() sólo identifica la instancia del supervisor
   * de fallos que hay que reiniciar.
   */

   scds_syslog(LOG_INFO, "Reiniciar el supervisor de fallos.");
   result = scds_pmf_restart_fm(scds_handle, 0);
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se ha podido reiniciar el supervisor de fallos.");
      /* Liberar toda la memoria asignada por scds_initialize */
      scds_close(&scds_handle);
```

```
   return (1);
}

scds_syslog(LOG_INFO,
"Se ha terminado satisfactoriamente.");
```

Nota – El segundo parámetro de `scds_pmf_restart_fm()` sólo identifica la instancia del supervisor de fallos que se va a reiniciar, en caso de que haya varias instancias. El valor `0` del ejemplo indica que sólo hay una instancia del supervisor de fallos.

CAPÍTULO 9

Agent Builder de SunPlex

En este capítulo se describe Agent Builder de SunPlex™ y el módulo Cluster Agent para Agent Builder, herramientas que automatizan la creación de tipos de recursos, o servicios de datos, para ejecutarlos bajo el Gestor de grupos de recursos (RGM). Un tipo de recurso es, básicamente, un envoltorio que rodea una aplicación para que pueda funcionar en un entorno del clúster, bajo el control de RGM.

Agent Builder proporciona una interfaz basada en pantallas para introducir información sencilla sobre la aplicación y el tipo de recurso que se desea crear. De acuerdo con la información introducida, Agent Builder genera el software siguiente:

- Un conjunto de archivos de origen (shells C, Korn (`ksh`) o GDS (servicio genérico de datos) para un tipo de recurso a prueba de fallos o escalable, según las rellamadas del método del tipo de recurso
- Un archivo personalizado de Registro de tipo de recurso (RTR) (si se genera un origen en los shells C o Korn)
- Secuencias de utilidades personalizadas para iniciar, detener y eliminar una instancia (recurso) del tipo de recurso, además de páginas de comando man personalizadas que explican cómo usar esos archivos
- Un paquete Solaris que incluye los binarios (si se genera un origen en C), un archivo RTR (si se genera un origen en los shells C o Korn) y las secuencias de utilidades

Agent Builder admite aplicaciones habilitadas para red, es decir, aplicaciones que utilizan la red para comunicarse con los clientes, y aplicaciones no habilitadas para red (autónomas). Agent Builder también permite generar un tipo de recurso para una aplicación que tenga múltiples árboles de procesos independientes que la Prestación del supervisor de procesos (PMF) deba supervisar y reiniciar individualmente (consulte «Creación de tipos de recursos con varios árboles de proceso independientes» en la página 173).

Los temas que abarca este capítulo son los siguientes:

- «Utilización de Agent Builder» en la página 164

Utilización de Agent Builder

Esta sección explica cómo utilizar Agent Builder, incluidas las tareas que se deben realizar antes de poder usarlo, y cómo usarlo después de haber generado el código del tipo de recurso.

Análisis de la aplicación

Antes de usar Agent Builder debe determinar si la aplicación cumple los criterios para tener una alta disponibilidad y escalabilidad. Agent Builder no puede realizar este análisis, que se basa únicamente en las características de tiempo de ejecución de la aplicación. En «Análisis de la validez de la aplicación» en la página 29 puede obtener más información sobre este tema.

Es posible que Agent Builder no pueda crear siempre un tipo de recurso completo para una aplicación, aunque en la mayoría de los casos facilitará al menos una solución parcial. Por ejemplo, algunas aplicaciones más complejas pueden requerir un código adicional que Agent Builder no genera de forma predeterminada, como el código para agregar comprobaciones de validación de propiedades adicionales o para ajustar parámetros que Agent Builder no muestra. En esos casos, es necesario realizar cambios en el código fuente generado o el archivo RTR. Agent Builder está diseñado precisamente para permitir esta flexibilidad.

Agent Builder sitúa comentarios en determinados puntos del código fuente generado para que se pueda añadir código del tipo de recurso concreto. Después de hacer cambios en el código fuente, se puede usar el makefile generado por Agent Builder para recompilar el código fuente y volver a generar el paquete del tipo de recurso.

Aunque se escriba todo el código del tipo de recurso sin utilizar ningún código generado por Agent Builder, se puede aún aprovechar el makefile y la estructura que facilita Agent Builder para crear el paquete de Solaris para ese tipo de recurso.

Instalación y configuración de Agent Builder

Agent Builder no requiere ninguna instalación especial. Agent Builder se incluye en el paquete `SUNWscdev` que se instala de forma predeterminada con la instalación estándar del software Sun Cluster (en *Sun Cluster 3.1 10/03: Guía de instalación del software* encontrará más información). Antes de utilizar Agent Builder, compruebe la información siguiente:

- Java se incluye en la variable `$PATH`. Agent Builder depende de Java (Java Development Kit versión 1.3.1 o superior). Si Java no está en `$PATH`, `scdsbuilder` retorna con un mensaje de error.
- Si ha instalado el grupo de software "Sistema de soporte del desarrollador" de Solaris 8 o superior.
- El compilador `cc` se incluye en la variable `$PATH`. Agent Builder utiliza la primera aparición de `cc` en la variable `$PATH` para identificar el compilador con el que generará código binario de C para el tipo de recurso. Si `cc` no está incluido en `$PATH`, Agent Builder inhabilita la opción para generar código en C (consulte «Utilización de la pantalla de creación» en la página 167.

Nota – Con Agent Builder puede usar un compilador diferente del compilador `cc` estándar. Para ello debe crear un vínculo simbólico en `$PATH` de `cc` a un compilador diferente, como `gcc`, o bien cambiar la especificación del compilador en el makefile (actualmente, `CC=cc`) a la ruta completa de un compilador diferente. Por ejemplo, en el makefile que genera Agent Builder, cambie `CC=cc` por `CC=`*`nombre_ruta`*`/gcc`. En este caso, no es posible ejecutar directamente Agent Builder. Utilice los comandos `make` y `make pkg` para generar un código de servicio de datos y un paquete.

Ejecución de Agent Builder

Para ejecutar Agent Builder escriba el comando siguiente:

```
% /usr/cluster/bin/scdsbuilder
```

Aparece la pantalla inicial de Sun Builder, como muestra la figura siguiente.

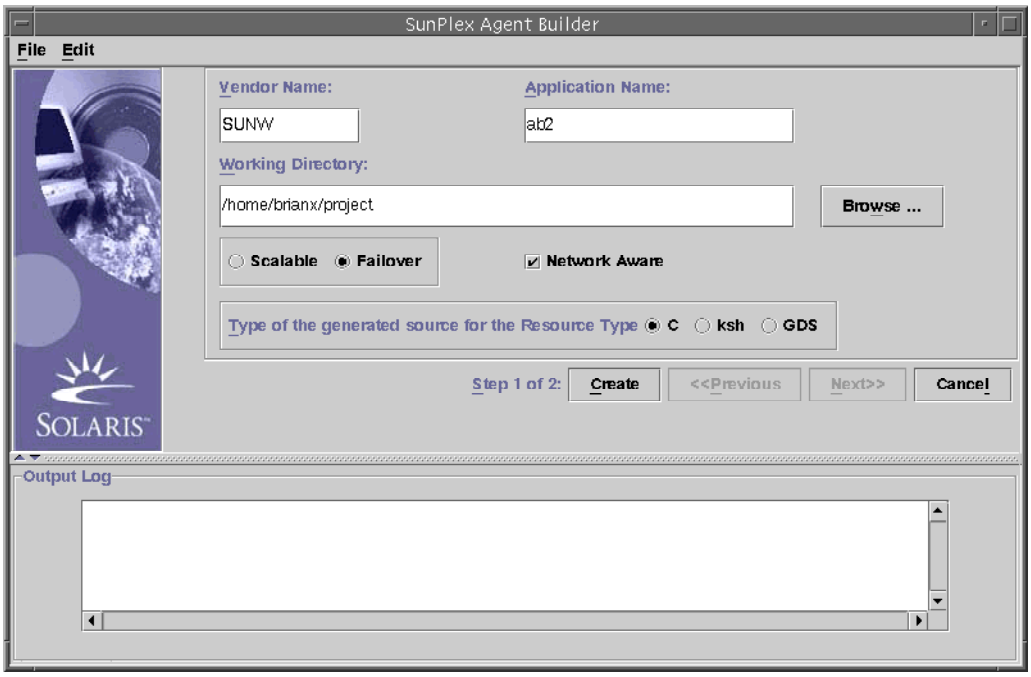


FIGURA 9–1 Pantalla inicial

Nota – Es posible acceder a Agent Builder a través de una interfaz de línea de comandos (consulte «Utilización de la versión de línea de comandos de Agent Builder» en la página 175) si la versión de la GUI no estuviera accesible.

Agent Builder proporciona dos pantallas que sirven de guía del proceso de creación de un nuevo tipo de recurso:

1. **Creación:** en esta pantalla se suministra información básica sobre el tipo de recurso que se va a crear, como el nombre y el directorio de trabajo (es decir, el directorio en el que se crea y configura la plantilla del tipo de recurso) de los archivos generados. También se indica el tipo de recurso que se va a crear (escalable o a prueba de fallos), si la aplicación básica va a estar habilitada para la red (es decir, si utiliza la red para comunicarse con los clientes) y el tipo de código (C, `ksh` o GDS) que se va a generar. Para obtener información sobre GDS (servicio genérico de datos), consulte el Capítulo 10. Debe introducir la información de esta pantalla y seleccionar **Crear** para generar la salida correspondiente, antes de poder mostrar la pantalla de configuración.
2. **Configuración:** en esta pantalla se le pide que proporcione un comando para iniciar la aplicación. También puede proporcionar comandos para detener y analizar la aplicación. Si no especifica estos comandos, la salida generada utiliza

señales para detener la aplicación y proporciona un mecanismo de análisis predeterminado (si desea ver la descripción del comando de análisis, consulte «Utilización de la pantalla de configuración» en la página 170). Esta pantalla también permite cambiar los valores de tiempo de espera para los tres siguientes comandos.

Nota – Si ejecuta Agent Builder desde el directorio de trabajo de un tipo de recurso, Agent Builder inicializa las pantallas de creación y configuración para incluir los valores del tipo de recurso.

Consulte «Desplazamientos por Agent Builder» en la página 180 si tiene alguna duda sobre cómo utilizar cualquiera de los botones o comandos de menú en las pantallas de Agent Builder.

Utilización de la pantalla de creación

El primer paso para crear un tipo de recurso consiste en cumplimentar la pantalla de creación que aparece al ejecutar Agent Builder. La figura siguiente muestra la pantalla de creación después de haber introducido la información en los campos.

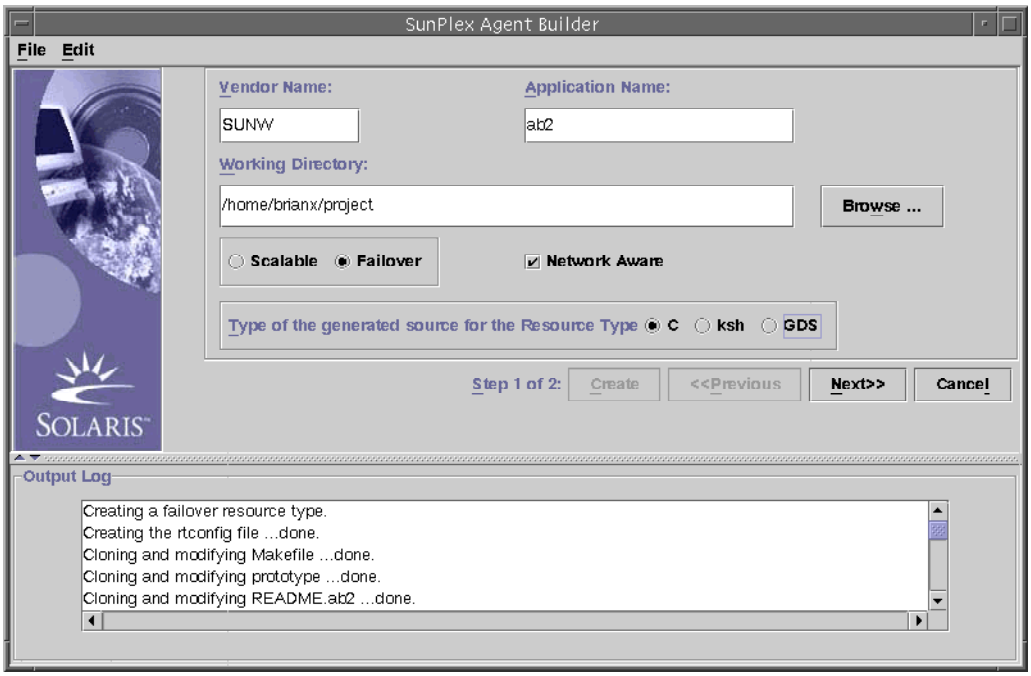


FIGURA 9–2 Pantalla de creación

La pantalla de creación contiene los campos, botones de selección y casillas de verificación siguientes:

- **Nombre proveedor:** un nombre que identifica al fabricante del tipo de recurso. Generalmente, se especifica el símbolo bursátil del fabricante, pero se puede utilizar cualquier nombre que lo identifique de forma única. Utilice únicamente caracteres alfanuméricos.
- **Nombre de aplicación:** el nombre del tipo de recurso. Utilice únicamente caracteres alfanuméricos.

Nota – El nombre de la aplicación y del fabricante, juntos, componen el nombre completo del tipo de recurso. El nombre completo no puede superar los nueve caracteres.

- **Directorio de trabajo:** el directorio en el cual Agent Builder crea una estructura de directorios para acoger todos los archivos que se creen para el tipo de recurso de destino. Sólo se puede crear un tipo de recurso por directorio de trabajo. Agent Builder indica en este campo la ruta al directorio desde el que se ha ejecutado Agent Builder, aunque se puede escribir un nombre diferente o utilizar el botón

Examinar para buscar un directorio diferente.

En el directorio de trabajo, Agent Builder crea un subdirectorio con el nombre del tipo de recurso. Por ejemplo, si `SUNW` es el nombre del fabricante y `ftp` el de la aplicación, Agent Builder llamará al subdirectorio `SUNWftp`.

Agent Builder coloca todos los directorios y archivos del tipo de recurso de destino en este subdirectorio (consulte «Estructura de directorios» en la página 175).

- **Escalable o A prueba de fallos:** especifique si el tipo de recurso de destino va a ser a prueba de fallos o escalable.
- **Preparado para la red:** especifique si la aplicación básica está habilitada para la red; es decir, si utiliza la red para comunicarse con sus clientes. Marque la casilla de verificación para especificar la habilitación para red; déjela en blanco para especificar que no está habilitada para la red. El código del shell Korn requiere que la aplicación esté habilitada para red. Por eso, Agent Builder marca la casilla y la ensombrece si usted selecciona los botones **ksh** o **GDS**.
- **C, ksh:** especifica el lenguaje del código fuente generado. Aunque estas opciones se excluyen mutuamente, con Agent Builder puede crear un tipo de recurso con código generado con `ksh` y reutilizar la misma información para crear código generado con C (consulte «Clonación de un tipo de recurso» en la página 173).
- **GDS:** especifica que este servicio es un servicio genérico de datos. Consulte el Capítulo 10 para obtener información sobre la creación y configuración de un servicio genérico de datos.

Nota – Si el compilador `cc` no está en `$PATH`, Agent Builder oscurece el botón de la opción de C y selecciona el botón **ksh**. Para especificar un compilador diferente, consulte la nota al final de «Instalación y configuración de Agent Builder» en la página 165.

Después de haber introducido la información solicitada, haga clic en el botón **Crear**. El registro de salida de la parte inferior de la pantalla muestra las acciones que está realizando Agent Builder. Puede utilizar el comando **Guardar registro de salida** del menú Editar para guardar la información en el registro.

Al finalizar, Agent Builder muestra un mensaje de realización satisfactoria o de advertencia que indica que no ha sido posible completar esta operación satisfactoriamente y que se debería comprobar el registro de salida para ver los detalles.

Si Agent Builder concluye con éxito, haga clic en el botón **Siguiente** para que aparezca la pantalla de configuración que permitirá terminar de generar el tipo de recurso.

Nota – Aunque la generación de un tipo de recurso completo es un proceso de dos pasos, es posible salir de Agent Builder después de terminar el primer paso (creación) sin perder la información introducida ni el trabajo que ha realizado Agent Builder (consulte «Reutilización del trabajo terminado» en la página 173).

Utilización de la pantalla de configuración

La pantalla de configuración (figura siguiente) aparece después de que Agent Builder haya terminado de crear el tipo de recurso y de que se haya seleccionado el botón **Siguiente** en la pantalla de creación. No se puede acceder a la pantalla de configuración sin haber creado el tipo de recurso con anterioridad.

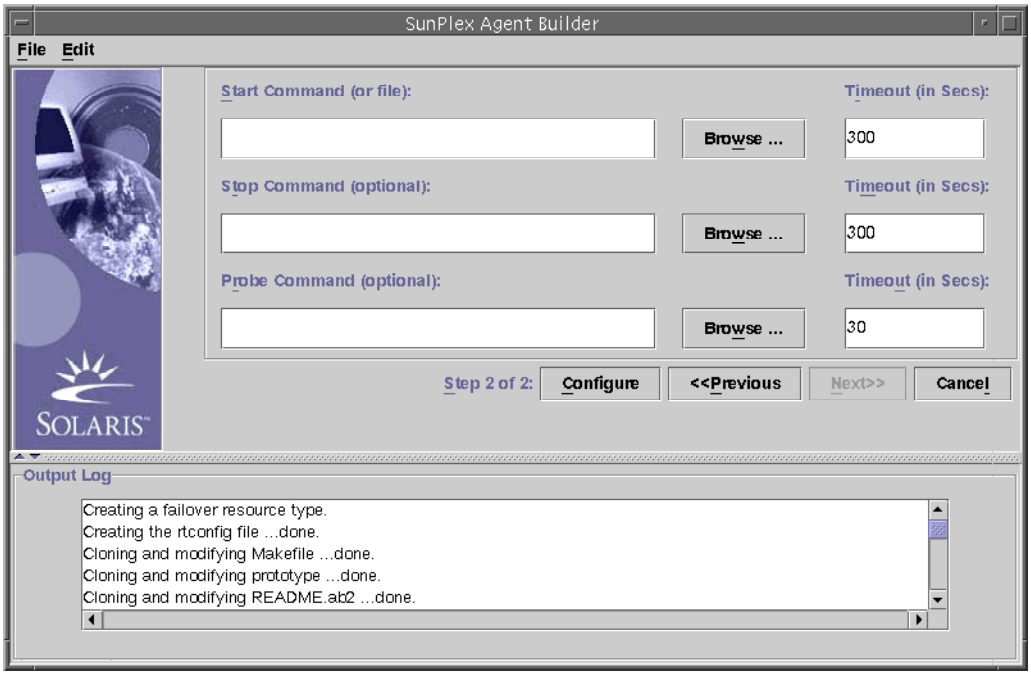


FIGURA 9–3 Pantalla de configuración

La pantalla de configuración contiene los campos siguientes:

- **Comando de inicio:** la línea de comandos completa que se puede pasar a cualquier shell de UNIX para iniciar la aplicación básica. Es necesario especificar este comando. Se puede escribir el comando en el campo suministrado o utilizar el botón **Examinar** para localizar un archivo que contenga el comando para iniciar la aplicación.

La línea de comandos completa debe incluir todo lo necesario para iniciar la aplicación, como nombres de sistema, números de puerto, una ruta a los archivos de configuración, etc. Si la aplicación requiere que se especifique un nombre de sistema en la línea de comandos, se puede usar la variable `$hostnames` que define Agent Builder (consulte «Utilización de la variable `$hostnames` de Agent Builder» en la página 172).

No ponga el comando entre comillas dobles (“”).

Nota – Si la aplicación básica tiene muchos árboles de procesos independientes, cada uno de los cuales se inicia con su propia etiqueta, bajo el control de PMF, no será posible especificar un solo comando. En su lugar, se debe crear un archivo de texto con comandos individuales para iniciar cada árbol de procesos y especificar la ruta a este archivo en el campo de texto Comando de inicio. Consulte «Creación de tipos de recursos con varios árboles de proceso independientes» en la página 173, donde se enumeran algunas características especiales, necesarias para que el archivo funcione correctamente.

- **Comando de parada:** la línea de comandos completa que se puede pasar a cualquier shell de UNIX para detener la aplicación básica. Puede escribir el comando en el campo suministrado o utilizar el botón Examinar para localizar un archivo que contenga el comando para detener la aplicación. Si la aplicación requiere que se especifique un nombre de sistema en la línea de comandos, se puede usar la variable `$hostnames` que define Agent Builder (consulte «Utilización de la variable `$hostnames` de Agent Builder» en la página 172).

  Este comando es opcional. Si no se especifica un comando de parada, el código generado emplea señales (en el método `Stop`) para detener la aplicación, como se muestra a continuación.

  - El método `Stop` envía `SIGTERM` para detener la aplicación y espera un 80% del valor del tiempo de espera para que la aplicación se cierre.
  - Si la señal `SIGTERM` no es satisfactoria, el método `Stop` envía `SIGKILL` para detener la aplicación y espera un 15% del valor del tiempo de espera para que la aplicación se cierre.
  - Si `SIGKILL` no es satisfactorio, el método `Stop` sale de forma no satisfactoria (el 5% restante del valor de tiempo de espera se considera una carga adicional indirecta).

Precaución – Asegúrese de que el comando de parada no retorne antes de que la aplicación se haya detenido por completo.

- **Comando de sonda:** un comando que se puede ejecutar periódicamente para comprobar el estado de la aplicación y devolver un estado de salida adecuado entre 0 (éxito) y 100 (fallo total). Este comando es opcional. Puede escribir la ruta

completa al comando o utilizar el botón Examinar para localizar un archivo que contenga los comandos para analizar la aplicación.

Generalmente, se especifica un cliente simple de la aplicación básica. Si no se especifica un comando de sonda, el código generado se limita a conectarse y desconectarse del puerto que utiliza el recurso y, si funciona, declara que la aplicación funciona correctamente. Los comandos de sonda sólo se pueden utilizar con aplicaciones habilitadas para la red. Agent Builder siempre genera un comando de sonda, pero lo desactiva para las aplicaciones que no estén habilitadas para la red.

Si la aplicación requiere que se especifique un nombre de sistema en la línea de comandos de sonda, puede usar la variable `$hostnames` que define Agent Builder (consulte «Utilización de la variable `$hostnames` de Agent Builder» en la página 172).

- **Tiempo de espera:** (para cada comando) un valor de tiempo de espera (en segundos) para cada comando. Se puede especificar un valor nuevo o aceptar el predeterminado de Agent Builder (300 segundos para inicio y parada y 30 para el análisis).

Utilización de la variable `$hostnames` de Agent Builder

Para muchas aplicaciones, especialmente las habilitadas para red, el nombre del sistema en el que la aplicación recibe y sirve las solicitudes del cliente se debe pasar a la aplicación en la línea de comandos. Por tanto, en muchos casos, el nombre del sistema es un parámetro que se debe especificar para iniciar, detener y analizar comandos del tipo de recurso de destino (en la pantalla de configuración). Sin embargo, el nombre del sistema en el que la aplicación recibe es específica del clúster; se determina cuando el recurso se ejecuta en un clúster y no se puede determinar cuando Agent Builder genera el código del tipo de recurso.

Para solucionar este problema, Agent Builder proporciona la variable `$hostnames` que se puede especificar en la línea de comandos de los comandos de inicio, parada y análisis. La variable `$hostnames` se especifica exactamente igual que un nombre de sistema real, por ejemplo:

```
/opt/network_aware/echo_server -p número_puerto -l $hostnames
```

Cuando un recurso del tipo de recurso de destino se ejecuta en un clúster, los nombres de sistema LogicalHostname o SharedAddress configurados para ese recurso (en la propiedad de recurso `Network_resources_used`) se reemplazan por el valor de la variable `$hostnames`.

Si configura la propiedad `Network_resources_used` con múltiples nombres de sistemas, la variable `$hostnames` los incluye todos, separados por comas.

Creación de tipos de recursos con varios árboles de proceso independientes

Agent Builder puede crear tipos de recursos para aplicaciones con varios árboles de procesos considerados independientes ya que PMF los supervisa e inicia individualmente. PMF inicia cada árbol de procesos con su propia etiqueta.

Nota – Agent Builder permite crear tipos de recursos con varios árboles de procesos independientes sólo cuando el código fuente generado que se especifique sea C. No es posible utilizar Agent Builder para crear estos tipos de recursos para `ksh` ni GDS; en estos casos, se debe escribir el código manualmente.

En el caso de una aplicación básica con varios árboles de procesos independientes, no se puede especificar una única línea de comandos para iniciar la aplicación. En su lugar, es necesario crear un archivo de texto, especificando en cada línea la ruta completa a un comando para iniciar cada uno de los árboles de procesos de la aplicación. Este archivo no debe contener líneas en blanco y se ha de especificar en el campo de texto del comando de inicio, en la pantalla de configuración.

También debe asegurarse de que este archivo de texto no tenga permisos de ejecución. Esto permite que Agent Builder diferencie este archivo, cuyo objeto es iniciar varios árboles de procesos, de una secuencia ejecutable simple con diferentes comandos. Si se dota al archivo de texto de permisos de ejecución, los recursos se ejecutarían correctamente en un clúster, pero todos los comandos se iniciarían bajo una única etiqueta de PMF, lo que impediría que éste pudiera supervisar y reiniciar individualmente los árboles de procesos.

Reutilización del trabajo terminado

Agent Builder permite aprovechar el trabajo finalizado de varias formas.

- Se puede clonar un tipo de recurso creado con Agent Builder.
- Se puede editar el código fuente que genera Agent Builder y recompilarlo para crear un paquete nuevo.

Clonación de un tipo de recurso

Siga este procedimiento para clonar un tipo de recurso, generado por Agent Builder.

1. **Cargue un tipo de recurso en Agent Builder; puede hacerlo de dos formas:**

   a. **Ejecute Agent Builder desde el directorio de trabajo (que contiene el archivo `rtconfig`) para un tipo de recurso creado con Agent Builder. Así, éste carga los valores de ese tipo de recurso en las pantallas de creación y configuración.**

b. Utilice el comando Cargar tipo de recurso del menú Archivo.

2. **Vaya al directorio de trabajo en la pantalla de creación.**

   Debe usar el botón **Examinar** para seleccionar un directorio; no basta con escribir el nombre del nuevo directorio. Una vez seleccionado un directorio, Agent Builder vuelve a habilitar el botón **Crear**.

3. **Realice los cambios.**

   Este procedimiento se puede utilizar para cambiar el tipo de código generado para el tipo de recurso. Por ejemplo, si inicialmente ha creado una versión en `ksh` de un tipo de recurso, pero más tarde se da cuenta de que necesita una versión en C, puede cargar el tipo de recurso en `ksh`, cambiar el lenguaje de salida a C y hacer que Agent Builder cree una versión en C del tipo de recurso.

4. **Cree el tipo de recurso clonado.**

   Seleccione **Crear** para crear el tipo de recurso. Seleccione **Siguiente** para que aparezca la pantalla de configuración. Seleccione **Configurar** para configurar el tipo de recurso y **Cancelar** para terminar.

Edición del código fuente generado

Para simplificar el proceso de creación de un tipo de recurso, Agent Builder limita el número de entradas, lo que reduce el alcance del tipo de recurso generado. Por tanto, para poder agregar funciones más avanzadas, como las comprobaciones de validación para las propiedades adicionales, o para ajustar parámetros que no muestra Agent Builder, es necesario modificar el código fuente generado o el archivo RTR.

Los archivos de origen están en el directorio *directorio_instalación*`/`*nombre_tr*`/src`. Agent Builder integra comentarios en el código fuente en los lugares donde se puede agregar código que son del tipo siguiente (para el código C):

```
/* ´Código agregado por el usuario -- INICIO vvvvvvvvvvvvvvv */
/* Código agregado por el usuario -- FIN   ^^^^^^^^^^^^^^^ */
```

Nota – Estos comentarios son iguales en el código del shell Korn, salvo por el hecho de que utilizan el símbolo de la libra esterlina (#) para iniciar la línea de comentario.

Por ejemplo, *nombre_tr*`.h` declara todas las rutinas de utilidades que emplean los diferentes programas. Al final de la lista de declaraciones se incluyen comentarios que permiten declarar las rutinas adicionales agregadas a cualquiera de los códigos.

Agent Builder genera también el makefile en el directorio *directorio_instalación*`/`*nombre_tr*`/src` con los destinos correspondientes. Utilice el comando `make` para volver a compilar el código fuente y el comando `make pkg` para regenerar el paquete del tipo de recurso.

El archivo RTR, que se encuentra en el directorio *directorio_instalación*/*nombre_tr*/`etc`, se puede editar con un editor de texto estándar (consulte «Establecimiento del recurso y las propiedades del tipo de recurso» en la página 34 para obtener más información sobre el archivo RTR y el Apéndice A para obtener información sobre las propiedades).

Utilización de la versión de línea de comandos de Agent Builder

La versión de línea de comandos de Agent Builder sigue el mismo proceso en dos pasos que la versión de interfaz gráfica de usuario, con la diferencia de que en lugar de introducir la información en la interfaz gráfica de usuarios, los parámetros se pasan a los comandos `scdscreate`(1HA) y `scdsconfig`(1HA).

Siga estos pasos para utilizar la versión de línea de comandos de Agent Builder:

1. **Utilice `scdscreate` para crear una plantilla del tipo de recurso de Sun Cluster para hacer que una aplicación tenga una alta disponibilidad (sea HA) o escalabilidad.**
2. **Utilice `scdsconfigure` para configurar la plantilla de tipo de recurso creada con `scdscreate`.**
3. **Cambie los directorios al subdirectorio `pkg` del directorio de trabajo.**
4. **Utilice el comando `pkgadd`(1M) para instalar los paquetes creados con `scdscreate`.**
5. **Si lo desea, edite el código fuente generado.**
6. **Ejecute la secuencia de inicio.**

Estructura de directorios

Agent Builder crea una estructura de directorios para contener todos los archivos que genera para el tipo de recurso de destino. El directorio de trabajo se especifica en la pantalla de creación. Se deben especificar directorios de instalación separados para cualquier tipo de recurso adicional que se desarrolle. En el directorio de trabajo, Agent Builder crea un subdirectorio cuyo nombre se forma con la unión de los nombres del fabricante y del tipo de recurso (desde la pantalla de creación). Por ejemplo, si especifica `SUNW` como nombre de fabricante y crea un tipo de recurso denominado `ftp`, Agent Builder creará un directorio llamado `SUNWftp` en el directorio de trabajo.

En este subdirectorio, Agent Builder crea y ocupa los directorios de la siguiente tabla.

Nombre del directorio	Contenido
`bin`	Para una salida en C, contiene los archivos binarios compilados de los archivos de origen. Para la salida en `ksh`, contiene los mismos archivos que el directorio `src`.
`etc`	Contiene el archivo RTR. Agent Builder une los nombres del fabricante y de la aplicación, separándolos con un punto, para formar el nombre del archivo RTR. Por ejemplo, si el nombre del fabricante es `SUNW` y el del tipo de recurso `ftp`, el nombre del archivo RTR será `SUNW.ftp`.
`man`	Contiene páginas de comando man personalizadas (man1m) para las secuencias de utilidades `start`, `stop` y `remove`. Por ejemplo, `startftp`(1M), `stopftp`(1M) y `removeftp`(1M). Para ver estas páginas de comando man, especifique la ruta con la opción `man -M`. Por ejemplo, `man -M` *directorio_instalación*`/SUNWftp/man removeftp`.
`pkg`	Contiene el paquete final.
`src`	Contiene los archivos de origen generados por Agent Builder.
`util`	Contiene las secuencias de utilidades `start`, `stop` y `remove` generadas por Agent Builder. Consulte «Secuencias de utilidades y páginas de comando man» en la página 178. Agent Builder adjunta el nombre de la aplicación a cada uno de estos nombres de secuencias; por ejemplo, `startftp`, `stopftp`, `removeftp`.

Salida

Esta sección describe la salida generada por Agent Builder.

Archivos binario y de origen

El Gestor de grupos de recursos (RGM), que gestiona grupos de recursos y, en última instancia, los recursos de un clúster, funciona sobre un modelo de rellamada. Cuando se producen eventos específicos, como fallos de nodos, RGM invoca los métodos del tipo de recurso para cada uno de los recursos en ejecución del nodo afectado. Por ejemplo, RGM invoca el método `Stop` para detener un recurso que se esté ejecutando en el nodo afectado y después invoca el método `Start` del recurso para iniciar éste en un nodo diferente. (Consulte «Modelo de RGM» en la página 21, «Métodos de rellamada» en la página 23 y la página de comando man `rt_callbacks`(1HA) para obtener más información sobre este modelo).

Para admitir este modelo, Agent Builder genera (en el directorio *directorio_instalación*/*nombre_tr*/bin) ocho programas ejecutables (C) o secuencias (ksh) que sirven de métodos de rellamada.

Nota – Estrictamente hablando, el programa *nombre_tr*_probe, que implementa un supervisor de fallos, no es un programa de rellamada. RGM no invoca directamente *nombre_tr*_probe, sino *nombre_tr*_monitor_start y *nombre_tr*_monitor_stop, que inician y detienen el supervisor de fallos, invocando *nombre_tr*_probe.

Los ocho métodos que genera Agent Builder son:

- *nombre_tr*_monitor_check
- *nombre_tr*_monitor_start
- *nombre_tr*_monitor_stop
- *nombre_tr*_probe
- *nombre_tr*_svc_start
- *nombre_tr*_svc_stop
- *nombre_tr*_update
- *nombre_tr*_validate

Consulte la página de comando man rt_callbacks(1HA) para obtener información específica sobre cada uno de estos métodos.

En el directorio *directorio_instalación*/*nombre_tr*/src (salida C), Agent Builder genera los archivos siguientes:

- Un archivo de cabecera (*nombre_tr*.h).
- Un archivo de origen (*nombre_tr*.c) que contiene el código común a todos los métodos.
- Un archivo objeto (*nombre_tr*.o) para el código común.
- Archivos de origen (*.c) para cada uno de los métodos
- Archivos objeto (*.o) para cada uno de los métodos.

Agent Builder vincula el archivo *nombre_tr*.o con cada uno de los archivos de método .o para crear los ejecutables del directorio *directorio_instalación*/*nombre_tr*/bin.

Para la salida ksh, los directorios *directorio_instalación*/*nombre_tr*/bin y *directorio_instalación*/*nombre_tr*/src son idénticos; ambos contienen las ocho secuencias ejecutables correspondientes a los siete métodos de rellamada y el método PROBE.

Nota – La salida ksh incluye dos programas de utilidad compilados (gettime y gethostnames) que algunos métodos de rellamada necesitan para obtener el tiempo y realizar un análisis.

Es posible editar el código fuente, ejecutar el comando `make` para recompilar el código y, al terminar, ejecutar el comando `make pkg` para generar otro paquete. Para poder realizar cambios en el código fuente, Agent Builder integra comentarios en el código fuente en los lugares donde es posible agregar nuevo código. Consulte «Edición del código fuente generado» en la página 174.

Secuencias de utilidades y páginas de comando man

Después de generar un tipo de recurso e instalar el paquete correspondiente en un clúster, todavía hay que obtener una instancia (recurso) del tipo de recurso que se ejecuta en un clúster, generalmente mediante comandos administrativos o SunPlex Manager. Sin embargo, para mayor comodidad, Agent Builder genera una secuencia de utilidades personalizada con este fin (la secuencia de inicio) así como secuencias para detener y eliminar un recurso del tipo de recurso de destino. Estas tres secuencias, ubicadas en el directorio *directorio_instalación*`/`*nombre_tr*`/util`, hacen lo siguiente:

- **Secuencia de inicio**: registra el tipo de recurso y crea los grupos de recursos y recursos necesarios. Crea también los recursos de dirección de red (LogicalHostname o SharedAddress) que permiten que la aplicación se comunique con los clientes de la red.
- **Secuencia de parada**: detiene e inhabilita el recurso.
- **Secuencia de eliminación**: deshace el trabajo de la secuencia de inicio, es decir, detiene y elimina los recursos, grupos de recursos y el tipo de recurso de destino del sistema.

Nota – La secuencia de eliminación sólo se puede utilizar con los recursos iniciados por la secuencia de inicio correspondiente, porque estas secuencias emplean convenciones internas para nombrar los recursos y grupos de recursos.

Agent Builder nombra estas secuencias añadiendo el nombre de la aplicación a los nombres de secuencias. Por ejemplo, si el nombre de la aplicación es `ftp`, las secuencias se denominan `startftp`, `stopftp` y `removeftp`.

Agent Builder proporciona páginas de comando man en el directorio *directorio_instalación*`/`*nombre_tr*`/man/man1m` de cada una de las secuencias de utilidades. Antes de ejecutar estas secuencias es recomendable leer estas páginas de comando man, porque documentan los parámetros que hay que pasar a las secuencias.

Para ver estas páginas de comando man, especifique la ruta al directorio del comando man con la opción `-M` del comando `man`. Por ejemplo, si `SUNW` es el fabricante y `ftp` es el nombre de la aplicación, utilice el comando siguiente para ver la página de comando man de `startftp`(1M):

```
man -M directorio_instalación/SUNWftp/man startftp
```

Las secuencias de utilidades de las páginas de comando man también están disponibles para el administrador del clúster. Cuando se instala un paquete generado por Agent Builder en un clúster, las páginas de comando man de las secuencias de utilidades se colocan en el directorio `/opt/`*nombre_tr*`/man`. Por ejemplo, utilice el comando siguiente para visualizar la página de comando man `startftp`(1M):

```
man -M /opt/SUNWftp/man startftp
```

Archivos de compatibilidad

Agent Builder coloca archivos de compatibilidad, como `pkginfo`, `postinstall`, `postremove` y `preremove` en el directorio *directorio_instalación*`/`*nombre_tr*`/etc`. Éste contiene también el archivo de registro del tipo de recurso (RTR), que declara las propiedades del recurso y tipo de recurso disponibles para el tipo de recurso de destino e inicializa los valores de las propiedades en el momento en que un recurso se registra en un clúster (consulte «Establecimiento del recurso y las propiedades del tipo de recurso» en la página 34 para obtener más información). El archivo RTR recibe el nombre de *nombre_fabricante*`.`*nombre_tipo_recurso*, por ejemplo, `SUNW.ftp`.

Este archivo se puede editar con un editor de texto estándar y no es necesario recompilar el código fuente para realizar los cambios. Sin embargo, se debe reconstruir el paquete con el comando `make pkg`.

Directorio de paquetes

El directorio *directorio_instalación*`/`*nombre_tr*`/pkg` contiene un paquete Solaris cuyo nombre es el resultado de la unión de los nombres del fabricante y de la aplicación, por ejemplo, `SUNWftp`. El `Makefile` del directorio *directorio_instalación*`/`*nombre_tr*`/src` admite la creación de un paquete nuevo. Por ejemplo, si realiza cambios en los archivos de origen y recompila el código o realiza cambios en las secuencias de utilidad del paquete, utilice el comando `make pkg` para crear un paquete nuevo.

Cuando se elimina un paquete de un clúster, el comando `pkgrm` puede fallar si se intenta ejecutarlo desde más de un nodo al mismo tiempo. Este problema se puede resolver de una de las dos formas siguientes:

- Ejecute la secuencia `remove` *nombre_tr* desde un nodo del clúster antes de ejecutar `pkgrm` desde cualquier nodo.
- Ejecute `pkgrm` desde un nodo del clúster, que se encarga de la reorganización necesaria. Después, ejecute `pkgrm` desde el resto de los nodos, simultáneamente si fuera necesario.

Si `pkgrm` falla porque se ha intentado ejecutarlo desde varios nodos al mismo tiempo, vuelva a ejecutarlo desde un solo nodo y después desde el resto.

El archivo `rtconfig`

Si se genera código fuente en C o `ksh`, Agent Builder genera, en el directorio de trabajo, un archivo de configuración `rtconfig` que contiene la información introducida en las pantallas de creación y configuración. Si ejecuta Agent Builder desde el directorio de trabajo para un tipo de recurso (o carga un tipo de recurso con el comando **Cargar tipo de recurso** del menú Archivo), Agent Builder lee el archivo `rtconfig` y cumplimenta las pantallas de creación y configuración con la información suministrada para el tipo de recurso. Esta característica resulta útil para clonar tipos de recursos (consulte «Clonación de un tipo de recurso» en la página 173.

Desplazamientos por Agent Builder

Desplazarse por Agent Builder es fácil e intuitivo. Agent Builder es un asistente en dos pasos que tiene una pantalla para cada paso (de creación y de configuración). En ambas pantallas, la información se introduce:

- Escribiendo la información en un campo.
- Examinando la estructura de directorios y seleccionando un archivo o un directorio.
- Marcando uno de los botones de selección que se excluyen mutuamente, por ejemplo, **Escalable** o **A prueba de fallos**.
- Marcando una casilla de activación y desactivación. Por ejemplo, si marca **Preparado para la red**, la aplicación básica estará habilitada para la red. Si no la marca, se tratará de una aplicación básica no habilitada para la red.

Los botones de la parte inferior de cada pantalla permiten finalizar la tarea, pasar a la pantalla anterior o siguiente o salir de Agent Builder. Agent Builder resalta u oscurece los botones según corresponda.

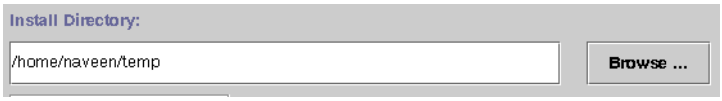
Por ejemplo, una vez cumplimentados los campos y seleccionadas las opciones deseadas en la pantalla de creación, haga clic en el botón **Crear** de la parte inferior de la pantalla. Los botones **Anterior** y **Siguiente** se oscurecen, porque no hay ninguna pantalla anterior y aún no se puede ir al paso siguiente, pues no se ha finalizado el actual.

Agent Builder muestra mensajes de progreso en el área de registro de salida de la parte inferior de la pantalla. Cuando termina, Agent Builder muestra un mensaje que indica que la operación se ha realizado satisfactoriamente o una advertencia para que se mire el registro de salida. En ese momento se resalta el botón **Siguiente** o, si se trata de la última pantalla, sólo se resalta el botón **Cancelar**.

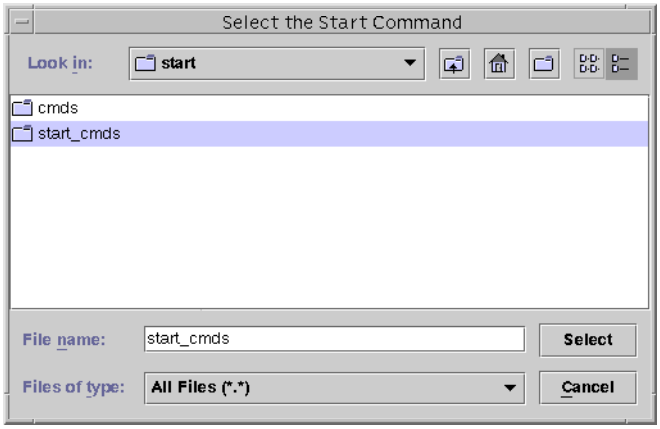
Si se desea, se puede pulsar **Cancelar** en cualquier momento para salir de Agent Builder.

Botón Examinar

Algunos campos de Agent Builder permiten escribir la información o hacer clic en el botón **Examinar** para recorrer la estructura de directorios y seleccionar un archivo o directorio.



Cuando se hace clic en **Examinar**, aparece una pantalla similar a ésta:



Haga doble clic en una carpeta para abrirla. Cuando se selecciona un archivo, el nombre aparece en el cuadro **Nombre de archivo**. Haga clic en **Seleccionar** cuando haya encontrado y seleccionado el archivo que desee.

Nota – Si está buscando un directorio, selecciónelo y pulse el botón **Abrir**. Si no hay subdirectorios, Agent Builder cierra la ventana Examinar y pone el nombre del directorio seleccionado en el campo correspondiente. Si el directorio tiene subdirectorios, haga clic en el botón Cerrar para cerrar la ventana Examinar y volver a la pantalla anterior. Agent Builder pone el nombre del directorio seleccionado en el campo correspondiente.

Los iconos de la esquina superior derecha de la pantalla hacen lo siguiente:

Este icono sube un nivel en el árbol de directorios.

Este icono vuelve a la carpeta de inicio.

Este icono crea una carpeta nueva en la carpeta seleccionada actualmente.

Ese icono, que sirve para cambiar entre las diferentes vistas, está reservado para usos futuros.

Menús

Agent Builder incluye los menús Archivo y Editar.

Menú Archivo

El menú Archivo tiene dos comandos:

- **Cargar tipo de recurso**: carga un tipo de recurso. Agent Builder proporciona una pantalla para examinar y seleccionar el directorio de trabajo para un tipo de recurso. Si éste existe en el directorio desde el cual se ejecuta Agent Builder, éste lo carga automáticamente. El comando **Cargar tipo de recurso** permite ejecutar Agent Builder desde cualquier directorio y seleccionar un tipo de recurso para usarlo como plantilla para crear un nuevo tipo de recurso (consulte «Clonación de un tipo de recurso» en la página 173).

- **Salir**: permite salir de Agent Builder. También se puede salir haciendo clic en **Cancelar** en las pantallas de creación o configuración.

Menú Editar

El menú Editar tiene comandos para borrar y guardar el registro de salida:

- **Borrar registro de salida**: borra la información del registro de salida. Cada vez que se seleccione Crear o Configurar, Agent Builder introduce mensajes de estado en el registro de salida. Si está realizando un proceso iterativo, realizando cambios en el código fuente y regenerando la salida de Agent Builder y desea separar los mensajes de estado, puede guardar y borrar el archivo de registro antes de cada utilización.
- **Guardar archivo de registro**: guarda la salida de registro en un archivo. Agent Builder proporciona una pantalla donde se puede elegir el directorio y especificar un nombre de archivo.

Módulo Cluster Agent para Agent Builder

El módulo Cluster Agent para Agent Builder es un módulo NetBeans™ que permite a los usuarios del producto Sun™ ONE Studio crear tipos de recursos, o servicios de datos, para el software Sun Cluster, en un entorno de desarrollo integrado. El módulo Cluster Agent proporciona una interfaz basada en pantallas para describir el tipo de recurso que se desea crear.

Nota – En la página web Documentation de Sun ONE Studio puede consultar información sobre cómo configurar, instalar y utilizar el producto Sun ONE Studio.

▼ Instalación y configuración del módulo Cluster Agent

El módulo Cluster Agent se instala al instalar el software Sun Cluster. La herramienta de instalación de Sun Cluster coloca el archivo del módulo Cluster Agent `scdsbuilder.jar` en `/usr/cluster/lib/scdsbuilder`. Para utilizar el módulo Cluster Agent con el software Sun ONE Studio, es necesario crear un enlace simbólico a este archivo.

Nota – Los productos Sun Cluster, Sun ONE Studio y Java™ 1.4 deben estar instalados y ser accesibles para el sistema en el que se va a ejecutar el módulo Cluster Agent.

1. **¿Desea permitir que todos los usuarios o sólo usted mismo utilicen el módulo Cluster Agent?**

   - Para permitir que lo utilicen todos los usuarios, conviértase en superusuario o asuma un papel equivalente y cree el enlace simbólico en el directorio del módulo global:

     ```
     # cd /opt/s1studio/ee/modules
     # ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

     Nota – Si ha instalado el software Sun ONE Studio en un directorio diferente de `/opt/s1studio/ee`, cambie esta ruta al directorio por la ruta correspondiente.

   - Para que sólo usted pueda usar el módulo, cree el enlace simbólico en el subdirectorio `modules`:

     ```
     % cd ~su_directorio_inicio/ffjuser40ee/modules
     % ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

2. **Detenga y reinicie el software Sun ONE Studio.**

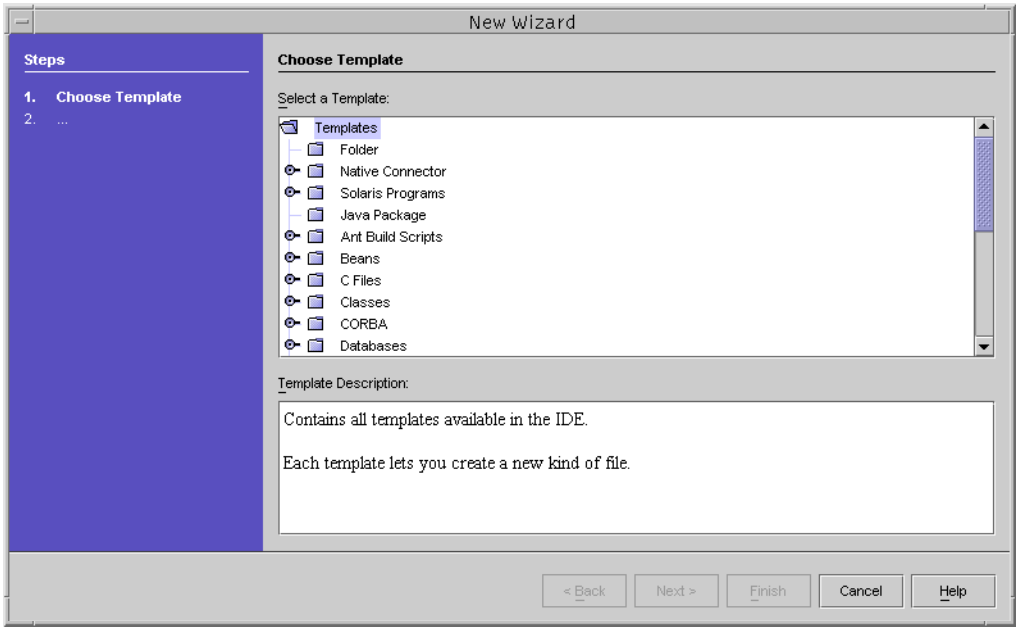
▼ Inicio del módulo Cluster Agent

Los pasos siguientes explican cómo iniciar el módulo Cluster Agent desde el software Sun ONE Studio.

1. **Desde el menú File de Sun ONE Studio, seleccione New o haga clic en este icono de la barra de herramientas:**

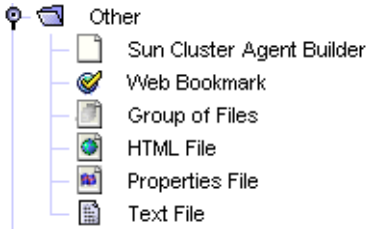
   

   Aparece la pantalla New Wizard.

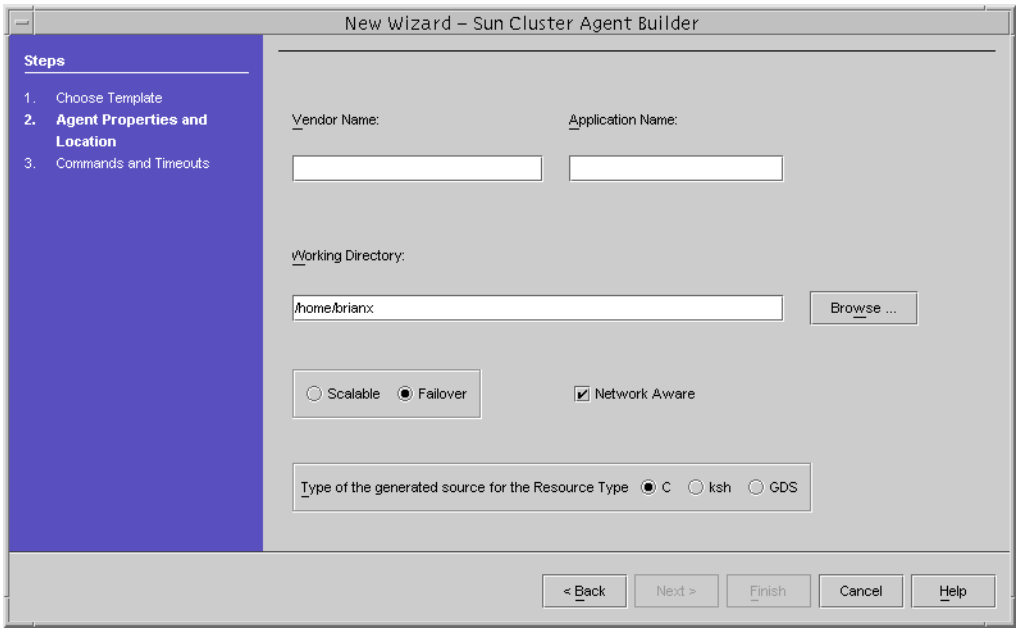


2. **En la ventana de selección de plantilla, desplácese hacia abajo (si fuera necesario) y haga clic en la tecla situada al lado de la carpeta Other:**

Se abre el menú Other.



3. **En el menú Other, seleccione Sun Cluster Agent Builder y haga clic en Next.**
   Se inicia el módulo Cluster Agent de Sun One Studio. Aparece la primera pantalla New Wizard - Sun Cluster Agent Builder.



Utilización del módulo Cluster Agent

Utilice el módulo Cluster Agent igual que el software Agent Builder. Las interfaces son idénticas. Por ejemplo, las figuras siguientes muestran que la pantalla de creación del software Agent Builder y la pantalla New Wizard - Sun Cluster Agent Builder del módulo Cluster Agent contienen los mismos campos y selecciones.

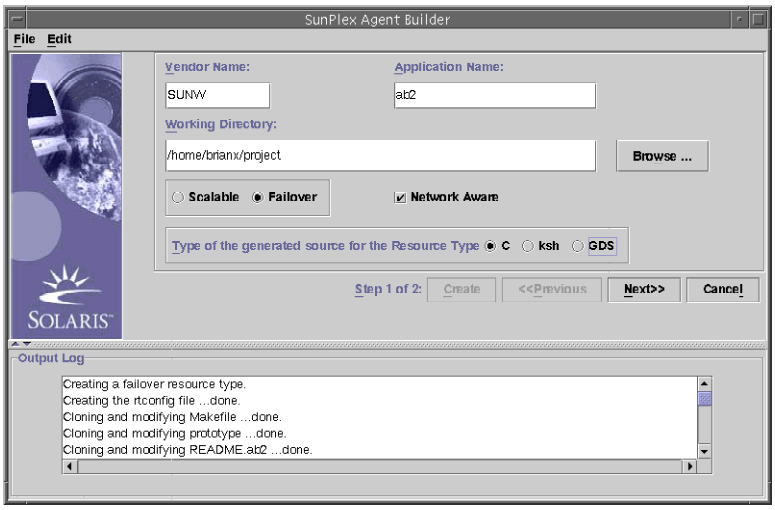


FIGURA 9–4 Pantalla de creación del software Agent Builder

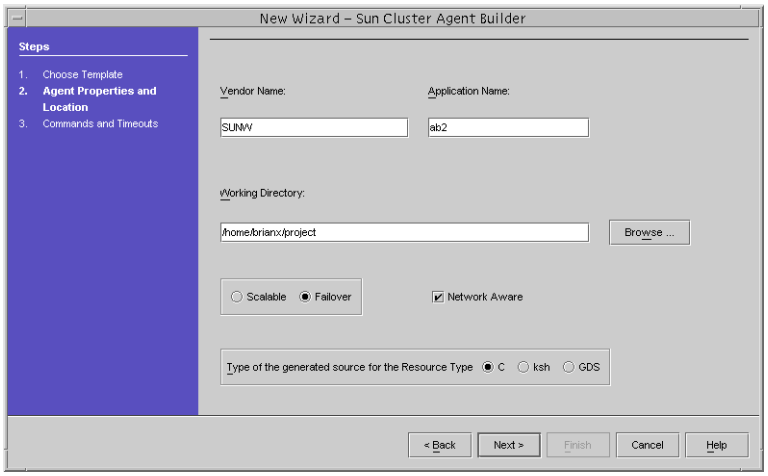


FIGURA 9–5 Pantalla New Wizard - Sun Cluster Agent Builder, en el módulo Cluster Agent

Diferencias entre el módulo Cluster Agent y Agent Builder

A pesar de sus semejanzas, existen pequeñas diferencias entre el módulo Cluster Agent y Agent Builder:

- En el módulo Cluster Agent, el tipo de recurso se crea y configura después de pulsar el botón Finish de la segunda pantalla New Wizard - Sun Cluster Agent Builder. El tipo de recurso *no* se crea al hacer clic en Next en la primera pantalla New Wizard - Sun Cluster Agent Builder.

  En Agent Builder, el tipo de recurso se crea al hacer clic en el botón Crear de la pantalla de creación y se configura al hacer clic en Configurar, en la pantalla de configuración.
- La información que aparece en la ventana del registro de salida de Agent Builder aparece en una ventana de salida aparte en el producto Sun ONE Studio.

CAPÍTULO 10

Servicios genérico de datos

Este capítulo proporciona información sobre el servicio genérico de datos (GDS) y muestra cómo crear un servicio que utilice GDS con SunPlex Agent Builder o los comandos administrativos estándar de Sun Cluster.

Información general de GDS

GDS es un mecanismo para dotar a aplicaciones sencillas habilitadas para la red de una alta disponibilidad o escalabilidad, mediante su conexión a una estructura de Gestión de grupos de recursos de Sun Cluster. Este mecanismo no requiere codificación de un agente, que es el método habitual para dotar una aplicación de alta disponibilidad y escalabilidad.

GDS es un servicio de datos único, precompilado. No es posible modificar el servicio de datos precompilado ni sus componentes, tampoco las implementaciones de método de rellamada (`rt_callbacks`(1HA)) ni el archivo de registro de tipo de recurso (`rt_reg`(4)).

Tipo de recurso precompilado

El tipo de recurso del servicio genérico de datos `SUNW.gds` se incluye en el paquete `SUNWscgds`. La utilidad `scinstall`(1M) instala este paquete durante la instalación del clúster. El paquete `SUNWscgds` incluye los archivos siguientes:

```
# 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
```

Razones para utilizar GDS

GDS tiene las ventajas siguientes con respecto a la utilización del modelo de código fuente generado por SunPlex Agent Builder (consulte `scdscreate`(1HA)) o los comandos administrativos estándar de Sun Cluster:

- GDS es fácil de usar.
- GDS y sus métodos están precompilados, por lo que no son modificables.
- SunPlex Agent Builder se puede utilizar para generar secuencias de control de la aplicación que se empaquetan en un paquete de Solaris que se puede reutilizar en varios clústers.

Formas de crear un servicio que utilice GDS

Hay dos formas de crear un servicio que utilice GDS:

- Con SunPlex Agent Builder
- Con los comandos administrativos estándar de Sun Cluster

GDS y SunPlex Agent Builder

Utilice SunPlex Agent Builder y seleccione GDS como tipo de código fuente generado. Los datos de entrada del usuario se utilizan para generar un conjunto de secuencias de control que configuren recursos para la aplicación en cuestión.

GDS y los comandos administrativos estándar de Sun Cluster

Este método utiliza el código de servicio de datos precompilado de `SUNWscgds` pero requiere que el administrador del sistema emplee los comandos administrativos estándar de Sun Cluster (`scrgadm`(1M) y `scswitch`(1M)) para crear y configurar el recurso.

Selección del método que se va a utilizar para crear el servicio basado en GDS

Como muestran los procedimientos «Cómo utilizar los comandos administrativos de Sun Cluster para crear un servicio de alta disponibilidad con GDS» en la página 199 y «Comandos administrativos estándar de Sun Cluster para crear un servicio escalable con GDS» en la página 200, hay que escribir mucho para emitir los comandos `scrgadm` y `scswitch` apropiados.

Con GDS, SunPlex Agent Builder simplifica el proceso, porque genera las secuencias de control que emiten los comandos `scrgadm` y `scswitch`.

Cuándo no se debe utilizar GDS

Aunque la utilización de GDS tiene muchas ventajas, hay casos en los que no es conveniente utilizarlo. GDS no resulta apropiado:

- Cuando se requiere un control mayor del que ofrecen los tipos de recursos precompilados, por ejemplo cuando hay que agregar propiedades de extensión o cambiar los valores predeterminados.
- Cuando hay que modificar el código fuente para agregar funciones especiales.
- Cuando desee utilizar árboles de procesos múltiples.
- Cuando desee utilizar aplicaciones no habilitadas para la red.

Propiedades necesarias para GDS

Se deben proporcionar las propiedades siguientes:

- `Start_command` (propiedad de extensión)
- `Port_list`

Propiedad de extensión `Start_command`

El comando de inicio, que se especifica en la propiedad de extensión `Start_command`, inicia la aplicación. Debe ser un comando UNIX, con sus argumentos, que se puede pasar directamente a un shell para iniciar la aplicación.

Propiedad `Port_list`

La propiedad `Port_list` identifica la lista de puertos en los que recibe la aplicación; se debe especificar en la secuencia de inicio que crea SunPlex Agent Builder o en el comando `scrgadm`, si se están utilizando los comandos administrativos estándar de Sun Cluster.

Propiedades opcionales para GDS

La propiedades opcionales para GDS son:

- `Network_resources_used`
- `Stop_command` (propiedad de extensión)
- `Probe_command` (propiedad de extensión)
- `Start_timeout`
- `Stop_timeout`
- `Probe_timeout` (propiedad de extensión)
- `Child_mon_level` (propiedad de extensión utilizada únicamente con los comandos administrativos estándar)
- `Failover_enabled` (propiedad de extensión)
- `Stop_signal` (propiedad de extensión)

Propiedad `Network_resources_used`

El valor predeterminado de esta propiedad es nulo. Esta propiedad se debe especificar si la aplicación debe vincularse a una o varias direcciones concretas. Si se omite esta propiedad o si se especifica como Null, se presupone que la aplicación recibe en todas las direcciones.

Antes de crear el recurso de GDS es necesario haber configurado un recurso `LogicalHostname` o `SharedAddress`. Consulte *Sun Cluster 3.1 Data Service Planning and Administration Guide* para obtener información sobre cómo configurar un recurso `LogicalHostname` o `SharedAddress`.

Para especificar un valor especifique uno o varios nombres de recurso; cada uno puede contener uno o varios `LogicalHostname` o `SharedAddress`. Consulte `r_properties`(5) para obtener más detalles.

Propiedad `Stop_command`

El comando de parada debe detener la aplicación y retornar sólo cuando la aplicación se haya detenido completamente. Debe ser un comando UNIX completo, que se pueda pasar directamente a un shell para detener la aplicación.

Si se da el comando `Stop_command`, el método de parada de GDS inicia el comando de parada con un 80% del tiempo de espera de parada. Independientemente del resultado del inicio del comando de parada, el método de parada de GDS envía `SIGKILL` después del 15% del tiempo de espera de parada. El 5% restante de ese tiempo se reserva para cargas adicionales indirectas de las tareas domésticas.

Si se omite el comando de parada, GDS intenta detener la aplicación con la señal que se especifica en `Stop_signal`.

Propiedad `Probe_command`

El comando de análisis comprueba periódicamente el estado de la aplicación. Debe ser un comando UNIX, con sus argumentos, que se puede pasar directamente a un shell para analizar la aplicación. El comando de análisis retorna con un estado de salida de 0 si la aplicación está bien.

El estado de salida del comando de análisis se utiliza para determinar la gravedad del fallo de la aplicación; se denomina estado de análisis y debe ser un número entero entre 0 (éxito) y 100 (fallo total). Puede tener también un valor especial de 201, lo que provoca una recuperación de fallos inmediata de la aplicación salvo que se establezca `Failover_enabled` en false. El estado de análisis se utiliza dentro del algoritmo de análisis de GDS (consulte `scds_fm_action`(3HA) para tomar la decisión de reiniciar la aplicación localmente en lugar de hacer una operación de recuperación de fallos a otro nodo; si el estado de salida es 201, se realiza inmediatamente una recuperación de fallos de la aplicación.

Si se omite el comando de análisis, GDS aporta su propio analizador sencillo, que conecta con la aplicación en el conjunto de direcciones IP derivadas de la propiedad `Network_resources_used` o la salida de `scds_get_netaddr_list`(3HA). Si la conexión es satisfactoria, se desconecta inmediatamente. Si tanto la conexión como la desconexión son satisfactorias, se considera que la aplicación está funcionando correctamente.

Nota – El analizador incluido con GDS sólo se pretende que sea un sustituto sencillo del analizador específico de la aplicación, que cuenta con todas las funciones.

Propiedad `Start_timeout`

Esta propiedad especifica el tiempo de espera de inicio del comando de inicio (consulte «Propiedad de extensión `Start_command`» en la página 191 para obtener información adicional). El valor predeterminado de `Start_timeout` es 300 segundos.

Propiedad `Stop_timeout`

Esta propiedad especifica el tiempo de espera de parada del comando de parada (consulte «Propiedad `Stop_command`» en la página 192 para obtener información adicional). El valor predeterminado de `Stop_timeout` es 300 segundos.

Propiedad `Probe_timeout`

Esta propiedad especifica el valor de tiempo de espera del comando de análisis (consulte «Propiedad `Probe_command`» en la página 193 para obtener información adicional). El valor predeterminado de `Probe_timeout` es 30 segundos.

Propiedad `Child_mon_level`

Esta propiedad proporciona control sobre los procesos que se supervisan a través de PMF. Indica el nivel máximo hasta el cual se supervisan los procesos secundarios bifurcados. Es similar al argumento `-C` del comando `pmfadm`(1M).

Omitir esta propiedad, o establecerla en su valor predeterminado de `-1`, tiene el mismo efecto que omitir la opción `-C` en el comando `pmfadm`; es decir, todos los secundarios (y sus descendientes) se supervisarán.

Nota – Esta opción sólo se puede especificar con los comandos administrativos estándar de Sun Cluster; no se puede especificar si se está usando SunPlex Agent Builder.

Propiedad `Failover_enabled`

Esta propiedad de extensión booleana controla el comportamiento de la recuperación de fallos del recurso. Si esta propiedad de extensión se establece en `true`, la aplicación realiza una operación de recuperación de fallos cuando el número de reinicios supere el valor `retry_count` dentro del número de segundos de `retry_interval`.

Si esta propiedad de extensión se fija en `false`, la aplicación no se reinicia ni realiza una recuperación de fallos a otro nodo cuando el número de reinicios supere el valor de `retry_count` del número de segundos de `retry_interval`.

Esta propiedad de extensión se puede utilizar para impedir que el recurso de aplicación inicie una recuperación de fallos del grupo de recursos. El valor predeterminado es `true`.

Propiedad `Stop_signal`

GDS utiliza el valor entero de esta propiedad de extensión para determinar la señal empleada para detener la aplicación con PMF. Consulte `signal`(3HEAD) para ver una lista de los valores enteros que se pueden especificar. El valor predeterminado es 15 (`SIGTERM`).

Utilización de SunPlex Agent Builder para crear un servicio con GDS

SunPlex Agent Builder puede utilizarse para crear el servicio que emplea GDS. SunPlex Agent Builder se describe con más detalle en el Capítulo 9.

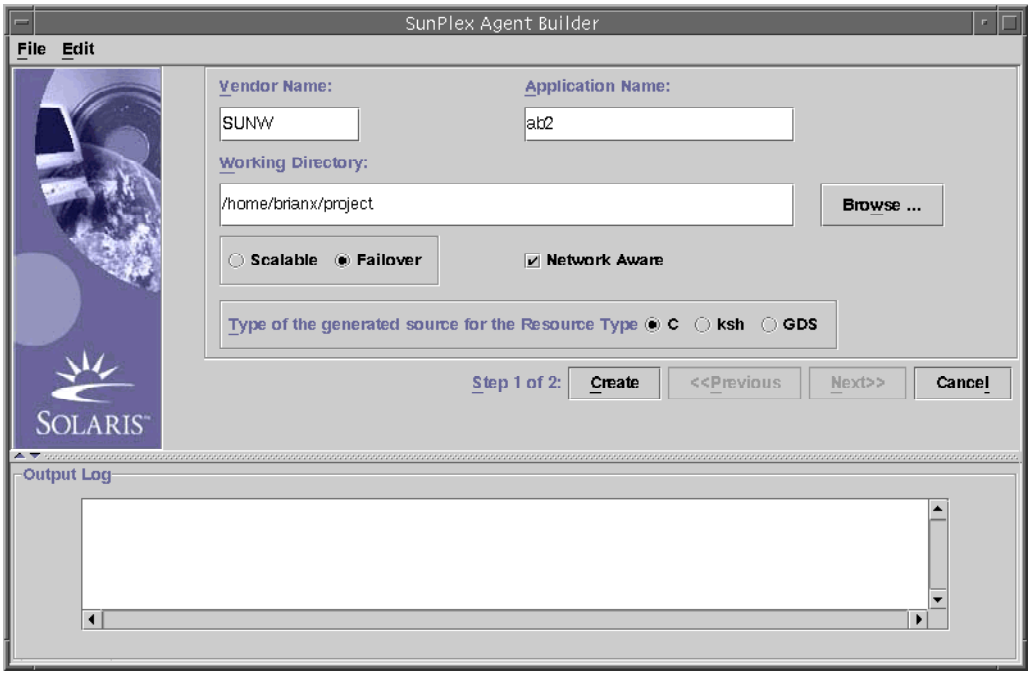
Creación de un servicio con GDS en SunPlex Agent Builder

▼ Creación de un servicio con GDS en Agent Builder

1. **Inicie SunPlex Agent Builder.**

   ```
   # /usr/cluster/bin/scdsbuilder
   ```

2. **Aparece el panel de SunPlex Agent Builder.**



3. **Escriba el nombre del fabricante.**

4. **Escriba el nombre de la aplicación.**

Nota – La combinación del nombre del fabricante y de la aplicación no puede contener más de nueve caracteres. Se utiliza como nombre del paquete de las secuencias de control.

---

5. **Vaya al directorio de trabajo.**

Se puede utilizar el desplegable del botón Examinar para seleccionar el directorio, en lugar de escribir la ruta.

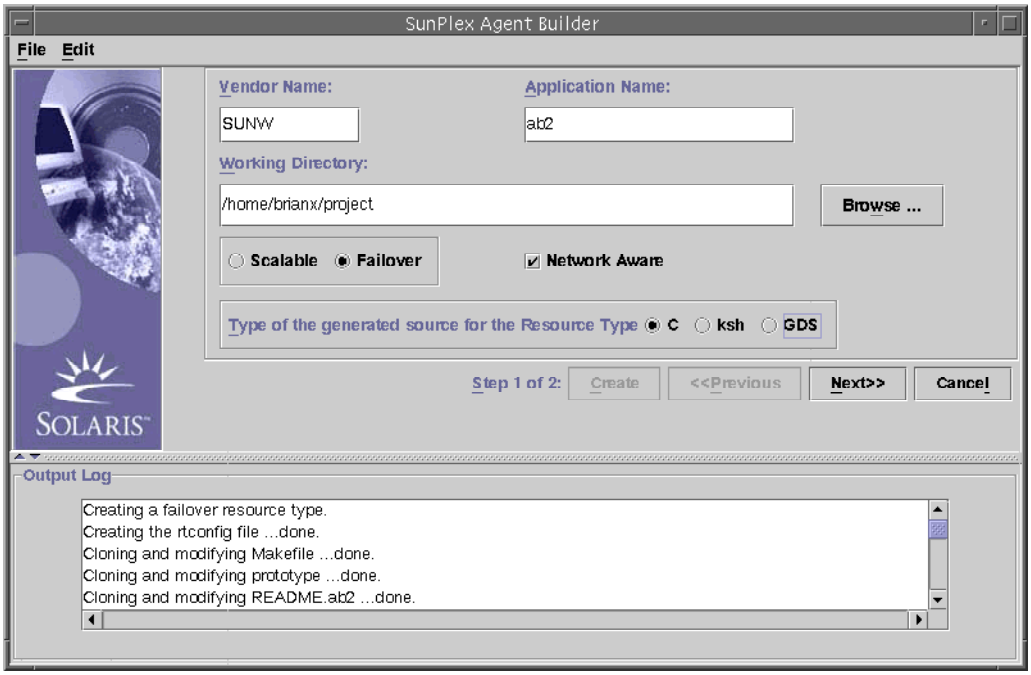
6. **Seleccione si el servicio de datos es escalable o a prueba de fallos.**

   No es necesario seleccionar la habilitación para red, porque es la opción predeterminada al crear GDS.

7. **Seleccione GDS.**

8. **Haga clic en el botón Crear para crear las secuencias de control.**

9. **El panel de SunPlex Agent Builder muestra los resultados de la creación del servicio. El botón Crear queda oscurecido; ya se puede utilizar el botón Siguiente.**



▼ Configuración de las secuencias de control

Después de crear las secuencias de control, es necesario utilizar SunPlex Agent Builder para configurar el nuevo servicio.

1. **Haga clic en Siguiente; se abrirá el panel de configuración.**

2. **Escriba la ubicación del comando de inicio o utilice el botón Examinar para buscarlo.**

3. **(Opcional) Escriba el comando de parada o utilice el botón Examinar para buscarlo.**

4. **(Opcional) Escriba el comando de análisis o utilice el botón Examinar para buscarlo.**

5. **(Opcional) Especifique los valores de tiempo de espera de los comandos de inicio, parada y análisis.**

6. **Haga clic en Configurar para iniciar la configuración de las secuencias de control.**

Nota – El nombre del paquete se forma con la unión del nombre del fabricante y el nombre de la aplicación.

Se crea un paquete de secuencias de control, que se ubica en:

<dir_trabajo>/*<nombre_fabricante><aplicación>*/pkg
Por ejemplo, /export/wdir/NETapp/pkg

7. **Instale el paquete terminado en todos los nodos del clúster.**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

8. **Los archivos siguientes se instalarán durante pkgadd:**

```
/opt/NETapp
/opt/NETapp/README.app
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Nota – Las páginas de comando man y los nombres de secuencias se corresponderán con el nombre de aplicación que se introdujo antes, precedido del nombre de la secuencia; por ejemplo, `startapp`.

Para ver las páginas de comando man hay que especificar la ruta a la página. Por ejemplo, para ver las páginas de comando man de `startapp` utilice:

```
# man -M /opt/NETapp/man startapp
```

9. **En un nodo del clúster configure los recursos e inicie la aplicación.**

```
# /opt/NETapp/util/startapp -h <nombre_sistema_lógico> -p <lista_puertos_y_protocolos>
```

Los argumentos de la secuencia de inicio variarán según el tipo de recurso: a prueba de fallos o escalable. Compruebe la página de comando man personalizada o ejecute la secuencia de inicio sin argumentos para obtener una instrucción de sintaxis.

```
# /opt/NETapp/util/startapp
 Se deben especificar los nombres de LogicalHostname o SharedAddress.
 Para servicios a prueba de fallos:
 Sintaxis: startapp -h <nombre_sistema_lógico>
           -p <lista_puertos_y_protocolos>
           [-n <grupo_IPMP/lista_adaptadores>]
 Para servicios escalables:
 Sintaxis: startapp
           -h <nombre_dirección_compartida>
           -p <lista_puertos_y_protocolos>
           [-l <norma_de_equilibrio_de_carga>]
           [-n <grupo_IPMP/lista_adaptadores>]
           [-w <pesos_de_equilibrio_de_cargas>]
```

Salida de SunPlex Agent Builder

SunPlex Agent Builder genera tres secuencias de control y un archivo de configuración de acuerdo con los datos que se han introducido durante la creación de paquetes. El archivo de configuración especifica los nombres del grupo de recursos y el tipo de recurso.

Las secuencias de control son las siguientes:

- Secuencia de inicio: se usa para configurar los recursos e iniciar la aplicación bajo control de RGM.
- Secuencia de parada: se usa para detener la aplicación y poner fuera de línea los recursos y grupos de recursos.
- Secuencia de eliminación: se utiliza para eliminar los recursos y grupos de recursos creados por la secuencia de inicio.

Estas secuencias de control tienen la misma interfaz y el mismo comportamiento que las secuencias de utilidad que genera SunPlex Agent Builder para agentes no basados en GDS. Las secuencias se empaquetan en un paquete instalable con Solaris, que se puede reutilizar en varios clústers.

El archivo de configuración se puede personalizar para proporcionar los nombres a los grupos de recursos u otros parámetros que suelen darse como entradas del comando `scrgadm`. Si no se personalizan las secuencias, SunPlex Agent Builder proporciona unos valores predeterminados razonables para los parámetros de `scrgadm`.

Utilización de los comandos administrativos estándar de Sun Cluster para crear un servicio con GDS

En este apartado se describe cómo se pueden introducir estos parámetros en GDS. GDS se utiliza y administra con los comandos administrativos de Sun Cluster, como `scrgadm` y `scswitch`.

Así, no será necesario introducir los comandos administrativos de nivel inferior que aparecen en esta sección, siempre que las secuencias de control proporcionen unas funciones adecuadas. Sin embargo, es posible hacerlo si se necesita un mayor control del recurso basado en GDS. Éstos son los comandos que ejecutan las secuencias de control.

▼ Cómo utilizar los comandos administrativos de Sun Cluster para crear un servicio de alta disponibilidad con GDS

1. **Registre el tipo de recurso `SUNW.gds`**

   ```
   # scrgadm -a -t SUNW.gds
   ```

2. **Cree el grupo de recursos que contenga el recurso LogicalHostname y el servicio a prueba de fallos.**

   ```
   # scrgadm -a -g haapp_rg
   ```

3. **Cree el recurso para LogicalHostname.**

   ```
   # scrgadm -a -L -g haapp_rs -l hhead
   ```

4. **Cree el recurso para el servicio a prueba de fallos.**

   ```
   # scrgadm -a -j haapp_rs -g haapp_rg -t SUNW.gds \
           -y Scalable=false -y Start_timeout=120 \
           -y Stop_timeout=120 -x Probe_timeout=120 \
           -y Port_list="2222/tcp" \
   ```

```
-x Start_command="/export/ha/appctl/start" \
-x Stop_command="/export/ha/appctl/stop" \
-x Probe_command="/export/app/bin/probe" \
-x Child_mon_level=0 -y Network_resources_used=hhead \
-x Failover_enabled=true -x Stop_signal=9
```

5. **Ponga el grupo de recursos `haapp_rg` en línea.**

```
# scswitch -Z -g haapp_rg
```

▼ Comandos administrativos estándar de Sun Cluster para crear un servicio escalable con GDS

1. **Registre el tipo de recurso SUNW.gds.**

```
# scrgadm -a -t SUNW.gds
```

2. **Cree el grupo de recursos para SharedAddress.**

```
# scrgadm -a -g sa_rg
```

3. **Cree el recurso SharedAddress en `sa_rg`.**

```
# scrgadm -a -S -g sa_rg -l hhead
```

4. **Cree el grupo de recursos para el servicio escalable.**

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

5. **Cree el grupo de recursos para el servicio escalable.**

```
# scrgadm -a -j app_rs -g app_rg -t SUNW.gds \
       -y Scalable=true -y Start_timeout=120 \
       -y Stop_timeout=120 -x Probe_timeout=120 \
       -y Port_list="2222/tcp" \
       -x Start_command="/export/app/bin/start" \
       -x Stop_command="/export/app/bin/stop" \
       -x Probe_command="/export/app/bin/probe" \
       -x Child_mon_level=0 -y Network_resource_used=hhead \
       -x Failover_enabled=true -x Stop_signal=9
```

6. **Ponga en línea el grupo de recursos que contiene los recursos de red.**

```
 # scswitch -Z -g sa_rg
```

7. **Ponga en línea el grupo de recursos `app_rg`.**

```
# scswitch -Z -g app_rg
```

Interfaz de línea de comandos de SunPlex Agent Builder

SunPlex Agent Builder tiene una interfaz de línea de comandos con una función equivalente a la interfaz GUI. Esta interfaz consta de los comandos `scdscreate`(1HA) y `scdsconfig`(1HA). En la sección siguiente «Creación de un servicio que utiliza GDS con la versión de línea de comandos de Agent Builder» en la página 201 se realiza la misma función que en el procedimiento basado en GUI, pero utiliza una interfaz no gráfica.

▼ Creación de un servicio que utiliza GDS con la versión de línea de comandos de Agent Builder

1. **Cree el servicio.**

   Para un servicio a prueba de fallos, utilice:

   ```
   # scdscreate -g -V NET -T app -d /export/wdir
   ```

   Para un servicio escalable, utilice:

   ```
   # scdscreate -g -s -V NET -T app -d /export/wdir
   ```

   Nota – Los parámetros *–d* son opcionales. Si no se especifica otro, el directorio de trabajo predeterminado es el actual.

   ---

2. **Configure el servicio.**

   ```
   # scdsconfig -s "/export/app/bin/start' -t "/export/app/bin/stop" \
   -m "/export/app/bin/probe" -d /export/wdir
   ```

   Nota – Sólo se requiere el comando de inicio. Todos los demás parámetros son opcionales.

   ---

3. **Instale el paquete terminado en todos los nodos del clúster.**

   ```
   # cd /export/wdir/NETapp/pkg
   # pkgadd -d . NETapp
   ```

4. **Los archivos siguientes se instalarán durante pkgadd:**

   ```
   /opt/NETapp
   /opt/NETapp/README.app
   ```

```
/opt/NETapp/man
/opt/NETapp/man/man1m
/opt/NETapp/man/man1m/removeapp.1m
/opt/NETapp/man/man1m/startapp.1m
/opt/NETapp/man/man1m/stopapp.1m
/opt/NETapp/man/man1m/app_config.1m
/opt/NETapp/util
/opt/NETapp/util/removeapp
/opt/NETapp/util/startapp
/opt/NETapp/util/stopapp
/opt/NETapp/util/app_config
```

Nota – Las páginas de comando man y los nombres de secuencias se corresponderán con el nombre de aplicación que se introdujo antes, precedido del nombre de la secuencia; por ejemplo, `startapp`.

Para ver las páginas de comando man hay que especificar la ruta a la página. Por ejemplo, para ver las páginas de comando man de `startapp` utilice:

```
# man -M /opt/NETapp/man startapp
```

5. **En un nodo del clúster configure los recursos e inicie la aplicación.**

   # **/opt/NETapp/util/startapp -h** *<nombre_sistema_lógico>* **-p** *<lista_puertos_y_protocolos>*

   Los argumentos de la secuencia de inicio variarán según el tipo de recurso: a prueba de fallos o escalable. Compruebe la página de comando man personalizada o ejecute la secuencia de inicio sin argumentos para obtener una instrucción de sintaxis.

```
# /opt/NETapp/util/startapp
 Se deben especificar los nombres de LogicalHostname o SharedAddress.
 Para servicios a prueba de fallos:
 Sintaxis: startapp -h <nombre_sistema_lógico>
           -p <lista_puertos_y_protocolos>
           [-n <grupo_IPMP/lista_adaptadores>]
 Para servicios escalables:
 Sintaxis: startapp
           -h <nombre_dirección_compartida>
           -p <lista_puertos_y_protocolos>
           [-l <norma_de_equilibrio_de_carga>]
           [-n <grupo_IPMP/lista_adaptadores>]
           [-w <pesos_de_equilibrio_de_cargas>]
```

CAPÍTULO 11

Referencia de la Biblioteca de desarrollo del servicio de datos (DSDL)

En este capítulo se enumeran y describen brevemente las funciones de la API de la Biblioteca de desarrollo del servicio de datos (DSDL). Consulte las páginas de comando man individuales de 3HA para ver una descripción completa de cada una de las funciones de DSDL. DSDL define únicamente una interfaz de C. No hay una interfaz de DSDL con secuencias.

DSDL proporciona funciones de las siguientes categorías.

Funciones de DSDL

Las siguientes subsecciones proporcionan una breve información general sobre cada una de las categorías de las funciones de DSDL. Sin embargo, las páginas de comando man de 3HA son la referencia definitiva para las funciones de DSDL.

Funciones de uso general

Las funciones de esta sección proporcionan una amplia gama de posibilidades; permiten:

- Inicializar el entorno de DSDL
- Recuperar recursos, tipos y nombres de grupos de recursos y valores de propiedades de extensión

- Recuperaciones de fallos y reiniciar un grupo de recursos y un recurso
- Convertir secuencias de errores en mensajes de error
- Ejecutar un comando tras un tiempo de espera

Las funciones siguientes inicializan el método de llamada.

- `scds_initialize`: asigna recursos e inicializa el entorno de DSDL.
- `scds_close`: libera recursos asignados por `scds_initialize`.

Las funciones siguientes recuperan información sobre recursos, tipos y grupos de recursos y propiedades de extensión.

- `scds_get_resource_name`: recupera el nombre del recurso para el programa de llamada.
- `scds_get_resource_type_name`: recupera el nombre del tipo de recurso para el programa de llamada.
- `scds_get_resource_group_name`: recupera el nombre del grupo de recursos para el programa de llamada.
- `scds_get_ext_property`: recupera el valor de la propiedad de extensión especificada.
- `scds_free_ext_property`: libera la memoria asignada por `scds_get_ext_property`.

La función siguiente recupera información del estado de los recursos de SUNW.HAStoragePlus que utiliza un recurso.

- `scds_hasp_check`: recupera información del estado de los recursos de `SUNW.HAStoragePlus` utilizados por un recurso. Esta información se obtiene del estado (en línea o no) de todos los recursos de `SUNW.HAStoragePlus` del que depende el recurso, con las propiedades de sistema `Resource_dependencies` o `Resource_dependencies_weak` definidas para el recurso.

  Consulte `SUNW.HAStoragePlus`(5) para obtener más información sobre `SUNW.HAStoragePlus`.

Las funciones siguientes realizan operaciones de recuperación de fallos o reinician un recurso o grupo de recursos.

- `scds_failover_rg`: realiza una operación de recuperación de fallos de un grupo de recursos.
- `scds_restart_rg`: reinicia un grupo de recursos.
- `scds_restart_resource`: reinicia un recurso.

Las dos siguientes funciones ejecutan un comando tras un tiempo de espera y convierten un código de error en un mensaje de error.

- `scds_timerun`: ejecuta un comando con un valor de tiempo de espera.
- `scds_error_string`: traduce un código de error en una secuencia de error.

Funciones de la propiedad

Estas funciones proporcionan API convenientes para acceder a las propiedades específicas del recurso, grupos y tipos de recursos, incluidas algunas propiedades de extensión utilizadas frecuentemente. DSDL proporciona la función `scds_initialize` para analizar los argumentos de la línea de comandos. La biblioteca *intercepta* entonces las diversas propiedades del recurso, grupo de recursos y tipo de recursos.

Una sola página de comando man, `scds_property_functions`(3HA), describe todas estas funciones. Esta sección contiene las funciones siguientes

- `scds_get_rs_`*nombre_propiedad*
- `scds_get_rg_`*nombre_propiedad*
- `scds_get_rt_`*nombre_propiedad*
- `scds_get_ext_`*nombre_propiedad*

Funciones de acceso a los recursos de la red

Las funciones enumeradas en esta sección recuperan, imprimen y liberan recursos de red empleados por recursos y grupos de recursos. Las funciones `scds_get_*` de esta sección proporcionan una forma conveniente de recuperar recursos de red sin necesidad de consultar propiedades específicas, como `Network_resources_used` y `Port_list` con las funciones de RMAPI. Las funciones `scds_print_`*nombre*`()` imprimen valores de las estructuras de datos devueltas por las funciones `scds_get_`*nombre*`()`. Las funciones `scds_free_`*nombre*`()` liberan la memoria asignada por las funciones `scds_get_`*nombre*`()`.

Las siguientes funciones están relacionadas con los nombres de sistema.

- `scds_get_rg_hostnames`: recupera una lista de nombres de sistema utilizados por los recursos de la red en un grupo de recursos.
- `scds_get_rs_hostnames`: recupera una lista de nombres de sistema empleados por el recurso.
- `scds_print_net_list`: imprime los contenidos de la lista de nombres de sistema que devuelven `scds_get_rg_hostnames` o `scds_get_rs_hostnames`.
- `scds_free_net_list`: libera la memoria asignada por `scds_get_rg_hostnames` o `scds_get_rs_hostnames`.

Las funciones siguientes están relacionadas con listas de puertos.

- `scds_get_port_list`: recupera una lista de pares puerto-protocolo empleados por un recurso.
- `scds_print_port_list`: imprime el contenido de la lista de pares puerto-protocolo que devuelve `scds_get_port_list`.
- `scds_free_port_list`: libera la memoria asignada por `scds_get_port_list`.

Las funciones siguientes están relacionadas con las direcciones de red.

- `scds_get_netaddr_list`: recupera una lista de las direcciones de red empleadas por un recurso.
- `scds_print_netaddr_list`: imprime el contenido de la lista de direcciones de red devuelta por `scds_get_netaddr_list`.
- `scds_free_netaddr_list`: libera la memoria asignada por `scds_get_netaddr_list`.

Supervisión de los fallos con las conexiones de TCP

Las funciones de esta sección habilitan la supervisión basada en TCP. Generalmente, un supervisor de fallos emplea estas funciones para establecer una conexión de zócalo sencilla con un servicio, leer y escribir datos en éste para asegurarse de su estado y desconectarse después de él.

Esta sección contiene las funciones siguientes.

- `scds_tcp_connect`: establece una conexión TCP con un proceso.
- `scds_tcp_read`: utiliza una conexión TCP para leer los datos del proceso que se está supervisando.
- `scds_tcp_write`: utiliza una conexión TCP para escribir datos en un proceso que se está supervisando.
- `scds_simple_probe`: analiza un proceso, mediante el establecimiento y la finalización de una conexión TCP con el proceso.
- `scds_tcp_disconnect`: finaliza la conexión con un proceso que se está supervisando.

Funciones de PMF

Estas funciones encapsulan la función de PMF. El modelo de DSDL de supervisión a través de PMF crea y emplea valores implícitos de *etiqueta* para `pmfadm`(1M). El recurso PMF también emplea valores implícitos para `Restart_interval`, `Retry_count` y `action_script` (las opciones `-t`, `-n` y `-a` en `pmfadm`). Es más, DSDL vincula el historial de terminación del proceso, como lo encuentra PMF, con el historial de fallos de la aplicación, como lo ha detectado el supervisor de fallos, para calcular la decisión de reinicio o recuperación de fallos.

Esta sección contiene las funciones siguientes.

- `scds_pmf_get_status`: determina si la instancia especificada está siendo supervisada bajo el control de PMF.
- `scds_pmf_restart_fm`: reinicia el supervisor de fallos con PMF.
- `scds_pmf_signal`: envía la señal especificada a un árbol de procesos que se ejecuta bajo el control de PMF.

- `scds_pmf_start`: ejecuta un programa especificado (incluido un supervisor de fallos) bajo el control de PMF.
- `scds_pmf_stop`: termina un proceso que se está ejecutando bajo el control de PMF.
- `scds_stop_monitoring`: detiene la supervisión de un proceso que se está ejecutando bajo el control de PMF.

Funciones del supervisor de fallos

Las funciones de esta sección proporcionan un modelo predeterminado de supervisión de fallos, mediante el mantenimiento del historial de fallos y su evaluación en combinación con las propiedades `Retry_count` y `Retry_interval`.

Esta sección contiene las funciones siguientes.

- `scds_fm_sleep`: espera un mensaje en un zócalo de control del supervisor de fallos.
- `scds_fm_action`: actúa tras finalizar un análisis.
- `scds_fm_print_probes`: escribe información del estado del análisis en el registro del sistema.

Funciones útiles

Las funciones de esta sección permiten escribir mensajes y depurar mensajes del registro del sistema. Esta sección incluye las dos funciones siguientes.

- `scds_syslog`: escribe mensajes en el registro del sistema.
- `scds_syslog_debug`: escribe un mensaje de depuración en el registro del sistema.

CAPÍTULO 12

CRNP

Este capítulo proporciona información sobre el Protocolo de notificación de reconfiguración del clúster (CRNP, Cluster Reconfiguration Notification Protocol) que permite que las aplicaciones a prueba de fallos y escalables estén "habilitadas para clúster." Es decir, proporciona un mecanismo que permite que las aplicaciones se registren para recibir, y reciban después, notificación asíncrona de los eventos de reconfiguración de Sun Cluster. Los servicios de datos que se ejecutan en el clúster y las aplicaciones que se ejecutan fuera de él se pueden registrar para recibir notificación de los eventos. Éstos se generan cuando cambian los miembros de un clúster y cuando cambia el estado de un grupo de recursos o de un recurso.

Información general sobre CRNP

CRNP proporciona mecanismos y daemons que generan eventos de reconfiguración del clúster, los encamina a través del clúster y los envía a los clientes interesados.

El daemon `cl_apid` interactúa con los clientes. El Gestor de grupos de recursos (RGM) de Sun Cluster genera eventos de reconfiguración del clúster. Estos daemons utilizan `syseventd`(1M) para transmitir eventos en todos los nodos locales. El daemon `cl_apid` utiliza Lenguaje de marcas extensible (XML) en TCP/IP para comunicarse con los clientes interesados.

El diagrama siguiente presenta información general sobre el flujo de eventos entre los componentes de CRNP. En este diagrama, un cliente se está ejecutando en el nodo del clúster 2 y el otro en un ordenador que no forma parte del clúster.

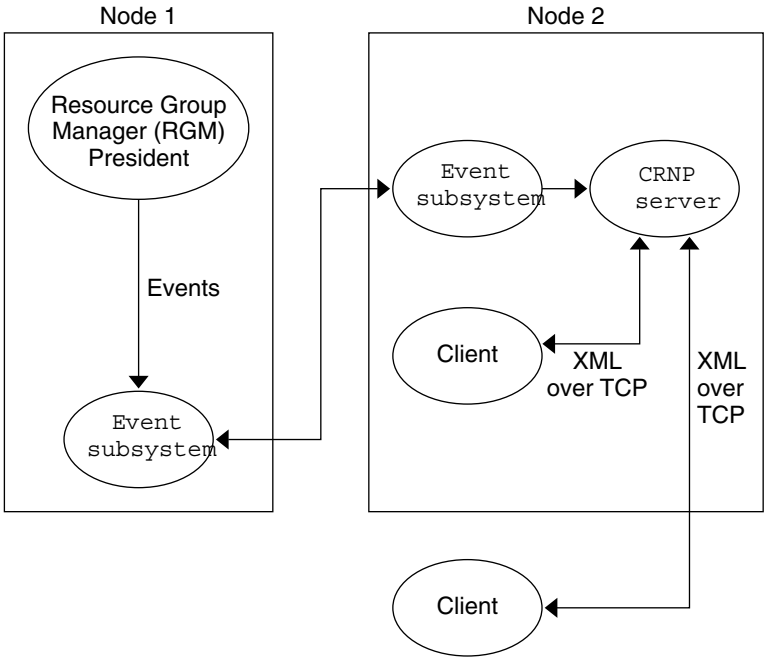


FIGURA 12–1 Cómo funciona CRNP

Información general sobre el protocolo de CRNP

CRNP define las capas de aplicación, presentación y sesión de la pila estándar del protocolo de interconexión de sistema abierto (OSI) de siete capas. La capa de transporte debe ser TCP y la de red, IP. CRNP es independiente de las capas física y de enlace de datos. Todos los mensajes de capa de aplicación intercambiados en CRNP se basan en XML 1.0.

Semántica del protocolo de CRNP

Los clientes inician una comunicación mediante el envío de un mensaje de registro (`SC_CALLBACK_RG`) que especifica los tipos de eventos sobre los cuales los clientes desean recibir notificaciones y el puerto al que se pueden enviar los eventos. El IP de origen de la conexión de registro y el puerto especificado, juntos, forman la dirección de rellamada.

Siempre que se genera un evento que pueda interesar a un cliente en el clúster, el servidor se pone en contacto con el cliente en su dirección de rellamada (IP y puerto) y le envía el evento (`SC_EVENT`). El servidor tiene una alta disponibilidad y se ejecuta en el propio clúster. El servidor almacena los registros del cliente en un almacenamiento que se conserva incluso después de rearrancar el clúster.

Los clientes anulan sus registros mediante el envío de un mensaje de registro (`SC_CALLBACK_RG` que contiene un mensaje `REMOVE_CLIENT`) al servidor. Cuando el cliente recibe un mensaje `SC_REPLY` del servidor, cierra la conexión.

El diagrama siguiente muestra el flujo de comunicación entre un cliente y un servidor.

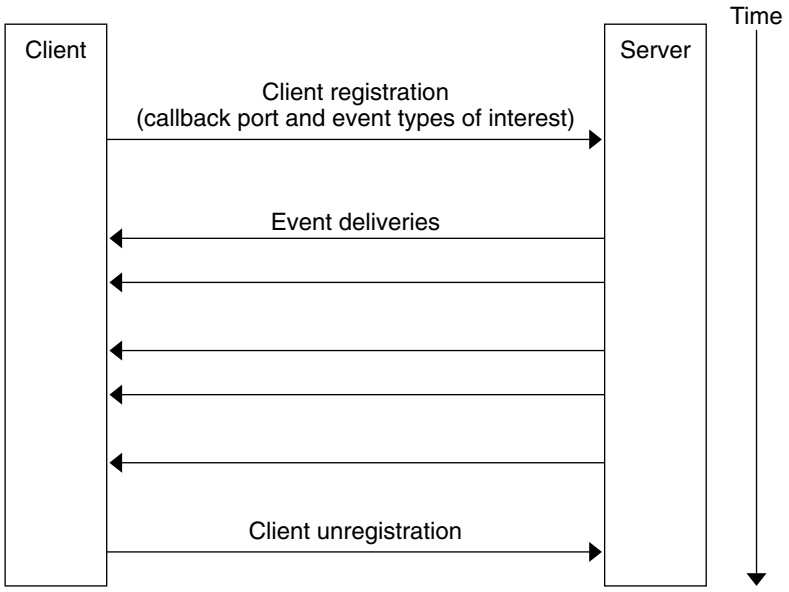


FIGURA 12–2 Flujo de comunicación entre un cliente y un servidor

Tipos de mensaje que emplea CRNP

CRNP emplea tres tipos de mensajes, todos basados en XML, como se detalla en la tabla siguiente. Estos tipos de mensaje y su sintaxis se explican con más detalle en este mismo capítulo.

Tipo de mensaje	Descripción
SC_CALLBACK_REG	Este mensaje adopta cuatro formas: ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS y REMOVE_EVENTS. Cada una de ellas contiene la información siguiente: ■ Versión del protocolo ■ Puerto de rellamada en formato ASCII (no formato binario) Las formas ADD_CLIENT, ADD_EVENTS y REMOVE_EVENTS contienen también una lista ilimitada de tipos de eventos, cada uno de los cuales incluye la información siguiente: ■ Clase de evento ■ Subclase de evento (opcional) ■ Lista de los pares de nombre y valor (opcional) Juntas, la clase y la subclase de eventos definen un único "tipo de evento". La definición de tipo de documento (DTD) desde la cual se generan las clases de SC_CALLBACK_REG es SC_CALLBACK_REG. Esta DTD se describe con más detalle en el Apéndice F.

SC_EVENT	Este mensaje contiene la información siguiente:
	■ Versión del protocolo
	■ Clase de evento
	■ Subclase de evento
	■ Fabricante
	■ Editor
	■ Lista de pares de nombres y valores (estructuras de datos de pares de 0 o más nombres y valores)
	  ■ Nombre (cadena)
	  ■ Valor (cadena o matriz de cadenas)
	
	Los valores en SC_EVENT no tienen un tipo. La definición de tipo de documento desde la cual se generan las clases de SC_EVENT es SC_EVENT. Esta DTD se describe con más detalle en el Apéndice F.

SC_REPLY	Este mensaje contiene la información siguiente: ■ Versión del protocolo ■ Código de error ■ Mensaje de error La definición de tipo de documento desde la cual se generan las clases de SC_REPLY es SC_REPLY. Esta DTD se describe con más detalle en el Apéndice F.

Cómo se registra un cliente en un servidor

Esta sección describe cómo un administrador debe configurar el servidor, cómo se identifican los clientes, cómo se envía información por las capas de aplicación y sesión y las condiciones de error.

Supuestos sobre cómo los administradores configuran el servidor

El administrador del sistema debe configurar el servidor con una dirección IP de alta disponibilidad (es decir, una dirección IP que no esté vinculada a una máquina concreta del clúster) y un número de puerto. El administrador debe publicar esta dirección de red para los clientes potenciales. CRNP no define cómo se pone este nombre de servidor a disposición de los clientes. Los administradores deberán utilizar un servicio de nombres, que permitirá a los clientes buscar la dirección de red del servidor de forma dinámica, o agregarán el nombre de red a un archivo de configuración que podrá leer el cliente. El servidor se ejecutará dentro del clúster como un tipo de recurso a prueba de fallos.

Cómo identifica el servidor a un cliente

Cada cliente está identificado de forma exclusiva por una dirección de rellamada, es decir, su dirección IP y número de puerto. El puerto se especifica en los mensajes `SC_CALLBACK_REG` y la dirección IP se obtiene de la conexión de registro de TCP. CRNP asume que los siguientes mensajes `SC_CALLBACK_REG` con la misma dirección de rellamada provienen del mismo cliente, aunque el puerto de origen desde el que se envíen los mensajes sea diferente.

Cómo se pasan los mensajes `SC_CALLBACK_REG` entre el cliente y el servidor

Un cliente inicia un registro, abriendo una conexión TCP con la dirección IP y el número de puerto del servidor. Una vez establecida la conexión TCP y lista para escribir, el cliente debe enviar su mensaje de registro. Éste debe ser un mensaje `SC_CALLBACK_REG` de formato correcto que no contenga bytes adicionales antes ni después del mensaje.

Una vez escritos todos los bytes en el canal, el cliente debe mantener la conexión abierta para recibir la respuesta del servidor. Si el cliente no le da el formato correcto al mensaje, el servidor no registrará el cliente y le enviará una respuesta de error. Si el cliente cierra la conexión de zócalo antes de que el servidor envíe una respuesta, éste registra el cliente como normal.

Un cliente puede ponerse en contacto con el servidor en cualquier momento. Cada vez que un cliente se pone en contacto con el servidor, el cliente debe enviar un mensaje `SC_CALLBACK_REG`. Si el servidor recibe un mensaje con formato erróneo, fuera de servicio o no válido, envía una respuesta de error al cliente.

Un cliente no puede enviar un mensaje `ADD_EVENTS`, `REMOVE_EVENTS` o `REMOVE_CLIENT` antes de enviar el mensaje `ADD_CLIENT`. Un cliente no puede enviar un mensaje `REMOVE_CLIENT` antes de haber enviado un mensaje `ADD_CLIENT`.

Si un cliente envía un mensaje `ADD_CLIENT` y el cliente ya está registrado, es posible que el servidor tolere el mensaje. En este caso, el servidor sustituye silenciosamente el registro anterior del cliente por el nuevo, especificado en el segundo mensaje `ADD_CLIENT`.

En la mayoría de los casos, un cliente se registra sólo una vez en el servidor, al comenzar, con un mensaje `ADD_CLIENT`. El cliente sólo cancela su registro una vez, enviando un mensaje `REMOVE_CLIENT` al servidor. Sin embargo, CRNP proporciona una mayor flexibilidad a aquellos clientes que tengan que modificar dinámicamente la lista de tipos de eventos.

Contenido de un mensaje `SC_CALLBACK_REG`

Cada mensaje `ADD_CLIENT`, `ADD_EVENTS` y `REMOVE_EVENTS` contiene una lista de eventos. La tabla siguiente describe los tipos de eventos que acepta CRNP, incluidos los valores requeridos de nombre y valor.

Si un cliente:

- Envía un mensaje `REMOVE_EVENTS` que especifique uno o varios tipos de eventos para los cuales el cliente no se ha registrado previamente
- Registra el mismo tipo de evento dos veces

En ambos casos el servidor ignora discretamente estos mensajes.

Clase y subclase	Pares nombre-valor	Descripción
`EC_Cluster`	Necesario: ninguno	Registra todos los eventos de cambio en los miembros del clúster (terminación de nodo o unión)
`ESC_cluster_membership`	Opcional: ninguno	

Clase y subclase	Pares nombre-valor	Descripción
`EC_Cluster` `ESC_cluster_rg_state`	Uno necesario, como sigue: `rg_name` Tipo de valor: cadena Opcional: ninguno	Registra todos los eventos de cambio de estado del grupo de recursos *nombre*
`EC_Cluster` `ESC_cluster_r_state`	Uno necesario, como sigue: `r_name` Tipo de valor: cadena Opcional: ninguno	Registra todos los eventos de cambio de estado del recurso *nombre*
`EC_Cluster` Ninguno	Necesario: ninguno Opcional: ninguno	Registra todos los eventos de Sun Cluster

Cómo responde el servidor a un cliente

Después de procesar el registro, el servidor envía el mensaje `SC_REPLY` sólo en la conexión TCP abierta desde el cliente en el que el servidor ha recibido la solicitud de registro; a continuación cierra la conexión. El cliente debe mantener abierta la conexión TCP hasta que reciba el mensaje `SC_REPLY` del servidor.

Por ejemplo, el cliente realiza las siguientes acciones:

1. Abre una conexión TCP al servidor
2. Espera que la conexión se pueda "escribir"
3. Envía un mensaje `SC_CALLBACK_REG` (que contiene un mensaje `ADD_CLIENT`)
4. Espera un mensaje `SC_REPLY`
5. Recibe un mensaje `SC_REPLY`
6. Recibe un indicador de que el servidor ha cerrado la conexión (lee 0 bytes del zócalo)
7. Cierra la conexión

Más tarde, el cliente realiza las acciones siguientes:

1. Abre una conexión TCP al servidor
2. Espera que la conexión se pueda "escribir"
3. Envía un mensaje `SC_CALLBACK_REG` (que contiene un mensaje `REMOVE_CLIENT`)

4. Espera un mensaje SC_REPLY
5. Recibe un mensaje SC_REPLY
6. Recibe un indicador de que el servidor ha cerrado la conexión (lee 0 bytes del zócalo)
7. Cierra la conexión

Cada vez que el servidor recibe un mensaje SC_CALLBACK_REG de un cliente, le envía un mensaje SC_REPLY en la misma conexión abierta. Este mensaje especifica si la operación se ha realizado de modo satisfactorio o no. En «DTD de XML de SC_REPLY» en la página 328 se explica la definición de tipo de documento XML de un mensaje SC_REPLY y los posibles mensajes de error que pueda contener.

Contenido de un mensaje SC_REPLY

Un mensaje SC_REPLY especifica si la operación se ha realizado de modo satisfactorio o no. Este mensaje contiene la versión del mensaje del protocolo CRNP, un código y un mensaje de estado que describe con más detalle el código de estado. La tabla siguiente describe los valores posibles del código de estado.

Código de estado	Descripción
OK	El mensaje se ha procesado satisfactoriamente.
RETRY	El servidor ha rechazado el registro del cliente debido a un error temporal (el cliente debería volver a intentar el registro con parámetros diferentes).
LOW_RESOURCE	Los recursos del clúster están bajos y el cliente sólo puede volver a intentarlo más tarde (el administrador del sistema del clúster también podría ampliar los recursos del clúster).
SYSTEM_ERROR	Se ha producido un error grave. Póngase en contacto con el administrador del sistema del clúster.
FAIL	La autorización ha fallado o se ha producido otro problema que ha provocado el fallo de registro.
MALFORMED	La solicitud XML tenía un formato erróneo y no se ha podido analizar.
INVALID	La solicitud XML no era válida (no cumple la especificación XML).
VERSION_TOO_HIGH	La versión del mensaje era demasiado elevada para procesar satisfactoriamente el mensaje.
VERSION_TOO_LOW	La versión del mensaje era demasiado baja para procesar satisfactoriamente el mensaje.

Cómo debe resolver el cliente las condiciones de error

En condiciones normales, un cliente que envía un mensaje `SC_CALLBACK_REG` recibe una respuesta que indica que el registro ha sido satisfactorio o no.

Sin embargo, el servidor puede experimentar una condición de error, durante el registro de un cliente, que le impida enviarle un mensaje `SC_REPLY` de vuelta. En este caso, el registro se podría haber realizado satisfactoriamente antes de que se produjera la condición de error, podría haber fallado o podría no haber sido procesado.

Dado que el servidor debe funcionar como un servidor de tipo a prueba de fallos o de alta disponibilidad en el clúster, esta condición de error no supondrá el fin del servicio. De hecho, es posible que el servidor pueda empezar muy pronto a enviar eventos al cliente recién registrado.

Para remediar estas condiciones, el cliente debería:

- Imponer un tiempo de espera de nivel de aplicación en una conexión de registro que esté esperando un mensaje `SC_REPLY`, pasado el cual el cliente deberá volver a intentar el registro.
- Empezar a recibir en su número de puerto y dirección IP de rellamada por si hubiera recepción de eventos antes de registrar las rellamadas de eventos. El cliente debería esperar a recibir un mensaje de confirmación de registro y los envíos de eventos paralelamente. Si el cliente empieza a recibir eventos antes de recibir un mensaje de confirmación, debería cerrar discretamente la conexión de registro.

Cómo envía eventos el servidor al cliente

Dado que los eventos se generan dentro del clúster, el servidor CRNP los envía a todos los clientes que solicitaron eventos de este tipo. El envío consiste en dirigir un mensaje `SC_EVENT` a la dirección de rellamada del cliente. El envío de cada evento se produce en una conexión TCP nueva.

Inmediatamente después de que un cliente se registre para un tipo de evento, a través de un mensaje `SC_CALLBACK_REG` que contenga un mensaje `ADD_CLIENT` o `ADD_EVENT`, el servidor enviará el evento más reciente de ese tipo al cliente. Éste podrá así descubrir el estado actual del sistema desde el que se le enviarán los siguientes eventos.

Cuando el servidor inicie una conexión TCP con el cliente, enviará exactamente un mensaje `SC_EVENT` por la conexión. El servidor después enviará un cierre full-duplex.

Por ejemplo, el cliente realiza las siguientes acciones:

1. Espera que el servidor inicie una conexión TCP
2. Acepta la conexión entrante del servidor
3. Espera un mensaje `SC_EVENT`
4. Lee un mensaje `SC_EVENT`
5. Recibe un indicador de que el servidor ha cerrado la conexión (lee 0 bytes del zócalo)
6. Cierra la conexión

Cuando todos los clientes estén registrados, deberán recibir en todas sus direcciones de rellamada (dirección IP y número de puerto) en todo momento, por si hubiera una conexión entrante para el envío de un evento.

Si el servidor no logra contactar con el cliente para enviarle un evento, vuelve a intentar enviarlo un cierto número de veces, en el intervalo que se defina. Si los intentos fallan, el cliente es eliminado de la lista de clientes del servidor. El cliente también debe volver a registrarse mediante el envío de otro mensaje `SC_CALLBACK_REG` que contenga un mensaje `ADD_CLIENT` antes de que el cliente pueda recibir más eventos.

Cómo se garantiza el envío de eventos

Hay un orden total de generación de eventos dentro del clúster, que se mantiene en el orden de envío a cada cliente. En otras palabras, si el evento A se genera en el clúster antes que el evento B, el cliente X recibe el evento A antes que el B. Sin embargo, el orden total de envío de eventos a *todos* los clientes *no* se mantiene. Es decir, el cliente Y podría recibir los eventos A y B antes de que el cliente X recibiera el evento A. Así, los clientes más lentos no retrasan el envío a los demás clientes.

Todos los eventos que envía el servidor (salvo el primero de una subclase y los eventos que siguen a los errores de servidor) se producen como respuesta a eventos reales generados por el clúster, salvo que el servidor sufra un error que haga que pierda eventos generados por el clúster. En ese caso, el servidor genera un evento para cada tipo de evento que representa el estado actual del sistema para ese tipo. Cada evento se envía a los clientes que registraron su interés en ese tipo de evento.

El envío de eventos sigue la semántica de “una vez como mínimo”. Es decir, el servidor puede enviar el mismo evento a un cliente más de una vez. Este margen es necesario en los casos en los que el servidor se desconecta temporalmente y, cuando se vuelve a conectar, no puede determinar si el cliente ha recibido la última información.

Contenido de un mensaje `SC_EVENT`

El mensaje `SC_EVENT` contiene el mensaje real generado en el clúster, traducido para encajar en el formato de mensaje XML `SC_EVENT`. La tabla siguiente describe los tipos de evento que envía CRNP, incluidos los pares nombre-valor, editor y fabricante.

Clase y subclase	Editor y fabricante	Pares nombre-valor	Notas
`EC_Cluster` `ESC_cluster_membership`	Editor: rgm Fabricante: SUNW	Nombre: `node_list` Tipo de valor: matriz de cadenas Nombre: `state_list` Tipo de valor: matriz de cadenas	Las posiciones de los elementos de la matriz de `state_list` están sincronizadas con las de `node_list`. Es decir, el estado del nodo que se indica en primer lugar en la matriz `node_list` es el primero de la matriz `state_list`. `state_list` contiene sólo números representados en ASCII, cada uno de los cuales representa el número que encarna actualmente ese nodo en el clúster. Si el número es el mismo que el recibido en un mensaje anterior, el nodo no ha modificado su relación con el clúster (se ha ido, se ha unido o se ha vuelto a unir). Si el número de representación es –1, el nodo no es miembro del clúster. Si el número de representación es un número positivo, el nodo es miembro del clúster. Otros nombres que empiezan por `ev_` y sus valores asociados pueden estar también presentes, pero no son para el uso del cliente.

Clase y subclase	Editor y fabricante	Pares nombre-valor	Notas
EC_Cluster ESC_cluster_rg_state	Editor: rgm Fabricante: SUNW	Nombre: rg_name Tipo de valor: cadena Nombre: node_list Tipo de valor: matriz de cadenas Nombre: state_list Tipo de valor: matriz de cadenas	Las posiciones de los elementos de la matriz de state_list están sincronizadas con las de node_list. Es decir, el estado del nodo que se indica en primer lugar en la matriz node_list es el primero de la matriz state_list. state_list contiene representaciones de cadenas del estado del grupo de recursos. Los valores válidos son los que se pueden recuperar con los comandos scha_cmds(1HA). Otros nombres que empiezan por ev_ y sus valores asociados pueden estar también presentes, pero no son para el uso del cliente.

Clase y subclase	Editor y fabricante	Pares nombre-valor	Notas
EC_Cluster ESC_cluster_r_state	Editor: rgm Fabricante: SUNW	Tres necesarios, como sigue: Nombre: r_name Tipo de valor: cadena Nombre: node_list Tipo de valor: matriz de cadenas Nombre: state_list Tipo de valor: matriz de cadenas	Las posiciones de los elementos de la matriz de state_list están sincronizadas con las de node_list. Es decir, el estado del nodo que se indica en primer lugar en la matriz node_list es el primero de la matriz state_list. state_list contiene representaciones de cadenas del estado del recurso. Los valores válidos son los que se pueden recuperar con los comandos scha_cmds(1HA). Otros nombres que empiezan por ev_ y sus valores asociados pueden estar también presentes, pero no son para el uso del cliente.

Cómo autentica CRNP los clientes y el servidor

El servidor autentica un cliente con una forma de envoltorio de TCP. La dirección IP de origen del mensaje de registro (que se usa también como dirección IP de rellamada a la que se deben enviar los eventos) debe estar en la lista de clientes admitidos en el servidor. La dirección IP de origen y el mensaje de registro no pueden estar en la lista de clientes rechazados. Si la dirección IP de origen y el registro no están en la lista, el servidor rechaza la solicitud y emite una respuesta de error para el cliente.

Cuando el servidor recibe un mensaje SC_CALLBACK_REG ADD_CLIENT, los mensajes SC_CALLBACK_REG posteriores para ese cliente deben contener una dirección IP de origen que es la misma dirección IP de origen del primer mensaje. Si el servidor CRNP recibe un mensaje SC_CALLBACK_REG que no cumple este requisito, el servidor:

- Ignora la solicitud y envía una respuesta de error al cliente o

- Asume que la solicitud proviene de un cliente nuevo (según el contenido del mensaje `SC_CALLBACK_REG`)

Este mecanismo de seguridad ayuda a evitar ataques de denegación de servicio, en los que alguien intenta anular el registro de un cliente legítimo.

Los clientes deberían autenticar del mismo modo al servidor. Los clientes sólo deben aceptar los envíos de un servidor cuya dirección IP de origen y número de puerto coincidan con la dirección IP y número de puerto de registro empleados por el cliente.

Dado que se espera que los clientes del servicio CRNP se encuentren dentro de un cortafuegos que proteja el clúster, CRNP no incluye otros mecanismos de seguridad adicional.

Creación de una aplicación Java que utilice CRNP

El ejemplo siguiente ilustra cómo se debe desarrollar una aplicación sencilla de Java, llamada `CrnpClient`, que utiliza CRNP. La aplicación se registra para rellamadas de eventos en el servidor CRNP del clúster, recibe las rellamadas de eventos y procesa éstos mediante la impresión del contenido. Antes de finalizar, la aplicación anula el registro de su solicitud de rellamadas de eventos.

Tenga los siguientes puntos presentes al revisar este ejemplo.

- La aplicación de ejemplo realiza una generación y análisis de XML con JAXP (API de Java para el procesamiento de XML). Este ejemplo no le enseña a utilizar JAXP. JAXP se describe con más detalle en `http://java.sun.com/xml/jaxp/index.html`.
- Este ejemplo presenta fragmentos de una aplicación completa, que se puede encontrar, íntegra, en el Apéndice G. Para ilustrar conceptos concretos de forma más eficaz, el ejemplo que se presenta en este capítulo es levemente diferente de la aplicación íntegra presentada en el Apéndice G.
- Para mayor brevedad, se excluyen los comentarios del código de ejemplo en este capítulo. La aplicación completa que se muestra en el Apéndice G incluye los comentarios.
- La aplicación que aparece en este ejemplo gestiona la mayoría de las condiciones de error simplemente cerrando la aplicación. La aplicación real debería tratar los errores con más firmeza.

▼ Configuración del entorno

En primer lugar hay que configurar el entorno.

1. **Descargue e instale JAXP y la versión correcta del compilador de Java y la máquina virtual Java.**

   En `http://java.sun.com/xml/jaxp/index.html` encontrará instrucciones.

Nota – Este ejemplo requiere la utilización de Java 1.3.1 o posterior.

---

2. **Asegúrese de que especifique una `classpath` en la línea de comandos de compilación para que el compilador pueda encontrar las clases de JAXP. Desde el directorio en el que se encuentra el archivo de origen, escriba:**

   ```
   % javac -classpath RAÍZ_JAXP/dom.jar:RAÍZ_JAXPjaxp-api. \
   jar:RAÍZ_JAXPsax.jar:RAÍZ_JAXPxalan.jar:RAÍZ_JAXP/xercesImpl \
   .jar:RAÍZ_JAXP/xsltc.jar -sourcepath . NOMBRE_ARCHIVO_ORIGEN.java
   ```

   donde *RAÍZ_JAXP* es la ruta absoluta o relativa al directorio en el que se encuentran los archivos jar de JAXP y *NOMBRE_ARCHIVO_ORIGEN* es el nombre de su archivo de origen Java.

3. **Al ejecutar la aplicación, se debe especificar la `classpath` para que la aplicación pueda cargar los archivos de clase de JAXP adecuados (tenga presente que la primera ruta de `classpath` es el directorio actual):**

   ```
   java -cp .:RAÍZ_JAXP/dom.jar:RAÍZ_JAXPjaxp-api. \
   jar:RAÍZ_JAXPsax.jar:RAÍZ_JAXPxalan.jar:RAÍZ_JAXP/xercesImpl \
   .jar:RAÍZ_JAXP/xsltc.jar NOMBRE_ARCHIVO_ORIGEN ARGUMENTOS
   ```

   Ahora que se ha configurado el entorno se puede desarrollar la aplicación.

▼ Procedimientos iniciales

En esta parte del ejemplo, se crea una clase básica denominada `CrnpClient`, con un método principal que analiza los argumentos de la línea de comandos y construye un objeto `CrnpClient`. Este objeto pasa los argumentos de línea de comandos a la clase, espera a que el usuario termine la aplicación, invoca `shutdown` en la clase `CrnpClient` y sale.

El constructor de la clase `CrnpClient` debe ejecutar las siguientes tareas:

- Configurar los objetos de proceso de XML.
- Crear un subproceso que reciba rellamadas de eventos.
- Contactar con el servidor CRNP y registrar rellamadas de eventos.

- **Crear el código Java que aplica la lógica anterior.**

El ejemplo siguiente muestra el código de la estructura básica de la clase `CrnpClient`. Las implementaciones de los cuatro métodos propios a los que se hace referencia en los métodos de cierre y el constructor aparecen más adelante. Observe que se muestra el código que importa todos los paquetes necesarios.

```
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("Pulsar Retorno para terminar la demostración...");
                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;
}
```

Las variables de los miembros se explican con más detalle más adelante.

▼ Análisis de los argumentos de la línea de comandos

- **Para ver cómo analizar los argumentos de línea de comandos, consulte el código del Apéndice G.**

▼ Definición del subproceso de recepción de eventos

En el código, hay que comprobar que la recepción de eventos se realice en un subproceso aparte de forma que la aplicación pueda seguir realizando el otro trabajo, mientras el subproceso de evento se bloquea y espera rellamadas de eventos.

Nota – La configuración del XML se detalla más adelante.

---

1. **En el código, defina una subclase de `Thread` llamada `EventReceptionThread` que cree `ServerSocket` y espere que lleguen eventos al zócalo.**

En esta parte del código de ejemplo, los eventos no se leen ni se procesan. La lectura y el procesamiento de eventos se explican más adelante. `EventReceptionThread` crea `ServerSocket` en una dirección comodín de protocolo de interred. `EventReceptionThread` mantiene también una referencia al objeto `CrnpClient` para que `EventReceptionThread` pueda enviar eventos al objeto `CrnpClient` para que los procese.

```
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();
                                // Construir un evento desde el flujo del zócalo y procesarlo
                                sock.close();
                        }
                        // INACCESIBLE

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* variables privadas de miembro*/
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. **Ahora que sabe cómo funciona la clase `EventReceptionThread`, cree un objeto `createEvtRecepThr`:**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
}
```

▼ Registro y anulación del registro de rellamadas

La tarea de registro consiste en:

- Abrir un zócalo de TCP básico para el puerto y protocolo de interred de registro
- Crear el mensaje de registro de XML
- Enviar el mensaje de registro de XML en el zócalo
- Leer el mensaje de respuesta de XML fuera del zócalo
- Cerrar el zócalo

1. **Crear el código Java que aplica la lógica anterior.**

   El ejemplo siguiente muestra la aplicación del método `registerCallbacks` de la clase `CrnpClient` (que invoca el constructor de `CrnpClient`). Las llamadas a `createRegistrationString()` y `readRegistrationReply()` se describen con más detalle más adelante.

   `regIp` y `regPort` son miembros objeto configurados por el constructor.

```
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. **Aplique el método `unregister`. Este método lo invoca el método `shutdown` de `CrnpClient`. La implementación de `createUnregistrationString` se describe más adelante con más detalle.**

```
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();
}
```

▼ Generación de XML

Ahora que se ha configurado la estructura de la aplicación y ha escrito todo el código de conexión a red, es necesario escribir el código que genera y analiza el XML. Empiece escribiendo el código que genera el mensaje de registro de XML `SC_CALLBACK_REG`.

Un mensaje SC_CALLBACK_REG consiste en un tipo de registro (ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS o REMOVE_EVENTS), un puerto de rellamada y una lista de eventos de interés. Cada evento consta de una clase y una subclase, seguidos de una lista de pares de nombres y valores.

En esta parte del ejemplo, se escribe una clase CallbackReg que guarda el tipo de registro, el puerto de rellamada y la lista de eventos de registro. Esta clase también puede serializarse en un mensaje de XML SC_CALLBACK_REG.

Un método interesante de esta clase es el convertToXml, que crea una cadena de mensaje de XML SC_CALLBACK_REG a partir de los miembros de la clase. La documentación de JAXP de http://java.sun.com/xml/jaxp/index.html describe el código de este método con más detalle.

La aplicación de la clase Event se muestra a continuación. Observe que la clase CallbackReg utiliza una clase Event que almacena un evento y puede convertirlo en un Element de XML.

1. Crear el código Java que aplica la lógica anterior.

```
class CallbackReg
{
        public static final int ADD_CLIENT = 0;
        public static final int ADD_EVENTS = 1;
        public static final int REMOVE_EVENTS = 2;
        public static final int REMOVE_CLIENT = 3;

        public CallbackReg()
        {
                port = null;
                regType = null;
                regEvents = new Vector();
        }

        public void setPort(String portIn)
        {
                port = portIn;
        }

        public void setRegType(int regTypeIn)
        {
                switch (regTypeIn) {
                case ADD_CLIENT:
                        regType = "ADD_CLIENT";
                        break;
                case ADD_EVENTS:
                        regType = "ADD_EVENTS";
                        break;
                case REMOVE_CLIENT:
                        regType = "REMOVE_CLIENT";
                        break;
                case REMOVE_EVENTS:
                        regType = "REMOVE_EVENTS";
                        break;
```

```
        default:
                System.out.println("Error, regType no válido " +
                    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) {
                // No se puede crear el analizador con las opciones
                // especificadas
                pce.printStackTrace();
                System.exit(1);
        }

        // Crear el elemento raíz
        Element root = (Element) document.createElement(
            "SC_CALLBACK_REG");

        // Añadir los atributos
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Añadir los eventos
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(
                    document));
        }
        document.appendChild(root);

        // Ahora, convertir el documento en cadena.
        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. **Implementar las clases `Event` y `NVPair`.**

   Observe que la clase `CallbackReg` utiliza una clase `Event`, que usa a su vez una clase `NVPair`.

```
class Event
{

        public Event()
        {
                regClass = regSubclass = null;
                nvpairs = new Vector();
        }

        public void setClass(String classIn)
        {
                regClass = classIn;
        }

        public void setSubclass(String subclassIn)
        {
                regSubclass = subclassIn;
        }

        public void addNvpair(NVPair nvpair)
        {
                nvpairs.add(nvpair);
        }

        public Element createXmlElement(Document doc)
        {
                Element event = (Element)
                    doc.createElement("SC_EVENT_REG");
                event.setAttribute("CLASS", regClass);
                if (regSubclass != null) {
                        event.setAttribute("SUBCLASS", regSubclass);
                }
                for (int i = 0; i < nvpairs.size(); i++) {
                         NVPair tempNv = (NVPair)
                             (nvpairs.elementAt(i));
                         event.appendChild(tempNv.createXmlElement(
                             doc));
                }
                return (event);
        }
```

```
        private String regClass, regSubclass;
        private Vector nvpairs;
}

class NVPair
{
        public NVPair()
        {
                name = value = null;
        }

        public void setName(String nameIn)
        {
                name = nameIn;
        }

        public void setValue(String valueIn)
        {
                value = valueIn;
        }

        public Element createXmlElement(Document doc)
        {
                Element nvpair = (Element)
                    doc.createElement("NVPAIR");
                Element eName = doc.createElement("NAME");
                Node nameData = doc.createCDATASection(name);
                eName.appendChild(nameData);
                nvpair.appendChild(eName);
                Element eValue = doc.createElement("VALUE");
                Node valueData = doc.createCDATASection(value);
                eValue.appendChild(valueData);
                nvpair.appendChild(eValue);

                return (nvpair);
        }

        private String name, value;
}
```

▼ Creación de los mensajes de registro y de anulación de registro

Ahora que se han creado las clases propias que generan los mensajes de XML, se puede escribir la aplicación del método `createRegistrationString`. Este método lo invoca el método `registerCallbacks`, descrito en «Registro y anulación del registro de rellamadas» en la página 227.

`createRegistrationString` construye un objeto `CallbackReg` y establece su puerto y tipo de registro. Después, `createRegistrationString` construye varios eventos, mediante los métodos propios `createAllEvent`,

`createMembershipEvent`, `createRgEvent` y `createREvent`. Cada evento se agrega al objeto `CallbackReg` después de crearlo. Finalmente, `createRegistrationString` invoca el método `convertToXml` en el objeto `CallbackReg` para recuperar el mensaje de XML en la forma de `String`.

Observe que la variable de miembro `regs` almacena los argumentos de la línea de comandos que proporciona un usuario a la aplicación. El quinto argumento y siguientes especifican los eventos para los cuales debería registrarse la aplicación. El cuarto argumento especifica el tipo de registro, pero se ignora en este ejemplo. El código completo del Apéndice G muestra cómo utilizar este cuarto argumento.

1. Crear el código Java que aplica la lógica anterior.

```
private String createRegistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);

        cbReg.setRegType(CallbackReg.ADD_CLIENT);

        // añadir eventos
        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. **Crear la cadena de anulación de registro.**

   Crear la cadena de anulación de registro es más fácil que crear la de registro, porque no es necesario incorporar eventos:

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}
```

▼ Configuración del analizador de XML

Una vez creado el código de generación de XML y de red para la aplicación, el paso final es analizar y procesar la respuesta de registro y las rellamadas de eventos. El constructor `CrnpClient` invoca un método `setupXmlProcessing` que crea un objeto `DocumentBuilderFactory` y establece en él varias propiedades de análisis. La documentación JAXP de `http://java.sun.com/xml/jaxp/index.html` describe este método con más detalle.

- **Crear el código Java que aplica la lógica anterior.**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        // No es necesario validar
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // Se desea ignorar comentarios y espacios en blanco
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Fusionar secciones CDATA en nodos de TEXT.
        dbf.setCoalescing(true);
}
```

▼ Analizar la respuesta de registro

Para analizar el mensaje de XML `SC_REPLY` que envía el servidor CRNP en respuesta a un mensaje de registro o anulación de registro, se necesita una clase propia `RegReply` que se puede construir a partir de un documento de XML. Esta clase proporciona accesores para el código de estado y el mensaje de estado. Para analizar el flujo de XML desde el servidor hay que crear un nuevo documento XML y utilizar el método de análisis del documento (la documentación de JAXP en `http://java.sun.com/xml/jaxp/index.html` describe el método con más detalle).

1. **Crear el código Java que aplica la lógica anterior.**

   Observe que el método `readRegistrationReply` utiliza la nueva clase `RegReply`.

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Crear el constructor de documento
        DocumentBuilder db = dbf.newDocumentBuilder();
        db.setErrorHandler(new DefaultHandler());

        //analizar el archivo de entrada
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
}
```

2. **Implementar la clase `RegReply`.**

   Observe que el método `retrieveValues` recorre el árbol DOM del documento XML y extrae el código y el mensaje de estado. La documentación de JAXP de `http://java.sun.com/xml/jaxp/index.html` ofrece más detalles.

```
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;

                // Buscar el elemento SC_REPLY.
                nl = doc.getElementsByTagName("SC_REPLY");
                if (nl.getLength() != 1) {
                        System.out.println("Error al analizar: "
                            " no se puede encontrar "
                            + "nodo SC_REPLY node.");
                        return;
                }

                n = nl.item(0);

                // Recuperar el valor del atributo STATUS_CODE
                statusCode = ((Element)n).getAttribute("STATUS_CODE");

                // Buscar el elemento SC_STATUS_MSG
                nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
                if (nl.getLength() != 1) {
                        System.out.println("Error al analizar: "
                            " no se puede encontrar "
                            + "nodo SC_STATUS_MSG.");
                        return;
                }
                // Obtener la sección TEXT, si la hubiera.
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                        // Sin error, si no lo hay
                         // se retorna discretamente.
```

```
                return;
            }

            // Recuperar el valor
            statusMsg = n.getNodeValue();
        }

        private String statusCode;
        private String statusMsg;
}
```

▼ Análisis de los eventos de rellamada

El paso final consiste en analizar y procesar los eventos reales de rellamada. Para facilitar esta tarea, se debe modificar la clase `Event` creada en «Generación de XML» en la página 227 de modo que pueda construir un `Event` a partir de un documento de XML y crear un `Element` de XML. Este cambio requiere un constructor adicional (que acepte un documento XML), un método `retrieveValues`, la adición de dos variables de miembro (`vendor` y `publisher`), métodos de accesor para todos los campos y un método de impresión.

1. **Crear el código Java que aplica la lógica anterior.**

   Observe que este código es similar al código de la clase `RegReply` descrito en «Analizar la respuesta de registro» en la página 234.

```
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;

        // Buscar el elemento SC_EVENT.
```

```
        nl = doc.getElementsByTagName("SC_EVENT");
        if (nl.getLength() != 1) {
                System.out.println("Error al analizar: "
                    " no se puede encontrar "
                    + "nodo SC_EVENT.");
                return;
        }

        n = nl.item(0);

        //
        // Recuperar los valores de los atributos
        // CLASS, SUBCLASS, VENDOR y PUBLISHER.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Recuperar todos los pares n-v
        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. **Implementar los métodos y constructores adicionales para la clase `NVPair` que admite el análisis de XML.**

   Los cambios en la clase `Event` que se muestran en el Paso 1 requieren cambios similares en la clase `NVPair`.

```
public NVPair(Element elem)
{
        retrieveValues(elem);
}
public void print(PrintStream out)
{
        out.println("NAME=" + nombre + " VALUE=" + valor);
}
private void retrieveValues(Element elem)
{
        Node n;
        NodeList nl;
        String nodeName;
        // Buscar el elemento NAME
        nl = elem.getElementsByTagName("NAME");
        if (nl.getLength() != 1) {
                System.out.println(""Error al analizar: "
                    " no se puede encontrar "
                    + "nodo NAME.");
                return;
        }
        // Obtener la sección TEXT
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                System.out.println(""Error al analizar: "
                    " no se puede encontrar "
                    + "sección TEXT.");
                return;
        }

        // Recuperar el valor
        name = n.getNodeValue();

        // Obtener ahora el elemento valor
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
                System.out.println(""Error al analizar: "
                    " no se puede encontrar "
                    + "nodo VALUE.");
                return;
        }
        // Obtener la sección TEXT
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        System.out.println(""Error al analizar: "
                    " no se puede encontrar "
                    + "sección TEXT.");
                return;
        }
```

```
                // Recuperar el valor
                value = n.getNodeValue();
                }

        public String getName()
        {
                return (name);
        }

        public String getValue()
        {
                return (value);
        }
}
```

3. **Implementar el bucle `while` en `EventReceptionThread`, que espera rellamadas de eventos (`EventReceptionThread` se describe en «Definición del subproceso de recepción de eventos» en la página 225).**

```
while(true) {
                Socket sock = listeningSock.accept();
                Document doc = db.parse(sock.getInputStream());
                Event event = new Event(doc);
                client.processEvent(event);
                sock.close();
        }
```

▼ Ejecución de la aplicación

- **Ejecutar la aplicación.**

  # **`java CrnpClient`** *sistema_crnp puerto_crnp puerto_local* **`...`**

  El código completo de la aplicación `CrnpClient` se puede consultar en el Apéndice G.

APÉNDICE **A**

Propiedades estándar

Este apéndice describe el tipo de recurso, grupo de recursos y propiedades de recurso estándar, así como los atributos de la propiedad de recurso disponibles para modificar las propiedades definidas por el sistema y crear las propiedades de extensión.

Este apéndice incluye las siguientes secciones principales:

Nota – Los valores de propiedad, como `True` y `False`, *no* distinguen entre mayúsculas y minúsculas.

Propiedades del tipo de recurso

La tabla siguiente describe las propiedades del tipo de recurso que define Sun Cluster. Los valores de propiedad están divididos en categorías como sigue (en la columna Categoría):

- **Necesario**: la propiedad requiere un valor explícito en el archivo de registro del tipo de recurso (RTR); si no no será posible crear el objeto al que pertenece. No se admite un espacio en blanco ni una cadena vacía como valor.
- **Condicional**; para que exista, la propiedad debe estar declarada en el archivo RTR; en caso contrario, RGM no la creará y no estará disponible para utilidades administrativas. Se permite un espacio en blanco o una cadena vacía. Si la propiedad está declarada en el archivo RTR, pero no se especifica ningún valor, RGM le proporciona un valor predeterminado.

- **Condicional/explícito**: para que exista, la propiedad debe estar declarada en el archivo RTR con un valor explícito; en caso contrario, RGM no la creará y no estará disponible para utilidades administrativas. No se permite un espacio en blanco ni una cadena vacía.
- **Opcional**: la propiedad se puede declarar en el archivo RTR. Si la propiedad no se declara, RGM la crea y le proporciona un valor predeterminado. Si la propiedad está declarada en el RTR pero no se le ha especificado ningún valor, RGM le proporciona el mismo valor predeterminado que si no se hubiera declarado.

Las propiedades del tipo de recurso no se pueden actualizar con utilidades administrativas, salvo `Installed_nodes`, que no se puede declarar en el archivo RTR y debe establecerla el administrador.

TABLA A–1 Propiedades del tipo de recurso

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Allow_hosts` (matriz de cadenas)	Controla el conjunto de clientes que están autorizados a registrarse con el daemon `cl_apid` para recibir eventos de reconfiguración de clúster. La forma general de esta propiedad es `ipaddress/masklength`, que define una subred desde la cual los clientes se pueden registrar. Por ejemplo, el valor `129.99.77.0/24` permite a los clientes de la subred `129.99.77` registrarse para recibir eventos. `192.9.84.231/32` sólo permite que el cliente `192.9.84.231` se registre para recibirlos. Esta propiedad aporta seguridad al CRNP. El daemon `cl_apid` se describe en `SUNW.Event`(5). Además, se reconocen las siguientes palabras clave especiales. `LOCAL` se refiere a todos los clientes situados en subredes del clúster conectadas directamente. `ALL` permite que todos los clientes se registren. Tenga en cuenta que si un cliente coincide con una entrada en las propiedades `Allow_hosts` y `Deny_hosts` no podrá registrarse en la implementación. El valor predeterminado es `LOCAL`.	N	Opcional
`API_version` (entero)	La versión de la API de gestión de recursos que utiliza la implementación de este tipo de recurso. El valor predeterminado de SC 3.1 es 2.	N	Opcional

TABLA A–1 Propiedades del tipo de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Boot` (cadena)	Un método de rellamada opcional: la ruta al programa que el gestor de grupos de recursos invoca en un nodo, que se une o vuelve a unirse al clúster cuando un recurso de este tipo ya está gestionado. Este método debe inicializar recursos de este tipo, similares al método `Init`.	N	Condicional/ explícito
`Client_retry_count` (entero)	Controla el número de intentos que realiza el daemon `cl_apid` al comunicarse con clientes externos. Si un cliente no responde en un número de intentos `Client_retry_count`, se acaba el tiempo de espera del cliente. Entonces, éste se elimina de la lista de clientes registrados que pueden recibir eventos de reconfiguración de clúster. El cliente debe volver a registrarse para empezar a recibir eventos otra vez. Consulte la descripción de la propiedad `Client_retry_interval` para obtener información sobre la frecuencia con que realiza esos intentos la implementación. El daemon `cl_apid` se describe en `SUNW.Event`(5). El valor predeterminado es 3.	S	Opcional
`Client_retry_interval` (entero)	Define el periodo de tiempo (en segundos) que utiliza el daemon `cl_apid` al comunicarse con clientes externos que no responden. Hasta `Client_retry_count` intentos se realizan en este intervalo para ponerse en contacto con el cliente. El daemon `cl_apid` se describe en `SUNW.Event`(5). El valor predeterminado es 1800.	S	Opcional
`Client_timeout` (entero)	El valor de tiempo de espera (en segundos) que emplea el daemon `cl_apid` al comunicarse con clientes externos. Sin embargo, el daemon `cl_apid` sigue intentando ponerse en contacto con el cliente durante un número de veces ajustable. Consulte las descripciones de las propiedades `Client_retry_count` y `Client_retry_interval` para obtener más información sobre cómo ajustar esta propiedad. El daemon `cl_apid` se describe en `SUNW.Event`(5). El valor predeterminado es 60.	S	Opcional

TABLA A–1 Propiedades del tipo de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Deny_hosts` (matriz de cadenas)	Controla el conjunto de clientes que no pueden registrarse para recibir eventos de reconfiguración del clúster. Para determinar el acceso, la configuración de esta propiedad tiene preferencia sobre las de la lista `Allow_hosts`. El formato de esta propiedad es el mismo que el que se define en la propiedad `Allow_hosts`. Esta propiedad aporta seguridad al CRNP. El valor predeterminado es `NULL`.	S	Opcional
`Failover` (Booleano)	`True` indica que los recursos de este tipo no se pueden configurar en ningún grupo que pueda estar en línea en varios nodos al mismo tiempo. El valor predeterminado es `False`.	N	Opcional
`Fini` (cadena)	Un método de rellamada opcional: la ruta al programa que invoca el gestor de grupos de recursos, cuando un recurso de este tipo se elimina del control del gestor de grupos de recursos.	N	Condicional/ explícito
`Init` (cadena)	Un método de rellamada opcional: la ruta al programa que invoca el gestor de grupos de recursos, cuando un recurso de este tipo entra bajo el control del gestor de grupos de recursos.	N	Condicional/ explícito
`Init_nodes` (enum.)	Los valores pueden ser `RG_primaries` (sólo los nodos que puede controlar el recurso) o `RT_installed_nodes` (todos los nodos en los que está instalado el tipo de recurso). Indica los nodos en los que el gestor de grupos de recursos llamará a los métodos `Init`, `Fini`, `Boot` y `Validate`. El valor predeterminado es `RG_primaries`.	N	Opcional
`Installed_nodes` (matriz de cadenas)	Una lista de los nombres de nodo del clúster en los que se puede ejecutar el tipo de recurso. RGM crea automáticamente esta propiedad. El administrador del clúster puede fijar el valor. No se puede declarar esta propiedad en el archivo RTR. El valor predeterminado es todos los nodos del clúster.	S	Puede configurarla el administrador del clúster

TABLA A–1 Propiedades del tipo de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Max_clients` (entero)	Controla el número máximo de clientes que se pueden registrar con el daemon `cl_apid` para recibir notificación de eventos del clúster. Los intentos de otros clientes de registrarse en eventos serán rechazados por la aplicación. Dado que cada registro de cliente utiliza recursos del clúster, ajustar esta propiedad permite a los usuarios controlar la utilización del recurso del clúster por parte de clientes externos. El daemon `cl_apid` se describe en `SUNW.Event`(5). El valor predeterminado es 1000.	S	Opcional
`Monitor_check` (cadena)	Un método de rellamada opcional: la ruta al programa que invoca el Gestor de grupos de recursos antes de hacer una operación de recuperación de fallos solicitada por el supervisor de un recurso de este tipo.	N	Condicional/ explícito
`Monitor_start` (cadena)	Un método de rellamada opcional: la ruta al programa que invoca el Gestor de grupos de recursos para empezar un supervisor de fallos para un recurso de este tipo.	N	Condicional/ explícito
`Monitor_stop` (cadena)	Un método de rellamada necesario si se fija `Monitor_start`: la ruta al programa que invoca el Gestor de grupos de recursos para detener un recurso de este tipo.	N	Condicional/ explícito
`Num_resource_restarts` en cada nodo de clúster (entero)	Esta propiedad está establecida por el RGM en el número de llamadas de `scha_control RESTART` que se han realizado para este recurso en este nodo en los últimos *n* segundos, donde *n* es el valor de la propiedad `Retry_interval` del recurso. Si un tipo de recurso no declara la propiedad `Retry_interval`, la propiedad `Num_resource_restarts` no estará disponible para recursos de ese tipo.	N	Sólo consulta
`Pkglist` (matriz de cadenas)	Una lista opcional de paquetes que están incluidos en la instalación del tipo de recursos.	N	Condicional/ explícito

TABLA A–1 Propiedades del tipo de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Postnet_stop` (cadena)	Un método de rellamada opcional: la ruta al programa que el Gestor de grupos de recursos invoca después de llamar al método `Stop` de cualquier recurso de dirección de red (`Network_resources_used`) del que depende un recurso de este tipo. Este método debe realizar acciones de `STOP` que deben completarse después de que se hayan configurado las interfaces de red para desconectarse.	N	Condicional/ explícito
`Prenet_start` (cadena)	Un método de rellamada opcional: la ruta al programa que el Gestor de grupos de recursos invoca después de llamar al método `Start` de cualquier recurso de dirección de red (`Network_resources_used`) del que depende un recurso de este tipo. Este método debe realizar acciones de `START` que deben completarse después de que se hayan configurado las interfaces de red para desconectarse.	N	Condicional/ explícito

`Resource_type` (cadena)	El nombre del tipo de recurso. Para ver los nombres de los tipos de recursos registrados actualmente, utilice:
	`scrgadm` `-p`
	Desde Sun Cluster 3.1, el formato de un nombre de tipo de recurso es:
	`id_fabricante.tipo_recurso:versión`
	Los tres componentes del nombre del tipo de recurso son propiedades que se especifican en el archivo RTR como *ID_fabricante*, *tipo_recurso* y *versión_TR*. El comando `scrgadm` introduce los separadores de punto y punto y coma. El sufijo `RT_version` del nombre del tipo de recurso es el mismo valor que la propiedad `RT_version`. Para garantizar que la propiedad *ID_fabricante* sea única, se recomienda utilizar el símbolo bursátil de la empresa que crea el tipo de recurso. Los nombres de tipo de recurso creados antes de Sun Cluster 3.1 siguen teniendo el formato:
	`ID_fabricante.tipo_recurso`
	El valor predeterminado es la cadena vacía.
		N	Necesaria

TABLA A–1 Propiedades del tipo de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`RT_basedir` (cadena)	La ruta de directorio que se usa para completar las rutas relativas de los métodos de rellamada. Esta ruta se debe establecer en el punto de instalación de los paquetes de tipos de recursos. Debe ser una ruta completa, es decir, tiene que empezar con una barra inclinada (/). Esta propiedad no es necesaria si todos los nombres de ruta de métodos son absolutos.	N	Necesario salvo que todos los nombres de rutas de métodos sean absolutos
`RT_description` (cadena)	Una descripción breve del tipo de recurso. El valor predeterminado es la cadena vacía.	N	Condicional
`RT_version` (cadena)	A partir de Sun Cluster 3.1, una cadena de versión obligatoria para esta implementación del tipo de recurso, `RT_version` es el componente de sufijo del nombre del tipo de recurso completo.	N	Condicional/ explícito
`Single_instance` (Booleano)	Si es `True`, indica que sólo puede existir un recurso de este tipo en el clúster. RGM sólo permite que se ejecute un recurso de este tipo en todo el clúster en cada momento. El valor predeterminado es `False`.	N	Opcional
`Start` (cadena)	Un método de rellamada: la ruta al programa que invoca el Gestor de grupos de recursos para empezar un recurso de este tipo.	N	Necesario salvo que el archivo RTR declare un método `Prenet_start`
`Stop` (cadena)	Un método de rellamada: la ruta al programa que invoca el Gestor de grupos de recursos para detener un recurso de este tipo.	N	Necesario salvo que el archivo RTR declare un método `Postnet_stop`
`Update` (cadena)	Un método de rellamada opcional: la ruta al programa que invoca el Gestor de grupos de recursos cuando se cambian las propiedades de un recurso de este tipo en ejecución.	N	Condicional/ explícito
`Validate` (cadena)	Un método de rellamada opcional: la ruta al programa que se invoca para comprobar los valores de las propiedades de los recursos de este tipo.	N	Condicional/ explícito
`Vendor_ID` (cadena)	Consulte la propiedad `Resource_type`.	N	Condicional

Propiedades de recurso

La Tabla A–2 describe las propiedades de recurso que define Sun Cluster. Los valores de propiedad están divididos en categorías como sigue (en la columna Categoría):

- **Necesario**: el administrador debe especificar un valor al crear un recurso con una utilidad administrativa.
- **Opcional**: si el administrador no especifica un valor al crear un grupo de recursos, el sistema suministra un valor predeterminado.
- **Condicional**: RGM crea la propiedad sólo si ésta se declara en el archivo RTR, en caso contrario, la propiedad no existirá y no estará disponible para los administradores de sistemas. Una propiedad condicional declarada en el archivo RTR es opcional o necesaria, según si se especifique o no un valor predeterminado en el archivo RTR. Para obtener más detalles, consulte la descripción de cada propiedad condicional.
- **Sólo consulta**: no se puede configurar directamente mediante una herramienta administrativa.

La Tabla A–2 indica también si es posible y, en su caso, cuándo se pueden actualizar las propiedades de recurso (en la columna ¿Se puede actualizar?), como sigue:

`None` o `False`	Nunca
`True` o `Anytime`	En cualquier momento
`At_creation`	Al agregar el recurso al clúster
`When_disabled`	Al desactivar el recurso

TABLA A–2 Propiedades de recurso

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Affinity_timeout` (entero)	Tiempo, en segundos, durante el cual las conexiones de una dirección IP de cliente determinado a cualquier servicio del recurso se enviarán al mismo nodo del servidor. Esta propiedad sólo es importante cuando `Load_balancing_policy` es `Lb_sticky` o `Lb_sticky_wild`. Además, `Weak_affinity` se debe establecer en false (el valor predeterminado). Esta propiedad se utiliza solamente en servicios escalables.	En cualquier momento	Opcional
`Cheap_probe_interval` (entero)	El número de segundos entre llamadas de un análisis rápido de fallos del recurso. Esta propiedad sólo la crea RGM y sólo está disponible para el administrador si se declara en el archivo RTR. Esta propiedad es opcional si se especifica un valor predeterminado en el archivo RTR. Si el atributo `Tunable` no se especifica en el archivo de tipo de recurso, el valor `Tunable` de la propiedad será `When_disabled`. Esta propiedad es necesaria si el atributo `Default` no se especifica en la declaración de propiedades del archivo RTR.	Cuando está inhabilitado	Condicional
Propiedades de extensión	Las propiedades de extensión, como se declaran en el archivo RTR del tipo de recurso. La implementación del tipo de recurso define estas propiedades. Para obtener información sobre los atributos individuales que se pueden fijar para las propiedades de extensión, consulte la Tabla A–4.	Depende de la propiedad en cuestión	Condicional

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Failover_mode` (enum.)	Los valores posibles son: None, Soft y Hard. Controla si RGM reubica un grupo de recursos o abandona un nodo en respuesta a un fallo de una llamada de los métodos `Start`, `Stop` o `Monitor_stop` en el recurso. `None` (ninguno) indica que RGM debe fijar el estado del recurso en un fallo de método y esperar la intervención del operador. `Soft` (dinámico) indica que el fallo de un método `Start` debería hacer que RGM reubicara el grupo de recursos en un nodo diferente, en tanto que el fallo de un método `Stop` o `Monitor_stop` debería hacer que RGM pusiera el recurso en el estado `STOP_FAILED` y el grupo de recursos en el estado `ERROR_STOP_FAILED` y que se esperara la intervención del operador. Para los fallos de `Stop` o `Monitor_stop`, los valores `None` y `Soft` son equivalentes. `Hard` (estático) indica que el fallo de un método `Start` debería provocar la reubicación del grupo y el fallo de un método `Stop` o `Monitor_stop` debería provocar la parada forzada del recurso, mediante la anulación del nodo del clúster. La opción predeterminada es `None`.	En cualquier momento	Opcional

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Load_balancing_policy` (cadena)	Una cadena que define la política de equilibrio de cargas que se está utilizando. Esta propiedad sólo se usa para servicios escalables. RGM crea automáticamente esta propiedad si la propiedad `Scalable` está declarada en el archivo RTR. Load_balancing_policy puede tomar los valores siguientes: `Lb_weighted` (el predeterminado). La carga se distribuye entre varios nodos, de acuerdo con los pesos fijados en la propiedad `Load_balancing_weights`. `Lb_sticky`. Un cliente determinado (identificado por la dirección IP de cliente) del servicio escalable se envía siempre al mismo nodo del clúster. `Lb_sticky_wild`. Un cliente determinado (identificado por la dirección IP de cliente) que se conecta a una dirección IP de un servicio adherente con comodín, siempre se envía al mismo nodo del clúster, independientemente del número de puerto al que llegue. La opción predeterminada es `Lb_weighted`.	Al crearse	Condicional/ opcional
`Load_balancing_weights` (matriz de cadenas)	Sólo para recursos escalables. RGM crea automáticamente esta propiedad si la propiedad `Scalable` está declarada en el archivo RTR. El formato es *peso@nodo,peso@nodo*, donde *peso* es un número entero que refleja la parte relativa de la carga distribuida al *nodo* especificado. La fracción de carga distribuida a un nodo es el peso de este nodo dividido entre la suma de todos los pesos. Por ejemplo, `1@1, 3@2` especifica que el nodo 1 recibe 1/4 de la carga y el nodo 2, 3/4. La cadena vacía (“”), la predeterminada, establece una distribución uniforme. Cualquier nodo que no tenga un peso asignado, recibe un peso predeterminado de 1. Si el atributo `Tunable` no se especifica en el archivo de tipo de recurso, el valor `Tunable` de la propiedad es `Anytime`. Cualquier cambio en esta propiedad revisa sólo la distribución de nuevas conexiones La opción predeterminada es la cadena vacía (“”).	En cualquier momento	Condicional/ opcional

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
*método*_timeout para cada método de rellamada en el tipo (entero)	Un margen de tiempo, en segundos, pasado el cual RGM concluye que la invocación del método ha fallado. El valor predeterminado es 3.600 (una hora) si el método en sí está declarado en el archivo RTR.	En cualquier momento	Condicional/ Opcional
Monitored_switch (enum.)	RGM lo establece en Enabled o Disabled si el administrador del clúster habilita o inhabilita el supervisor con una utilidad administrativa. Si es Disabled, no se invoca el método Start del supervisor hasta que se vuelva a habilitar. Si el recurso no tiene un método de rellamada del supervisor, la propiedad no existirá. El valor predeterminado es Enabled.	Nunca	Sólo consulta
Network_resources_used (matriz de cadenas)	Una lista de recursos de red de nombre lógico de servidor o dirección compartida que utiliza el recurso. Para los servicios escalables, esta propiedad se debe referir a recursos de dirección compartida existentes en un grupo de recursos aparte. En el caso de servicios a prueba de fallos, esta propiedad se refiere a recursos de nombre lógico de servidor o dirección compartida existentes en el mismo grupo de recursos. RGM crea automáticamente esta propiedad si la propiedad Scalable está declarada en el archivo RTR. Si Scalable no está declarada en el archivo RTR, Network_resources_used no estará disponible, salvo que se declare específicamente en el archivo RTR. Si el atributo Tunable no se especifica en el archivo de tipo de recurso, el valor Tunable de la propiedad será At_creation.	Al crearse	Condicional/ necesario
On_off_switch (enum.)	RGM lo establece en Enabled o Disabled si el administrador del clúster habilita o inhabilita el recurso con una utilidad administrativa. Si está inhabilitado, no se invocan rellamadas del recurso hasta que se vuelva a habilitar. El valor predeterminado es Disabled.	Nunca	Sólo consulta

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Port_list` (matriz de cadenas)	Una lista de números de puerto en los que recibe el servidor. Anexo a cada número de puerto está el protocolo que utiliza dicho puerto, por ejemplo `Port_list=80/tcp`. Si la propiedad `Scalable` está declarada en el archivo RTR, RGM crea automáticamente `Port_list`. En caso contrario, esta propiedad no estará disponible salvo que se declare explícitamente en el archivo RTR. La configuración de esta propiedad para Apache se detalla en *Sun Cluster 3.1 Data Service for Apache Guide*.	Al crearse	Condicional/ Necesaria
`R_description` (cadena)	Una descripción breve del recurso. El valor predeterminado es la cadena vacía.	En cualquier momento	Opcional
`Resource_name` (cadena)	El nombre de la instancia del recurso. Este nombre debe ser único en la configuración del clúster y no se puede cambiar después de su creación.	Nunca	Necesaria
`Resource_project_name` (cadena)	El nombre de proyecto Solaris asociado al recurso. Esta propiedad permite que los servicios de datos del clúster accedan al uso de las características de gestión de recursos de Solaris, como la CPU y la agrupación de recursos. Cuando RGM pone en línea los recursos, inicia los procesos relacionados bajo este nombre de proyecto. Si la propiedad no está especificada, el nombre del proyecto se obtendrá de la propiedad `RG_project_name` del grupo de recursos que contiene el recurso (consulte `rg_properties` (5)). Si no se especifica ninguna propiedad, RGM utilizará el nombre de proyecto predefinido `default`. El nombre de proyecto especificado debe existir en la base de datos de proyectos y el usuario `root` se debe configurar como miembro del proyecto nombrado. Esta propiedad sólo se admite a partir de Solaris 9. **Nota –** Los cambios a esta propiedad surten efecto después de reiniciar el recurso. La opción predeterminada es null.	En cualquier momento	Opcional

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Resource_state` en cada nodo del clúster (enum.)	El estado determinado por RGM del recurso en cada nodo del clúster. Los estados posibles son `Online`, `Offline`, `Stop_failed`, `Start_failed`, `Monitor_failed` y `Online_not_monitored`. El usuario no puede configurar esta propiedad.	Nunca	Sólo consulta
`Retry_count` (entero)	Las veces que el monitor intenta reiniciar un recurso si éste falla. Esta propiedad sólo la crea RGM y sólo está disponible para el administrador si se declara en el archivo RTR. Esta propiedad es opcional si se especifica un valor predeterminado en el archivo RTR. Si el atributo `Tunable` no se especifica en el archivo de tipo de recurso, el valor `Tunable` de la propiedad será `When_disabled`. Esta propiedad es necesaria si el atributo `Default` no se especifica en la declaración de propiedades del archivo RTR.	Cuando está inhabilitado	Condicional
`Retry_interval` (entero)	Los segundos que transcurren entre cada intento de reiniciar un recurso que ha fallado. El supervisor de recursos utiliza esta propiedad con `Retry_count`. Esta propiedad sólo la crea RGM y sólo está disponible para el administrador si se declara en el archivo RTR. Esta propiedad es opcional si se especifica un valor predeterminado en el archivo RTR. Si el atributo `Tunable` no se especifica en el archivo de tipo de recurso, el valor `Tunable` de la propiedad será `When_disabled`. Esta propiedad es necesaria si el atributo `Default` no se especifica en la declaración de propiedades del archivo RTR.	Cuando está inhabilitado	Condicional

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Scalable` (Booleano)	Indica si el recurso es escalable. Si esta propiedad se declara en el archivo RTR, RGM crea automáticamente las siguientes propiedades de servicio escalables para recursos de ese tipo: `Network_resources_used`, `Port_list`, `Load_balancing_policy` y `Load_balancing_weights`. Estas propiedades tienen valores predeterminados si no se declaran explícitamente en el archivo RTR. El valor predeterminado de `Scalable` (cuando se declara en el archivo RTR) es `True`. Cuando esta propiedad se declara en el archivo RTR, el atributo `Tunable` se debe establecer en `At_creation` o la creación del recurso fallará. Si esta propiedad no se declara en el archivo RTR, el recurso no será escalable, el administrador no podrá ajustarla y RGM no establecerá ninguna propiedad de servicio escalable. Sin embargo, se pueden declarar explícitamente las propiedades `Network_resources_used` y `Port_list` en el archivo RTR, si se desea, porque pueden ser útiles en servicios no escalables y escalables.	Al crearse	Opcional
`Status` en cada nodo de clúster (enum.)	Configurado por el monitor de recursos. Los valores posibles son: `OK`, `degraded`, `faulted`, `unknown` y `offline`. RGM establece el valor en `unknown` cuando el recurso se pone en línea y en `Offline` cuando se pone fuera de línea.	Nunca	Sólo consulta
`Status_msg` en cada nodo del clúster (cadena)	Establecido por el supervisor de recursos al mismo tiempo que la propiedad `Status`. Esta propiedad se puede establecer por recurso o por nodo. RGM la establece en la cadena vacía cuando el recurso se pone fuera de línea.	Nunca	Sólo consulta

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Thorough_probe_interval` (entero)	Los segundos que transcurren entre las invocaciones a una sonda de fallos de alta sobrecarga del recurso. Esta propiedad sólo la crea RGM y sólo está disponible para el administrador si se declara en el archivo RTR. Esta propiedad es opcional si se especifica un valor predeterminado en el archivo RTR. Si el atributo `Tunable` no se especifica en el archivo de tipo de recurso, el valor `Tunable` de la propiedad será `When_disabled`. Esta propiedad es necesaria si el atributo `Default` no se especifica en la declaración de propiedades del archivo RTR.	Cuando está inhabilitado	Condicional
`Type` (cadena)	El tipo de recurso del cual este recurso es una instancia.	Nunca	Necesaria
`Type_version` (cadena)	Indica la versión del tipo de recurso actualmente asociada al recurso. RGM crea automáticamente esta propiedad, que no se puede declarar en el archivo RTR. El valor de esta propiedad es el mismo que el de la propiedad `RT_version` del tipo de recurso. Cuando se crea un recurso, la propiedad `Type_version` no se especifica explícitamente, aunque pueda parecer como un sufijo del nombre del tipo de recurso. Cuando se edita un recurso, `Type_version` puede cambiar a un valor nuevo. Las posibilidades de configuración se derivan de: ■ La versión actual del tipo de recurso ■ La directiva `#$upgrade_from` del archivo RTR	Consulte la descripción	Consulte la descripción

TABLA A–2 Propiedades de recurso *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`UDP_affinity` (Booleano)	Si es verdadero, todo el tráfico de UDP de un cliente determinado se envía al mismo nodo de servidor que gestiona actualmente todo el tráfico de TCP del cliente. Esta propiedad sólo es importante cuando `Load_balancing_policy` es `Lb_sticky` o `Lb_sticky_wild`. Además, `Weak_affinity` se debe establecer en false (el valor predeterminado). Esta propiedad se utiliza solamente en servicios escalables.	Cuando está inhabilitado	Opcional
`Weak_affinity` (Booleano)	En caso de ser true, habilita la forma débil de afinidad del cliente. Esto permite las conexiones de un determinado cliente a un mismo nodo de servidor excepto: ■ Si el oyente del servidor se inicia, por ejemplo, debido a arranques erróneos del monitor, una recuperación de fallos del recurso, una conmutación o un nodo que se vuelve a unir al clúster después de fallar. ■ Cuando `Load_balancing_weights` del recurso escalable cambia debido a una acción administrativa. La afinidad débil proporciona una alternativa de baja sobrecarga a la forma predeterminada, tanto en consumo de memoria como en ciclos de procesador. Esta propiedad sólo es importante cuando `Load_balancing_policy` es `Lb_sticky` o `Lb_sticky_wild`. Esta propiedad se utiliza solamente en servicios escalables.	Cuando está inhabilitado	Opcional

Propiedades de grupo de recursos

La tabla siguiente describe las propiedades de grupo de recursos que define Sun Cluster. Los valores de propiedad están divididos en categorías como sigue (en la columna Categoría):

- **Necesario**: el administrador debe especificar un valor al crear un recurso con una utilidad administrativa.
- **Opcional**: si el administrador no especifica un valor al crear un grupo de recursos, el sistema suministra un valor predeterminado.
- **Sólo consulta**: no se puede configurar directamente mediante una herramienta administrativa.

La columna ¿Se puede actualizar? muestra si la propiedad se puede actualizar (S) o no (N) después de configurarla inicialmente.

TABLA A–3 Propiedades del grupo de recursos

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Auto_start_on_new_cluster` (Booleano)	Esta propiedad impide el inicio automático del grupo de recursos cuando se está formando un clúster nuevo. El valor predeterminado es `TRUE`. Si se establece en `TRUE`, el Gestor de grupos de recursos intentará iniciar automáticamente el grupo de recursos para lograr `Desired_primaries` cuando se rearranque el clúster. Si se establece en `FALSE`, el grupo de recursos no se iniciará automáticamente al rearrancar el clúster.	S	Opcional
`Desired_primaries` (entero)	El número de nodos en los que se desea que el grupo esté en línea simultáneamente. El valor predeterminado es 1. Si la propiedad `RG_mode` es `Failover`, el valor de esta propiedad no debe ser mayor que 1. Si la propiedad `RG_mode` es `Scalable`, se permite un valor mayor que 1.	S	Opcional
`Failback` (Booleano)	Un valor booleano que indica si se debe recalcular el conjunto de nodos en los que el grupo está en línea cuando cambian los miembros del clúster. Un cálculo nuevo puede hacer que RGM ponga el grupo fuera de línea en nodos no prioritarios y en línea en los más prioritarios. El valor predeterminado es `False`.	S	Opcional
`Global_resources_used` (matriz de cadenas)	Indica si algún recurso del grupo utiliza o no los sistemas de archivos del clúster. Los valores legítimos que puede especificar el administrador son un asterisco (*) para indicar todos los recursos globales o una cadena vacía (“”) que indica que no hay ningún recurso global. El valor predeterminado es todos los recursos globales.	S	Opcional

TABLA A–3 Propiedades del grupo de recursos *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Implicit_network_dependencies` (Booleano)	Un valor booleano que indica, cuando es `True`, que RGM debería imponer dependencias fuertes implícitas de recursos de dirección no de red a los recursos de dirección de red del grupo. Ejemplos de servicios de red-dirección incluyen el nombre de host lógico y los tipos de recursos de dirección compartida. En un grupo de recursos escalable, esta propiedad no tiene ningún efecto, porque un grupo escalable no tiene recursos de red-dirección. El valor predeterminado es `True`.	S	Opcional
`Maximum_primaries` (entero)	El número máximo de nodos en los que el grupo puede estar en línea simultáneamente. El valor predeterminado es 1. Si la propiedad `RG_mode` es `Failover`, el valor de esta propiedad no debe ser mayor que 1. Si la propiedad `RG_mode` es `Scalable`, se permite un valor mayor que 1.	S	Opcional
`Nodelist` (matriz de cadenas)	Una lista de nodos de clúster en los que el grupo se puede poner en línea en orden de preferencia. Estos nodos se denominan los primarios potenciales o los maestros del grupo de recursos. El valor predeterminado es la lista de todos los nodos del clúster.	S	Opcional
`Pathprefix` (cadena)	Un directorio del sistema de archivos del clúster en los que los recursos del grupo pueden escribir los archivos administrativos esenciales. Esta propiedad puede ser imprescindible para algunos recursos. `Pathprefix` debe ser único para cada grupo de recursos. El valor predeterminado es la cadena vacía.	S	Opcional

TABLA A–3 Propiedades del grupo de recursos *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`Pingpong_interval` (entero)	Un valor entero no negativo (en segundos) utilizado por RGM para determinar dónde debe ponerse en línea el grupo de recursos en caso de una reconfiguración o después de ejecutar un comando `scha_control -O GIVEOVER` o una función `scha_control()` con el argumento `SCHA_GIVEOVER`. En caso de reconfiguración, si el grupo de recursos no se pone en línea más de una vez en los últimos `Pingpong_interval` segundos en un nodo determinado (porque los métodos `Start` o `Prenet_start` del recurso devolvieron un valor diferente de cero o agotaron el tiempo de espera), ese nodo no se considera susceptible de alojar el grupo de recursos y RGM busca otro maestro. Si una llamada a un comando `scha_control` o función `scha_control()` de recurso hace que el grupo de recursos se ponga fuera de línea en un nodo concreto en los últimos `Pingpong_interval` segundos, ese nodo no podrá alojar el grupo de recursos, como resultado de una llamada posterior a `scha_control()` proveniente de otro nodo. El valor predeterminado es 3.600 (una hora).	S	Opcional
`Resource_list` (matriz de cadenas)	La lista de recursos que un grupo contiene. El administrador no puede configurar esta propiedad directamente. En su lugar, RGM la actualiza cuando el administrador agregue o elimine recursos del grupo de recursos. El valor predeterminado es la lista vacía.	N	Sólo consulta
`RG_description` (cadena)	Descripción breve del grupo de recursos. El valor predeterminado es la cadena vacía.	S	Opcional

TABLA A–3 Propiedades del grupo de recursos *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`RG_mode` (enum.)	Indica si el grupo de recursos es a prueba de fallos o escalable. Si el valor es `Failover`, RGM establece la propiedad `Maximum_primaries` del grupo en 1 y limita el grupo de recursos para que lo controle un solo nodo. Si el valor de esta propiedad es `Scalable`, RGM permite que la propiedad `Maximum_primaries` tenga un valor superior a 1, lo que quiere decir que el grupo puede tener varios nodos maestros simultáneamente. RGM no permite que un recurso con una propiedad `Failover` establecida en `True` se agregue a un grupo de recursos cuyo `RG_mode` sea `Scalable`. El valor predeterminado es `Failover` si `Maximum_primaries` es 1 y `Scalable` si `Maximum_primaries` es mayor que 1.	N	Opcional
`RG_name` (cadena)	El nombre del grupo de recursos. Es obligatorio que el nombre no se repita en el clúster.	N	Necesaria
`RG_project_name` (cadena)	El nombre de proyecto Solaris asociado al grupo de recursos. Esta propiedad permite que los servicios de datos del clúster accedan al uso de las características de gestión de recursos de Solaris, como la CPU y la agrupación de recursos. Cuando RGM pone los grupos de recursos en línea, inicia los procesos relacionados bajo este nombre de proyecto para los recursos que no tienen establecida la propiedad `Resource_project_name`. El nombre de proyecto especificado debe existir en la base de datos de proyectos y el usuario `root` se debe configurar como miembro del proyecto nombrado. Esta propiedad sólo se admite a partir de Solaris 9. **Nota –** Los cambios a esta propiedad surten efecto después de reiniciar el recurso.	En cualquier momento	Necesaria

TABLA A–3 Propiedades del grupo de recursos *(Continuación)*

Nombre de propiedad	Descripción	¿Se puede actualizar?	Categoría
`RG_state` en cada nodo del clúster (enum.)	Establecido por RGM en `Online`, `Offline`, `Pending_online`, `Pending_offline`, `Pending_online_blocked`, `Error_stop_failed` o `Online_faulted` para describir el estado del grupo en cada nodo del clúster. El usuario no puede configurar esta propiedad. Sin embargo, es posible establecer indirectamente esta propiedad, invocando `scswitch`(1M) (o utilizando el equivalente `scsetup`(1M) o comandos de SunPlex Manager).	N	Sólo consulta

Atributos de las propiedades de recursos

La tabla siguiente describe los atributos de la propiedad del recurso que se pueden usar para cambiar propiedades definidas por el sistema o crear propiedades de extensión.

Precaución – No puede especificar `NULL` o la cadena vacía (“”) como valor predeterminado para los tipos `boolean`, `enum` o `int`.

TABLA A–4 Atributos de las propiedades de recursos

Propiedad	Descripción
`Property`	El nombre de la propiedad de recurso.
`Extension`	Si se utiliza, indica que la entrada del archivo RTR declara una propiedad de extensión definida por la implementación del tipo de recurso. En caso contrario, la entrada será una propiedad definida por el sistema.
`Description`	Una anotación de cadena que se pretende sea una descripción breve de la propiedad. El atributo de descripción no se puede configurar en el archivo RTR para las propiedades que el sistema define.

TABLA A–4 Atributos de las propiedades de recursos *(Continuación)*

Propiedad	Descripción
Tipo de la propiedad	Los tipos permitidos son: `string`, `boolean`, `int`, `enum` y `stringarray`. El atributo de tipo no se puede configurar en el archivo RTR para las propiedades que el sistema define. El tipo determina los valores aceptables de la propiedad y los atributos específicos del tipo permitidos en la entrada del archivo RTR. Un tipo `enum` es un conjunto de valores de cadena.
`Default`	Indica un valor de propiedad predeterminado.
`Tunable`	Indica si el administrador del clúster puede configurar el valor de esta propiedad en un recurso. Se puede establecer en `None` o `False` para evitar que el administrador fije la propiedad. Los valores que permiten el ajuste del administrador son: `True` o `Anytime` (en cualquier momento), `At_creation` (sólo cuando se crea el recurso) o `When_disabled` (cuando el recurso está fuera de línea). El valor predeterminado es `True` (`Anytime`).
`Enumlist`	Para un tipo `enum`, se permite un conjunto de valores de cadena para la propiedad.
`Min`	Para un tipo `int`, el valor mínimo permitido para la propiedad.
`Max`	Para un tipo `int`, el valor máximo permitido para la propiedad.
`Minlength`	Para los tipos `string` y `stringarray`, la longitud de cadena mínima permitida.
`Maxlength`	Para los tipos `string` y `stringarray`, la longitud de cadena máxima permitida.
`Array_minsize`	Para el tipo `stringarray`, el número mínimo de elementos de matriz permitido.
`Array_maxsize`	Para el tipo `stringarray`, el número máximo de elementos de matriz permitido.

APÉNDICE **B**

Listados del código del servicio de datos de ejemplo

Este apéndice proporciona el código completo de cada método en el servicio de datos de ejemplo. Enumera también el contenido del archivo de registro del tipo de recurso.

Este apéndice incluye los siguientes listados de código.

Listado del archivo de registro del tipo de recurso

El archivo RTR (registro de tipo de recurso) contiene las declaraciones de propiedad del recurso y del tipo de recurso que define la configuración inicial del servicio de datos en el momento en que el administrador del clúster registra el servicio de datos.

EJEMPLO B–1 Archivo RTR de `SUNW.Ejemplo`

```
#
# Copyright (c) 1998-2003 de Sun Microsystems, Inc.
# Reservados todos los derechos.
#
# Información de registro para el Servicio de nombres de dominio (DNS)
```

EJEMPLO B–1 Archivo RTR de `SUNW.Ejemplo` *(Continuación)*

```
#


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

RESOURCE_TYPE = "sample";
VENDOR_ID = SUNW;
RT_DESCRIPTION = "Servicio de nombres de dominio de Sun Cluster";

RT_VERSION ="1.0";
API_VERSION = 2;
FAILOVER = TRUE;

RT_BASEDIR=/opt/SUNWsampleo/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;

# Una lista de declaraciones de propiedades de recurso entre llaves
# sigue a las declaraciones del tipo de recurso. La declaración del
# nombre de la propiedad debe ser el primer atributo después de la llave
# de apertura de cada entrada.
#

# Las propiedades de <method>_timeout fijan el valor en segundo
# tras el cual RGM determina que la invocación del método no ha
# sido satisfactoria.

# El valor MIN para todos los tiempos de espera de métodos es de 60
# segundos. Así se impide que los administradores definan tiempos de
# espera menores, que no mejoran el rendimiento de las operaciones
# de conmutación/recuperación de fallos y pueden provocar acciones
# indeseadas de RGM (recuperaciones de fallos falsas, rearranques de
# nodo o desplazamiento del grupo de recursos a un estado
# ERROR_STOP_FAILED, que requiera la intervención de un operador).
# Definir tiempos de espera demasiado cortos provoca una *disminución*
# de la disponibilidad global del servicio de datos.
{
   PROPERTY = Start_timeout;
   MIN=60;
   DEFAULT=300;
}

{
        PROPERTY = Stop_timeout;
```

EJEMPLO B–1 Archivo RTR de `SUNW.Ejemplo` *(Continuación)*

```
        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;
}

# El número de reintentos que se va a realizar en un determinado
# periodo antes de determinar que la aplicación no se puede iniciar
# satisfactoriamente en este nodo.
{
        PROPERTY = Retry_Count;
        MIN=0;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

# Definir Retry_Interval como múltiplo de 60, porque se convierte de
# segundos a minutos, en un redondeo. Por ejemplo, un valor de 50
# (segundos) se convierte en 1 minuto. Utilizar esta propiedad
# para cronometrar el número de reintentos (Retry_Count).
{
        PROPERTY = Retry_Interval;
        MIN=60;
        MAX=3600;
        DEFAULT=300;
        TUNABLE = ANYTIME;
}
```

EJEMPLO B–1 Archivo RTR de `SUNW.Ejemplo` *(Continuación)*

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

#
# Propiedades de extensión
#

#  El administrador del clúster debe definir el valor de esta propiedad para
#  que apunte al directorio que contiene los archivos de configuración
#  que utiliza la aplicación. Para esta aplicación, DNS, la ruta del archivo
#  de configuración de DNS se especifica en PXFS (normalmente
#  named.conf).
{
   PROPERTY = Confdir;
   EXTENSION;
   STRING;
   TUNABLE = AT_CREATION;
   DESCRIPTION = "La ruta al directorio de configuración";
}

# Valor de tiempo de espera en segundos antes de declarar que el
# análisis no ha sido satisfactorio.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 30;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Valor del tiempo de espera del análisis (segundos)";
}
```

Método `Start`

RGM invoca el método `Start` en un nodo del clúster cuando el grupo de recursos que contiene el recurso de servicio de datos se pone en línea en ese nodo o cuando se habilita el recurso. En la aplicación de ejemplo, el método `Start` activa el daemon `in.named` (DNS) en ese nodo.

EJEMPLO B–2 Método `dns_svc_start`

```
#!/bin/ksh
#
# Método Start para HA-DNS.
```

EJEMPLO B–2 Método `dns_svc_start` *(Continuación)*

```
#
# Este método inicia el servicio de datos bajo control de PMF. Antes de
# empezar el proceso in.named para DNS, realiza ciertas verificaciones de
# estado. La etiqueta de PMF para el servicio de datos es $RESOURCE_NAME.named.
# PMF intenta iniciar el servicio un número especificado de veces (Retry_count)
# y si el número de intentos supera este valor en un intervalo especificado
# (Retry_interval) PMF informa de que se ha producido un fallo al iniciar
# el servicio. Retry_count y Retry_interval son propiedades del recurso
# establecido en el archivo RTR.


#pragma ident   "@(#)dns_svc_start   1.1   00/05/24 SMI"

################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nombre del recurso de DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que se configura
                        # el recurso.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    "ERROR: Opción $OPTARG desconocida"
                     exit 1
                     ;;

                esac
        done

}
```

EJEMPLO B–2 Método `dns_svc_start` *(Continuación)*

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

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

# Obtener el recurso syslog que hay que utilizar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

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

# Obtener el valor de la propiedad Confdir del recurso para iniciar DNS.
# Con el nombre de recurso y el grupo de recursos introducidos, buscar
# el valor de Confdir fijado por el administrador de clúster al agregar el
# recurso.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get devuelve el "tipo" y el "valor" de las propiedades de
# extensión. Obtener sólo el valor de la propiedad de extensión.
CONFIG_DIR=`echo $config_info | awk `{print $2}'`

# Comprobar si $CONFIG_DIR es accesible.
if [ ! -d $CONFIG_DIR ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} Directorio $CONFIG_DIR falta o no está montado"
   exit 1
fi

# Cambiar al directorio $CONFIG_DIR en caso de que haya nombres de ruta
# relativos en los archivos de datos.
cd $CONFIG_DIR

# Comprobar que el archivo named.conf esté presente en el directorio $CONFIG_DIR.
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} El archivo $CONFIG_DIR/named.conf falta o está vacío"
   exit 1
fi

# Obtener el valor de Retry_count del archivo RTR.
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME
-G \ $RESOURCEGROUP_NAMÈ

# Obtener el valor de Retry_interval del archivo RTR. Convertir este valor, que
# está en segundos, a minutos para pasarlo a pmfadm. Observar que es una
# conversión con redondeo, por ejemplo, 50 segundos se redondea a un minuto.
((RETRY_INTRVAL = `scha_resource_get -O Retry_Interval
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ / 60))
```

EJEMPLO B–2 Método dns_svc_start *(Continuación)*

```
# Iniciar el daemon in.named bajo el control del PMF. Permitir una caída y reiniciar
# hasta $RETRY_COUNT veces en un periodo de $RETRY_INTERVAL; si se cae más a menudo,
# PMF dejará de intentar reiniciarlo. Si  hay un proceso ya registrado con la etiqueta
# <$PMF_TAG>, PMF enviará un mensaje de alerta indicando que el proceso ya está en
# ejecución.
echo "Retry interval is "$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Registrar un mensaje que indique que se ha iniciado HA-DNS.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
           "${ARGV0} HA-DNS iniciado satisfactoriamente"
fi
exit 0
```

Método Stop

El método Stop se invoca en un nodo del clúster cuando el grupo de recursos que contiene el recurso HA-DNS se pone fuera de línea en ese nodo o se inhabilita el recurso. Este método detiene el daemon in.named (DNS) en ese nodo.

EJEMPLO B–3 Método dns_svc_stop

```
#!/bin/ksh
#
# Método Stop para HA-DNS
#
# Detener el servicio de datos con PMF. Si el servicio no está en ejecución
# el método sale con estado 0, porque devolver otro valor pone el recurso
# en estado STOP_FAILED.


#pragma ident   "@(#)dns_svc_stop   1.1   00/05/24 SMI"

################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
```

EJEMPLO B–3 Método `dns_svc_stop` *(Continuación)*

```
                R)
                        # Nombre del recurso de DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que el recurso
                        # está configurado.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    "ERROR: Opción $OPTARG desconocida"
                     exit 1
                     ;;

                esac
        done

}


###############################################################################
# MAIN
#
###############################################################################

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

# Obtener el recurso syslog para utilizar mensajes de registro.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

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

# Obtener el valor Stop_timeout del archivo RTR.
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME
-G \ $RESOURCEGROUP_NAMÈ

# Intentar detener el servicio de datos de forma ordenada con una señal SIGTERM
# mediante PMF. Esperar un 80% del valor de Stop_timeout para ver si SIGTERM
# detiene satisfactoriamente el servicio de datos. Si no es así, enviar SIGKILL
# para detener el servicio de datos. Utilizar hasta un 15% del valor de Stop_timeout
# para ver si SIGKILL tiene éxito. Si no, hay un fallo y el método sale con estado
```

EJEMPLO B–3 Método `dns_svc_stop` *(Continuación)*

```
# diferente de cero. El 5% restante del valor Stop_timeout es para otros usos.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

# Consultar si in.named está en ejecución; si es así, terminarlo.
if pmfadm -q $PMF_TAG.named; then
   # Enviar una señal SIGTERM al servicio de datos y esperar un 80% del
   # valor total del tiempo de espera.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
          "${ARGV0} No se ha podido detener HA-DNS
          con SIGTERM; Reintertarlo con \
          SIGKILL"

      # Dado que el servicio de datos no se ha detenido con una señal SIGTERM,
      # usar SIGKILL ahora y esperar otro 15% del valor del tiempo de espera total.
      pmfadm -s $PMF_TAG.named -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [SYSLOG_TAG]
\
          "${ARGV0} No se ha podido detener HA-DNS;
          Salida NO SATISFACTORIA"

          exit 1
      fi
fi
else
   # El servicio de datos no se está ejecutando actualmente. Registrar
   # un mensaje y salir con resultado satisfactorio.
   logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
           "HA-DNS no se ha iniciado"

   # Aunque HA-DNS no esté en ejecución, salir con resultado satisfactorio
   # para no poner el recurso del servicio de datos en estado STOP_FAILED.

   exit 0

fi

# Se ha podido detener satisfactoriamente DNS. Registrar un mensaje y salir
# con resultado satisfactorio.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS se ha detenido satisfactoriamente"
exit 0
```

Utilidad gettime

La utilidad gettime es un programa en C utilizado por el programa PROBE para realizar un seguimiento del tiempo que pasa entre los reinicios del analizador. Debe compilar este programa y colocarlo en el mismo directorio que los métodos de rellamada, es decir, el directorio al que apunta la propiedad RT_basedir.

EJEMPLO B–4 Programa de utilidad gettime.c

```
#
# Este programa, utilizado por el método de análisis del servicio de datos,
# realiza un seguimiento del tiempo transcurrido, en segundos, desde un
# punto de referencia conocido (hito). Debe compilarse y situarse en el mismo
# directorio que los métodos de rellamada de servicios de datos (RT_basedir).


#pragma ident   "@(#)gettime.c   1.1   00/05/24 SMI"


#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
{
    printf("%d\n", time(0));
    exit(0);
}
```

Programa PROBE

El programa PROBE comprueba la disponibilidad del servicio de datos con los comandos nslookup(1M). El método de rellamada Monitor_start inicia este programa y el método de rellamada Monitor_start lo detiene.

EJEMPLO B–5 Programa dns_probe

```
#!/bin/ksh
#pragma ident   "@(#)dns_probe   1.1   00/04/19 SMI"
#
# Método Probe de HA-DNS.
#
# Este programa comprueba la disponibilidad del servicio de datos con nslookup, que
# consulta al servidor de DNS para que busque el propio servidor DNS. Si éste no
# responde o si la consulta la responde otro servidor, el análisis concluye que hay un
```

EJEMPLO B–5 Programa `dns_probe` *(Continuación)*

```
# problema con el servicio de datos y realiza una operación de recuperación de
# fallos  del servicio a otro nodo del clúster. El análisis se realiza en
# intervalos específicos, determinados por THOROUGH_PROBE_INTERVAL en el archivo RTR.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"


################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nombre del recurso de DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que está
                        # configurado el recurso.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    "ERROR: Opción $OPTARG desconocida"
                     exit 1
                     ;;

                esac
        done

}


################################################################################
# restart_service ()
#
# Esta función intenta reiniciar el servicio de datos invocando el método Stop,
# y el método Start después, del servicio de datos. Si el servicio de datos
# ya se ha terminado y no se ha registrado ninguna etiqueta para el servicio
# de datos en PMF, esta función realiza una operación de recuperación de fallos
# del servicio a otro nodo del clúster.
```

EJEMPLO B–5 Programa dns_probe *(Continuación)*

```
#
function restart_service
{

        # Para reiniciar el servicio de datos, verificar primero que éste
        # siga registrado bajo PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Dado que TAG para el servicio de datos sigue registrado
                # en PMF, detener el servicio de datos y volver a iniciarlo.

                # Obtener el nombre del método Stop y el valor de STOP_TIMEOUT
                # para este recurso.
                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} Método de parada no satisfactorio."
                        return 1
                fi

                # Obtener el nombre del método Start y el valor de START_TIMEOUT
                # para este recurso.
                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} Método de inicio no satisfactorio."
                        return 1
                fi
```

EJEMPLO B–5 Programa `dns_probe` *(Continuación)*

```
        else
                # La ausencia de TAG del servicio de datos
                # implica que el servicio de datos ya ha superado
                # el número máximo de reintentos permitidos en PMF.
                # No intentar reiniciar el servicio de datos
                # otra vez; intentar realizar una operación de
                # recuperación de fallos a otro nodo del clúster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME
\
                        -R $RESOURCE_NAME
        fi

        return 0
}



################################################################################
# decide_restart_or_failover ()
#
# Esta función decide la acción que hay que realizar cuando se produce el fallo
# de un analizador: reiniciar el servicio de datos localmente o realizar una
# recuperación de fallos a otro nodo del clúster.
#
function decide_restart_or_failover
{

   # Comprobar si es el primer intento de reinicio.
   if [ $retries -eq 0 ]; then
         # Es el primer fallo. Observar la hora del
         # primer intento.
         start_time=`$RT_BASEDIR/gettimè
         retries=`expr $retries + 1`
         # Dado que es el primer fallo, intentar reiniciar
         # el servicio de datos.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} No se ha podido iniciar el
                servicio de datos."
            exit 1
         fi
   else
      # No es el primer fallo
      current_time=`$RT_BASEDIR/gettimè
      time_diff=`expr $current_time - $start_timè
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # Este fallo se ha producido tras la ventana
         # de tiempo. Poner a cero el contador de reintentos,
         # deslizar la ventana y realizar un reintento.
         retries=1
```

EJEMPLO B–5 Programa dns_probe *(Continuación)*

```
         start_time=$current_time
         # Dado que el fallo anterior se produjo hace más de
         # Retry_interval, intentar reiniciar el servicio de datos.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG
                "${ARGV0} No se ha podido reiniciar HA-DNS."
            exit 1
         fi
      elif [ $retries -ge $RETRY_COUNT ]; then
         # Aún dentro de la ventana de tiempo,
         # y el contador de reintentos vencido, así que se
         # realiza una recuperación de fallos.
         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} Intento de recuperación de fallos
                no satisfactorio."
            exit 1
         fi
      else
         # Aún dentro de la ventana de tiempo,
         # y el contador de reintentos no ha vencido,
         # así que hacer otro reintento.
         retries=`expr $retries + 1`
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} No se ha podido reiniciar HA-DNS."
            exit 1
         fi
      fi
fi
}


################################################################################
# MAIN
################################################################################

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

# Obtener el recurso syslog que se debe usar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
```

EJEMPLO B–5 Programa `dns_probe` *(Continuación)*

```
# El intervalo al que se debe realizar el análisis se especifica en la propiedad
# THOROUGH_PROBE_INTERVAL. Obtener el valor de esta propiedad con scha_resource_get
PROBE_INTERVAL=`scha_resource_get -O THOROUGH_PROBE_INTERVAL
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtener el valor de tiempo de espera permitido para el análisis, establecido en
# la propiedad de extensión PROBE_TIMEOUT en el archivo RTR. El valor de tiempo
# de espera predeterminado para nslookup es 1,5 minutos.
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}'`

# Identificar el servidor en el que DNS está sirviendo para obtener el valor de
# la propiedad NETWORK_RESOURCES_USED del recurso.
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R
$RESOURCE_NAME -G \$RESOURCEGROUP_NAMÈ

# Obtener el valor de recuento de reintentos de la propiedad definida por el
# sistema Retry_count
RETRY_COUNT=`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# Obtener el valor del intervalo de reintentos de la propiedad definida por el
# sistema Retry_interval
RETRY_INTERVAL=`scha_resource_get -O RETRY_INTERVAL -R
$RESOURCE_NAME -G \$RESOURCEGROUP_NAMÈ

# Obtener la ruta completa para la utilidad gettime desde la propiedad RT_basedir
# del tipo de recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# El análisis se ejecuta en un bucle infinito, probando los comandos nslookup.
# Establecer un archivo temporal para las respuestas de nslookup.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
   # El intervalo al que debe ejecutarse el analizador se especifica en la
   # propiedad THOROUGH_PROBE_INTERVAL. Por tanto, poner el análisis
   # en reposo durante el tiempo que indica <THOROUGH_PROBE_INTERVAL>
   sleep $PROBE_INTERVAL

   # Ejecutar el análisis, que consulta la dirección IP en
   # la que sirve el DNS.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST
\
> $DNSPROBEFILE 2>&1

   retcode=$?
```

EJEMPLO B–5 Programa `dns_probe` *(Continuación)*

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

   # Asegurarse de que la respuesta a nslookup provenga del servidor
   # de HA-DNS y no de otro servidor de nombres mencionado en el
   # archivo /etc/resolv.conf.
   if [ $probefail -eq 0 ]; then
   # Obtener el nombre del servidor que ha respondido a la consulta de nslookup.
                   SERVER=` awk ` $1=="Server:" {
print $2 }' \
                   $DNSPROBEFILE | awk -F. ` { print $1 } ` `
                if [ -z "$SERVER" ];
then
                        probefail=1
                else
                        if [ $SERVER != $DNS_HOST ]; then
                                probefail=1
                        fi
                fi
        fi

   # Si no se establece la variable probefail en 0, el comando nslookup
   # ha agotado el tiempo de espera o la respuesta a la consulta provino de otro
   # (especificado en el archivo /etc/resolv.conf). En cualquier caso, el servidor
   # de DNS no responde y el método invoca decide_restart_or_failover, que evalúa si
   # hay que reiniciar el servicio de datos o realizar un recuperación de fallos
   # a otro nodo.

   if [ $probefail -ne 0 ]; then
         decide_restart_or_failover
   else
         logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
         "${ARGV0} Análisis del recurso HA-DNS satisfactorio"
   fi
done
```

Método `Monitor_start`

Este método inicia el programa `PROBE` del servicio de datos.

EJEMPLO B–6 Método `dns_monitor_start`

```
#!/bin/ksh
#
# Método Monitor start para HA-DNS.
#
```

EJEMPLO B–6 Método dns_monitor_start *(Continuación)*

```
# Este método inicia el supervisor (analizador) del servicio de datos
# controlado por PMF. El supervisor es un proceso que analiza el servicio de
# datos a intervalos periódicos y, si hay un problema, lo reinicia en el mismo
# nodo o realiza una operación de recuperación de fallos a otro nodo del clúster.
# La etiqueta de PMF para el supervisor es $RESOURCE_NAME.monitor.


#pragma ident   "@(#)dns_monitor_start   1.1   00/05/24 SMI"

##############################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nombre del recurso DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que el
                        # recurso se ha configurado.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
          logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                     "ERROR: Opción $OPTARG desconocida"
                      exit 1
                      ;;
                esac
        done

}



##############################################################################
# MAIN
#
##############################################################################

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

EJEMPLO B–6 Método dns_monitor_start *(Continuación)*

```
# Obtener el recurso syslog que hay que usar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

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

# Buscar dónde reside el método de análisis mediante la obtención
# del valor de la propiedad RT_BASEDIR del recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G \$RESOURCEGROUP_NAMÈ

# Iniciar el análisis del servicio de datos en PMF. Utilizar la opción de
# reintentos infinitos para iniciar el análisis. Pasar el nombre de recurso,
# tipo y grupo al programa de análisis.
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
\
    -T $RESOURCETYPE_NAME

# Registrar un mensaje que indique que el supervisor de HA-DNS se ha iniciado.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} Se ha iniciado el supervisor para HA-DNS"
fi
exit 0
```

Método Monitor_stop

Este método detiene el programa PROBE del servicio de datos.

EJEMPLO B–7 Método dns_monitor_stop

```
#!/bin/ksh
#
# Método Monitor stop para HA-DNS
#
# Detiene el supervisor que se está ejecutando con PMF.


#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"


###############################################################################
```

EJEMPLO B–7 Método dns_monitor_stop *(Continuación)*

```
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nombre del recurso de DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que el
                        # recurso se ha configurado.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    "ERROR: Opción $OPTARG desconocida"
                     exit 1
                     ;;

                esac
        done

}


###############################################################################
# MAIN
#
###############################################################################

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

# Obtener el recurso syslog que hay que utilizar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

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

EJEMPLO B–7 Método dns_monitor_stop *(Continuación)*

```
# Consultar si el supervisor está en ejecución. Si es así, terminarlo.
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} No se puede detener el supervisor del recurso" \
          $RESOURCE_NAME
           exit 1
   else
      # Se ha detenido satisfactoriamente el supervisor. Registrar un mensaje.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
          "${ARGV0} Supervisor del recurso " $RESOURCE_NAME
\
          " detenido satisfactoriamente"
   fi
fi

exit 0
```

Método Monitor_check

Este método verifica la existencia del directorio al que señala la propiedad Confdir. RGM invoca Monitor_check cuando el método PROBE realiza una operación de recuperación de fallos del servicio de datos a un nodo nuevo y para comprobar los nodos que son maestros potenciales.

EJEMPLO B–8 Método dns_monitor_check

```
#!/bin/ksh
#
# Método Monitor check para DNS.
#
# RGM invoca este método cuando el supervisor de fallos realice una recuperación
# de fallos del servicio de datos a un nodo nuevo. Monitor_check invoca el método
# Validate para comprobar si el directorio y los archivos de configuración están
# disponibles en el nuevo nodo.


#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
   typeset opt
```

EJEMPLO B–8 Método dns_monitor_check *(Continuación)*

```
   while getopts `R:G:T:' opt
   do
      case "$opt" in

      R)
      # Nombre del recurso DNS.
      RESOURCE_NAME=$OPTARG
      ;;

      G)
      # Nombre del grupo de recursos en el que se ha configurado
      # el recurso.
      RESOURCEGROUP_NAME=$OPTARG
      ;;

      T)
      # Nombre del tipo de recurso.
      RESOURCETYPE_NAME=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
      "ERROR: Opción $OPTARG desconocida"
      exit 1
      ;;

      esac
   done

}

###############################################################################
# MAIN
###############################################################################

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

# Obtener el recurso syslog que hay que utilizar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método.
parse_args "$@"

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

# Obtener la ruta completa del método Validate desde la
# propiedad RT_BASEDIR del tipo de recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
\
   -G $RESOURCEGROUP_NAMÈ
```

EJEMPLO B–8 Método `dns_monitor_check` *(Continuación)*

```
# Obtener el nombre del método Validate para este recurso.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAMÈ

# Obtener el valor de la propiedad Confdir para iniciar el servicio de
# datos. Utilizar el nombre de recurso y grupo de recursos introducidos
# para obtener el valor Confdir establecido al agregar el recurso.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get devuelve el tipo y el valor para las propiedades de
# extensión. Usar awk para obtener sólo el valor de la propiedad de extensión.
CONFIG_DIR=`echo $config_info | awk `{print $2}'`

# Invocar el método Validate para que el servicio de datos se pueda
# recuperar de un fallo a otro nodo satisfactoriamente.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME
\
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

# Registrar un mensaje que indique que la comprobación del supervisor
# ha sido satisfactoria.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
      "${ARGV0} Comprobación del supervisor para DNS satisfactoria."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Comprobación del supervisor para DNS no satisfactoria."
   exit 1
fi
```

Método `Validate`

Este método verifica la existencia del directorio al que señala la propiedad `Confdir`. RGM invoca este método cuando se crea el servicio de datos y cuando el administrador del clúster actualiza las propiedades del servicio de datos. El método `Monitor_check` invoca este método cuando el supervisor de fallos realiza una recuperación de fallos del servicio de datos a otro nodo.

EJEMPLO B–9 Método `dns_validate`

```
#!/bin/ksh
#
# Método Validate para HA-DNS.
```

EJEMPLO B–9 Método `dns_validate` *(Continuación)*

```
# Este método valida la propiedad Confdir del recurso. El método Validate
# se invoca en dos situaciones: cuando el recurso se está creando y cuando
# se está actualizando una propiedad de recurso. Cuando el recurso se está
# creando, este método se invoca con el indicador -c y todas las propiedades
# definidas por el sistema y de extensión se pasan como argumentos de línea
# de comandos. Cuando se está actualizando una propiedad de recurso, el método
# Validate se invoca con el indicador -u, y sólo el par propiedad-valor de la
# propiedad que se está actualizando se pasa como argumento de línea de comandos.
#
# ej.: cuando el recurso se está creando, los argumentos de comandos serán
#
# dns_validate -c -R <..> -G <...> -T <..> -r <prop-def-sis=valor>...
# -x <prop-extensión=valor>.... -g <prop-grupo-recurso=valor>....
#
# cuando la propiedad de recurso se está actualizando
#
# dns_validate -u -R <..> -G <...> -T <..> -r <prop-sis_actualizando=valor>
#   O
# dns_validate -u -R <..> -G <...> -T <..> -x <prop-exten_actualizando=valor>
#


#pragma ident   "@(#)dns_validate   1.1   00/05/24 SMI"

################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
   typeset opt

   while getopts `cur:x:g:R:T:G:' opt
   do
               case "$opt" in
               R)
                       # Nombre del recurso de DNS.
                       RESOURCE_NAME=$OPTARG
                       ;;
               G)
                       # Nombre del grupo de recursos en el que se ha
                       # configurado el recurso.
                       RESOURCEGROUP_NAME=$OPTARG
                       ;;
               T)
                       # Nombre del tipo de recurso.
                       RESOURCETYPE_NAME=$OPTARG
                       ;;

               r)
                       # El método no está accediendo a ninguna propiedad
                       # definida por el sistema, por lo que no tiene ninguna
                       # operación.
                       ;;
```

EJEMPLO B–9 Método `dns_validate` *(Continuación)*

```
                g)
                        # El método no está accediendo a ninguna propiedad de
                        # grupo de recursos, por lo que no tiene operación.
                        ;;

                c)
                        # Indica que se está invocando el método Validate mientras
                        # se crea el recurso, por lo que el indicador
                        # no tiene ninguna operación.
                        ;;

                u)
                        # Indica la actualización de una propiedad cuando el recurso
                        # ya existe. Si la actualización es de la propiedad Confdir,
                        # Confdir debería aparecer en los argumentos de línea de
                        # comandos. En caso contrario, el método debe buscarlo
                        # específicamente con scha_resource_get.
                        UPDATE_PROPERTY=1
                        ;;

                x)
                        # Lista de propiedades de extensión. Separar los pares de
                        # propiedad valor con "=" como separador.
                        PROPERTY=`echo $OPTARG | awk -F= `{print $1}'`
                        VAL=`echo $OPTARG | awk -F= `{print $2}'`

                        # Si la propiedad de extensión Confdir se encuentra en la
                        # línea de comandos, anotar su valor.
                        if [ $PROPERTY == "Confdir" ];
                        then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                        fi
                        ;;

                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        "ERROR: Opción $OPTARG desconocida"
                        exit 1
                        ;;
                esac
   done
}

##############################################################################
# MAIN
#
##############################################################################

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

EJEMPLO B–9 Método `dns_validate` *(Continuación)*

```
# Obtener el recurso syslog que hay que usar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Establecer el valor de CONFDIR en null. Más tarde, este método recupera el
# valor de la propiedad Confdir de la línea de comandos o con scha_resource_get.
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0

# Analizar los argumentos que se han pasado a este método.
parse_args "$@"

# Si el método Validate se invoca por una actualización de propiedades
# intentar recuperar el valor de la propiedad de extensión Confdir de línea de
# comandos. Si no, obtener el valor de Confdir con 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

# Verificar que la propiedad Confdir tiene un valor. En caso contrario,
# hay un fallo y sale con estado 1.
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
       "${ARGV0} Método de validar el recurso
       "$RESOURCE_NAME " no satisfactorio"
   exit 1
fi

# Validar ahora el valor real de la propiedad Confdir.

# Comprobar si $CONFDIR es accesible.
if [ ! -d $CONFDIR ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]\
            "${ARGV0} Directorio $CONFDIR falta o no está montado"
        exit 1
fi

# Comprobar que el archivo named.conf esté presente en el directorio Confdir.
if [ ! -s $CONFDIR/named.conf ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
\
            "${ARGV0} El archivo $CONFDIR/named.conf falta o está vacío"
        exit 1
fi

# Registrar un mensaje que indique que el método Validate ha sido satisfactorio.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
   "${ARGV0} Método de validar para el recurso
   "$RESOURCE_NAME \ se ha completado
```

EJEMPLO B–9 Método `dns_validate` *(Continuación)*

```
   "satisfactoriamente"

exit 0
```

Método `Update`

RGM invoca el método `Update` para notificar a un recurso en ejecución que se han modificado sus propiedades.

EJEMPLO B–10 Método `dns_update`

```
#!/bin/ksh
#
# Método Update para HA-DNS.
#
# Las actualizaciones reales a las propiedades las realiza RGM. Las
# actualizaciones sólo afectan al supervisor de fallos, por lo que este método
# debe reiniciar el supervisor de fallos.


#pragma ident   "@(#)dns_update   1.1   00/05/24 SMI"

################################################################################
# Analizar argumentos de programa.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts `R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Nombre del recurso DNS.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Nombre del grupo de recursos en el que el
                        # recurso se ha configurado.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Nombre del tipo de recurso.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
```

EJEMPLO B–10 Método `dns_update` *(Continuación)*

```
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME]
\
                    "ERROR: Opción $OPTARG desconocida"
                     exit 1
                     ;;

                esac
        done

}



#############################################################################
# MAIN
#
#############################################################################

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

# Obtener el recurso syslog que se debe utilizar para registrar mensajes.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Analizar los argumentos que se han pasado a este método
parse_args "$@"

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

# Descubrir dónde reside el método de análisis mediante la obtención del valor
# de la propiedad RT_BASEDIR del recurso.
RT_BASEDIR=`scha_resource_get -O RT_BASEDIR -R $RESOURCE_NAME
-G $RESOURCEGROUP_NAMÈ

# Cuando se invoca el método Update, RGM actualiza el valor de la propiedad que
# se está actualizando. Este método debe comprobar si el supervisor de fallos
# (analizador) está en ejecución y, si fuera así, terminarlo y reiniciarlo.
if pmfadm -q $PMF_TAG.monitor; then

   # Terminar el supervisor que se está ejecutando
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]
\
                    "${ARGV0} No se se puede detener el supervisor"
                exit 1
        else
                # Se ha podido detener satisfactoriamente el DNS.
                # Registrar un mensaje.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]
\
```

EJEMPLO B–10 Método `dns_update` *(Continuación)*

```
                    "Supervisor para HA-DNS detenido satisfactoriamente"
        fi

   # Reiniciar el supervisor.
   pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
      -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCETYPE_NAME
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG]\
                  "${ARGV0} No se ha podido reiniciar el supervisor para HA-DNS "
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG]\
                    "El supervisor para HA-DNS se ha reiniciado satisfactoriamente"

   fi
fi
exit 0
```

APÉNDICE **C**

Listado del código del tipo de recurso de ejemplo de la Biblioteca de desarrollo del servicio de datos

Este apéndice enumera el código completo de cada método del tipo de recurso `SUNW.xfnts`. Incluye la lista de `xfnts.c`, que contiene código para las subrutinas invocadas por los métodos de rellamada. Los listados del código de este apéndice son los siguientes.

`xfnts.c`

Este archivo implementa las subrutinas invocadas por los métodos `SUNW.xfnts`.

EJEMPLO C–1 `xfnts.c`

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts.c - Utilidades comunes de HA-XFS
 *
 * Esta utilidad tiene los métodos para realizar la validación, iniciar
 * y detener el servicio de datos y el supervisor de fallos. Contiene
 * también el método para analizar el estado del servicio de datos. El
 * analizador sólo devuelve satisfactorio o no satisfactorio. La acción se
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
 * decide en función de este valor devuelto en el método encontrado en
 * el archivo 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"

/*
 * El tiempo de espera inicial permitido para que el servicio de datos
 * HAXFS esté conectado y en ejecución. Se esperará un %
 * (SVC_WAIT_PCT) del tiempo start_timeout antes de analizar el
 * servicio.
 */
#define   SVC_WAIT_PCT      3

/*
 * Hay que usar 95% de probe_timeout para conectar al puerto y el
 * resto del tiempo se usa para desconectarse del puerto de la
 * función svc_probe.
 */
#define   SVC_CONNECT_TIMEOUT_PCT      95

/*
 * SVC_WAIT_TIME se usa sólo durante el inicio en svc_wait().
 * En svc_wait() hay que estar seguro de que el servicio esté activo
 * antes del retorno, por lo que hay que invocar a svc_probe() para
 * que supervise el servicio. SVC_WAIT_TIME es el tiempo que pasa
 * entre esos análisis.
 */

#define   SVC_WAIT_TIME      5

/*
 * Este valor se usará como tiempo de espera de desconexión, si no
 * queda tiempo de probe_timeout.
 */

#define   SVC_DISCONNECT_TIMEOUT_SECONDS      2
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
/*
 * svc_validate():
 *
 * Realizar una validación específica de HA-XFS de la configuración del
 * recurso.
 *
 * svc_validate comprobará:
 * 1. Propiedad de extensión Confdir_list
 * 2. Archivo fontserver.cfg
 * 3. Binario xfs
 * 4. Propiedad port_list
 * 5. Recursos de red
 * 6. Otras propiedades de extensión
 *
 * Si falla cualquiera de esas validaciones, devolver > 0; en caso
 * contrario, devolver 0 para indicar satisfactorio
 */

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;

   /*
    * Obtener el directorio de configuración del servicio de datos XFS
    * de la propiedad de extensión confdir_list.
    */
   confdirs = scds_get_ext_confdir_list(scds_handle);

   /* Devolver un error si no hay propiedad de extensión confdir_list*/
   if (confdirs == NULL || confdirs->array_cnt != 1) {
      scds_syslog(LOG_ERR,
          "Propiedad Confdir_list no establecida adecuadamante.");
      return (1); /* Fallo de validación */
   }

   /*
    * Construir la ruta al archivo de configuración desde la propiedad
    * de extensión confdir_list. Dado que HA-XFS sólo tiene una
    * configuración hay que usar la primera entrada de la propiedad
    * confdir_list.
    */
   (void) sprintf(xfnts_conf, "%s/fontserver.cfg",
confdirs->str_array[0]);

   /*
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
    * Comprobar si el archivo de configuración HA-XFS está en el lugar
    * correcto. Intentar acceder al archivo de configuración HA-XFS y
    * asegurarse de que se hayan configurado correctamente los
    * permisos
    */
   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suprimir error lint porque el prototipo errno.h
       * no tiene argumento vacío
       */
      scds_syslog(LOG_ERR,
          "No se ha podido acceder al archivo <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }

   /*
    * Asegurarse de que el binario xfs exista y de que los permisos
    * sean correctos. El binario XFS se supone que está en el sistema
    * local de archivos, no en el sistema global de archivos
    */
   if (stat("/usr/openwin/bin/xfs", &statbuf)
!= 0) {
      scds_syslog(LOG_ERR,
          "No se puede acceder al binario XFS: <%s> ",
strerror(errno));
      return (1);
   }

   /* HA-XFS sólo tendrán puerto */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se puede acceder a la propiedad Port_list: %s.",
         scds_error_string(err));
      return (1); /* Fallo de validación */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
      scds_syslog(LOG_ERR,
          "La propiedad Port_list sólo debe tener un valor.");
      scds_free_port_list(portlist);
      return (1); /* Fallo de validación */
   }
#endif

   /*
    * Devolver un error si hay un error al intentar obtener los recursos
    * de dirección de red disponibles para este recurso
    */
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
      != SCHA_ERR_NOERR) {
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
      scds_syslog(LOG_ERR,
          "No hay recursos de dirección de red en el grupo de recursos: %s.",
         scds_error_string(err));
      return (1); /* Fallo de validación */
   }

   /* Devolver un error si no hay recursos de dirección de red */
   if (snrlp == NULL || snrlp->num_netresources == 0) {
      scds_syslog(LOG_ERR,
          ";No hay recursos de dirección de red en el grupo de recursos.");
      rc = 1;
      goto finished;
   }

   /* Comprobar si hay otras propiedades de extensión importantes
    * establecidas */
   if (scds_get_ext_monitor_retry_count(scds_handle) <= 0)
{
      scds_syslog(LOG_ERR,
          "La propiedad Monitor_retry_count no está establecida.");
      rc = 1; /* Fallo de validación */
      goto finished;
   }
   if (scds_get_ext_monitor_retry_interval(scds_handle) <=
0) {
      scds_syslog(LOG_ERR,
          "La propiedad  Monitor_retry_interval no está establecida.");
      rc = 1; /* Fallo de validación */
      goto finished;
   }

   /* Todas las comprobaciones han sido satisfactorias */
   scds_syslog(LOG_INFO, "Validación satisfactoria.");
   rc = 0;

finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* devolver resultado de validación */
}


/*
 * svc_start():
 *
 * Iniciar el servidor de fuentes X
 * Devolver 0 si es satisfactorio, > 0 si hay fallos.
 *
 * El servicio XFS se iniciará al ejecutar el comando
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS se iniciará bajo PMF. XFS se iniciará como servicio de instancia
 * única. La etiqueta de PMF para el servicio de datos tendrá el formato
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
 * <nombre_grupo_recurso,nombre_recurso,número_instancia.svc>. En el caso
 * de XFS, dado que sólo habrá una instancia, instance_number de la etiqueta
 * tag será 0.
 */

int
svc_start(scds_handle_t scds_handle)
{
   char    xfnts_conf[SCDS_ARRAY_SIZE];
   char   cmd[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_port_list_t    *portlist;
   scha_err_t   err;

   /* obtener el directorio de configuración de la propiedad confdir_list */
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

   /* obtener el puerto que debe usar XFS de la propiedad Port_list */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se ha podido acceder a la propiedad Port_list.");
      return (1);
   }

   /*
    * Construir el comando para iniciar HA-XFS.
    * NOTA: el daemon de XFS imprime el siguiente mensaje mientras
    * detiene XFS "/usr/openwin/bin/xfs notice: terminating"
    * Para suprimir el mensaje del daemon, la salida se redirige a /dev/null.
    */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);

   /*
    * Iniciar HA-XFS bajo PMF. Observar que HA-XFS se inicia como un
    * servicio de instancia única. El último argumento de la función
    * scds_pmf_start denota el nivel de los secundario que hay que supervisar.
    * Un valor de -1 para este parámetro significa que todos los secundarios
    * y el proceso original deben supervisarse.
    */
   scds_syslog(LOG_INFO, "Emitir una solicitud de inicio.");
   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,
          "Comando de inicio completado satisfactoriamente.");
   } else {
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
      scds_syslog(LOG_ERR,
          "No se ha podido iniciar HA-XFS ");
   }

   scds_free_port_list(portlist);
   return (err); /* devolver estado satisfactorio/no satisfactorio */
}


/*
 * svc_stop():
 *
 * Detener el servidor XFS
 * Devolver 0 si es satisfactorio,> 0 si hay fallos.
 *
 * svc_stop detendrá el servidor, invocando la función toolkit:
 * scds_pmf_stop.
 */
int
svc_stop(scds_handle_t scds_handle)
{
   scha_err_t   err;

   /*
    * El valor de tiempo de espera necesario para que el método de parada
    * sea satisfactorio se define en la propiedad Stop_Timeout (definida
    * por el sistema)
    */
   scds_syslog(LOG_ERR, "Emitir una solicitud de parada.");
   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,
          "No se ha podido detener HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Se ha detenido satisfactoriamente HA-XFS.");
   return (SCHA_ERR_NOERR); /* Se ha detenido satisfactoriamente */
}

/*
 * svc_wait():
 *
 * esperar que el servicio de datos se inicie por completo y asegurarse de que
 * se ejecute correctamente
 */

int
svc_wait(scds_handle_t scds_handle)
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
{
   int rc, svc_start_timeout, probe_timeout;
   scds_netaddr_list_t   *netaddr;

   /* obtener el recurso de red que hay que usar para analizar */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
       "No se ha encontrado recurso de dirección de red en el grupo de recursos.");
      return (1);
   }

   /* Devolver un error si no hay recursos de red */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
      scds_syslog(LOG_ERR,
       "No se ha encontrado recurso de dirección de red en el grupo de recursos.");
      return (1);
   }

   /*
    * Obtener el tiempo de espera del método Start, número de puerto en
    * el que analizar, el valor de tiempo de espera de Probe
    */
   svc_start_timeout = scds_get_rs_start_timeout(scds_handle);
   probe_timeout = scds_get_ext_probe_timeout(scds_handle);

   /*
    * reposar durante un porcentaje de SVC_WAIT_PCT del tiempo
    * start_timeout antes de analizar efectivamente el servicio de datos.
    * Así se permite que el servicio de datos esté totalmente activo para
    * responder al análisis.
    * NOTA: el valor de SVC_WAIT_PCT puede variar según el servicio de datos.
    * En lugar de invocar sleep(), invocar scds_svc_wait() para que si el
    * servicio falla demasiadas veces, se deje de intentar y se dé antes
    * una respuesta.
    */
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
      return (1);
   }

   do {
      /*
       * analizar el servicio de datos en la dirección IP del
       * recurso de red y el nombre de puerto
       */
      rc = svc_probe(scds_handle,
          netaddr->netaddrs[0].hostname,
          netaddr->netaddrs[0].port_proto.port, probe_timeout);
      if (rc == SCHA_ERR_NOERR) {
         /* Satisfactorio. Liberar recursos y retornar */
         scds_free_netaddr_list(netaddr);
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
         return (0);
      }

      /*
       * El servicio de datos sigue intentando activarse. Reposar antes de
       * volver a analizar. En lugar de invocar sleep(), invocar scds_svc_wait()
       * para que si el servicio falla demasiadas veces, se deje de intentar
       * y se dé una respuesta antes.
       */
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
         return (1);
      }

   /* RGM debe agotar el tiempo de espera y terminar el programa */
   } while (1);

}

/*
 * Esta función inicia el supervisor de fallos para un recurso de HA-XFS.
 * Esto se realiza iniciando el análisis bajo PMF. La etiqueta PMF
 * se deriva como <RG-name,RS-name,instance_number.mon>.
 * La opción de reinicio de PMF se usa pero no el "reinicio infinito".
 * En su lugar, se  obtiene interval/retry_time del archivo RTR.
 */

int
mon_start(scds_handle_t scds_handle)
{
   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Llamar al metodo MONITOR_START del recurso <%s>.",
      scds_get_resource_name(scds_handle));

   /*
    * El analizador xfnts_probe se supone disponible en el mismo
    * subdirectorio en el que hay instalados otros métodos de rellamada
    * para el TR. El último parámetro de scds_pmf_start indica el nivel
    * de supervisión de secundarios. Dado que el análisis se inició bajo PMF,
    * hay que supervisar sólo el proceso de análisis, por lo que se usa
    * un valor de 0.
    */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe",
0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se ha podido iniciar el supervisor de fallos.");
      return (1);
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
   }

   scds_syslog(LOG_INFO,
       "Se ha iniciado el supervisor de fallos.");

   return (SCHA_ERR_NOERR); /* Monitor iniciado satisfactoriamente*/
}


/*
 * Esta función detiene el supervisor de fallos de un recurso HA-XFS.
 * Esto se hace a través de PMF. La etiqueta de PMF para el supervisor
 * de fallos se construye basándose en <RG-name_RS-name,instance_number.mon>.
 */

int
mon_stop(scds_handle_t scds_handle)
{

   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Llamar al método scds_pmf_stop");

   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,
          "No se ha podido detener el supervisor de fallos.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Se ha detenido el supervisor de fallos.");

   return (SCHA_ERR_NOERR); /* Supervisor detenido satisfactoriamente*/
}

/*
 * svc_probe(): realizar análisis específico de servicio de datos.
 * Devolver un valor flotante entre 0 (satisfactorio) y 100 (fallo total).
 *
 * El análisis realiza una conexión de zócalo simple al servidor XFS en el
 * puerto especificado, que se configura como la propiedad de extensión de
 * recurso (Port_list) y realiza un pulso de sondeo del servicio de datos.
 * Si el análisis no puede conectarse al puerto, se devuelve un valor de 100,
 * que indica que el fallo es total. Si la conexión se realiza y falla
 * la desconexión del puerto, se devuelve un valor de 50, que indica un
 * fallo parcial.
 *
 */
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
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;


   /*
    * Analizar el servicio de datos con una conexión de zócalo al puerto
    * especificado en la propiedad port_list del sistema que sirve al
    * servicio de datos XFS. Si el servicio XFS configurado para
    * recibir en el puerto especificado responde a la conexión, el
    * análisis es satisfactorio. En caso contrario, se esperará durante un
    * tiempo definido en la propiedad probe_timeout antes de concluir
    * que el análisis ha fallado.
    */

   /*
    * Utilizar el porcentaje SVC_CONNECT_TIMEOUT_PCT del tiempo de
    * espera para conectarse al puerto
    */
   connect_timeout = (SVC_CONNECT_TIMEOUT_PCT * timeout)/100;
   t1 = (hrtime_t)(gethrtime()/1E9);

   /*
    * el analizador establece una conexión al puerto y nombre de sistema
    * especificados. La conexión debe ocupar un 95% del probe_timeout real.
    */
   rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
       connect_timeout);
   if (rc) {
      scds_syslog(LOG_ERR,
          "No se ha podido conectar con el puero <%d> del recurso <%s>.",
          port, scds_get_resource_name(scds_handle));
      /* es un fallo total*/
      return (SCDS_PROBE_COMPLETE_FAILURE);
   }

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

   /*
    * Calcular el tiempo real que tardó la conexión. Debe ser inferior o igual
    * a connect_timeout, el tiempo asignado para la conexión. Si la conexión
    * emplea todo el tiempo que se le ha asignado, el valor restante de
    * probe_timeout pasado a esta función se utilizará como tiempo de espera
    * de desconexión. En caso contrario, el tiempo restante de la llamada de
    * conexión se agregará también al tiempo de espera de desconexión.
    *
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
 */

time_used = (int)(t2 - t1);

/*
 * Utilizar el tiempo restante (tiempo de espera - time_took_to_connect)
 * para la desconexión
 */

time_remaining = timeout - (int)time_used;

/*
 * Si se usa todo el tiempo, utilizar un tiempo de espera estático reducido
 * para intentar la desconexión. Así se evitan fugas de fd.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe ha utilizado el tiempo de espera completo de "
       "%d segundos durante la operacion de conexion y ha superado el "
       "tiempo de espera en %d segundos. Intentando desconectar con
       "un tiempo de espera de %d ",
       connect_timeout,
       abs(time_used),
       SVC_DISCONNECT_TIMEOUT_SECONDS);

   time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Devolver fallo parcial en caso de un fallo de desconexión.
 * Motivo: la llamada de conexión ha sido satisfactoria, lo que significa
 * que la aplicación está activa. Un fallo de desconexión puede
 * deberse a una aplicación bloqueada o a una carga pesada. Si es éste
 * el caso, no declarar la aplicación terminada devolviendo un fallo total.
 * Declarar un fallo parcial en su lugar. Si la situación persiste, la
 * llamada de desconexión fallará otra vez y se reiniciará la aplicación.
 */
rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
if (rc != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "No se ha podido desconectar el puerto %d del recurso %s.",
       port, scds_get_resource_name(scds_handle));
   /* es un fallo parcial */
   return (SCDS_PROBE_COMPLETE_FAILURE/2);
}

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

/*
 * Si no hay tiempo, no realizar la comprobación total con
 * fsinfo. Devolver SCDS_PROBE_COMPLETE_FAILURE/2. Así, si
```

EJEMPLO C–1 xfnts.c *(Continuación)*

```
     * persiste el tiempo de espera agotado, el servidor se reiniciará.
     */
    if (time_remaining <= 0) {
       scds_syslog(LOG_ERR, "Tiempo de espera de análisis excedido.");
       return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }

    /*
     * La conexión y desconexión del puerto son satisfactorias,
     * Ejecutar el comando fsinfo para realizar una comprobación total
     * del estado del servidor. Redirigir stdout, si no, la salida de
     * fsinfo terminará en la consola.
     */
    (void) sprintf(testcmd,
        "/usr/openwin/bin/fsinfo -server %s:%d> /dev/null",
        hostname, port);
    scds_syslog_debug(DBG_LEVEL_HIGH,
        "Comprobar el estado del servidor con %s.", testcmd);
    if (scds_timerun(scds_handle, testcmd, time_remaining,
       SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

       scds_syslog(LOG_ERR,
          "No se ha podido comprobar el estado del servidor con el comando <%s>",
          testcmd);
       return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }
    return (0);
}
```

Método xfnts_monitor_check

Este método verifica si la configuración básica del tipo de recurso es válida.

EJEMPLO C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_monitor_check.c - Método del comprobación del supervisor para HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"
```

EJEMPLO C–2 xfnts_monitor_check.c *(Continuación)*

```
/*
 * Realizar una comprobación de validez sencilla del servicio
 */

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

   /* Procesar los argumentos que pasa RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "método monitor_check "
       "se ha llamado y ha devuelto <%d>.", rc);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado del método Validate ejecutado en la
    * comprobación del supervisor */
   return (rc);
}
```

Método xfnts_monitor_start

Este método inicia el método xfnts_probe.

EJEMPLO C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_monitor_start.c - Método para iniciar el supervisor para HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"
```

EJEMPLO C–3 xfnts_monitor_start.c *(Continuación)*

```
#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Este método inicia el supervisor de fallos para el recurso HA-XFS.
 * Esto se hace iniciando el análisis bajo PMF. La etiqueta PMF se
 * deriva como RG-name,RS-name.mon. La opción de reinicio de PMF
 * se usa, pero no el "reinicio infinito". En su lugar,
 * se obtiene interval/retry_time del archivo RTR.
 */

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

   /* Procesar argumentos que pasa RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }

   rc = mon_start(scds_handle);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado del método monitor_start */
   return (rc);
}
```

Método xfnts_monitor_stop

Este método detiene el método xfnts_probe.

EJEMPLO C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_monitor_stop.c - Método para detener el supervisor para HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"
```

EJEMPLO C–4 xfnts_monitor_stop.c *(Continuación)*

```
#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Este método detiene el supervisor de fallos del recurso HA-XFS.
 * Esto se hace mediante PMF. La etiqueta PMF para el supervisor de
 * fallos se construye basándose en RG-name_RS-name.mon.
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int    rc;

   /* Procesar argumentos que pasa RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }
   rc = mon_stop(scds_handle);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado del método para detener el supervisor */
   return (rc);
}
```

Método xfnts_probe

El método xfnts_probe comprueba la disponibilidad de la aplicación y decide si debe realizar una operación de recuperación de fallos o reiniciar el servicio de datos. El método de rellamada xfnts_monitor_start inicia este programa y el método de rellamada xfnts_monitor_stop lo detiene.

EJEMPLO C–5 xfnts_probe.c+

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_probe.c - Análisis de HA-XFS
 */
```

EJEMPLO C–5 xfnts_probe.c+ *(Continuación)*

```
#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"


/*
 * main():
 * Sólo un bucle infinito que permanece en sleep()s durante un tiempo,
 * a la espera de que la secuencia de acciones de PMF lo interrumpa.
 * Cuando eso ocurre, invoca el método de inicio para que HA-XFS lo reinicie.
 *
 */

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, "No se ha podido inicializar el manejo.");
      return (1);
   }


   /* Obtener la dirección IP disponible para este recurso */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No hay recurso de dirección de red en el grupo de recursos.");
      scds_close(&scds_handle);
      return (1);
   }

   /* Devolver un error si no hay recursos de red */
   if (netaddr == NULL || netaddr->num_netaddrs == 0) {
```

EJEMPLO C–5 xfnts_probe.c+ *(Continuación)*

```
        scds_syslog(LOG_ERR,
            "No hay recurso de dirección de red en el grupo de recursos.");
        return (1);
    }


    /*
     * Establecer el tiempo de espera de las propiedades de X. Esto significa
     * que cada iteración de análisis recibirá un tiempo de espera completo
     * en cada recurso de red, sin necesidad de dividir el tiempo de espera
     * entre todos los recursos de red configurados para este recurso.
     */
    timeout = scds_get_ext_probe_timeout(scds_handle);

    for (;;) {

        /*
         * Reposar durante un tiempo thorough_probe_interval entre
         * análisis sucesivos.
         */
        (void) scds_fm_sleep(scds_handle,
            scds_get_rs_thorough_probe_interval(scds_handle));

        /*
         * Ahora, analizar todas las direcciones IP que se usan. Recorrer
         * 1. Todos los recursos de red que se emplean.
         * 2. Todas las direcciones IP de un recurso determinado.
         * Para cada dirección IP analizada, calcular
         * el historial de fallos.
         */
        probe_result = 0;
        /*
         * Repetir con todos los recursos para obtener todas
         * las direcciones IP que hay que usar para invocar svc_probe()
         */
        for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
            /*
             * Obtener el nombre de sistema y puerto en los que hay que
             * supervisar el estado.
             */
            hostname = netaddr->netaddrs[ip].hostname;
            port = netaddr->netaddrs[ip].port_proto.port;
            /*
             * HA-XFS sólo admite un puerto, por lo que
             * se debe obtener el valor del puerto de la
             * primera entrada de la matriz de puertos.
             */
            ht1 = gethrtime(); /* Bloquear la hora de inicio del análisis */
            scds_syslog(LOG_INFO, "Analizar el servicio de "
                "port: %d.", port);

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

EJEMPLO C–5 xfnts_probe.c+ *(Continuación)*

```
         /*
          * Actualizar el historial de análisis del servicio,
          * y actuar si es necesario.
          * Bloquear la hora de finalización del análisis.
          */
         ht2 = gethrtime();

         /* Convertir a milisegundos */
         dt = (ulong_t)((ht2 - ht1) / 1e6);

         /*
          * Calcular el historial de fallos y
          * actuar si es necesario
          */
         (void) scds_fm_action(scds_handle,
             probe_result, (long)dt);
      }   /* Todos los recursos de red */
   }    /* Analizar para siempre */
}
```

Método xfnts_start

RGM invoca el método Start en un nodo del clúster cuando el grupo de recursos que contiene el recurso de servicio de datos se pone en línea en ese nodo o cuando se habilita el recurso. El método xfnts_start activa el daemon xfs en ese nodo.

EJEMPLO C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_svc_start.c - Método de inicio para HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * El método de inicio de HA-XFS. Realiza comprobaciones de estado
 * en los valores de recurso e inicia el HA-XFS bajo PMF con una
 * secuencia de acciones.
 */
```

EJEMPLO C–6 xfnts_start.c *(Continuación)*

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

   /*
    * Procesar todos los argumentos que ha pasado de RGM
    * y realizar una inicialización para syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }

   /* Validar la configuración; si hay un error, volver atrás */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "No se ha podido validar la configuración.");
      return (rc);
   }

   /* Iniciar el servicio de datos; si falla, volver con un error */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

   /* Esperar que el servicio se inicie por completo */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Llamar a svc_wait para comprobar que el servicio se haya iniciado.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Devuelto desde svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "El servicio se ha iniciado satisfactoriamente.");
   } else {
      scds_syslog(LOG_ERR, "No se ha podido iniciar el servicio.");
   }


finished:
   /* Liberar los recursos de entorno asignados */
   scds_close(&scds_handle);

   return (rc);
}
```

El método `xfnts_stop`

RGM invoca el método `Stop` en un nodo del clúster cuando el grupo de recursos que contiene el recurso de HA-XFS se pone fuera de línea en ese nodo o cuando se inhabilita el recurso. Este método detiene el daemon `xfs` en ese nodo.

EJEMPLO C–7 `xfnts_stop.c`

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_svc_stop.c - Método de parada para HA-XFS
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Detiene el proceso de HA-XFS con PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

   /* Procesar los argumentos pasados por RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }

   rc = svc_stop(scds_handle);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado del método svc_stop */
   return (rc);
}
```

El método `xfnts_update`

RGM invoca el método `Update` para notificar a un recurso en ejecución que sus propiedades han cambiado. RGM invoca `Update` después de que una acción administrativa logre establecer satisfactoriamente las propiedades de un recurso o su grupo.

EJEMPLO C–8 `xfnts_update.c`

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_update.c - Método Update para HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Es posible que se hayan actualizado algunas propiedades del recurso.
 * Todas las propiedades actualizables y actualizadas están relacionadas
 * con el supervisor de fallos. Por tanto, debería bastar con reiniciar éste.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

   /* Procesar los argumentos que ha pasado RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }

   /*
    * Comprobar si el supervisor de fallos ya está en ejecución. Si es así,
    * detenerlo y reiniciarlo. El segundo parámetro de scds_pmf_restart_fm()
    * sólo identifica la instancia del supervisor de fallos que hay que
    * reiniciar.
    */

   scds_syslog(LOG_INFO, "Reinicio del supervisor de fallos.");
   result = scds_pmf_restart_fm(scds_handle, 0);
```

EJEMPLO C–8 xfnts_update.c *(Continuación)*

```
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "No se ha podido reiniciar el supervisor de fallos.");
      /* Liberar toda la memoria asignada por scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Terminado satisfactoriamente.");

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   return (0);
}
```

El listado del código del método xfnts_validate

Este método verifica la existencia del directorio al que apunta la propiedad Confdir_list. RGM invoca este método cuando se crea el servicio de datos y cuando el administrador del clúster actualiza las propiedades del servicio de datos. El método Monitor_check invoca este método cuando el supervisor de fallos realiza una operación de recuperación de fallos del servicio de datos a otro nodo.

EJEMPLO C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2003 de Sun Microsystems, Inc.
 * Reservados todos los derechos.
 *
 * xfnts_validate.c - Método de validación para HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Comprobar si las propiedades se han establecido correctamente.
 */

int
```

EJEMPLO C–9 xfnts_validate.c *(Continuación)*

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

   /* Procesar los argumentos que ha pasado RGM e inicializar syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "No se ha podido inicializar el manejo.");
      return (1);
   }
   rc = svc_validate(scds_handle);

   /* Liberar toda la memoria asignada por scds_initialize */
   scds_close(&scds_handle);

   /* Devolver el resultado del método de validación */
   return (rc);

}
```

APÉNDICE **D**

Valores y nombres válidos de RGM

Este apéndice enumera los requisitos de los caracteres válidos para los nombres y valores de RGM.

Nombres válidos de RGM

Los nombres de RGM se dividen en cinco categorías:

- Nombres de grupo de recursos
- Nombres de tipo de recurso
- Nombres de recurso
- Nombres de propiedad
- Nombres literales de enumeración

Todos los nombres deben cumplir las normas siguientes, salvo los nombres de tipo de recurso:

- Deben estar en ASCII.
- Deben empezar con una letra.
- Pueden contener letras mayúsculas y minúsculas, números, guiones (-) y guiones bajos (_).
- No deben superar los 255 caracteres.

Un nombre de tipo de recurso puede ser un nombre simple (especificado por la propiedad `Resource_type` del archivo RTR) o un nombre completo (especificado por las propiedades `Vendor_id` y `Resource_type` del archivo RTR). Cuando se especifican ambas propiedades, RGM introduce un punto entre `Vendor_id` y `Resource_type` para formar el nombre completo. Por ejemplo, si `Vendor_id=SUNW` y `Resource_type=ejemplo`, el nombre completo será `SUNW.ejemplo`. Éste es el único caso en el que el punto es un carácter admitido en un nombre de RGM.

Valores de RGM

Los valores de RGM se dividen en dos categorías: valores de propiedad y valores de descripción. Las reglas son las mismas para ambos, a saber:

- Los valores deben estar en ASCII.
- La longitud máxima de un valor es de 4 MB menos 1, es decir, 4.194.303 bytes.
- Los valores no pueden contener ninguno de los caracteres siguientes: nulo, línea nueva, coma o punto y coma.

APÉNDICE **E**

Requisitos para aplicaciones no habilitadas para el clúster

Una aplicación normal, no habilitada para el clúster, debe cumplir ciertos requisitos para poder adquirir una alta disponibilidad (HA). La sección «Análisis de la validez de la aplicación» en la página 29 enumera estos requisitos. Este apéndice proporciona detalles adicionales sobre elementos específicos de esta lista.

Una aplicación adquiere alta disponibilidad cuando la configuración de sus recursos se transforma en grupos de recursos. Los datos de la aplicación se sitúan en un sistema global de archivos de alta disponibilidad, lo que permite que los datos estén accesibles desde un servidor encendido en caso de que falle el otro. Consulte la información sobre los sistemas de archivos del clúster en *Sun Cluster 3.1: Guía de conceptos*.

Para que los clientes de la red dispongan de acceso, se debe configurar una dirección IP de red lógica en recursos de nombre lógico de servidor, contenidos en el mismo grupo de recursos que el recurso del servicio de datos. El recurso de servicio de datos y los recursos de dirección de red experimentan una recuperación de fallos simultáneamente, lo que hace que los clientes de red del servicio de datos accedan al recurso del servicio de red en su nuevo servidor.

Datos de sistemas múltiples

Los conjuntos del disco de sistemas globales de archivos de alta disponibilidad se alojan en diferentes sistemas para que, en caso de caída de un sistema físico, uno de los sistemas activos pueda acceder al disco. Para que una aplicación tenga una alta disponibilidad, sus datos deben también tenerla, por lo que deben residir en los sistemas globales de archivos de HA (alta disponibilidad).

El sistema global de archivos está montado en grupos de discos creados como entidades independientes. El usuario puede elegir utilizar algunos grupos de discos como sistemas globales de archivos montados y otros como dispositivos básicos para utilizarlos con un servicio de datos, como HA Oracle.

Una aplicación puede tener conmutadores de línea de comandos o archivos de configuración que apunten a la ubicación de los archivos de datos. Si la aplicación utiliza nombres de rutas invariables, es posible cambiar los nombres de éstas por vínculos simbólicos que señalen a archivos del sistema global de archivos, sin cambiar el código de aplicación. Consulte «Utilización de vínculos simbólicos para la colocación de datos de varios sistemas» en la página 320 para ver más detalles sobre la utilización de los vínculos simbólicos.

En el peor de los casos, el código fuente de la aplicación se debe modificar para proporcionar un mecanismo de direccionamiento a la ubicación real de los datos. Esto se puede hacer implementando conmutadores de líneas de comandos adicionales.

Sun Cluster admite la utilización de sistemas UFS de UNIX y dispositivos básicos de alta disponibilidad, configurados en un gestor de volúmenes. Durante la instalación y configuración, el administrador del sistema debe especificar qué recursos de disco se deben utilizar con los sistemas de archivos UFS y cuáles con los dispositivos básicos. Generalmente, los dispositivos básicos sólo los utilizan servidores de bases de datos y servidores multimedia.

Utilización de vínculos simbólicos para la colocación de datos de varios sistemas

En ocasiones, los nombres de ruta de los archivos de datos de una aplicación son invariables y no existe ningún mecanismo para anularlos. Para evitar modificar el código de la aplicación, en ocasiones se pueden usar enlaces simbólicos.

Por ejemplo, supongamos que la aplicación le asigna al archivo de datos el nombre de ruta invariable `/etc/mi_archivo_de_datos`. Esa ruta se puede modificar de un archivo a un enlace simbólico cuyo valor señale a un archivo de uno de los sistemas de archivos del servidor lógico. Por ejemplo, se puede establecer un enlace simbólico a `/global/phys-schost-2/mi_archivo_de_datos`.

Es posible que se produzca un problema si se utilizan así los enlaces simbólicos cuando la aplicación, o uno de sus procedimientos administrativos, modifique el nombre del archivo de datos y su contenido. Por ejemplo, supongamos que la aplicación realiza una actualización, creando primero un archivo temporal nuevo, `/etc/mi_archivo_de_datos.nuevo`. Después, renombra el archivo temporal para que tenga el nombre del archivo real, con la llamada al sistema `rename(2)` (o el programa `mv(1)`). Al crear el archivo temporal y darle el nombre del archivo real, el servicio de datos intentará garantizar que el contenido del archivo de datos siempre esté bien formado.

Desgraciadamente, la acción `rename(2)` acaba con el enlace simbólico. El nombre `/etc/mi_archivo_de_datos` ahora es un archivo regular y está en el mismo sistema de archivos que el directorio `/etc`, no en el sistema global de archivos del clúster. Dado que el sistema de archivos `/etc` es privado para cada sistema, los datos no estarán disponibles después de una operación de control o conmutación.

El problema subyacente en esta situación es que la aplicación no conoce la existencia del enlace simbólico y no se ha escrito teniendo presentes los enlaces simbólicos. Para utilizar éstos a fin de redirigir el acceso a los datos en los sistemas de archivos de servidores lógicos, la implementación de la aplicación debe comportarse de forma que no anule los enlaces simbólicos. Por tanto, éstos no resuelven totalmente el problema de situar los datos en los sistemas globales de archivos del clúster.

Nombres de sistema

Hay que determinar si el servicio de datos necesita conocer en algún momento el nombre de sistema del servidor en el que se está ejecutando. En caso afirmativo, es posible que haya que modificar el servicio de datos para que utilice un nombre lógico de servidor (es decir, un nombre de sistema configurado en un recurso de nombre lógico de servidor que resida en el mismo grupo de recursos que el recurso de aplicación), en lugar del nombre físico.

En ocasiones, en el protocolo cliente-servidor de un servicio de datos, el servidor devuelve su propio nombre del sistema al cliente, en el contenido de un mensaje. En esos protocolos, es posible que, para ponerse en contacto con el servidor, el cliente tenga que usar este nombre de sistema devuelto, el cual, para que resulte utilizable tras una operación de control o conmutación, debería ser un nombre lógico de servidor del grupo de recursos, no el nombre del servidor físico. En este caso, es necesario modificar el código del servicio de datos para devolver el nombre lógico de servidor al cliente.

Sistemas multienlace

La expresión *sistema multienlace* describe un sistema que se encuentra en más de una red pública. Un sistema así tiene varios nombres de sistema y direcciones IP. Tiene un par nombre de sistema-dirección IP para cada red. Sun Cluster está diseñado para permitir que un sistema aparezca en tantas redes como se desee, o en una sola (el caso de los sistemas que no tienen varios directorios iniciales). Al igual que el nombre físico de servidor tiene varios pares nombre de sistema-dirección IP, cada grupo de recursos

puede tener múltiples pares nombre de sistema-dirección IP, uno para cada red pública. Cuando Sun Cluster mueve un grupo de recursos de un servidor físico a otro, el juego completo de pares nombre de sistema-dirección IP de ese recurso se mueve también.

El conjunto de pares de nombre de sistema-dirección IP de un grupo de recursos se configura como recursos de nombre lógico de servidor contenidos en el grupo de recursos. Estos recursos de dirección de red los especifica el administrador del sistema cuando se crea y configura el grupo de recursos. La API del servicio de datos de Sun Cluster contiene recursos para consultar estos pares de nombre de sistema-dirección IP.

La mayoría de los daemons de servicios de datos ya preparados, que se han escrito para el entorno Solaris, gestionan adecuadamente los sistemas principales con varios directorios iniciales. Muchos servicios de datos realizan todas sus comunicaciones de red mediante el vínculo con la dirección comodín de Solaris `INADDR_ANY`. Esta vinculación hace, automáticamente, que los servicios de datos manejen todas las direcciones IP de todas las interfaces de red. `INADDR_ANY` se vincula efectivamente con todas las direcciones IP configuradas actualmente en la máquina. Un daemon de servicio de datos que utiliza `INADDR_ANY` generalmente no necesita cambios para manejar las direcciones de red lógicas de Sun Cluster.

Vinculación con `INADDR_ANY` frente a vinculación con direcciones IP específicas

Aún cuando se utilicen sistemas que no tengan varias direcciones iniciales, el concepto de dirección lógica de Sun Cluster permite que la máquina disponga de más de una dirección IP, una para su propio servidor físico y varias adicionales para cada recurso de dirección de red (nombre lógico de servidor) que controla actualmente. Cuando una máquina se convierte en maestro de un recurso de dirección de red, adquiere dinámicamente direcciones IP adicionales. Cuando renuncia a ser maestro de un recurso de dirección de red, dinámicamente cede las direcciones IP.

Algunos servicios de datos no pueden funcionar adecuadamente en un entorno Sun Cluster si están vinculados con `INADDR_ANY`. Estos servicios de datos deben modificar dinámicamente el conjunto de direcciones IP al que están vinculados, según si el grupo de recursos tiene un maestro o no. Una estrategia para lograr volver a establecer los vínculos consiste en hacer que los métodos de inicio y parada de esos servicios de datos terminen y reinicien los daemons del servicio de datos.

La propiedad del recurso `Network_resources_used` permite que el usuario final configure un conjunto específico de recursos de direcciones de red con los que se debe vincular el recurso de aplicación. Para los tipos de recursos que requieran esta función, la propiedad `Network_resources_used` debe declararse en el archivo RTR del tipo de recurso.

Cuando RGM pone el grupo de recursos en línea o fuera de línea, sigue un orden específico para conectar, desconectar, asignar y desasignar direcciones de red dependiendo de cuándo invoca los métodos de recurso del servicio de datos. Consulte «Elección de los métodos `Start` y `Stop` que se van a utilizar» en la página 46.

Para cuando el método `Stop` del servicio de datos retorna, éste debe haberse detenido con las direcciones de red del grupo de recursos. Del mismo modo, para cuando el método `Start` retorna, el servicio de datos debe haber empezado a utilizar las direcciones de red.

Si el servicio de datos se vincula con `INADDR_ANY` en lugar de con direcciones IP individuales, el orden en el que se invocan los métodos del recurso del servicio de datos y los métodos de la dirección de red no tiene importancia.

Si los métodos de inicio y parada del servicio de datos cumplen su labor, terminando y reiniciando los daemons del servicio de datos, el servicio de datos se detiene e inicia mediante las direcciones de red, en los momentos adecuados.

Reintento del cliente

Para un cliente de red, una operación de control o conmutación aparece como una caída del servidor lógico, seguida de un rearranque rápido. Idealmente, la aplicación del cliente y el protocolo cliente-servidor se estructuran para realizar un cierto número de reintentos. Si la aplicación y el protocolo se ocupan del caso de un único servidor que se desconecta y se reinicia, se encargarán también del caso de un grupo de recursos que está experimentando una operación de control o conmutación. Algunas aplicaciones optan por realizar un número infinito de reintentos. Las aplicaciones más complejas notifican al usuario que hay un reintento prolongado en curso y permiten que sea él quien elija si desea continuar.

APÉNDICE **F**

Definiciones del tipo de documento para CRNP

Este apéndice enumera las DTD (definiciones de tipo de documento) para el Protocolo de notificación de reconfiguración de clúster (CRNP).

DTD de XML SC_CALLBACK_REG

Nota – La estructura de datos NVPAIR que utilizan SC_CALLBACK_REG y SC_EVENT se define sólo una vez.

```
<!— especificación del formato XML de SC_CALLBACK_REG
        Copyright 2001-2003 Sun Microsystems, Inc. Reservados todos los derechos.
        Uso sujeto a los términos de la licencia.

        Uso esperado:

Un cliente de CRNP debería utilizar este formato XML para registrarse inicialmente
en el servicio, para registrarse o eliminar su registro de algunos eventos más
adelante o para borrarse por completo del servicio.

Un cliente se identifica de forma exclusiva por su puerto e IP de rellamada. El puerto
se define en el elemento SC_CALLBACK_REG y el IP se toma como el IP de origen de la
conexión de registro. El atributo final del elemento SC_CALLBACK_REG raíz es ADD_CLIENT,
ADD_EVENTS, REMOVE_CLIENT o REMOVE_EVENTS, según la forma del mensaje que esté utilizando
el cliente.

SC_CALLBACK_REG contiene 0 o varios subelementos SC_EVENT_REG.

Un SC_EVENT_REG es la especificación de un tipo de evento. Un cliente puede
especificar sólo CLASS (un atributo del elemento SC_EVENT_REG) o SUBCLASS (un
atributo opcional) para una mayor granularidad. Asimiso, SC_EVENT_REG
tiene como subelementos 0 o más NVPAIRs, que se pueden utilizar para especificar
```

```
aún más el evento.

Así, el cliente puede especificar eventos en la granularidad que desee. Observe que un
cliente no se puede registrar y no registrar para eventos del mismo mensaje. Sin
embargo, un cliente puede suscribirse al servicio y suscribirse para recibir eventos
del mismo mensaje.

Nota sobre la versión: el atributo VERSION de cada elemento raíz se marca como "fijo",
lo que significa que todos los mensajes que cumplan estas DTD deben tener el valor
de versión especificado. Si se crea una nueva versión del protocolo, las DTD revisadas
tendrán un valor nuevo para este atributo de VERSION fijo, de forma que todos los mensajes
de la nueva versión deben tener el número de la nueva versión.
—>

<!— definición de SC_CALLBACK_REG

El elemento raíz del documento XML es un mensaje de registro. Estos mensajes incluyen
el puerto de rellamada y la versión del protocolo como atributos y un atributo
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT o REMOVE_EVENTS que especifica el tipo de registro.
Los tipos ADD_CLIENT, ADD_EVENTS y REMOVE_EVENTS deben tener uno o varios subelementos
SC_EVENT_REG. REMOVE_CLIENT no debe especificar un subelemento SC_EVENT_REG.
        ATRIBUTOS:
               VERSION       La versión del protocolo de CRNP del mensaje.
               PORT          El puerto de rellamada.
               REG_TYPE      El tipo de registro. Uno de los siguientes:
                       ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENIDO:
SUBELEMENTS: SC_EVENT_REG (0 o más)
—>
<!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

>
<!— definición de SC_EVENT_REG

SC_EVENT_REG define un evento para el cual el cliente se está registrando o cancelando
el registro para recibir notificaciones de eventos. El registro puede ser para cualquier
nivel de granularidad, desde sólo una clase de evento hasta pares de nombre-valor
específicos que deben estar presentes. Así, el único atributo necesario es CLASS.
El atributo SUBCLASS y los subelementos NVPAIRS son opcionales, para una mayor
granularidad.

Los registros que especifican los pares de nombre-valor registran un interés en la
notificación de mensajes de la clase/subclase especificada con TODOS los pares de
nombre-valor presentes. Las cancelaciones de registro que especifican los pares de
nombre-valor anulan el registro para notificaciones que tienen PRECISAMENTE esos pares
de nombre-valor en una granularidad especificada anteriormente. Las cancelaciones de
registro que no especifican los pares nombre-valor anulan el registro de TODAS las
notificaciones de eventos de la clase y subclase especificada.

        ATRIBUTOS:
```

```
                CLASS:       La clase de evento en la que el elemento registra
                             o anula el registro.
                SUBCLASS:    La subclase del evento (opcional).

        CONTENIDO:
                SUBELEMENTS: 0 o varios NVPAIRs.
—>

<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA               #REQUIRED
       SUBCLASS       CDATA               #IMPLIED
>
```

DTD de XML de NVPAIR

```
<!— especificación de fomrato XML de NVPAIR

        Copyright 2001-2003 Sun Microsystems, Inc. Reservados todos los derechos.
        El uso está sujeto a los términos de la licencia.

        Uso esperado:
        Un elemento nvpair se debe usar en un elemento SC_EVENT o SC_CALLBACK_REG.

—>

<!— definición de NVPAIR

       NVPAIR es un par de nombre-valor que representa combinaciones arbitrarias
       de nombre y valor. Se pretende que sea una traducción directa y genérica de
       la estructura nvpair_t de Solaris, empleada por la estructura sysevent.
       Sin embargo, en el caso de este elemento de xml no hay información de tipo
       asociada al nombre o valor (ambos son textos arbitrarios).


       NVPAIR consta sólo de un elemento NAME y uno o varios elementos VALUE.
       Un elemento VALUE representa un valor escalar; varios, representan un VALUE
       de matriz.

       ATRIBUTOS:

       CONTENIDO:
               SUBELEMENTS: NAME(1), VALUE(1 o varios)
—>

<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— definición de NAME

        NAME es simplemente una cadena de longitud arbitraria.
```

```
        ATRIBUTOS:

        CONTENIDO:
                Datos de texto arbitrario.  Debería ir envuelto con <![CDATA[...]]>
                para evitar un análisis de XML interno.
—>
<!ELEMENT NAME (#PCDATA)>

<!— definición de VALUE
        VALUE es simplemente una cadena de longitud arbitraria.

        ATRIBUTOS:

        CONTENIDO:
                Datos de texto arbitrarios.  Debería estar envuelto con <![CDATA[...]]>
                para evitar análisis de XML internos.
—>

<!ELEMENT VALUE (#PCDATA)>
```

DTD de XML de SC_REPLY

```
<!— Especificación de formato XML de SC_REPLY

        Copyright 2001-2003 Sun Microsystems, Inc. Reservados todos los derechos.
        Uso sujeto a los términos de la licencia.
—>

<!— definición de SC_REPLY

        El elemento raíz del documento XML representa una respuesta a un mensaje.
        La respuesta contiene un código y un mensaje de estado.

        ATRIBUTOS:
                VERSION:        La versión de protocolo de CRNP del mensaje.
                STATUS_CODE:    El código de retorno del mensaje. Uno de los siguientes:
                                OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL, MALFORMED,
                                INVALID_XML, VERSION_TOO_HIGH o VERSION_TOO_LOW.

                CONTENIDO:
                        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
```

```
>

<!— definición de SC_STATUS_MSG
        SC_STATUS_MSG es simplemente una cadena de texto arbitrario que informa sobre
        el código de estado. Debería estar envuelta con <![CDATA[...]]> para evitar un
        análisis de XML interno.

        ATRIBUTOS:

        CONTENIDO:
                Cadena arbitraria.
—>

<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

DTD de XML de SC_EVENT

Nota – La estructura de datos NVPAIR que utilizan SC_CALLBACK_REG y SC_EVENT se define sólo una vez.

```
<!— Especificación de formato XML de SC_EVENT

        Copyright 2001-2003 Sun Microsystems, Inc. Reservados todos los derechos.
        Uso sujeto a los términos de la licencia.

        El elemento raíz del documento XML debe ser una traducción directa y genérica
        del formato de mensaje sysseventd de Solaris. Tiene atributos que representan
        la clase, subclase, fabricante y editor y contiene una cantidad variable de
        elementos NVPAIR.

        ATRIBUTOS:
                VERSION:      La versión del protocolo CRNP del mensaje.
                CLASS:        La clase sysevent del evento
                SUBCLASS:     La subclase del evento
                VENDOR:       El fabricante asociado al evento
                PUBLISHER:    El editor del evento
        CONTENIDO:
                SUBELEMENTS: NVPAIR (0 o varios)
—>

<!ELEMENT SC_EVENT (NVPAIR*)>
<!ATTLIST SC_EVENT
        VERSION         NMTOKEN         #FIXED "1.0"
        CLASS           CDATA           #REQUIRED
        SUBCLASS        CDATA           #REQUIRED
        VENDOR          CDATA           #REQUIRED
        PUBLISHER       CDATA           #REQUIRED
```

>

APÉNDICE **G**

Aplicación `CrnpClient.java`

Este apéndice muestra la aplicación `CrnpClient.java` completa, que se explica con más detalle en el Capítulo 12.

Contenido de `CrnpClient.java`

```
/*
 * CrnpClient.java
 * ================
 *
 * Nota sobre el análisis de XML:
 *
 * Este programa usa la arquitectura Java de Sun para las API de procesos de XML (JAXP).
 * Consulte http://java.sun.com/xml/jaxp/index.html para ver documentación sobre API y
 * obtener información sobre la disponibilidad.
 *
 * Este programa se ha escrito para Java 1.3.1 o superior.
 *
 * Información general del programa:
 *
 * El subproceso principal del programa crea un objeto CrnpClient, espera que el
 * usuario termine la demostración e invoca el cierre en el objeto CrnpClient
 * y sale del programa.
 *
 * El constructor de CrnpClient crea un objeto EventReceptionThread, abre una
 * conexión con el servidor CRNP (con el sistema y puerto especificados en la líne
 * a de comandos), construye un mensaje de registro (basado en las especificaciones
 * de la línea de comandos), envía el mensaje de registro y lee y analiza la respuesta.
 *
 * EventReceptionThread crea un zócalo de recepción vinculado al nombre de sistema
 * de la máquina en la que se ejecuta el programa y el puerto especificado en la línea
 * de comandos. Espera una rellamada de evento entrante y, cuando la recibe, construye
 * un documento XML desde el canal del zócalo de entrada, que se devuelve al objeto
```

```
 * CrnpClient para ser procesado.
 *
 * El método de apagado de CrnpClient se limita a enviar un mensaje de cancelación
 * de registro (REMOVE_CLIENT) SC_CALLBACK_REG al servidor crnp.
 *
 * Nota sobre el manejo de errores: en pro de la brevedad, este programa suele salir
 * cuando se producen la mayoría de los errores. Obviamente, una aplicación real
 * intentaría manejar algunos errores de formas diversas, por ejemplo, mediante
  * reintentos cuando corresponda.
 */

// 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.*;

/*
 * clase CrnpClient
 * ----------------
 * Consulte los comentarios de cabecera de archivos anteriores.
 */
class CrnpClient
{
    /*
     * main
     * ----
     * El punto de entrada de la ejecución main, sólo verifica
     * el número de argumentos de línea de comandos y construye
     * una instancia de CrnpClient para que realice todo el trabajo.
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

        /* Verificar el número de argumentos de línea de comandos */
        if (args.length < 4) {
            System.out.println(
                "Sintaxis: java CrnpClient sistemacrnp puertocrnp "
                + "puertoLocal (-ac | -ae | -re) "
                + "[(M | A | RG=nombre | R=nombre) [...]]");
            System.exit(1);
        }


        /*
```

```
     * Se espera que la línea de comandos contenga la IP/el puerto del
     * servidor crnp, el puerto local en el que se debe recibir y
     * argumentos que especifiquen el tipo de registro.
     */
    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);
    }

    // Crear CrnpClient
    CrnpClient client = new CrnpClient(regIp, regPort, localPort,
        args);

    // Esperar hasta que el usuario desee terminar el programa
    System.out.println("Pulsar Retorno para terminar la demostración...");

    // la lectura se bloquea hasta que el usuario escriba algo
    try {
        System.in.read();
    } catch (IOException e) {
        System.out.println(e.toString());
    }

    // apagar el cliente
    client.shutdown();
    System.exit(0);
}

/*
 * ======================
 * métodos públicos
 * ======================
 */

/*
 * Constructor CrnpClient
 * ----------------------
 * Analiza los argumentos de línea de comandos para saber cómo contactar con
 * el servidor crnp, crea el subproceso de recepción de eventos y lo pone
 * en ejecución, crea el objeto XML DocumentBuilderFactory y, finalmente, se
 * registra para recibir rellamadas en el servidor crnp.
 */
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
        localPort = localPortIn;
        regs = clArgs;
```

```
        /*
         * Configurar el creador de documentos para
         * el proceso de xml.
         */
        setupXmlProcessing();

        /*
         * Crear EventReceptionThread, que crea
         * ServerSocket y lo vincula a un puerto e ip local.
         */
        createEvtRecepThr();

        /*
         * Registrar en el servidor crnp.
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * ---------------
 * Rellamada a CrnpClient, utilizada por EventReceptionThread
 * cuando recibe rellamadas de eventos.
 */
public void processEvent(Event event)
{
    /*
     * Con fines demostrativos, imprima el evento en System.out.
     * Por supuesto, una aplicación real utilizaría el evento
     * de algún modo.
     */
    event.print(System.out);
}

/*
 * apagado
 * -------------
 * Cancelar el registro del servidor CRNP.
 */
public void shutdown()
{
    try {
        /* Enviar un mensaje de cancelación de registro al servidor */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}
```

```
/*
 * ======================
 * métodos de ayuda privados
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Crear el generador de documentos para
 * analizar los eventos y respuestas de xml.
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // No es necesario validar
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // Se desea ignorar comentarios y espacios en blanco
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // Fusionar secciones CDATA en nodos de TEXT.
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -------------------
 * Crea un objeto EventReceptionThread nuevo, guarda el ip
 * y puerto al que está vinculado el zócalo receptor y
 * pone el subproceso en ejecución.
 */
private void createEvtRecepThr() throws Exception
{
    /* crear el objeto de subproceso */
    evtThr = new EventReceptionThread(this);

    /*
     * Ahora, iniciar la ejecución del subproceso para empezar a
     * recibir rellamadas de entrega de eventos.
     */
    evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
 * Crea una conexión de zócalo al servidor crnp y envía
 * un mensaje de registro de evento.
```

```
 */
private void registerCallbacks() throws Exception
{
    System.out.println("Acerca del registro");

    /*
     * Crear un zócalo conectado al ip/puerto de registro
     * del servidor crnp y enviar la información de registro.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Leer la respuesta
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Cerrar la conexión de zócalo.
     */
    sock.close();
}

/*
 * unregister
 * ----------
 * Al igual que en registerCallbacks, crear una conexión de zócalo al
 * servidor crnp, enviar un mensaje de cancelación de registro, esperar
 * la respuesta del servidor y cerrar el zócalo.
 */
private void unregister() throws Exception
{
    System.out.println("Acerca de la cancelación del registro");

    /*
     * Crear un zócalo conectado al ip/puerto de registro del
     * servidor crnp y enviar información de cancelación de registro.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Leer la respuesta
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Cerrar la conexión de zócalo.
     */
    sock.close();
}
```

```
/*
 * createRegistrationString
 * ------------------
 * Construye un objeto CallbackReg basado en los argumentos de línea de
 * comandos de este programa; después, recupera la cadena XML del objeto
 * CallbackReg.
 */
private String createRegistrationString() throws Exception
{
    /*
     * Crear la clase real de CallbackReg y establecer el puerto.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    // Establecer el tipo de registro
    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("Tipo de registro no válido: " + regs[3]);
        System.exit(1);
    }

    // Agregar los eventos
    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
 * ----------------
 * Crea un evento de registro de XML con clase EC_Cluster, y sin
 * subclase.
```

```
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
 * ---------------------
 * Crea un evento de registro de XML con EC_Cluster, subclase
 * ESC_cluster_membership.
 */
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Crea un evento de registro de XML con clase EC_Cluster,
 * subclase ESC_cluster_rg_state y un nvpair "rg_name" (basado
 * en parámetro de entrada).
 */
private Event createRgEvent(String rgname)
{
    /*
     * Crea un evento de cambio de estado de grupo de recursos para el
     * grupo de recursos nombre_registro. Observar que se da
     * un par nombre-valor (nvpair) para este tipo de evento para
     * especificar el grupo de recursos que interesa.
     */
    /*
     * Construir el objeto de evento y fijar la clase y subclase.
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * Crear el objeto nvpair y agregarlo al evento.
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
}
```

```
/*
 * createREvent
 * ----------------
 * Crea un evento de registro de XML con clase EC_Cluster,
 * subclase ESC_cluster_r_state y un nvpair "r_name" (basado
 * en parámetro de entrada).
 */
private Event createREvent(String rname)
{
    /*
     * Crea un evento de cambio de estado de recurso para el
     * recurso nombre_registro. Observar que se indica un par
     * nombre-valor (nvpair) para este tipo de evento, para
     * especificar el grupo de recursos que interesa.
     */
    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
 * ------------------
 * Construye un objeto REMOVE_CLIENT CallbackReg, después recupera
 * la cadena XML del objeto CallbackReg.
 */
private String createUnregistrationString() throws Exception
{
    /*
     * Crear el objeto CallbackReg.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
     * se dirige el registro hacia OutputStream
     */
    String xmlStr = cbReg.convertToXml();

    // Imprimir la cadena para la depuración
    System.out.println(xmlStr);
    return (xmlStr);
}


/*
 * readRegistrationReply
```

```
     * ------------------------
     * Analizar el xml en un documento, construir un objeto RegReply
     * a partir del documento e imprimir el objeto RegReply.  Observar
     * que una aplicación real actuaría según el status_code
     * del objeto RegReply.
     */
    private void readRegistrationReply(InputStream stream)
        throws Exception
    {
        // Crear el constructor de documento
        DocumentBuilder db = dbf.newDocumentBuilder();

        //
        // Establecer ErrorHandler antes del análisis
        // Utilizar el descriptor predeterminado.
        //
        db.setErrorHandler(new DefaultHandler());

        // Analizar el archivo de entrada
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
    }

    /* variables de miembro privado */
    private InetAddress regIp;
    private int regPort;
    private EventReceptionThread evtThr;
    private String regs[];

    /* variables de miembro público */
    public int localPort;
    public DocumentBuilderFactory dbf;
}

/*
 * clase EventReceptionThread
 * --------------------------
 * Consultar los comentarios de cabecera de archivo de antes.
 */
class EventReceptionThread extends Thread
{
    /*
     * EventReceptionThread constructor
     * --------------------------------
     * Crea un ServerSocket nuevo, vinculado al nombre de sistema local y
     * un puerto comodín.
     */
    public EventReceptionThread(CrnpClient clientIn) throws IOException
    {
        /*
         * mantener una referencia al cliente para que se pueda rellamar
         * al recibir un evento.
         */
```

```
    client = clientIn;

    /*
     * Especificar la IP al que se debería establecer el vínculo.
     * Es sencillamente el ip del sistema local. Si hay más de una
     * interfaz pública configurada en esta máquina, se irá a
     * cualquiera que indique InetAddress.getLocalHost.
     *
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
        System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Invocado por el método Thread.Start.
 *
 * Realiza un bucle infinito, a la espera de conexiones de entrada
 * de ServerSocket.
 *
 * A medida que se acepta cada conexión entrante, se crea un objeto de
 * de evento del canal de xml, que luego se pasa al objeto
 * CrnpClient para su proceso.
 */
public void run()
{
    /*
     * Bucle infinito.
     */
    try {
        //
        // Crea el generador de documento con el
        // generador de documentos de CrnpClient.
        //
        DocumentBuilder db = client.dbf.newDocumentBuilder();

        //
        // Establecer un ErrorHandler antes de analizar
        // Utilizar el descriptor predeterminado.
        //
        db.setErrorHandler(new DefaultHandler());

        while(true) {
            /* Esperar rellamada del servidor */
            Socket sock = listeningSock.accept();

            // Analizar el archivo de entrada
            Document doc = db.parse(sock.getInputStream());

            Event event = new Event(doc);
            client.processEvent(event);

            /* Cerrar el zócalo */
```

```
                sock.close();
            }
            // INACCESIBLE

        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }

    /* variables de miembro privado */
    private ServerSocket listeningSock;
    private CrnpClient client;
}

/*
 * clase NVPair
 * -----------
 * Esta clase guarda un par de nombre-valor (ambas cadenas). Sabe como crear
 * un mensaje XML NVPAIR desde sus miembros y cómo analizar un elemento
 * XML NVPAIR en sus miembros.
 *
 * Observar que la especificación de formato de un NVPAIR permite varios
 * valores. Se ha realizado la simplificación a un solo valor.
 */
class NVPair
{
    /*
     * Dos constructores: el primero crea un NVPair vacío; el segundo
     * crea un NVPair a partir de un elemento XML NVPAIR.
     */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
     * Configuradores públicos.
     */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }

    /*
```

```
 * Imprime el nombre y valor en una sola línea.
 */
public void print(PrintStream out)
{
    out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Construye un elemento XML NVPAIR a partir de las variables
 * de miembro. Toma el documento como parámetro para que pueda
 * crear el elemento.
 */
public Element createXmlElement(Document doc)
{
    // Crear el elemento.
    Element nvpair = (Element)
        doc.createElement("NVPAIR");
    //
    // Agregar el nombre. Observar que el nombre real es
    // una sección CDATA separada.
    //
    Element eName = doc.createElement("NAME");
    Node nameData = doc.createCDATASection(name);
    eName.appendChild(nameData);
    nvpair.appendChild(eName);
    //
    // Agregar el valor. Observar que el valor real es
    // una sección CDATA separada.
    //
    Element eValue = doc.createElement("VALUE");
    Node valueData = doc.createCDATASection(value);
    eValue.appendChild(valueData);
    nvpair.appendChild(eValue);

    return (nvpair);
}

/*
 * retrieveValues
 * ----------------
 * Analizar el elemento XML para recuperar el nombre y valor.
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // Buscar el elemento NAME
    //
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error al analizar: no se puede encontrar "
```

```
            + "nodo NAME.");
        return;
    }

    //
    // Obtener la sección TEXT
    //
    n = nl.item(0).getFirstChild();
    if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        System.out.println("Error al analizar: no se puede encontrar "
            + "sección TEXT.");
        return;
    }

    // Recuperar el valor
    name = n.getNodeValue();

    //
    // Ahora, obtener el elemento de valor
    //
    nl = elem.getElementsByTagName("VALUE");
    if (nl.getLength() != 1) {
        System.out.println("Error al analizar: no se puede encontrar "
            + "nodo VALUE.");
        return;
    }

    //
    // Obtener la sección TEXT
    //
    n = nl.item(0).getFirstChild();
    if (n == null || n.getNodeType() != Node.TEXT_NODE) {
        System.out.println("Error al analizar: no se puede encontrar "
            + "sección TEXT.");
        return;
    }

    // Recuperar el valor
    value = n.getNodeValue();
}


/*
 * Accesores públicos
 */
public String getName()
{
    return (name);
}

public String getValue()
{
    return (value);
}
```

```
    // Variables de miembro privado
    private String name, value;
}


/*
 * clase Event
 * -----------
 * Esta clase guarda un evento que consta de una clase, subclase, fabricante,
 * editor y lista de pares de nombre-valor. Sabe cómo construir un elemento
 * XML SC_EVENT_REG a partir de sus miembros y cómo analizar un elemento
 * XML SC_EVENT en sus miembros. Observar que existe una asimetría aquí: se
 * analizan los elementos SC_EVENT, pero se construyen elementos SC_EVENT_REG.
 * Esto se debe a que los elementos SC_EVENT_REG se utilizan en los mensajes
 * de registro (que hay que construir), en tanto que los elementos SC_EVENT se
 * utilizan en la entrega de eventos (que hay que analizar). La única diferencia
 * es que los elementos SC_EVENT_REG no tienen fabricante ni editor.
 */
class Event
{

    /*
     * Dos constructores: el primero crea un evento vacío; el segundo
     * crea un evento a partir de un documento XML SC_EVENT.
     */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // Convertir el documento en una cadena para imprimir con fines
        // de depuración.
        //
        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());

        // Realizar el análisis real.
        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
 * ------------------
 * Construye un elemento XMl SC_EVENT_REG a partir de las variables de
 * los miembros. Toma el documento como parámetro para que pueda crear
 * el elemento. Depende de la capacidad del NVPair createXmlElement.
 */
public Element createXmlElement(Document doc)
{
    Element event = (Element)
        doc.createElement("SC_EVENT_REG");
    event.setAttribute("CLASS", regClass);
    if (regSubclass != null) {
        event.setAttribute("SUBCLASS", regSubclass);
    }
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        event.appendChild(tempNv.createXmlElement(
            doc));
    }
    return (event);
}

/*
 * Imprime las variables de miembros en varias líneas.
 */
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
 * ---------------
 * Analizar el documento XML para recuperar la clase, subclase,
 * fabricante, editor y nvpairs.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Buscar el elemento SC_EVENT.
    //
    nl = doc.getElementsByTagName("SC_EVENT");
    if (nl.getLength() != 1) {
        System.out.println("Error al analizar: no se puede encontrar"
            + "nodo SC_EVENT.");
        return;
    }

    n = nl.item(0);

    //
    // Recuperar los valores de los atributos CLASS, SUBCLASS,
    // VENDOR y PUBLISHER.
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // Recuperar todos los pares n-v
    //
    for (Node child = n.getFirstChild(); child != null;
         child = child.getNextSibling())
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 * Métodos de accesores públicos.
 */
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);
    }

    // Variables de miembro privado.
    private String regClass, regSubclass;
    private Vector nvpairs;
    private String vendor, publisher;
}


/*
 * clase CallbackReg
 * -----------
 * Esta clase guarda un puerto y regType (ambas cadenas) y una lista de
 * eventos. Sabe cómo construir un mensaje XML SC_CALLBACK_REG de sus
 * miembros.
 *
 * Observar que esta clase no necesita poder analizar mensajes
 * SC_CALLBACK_REG porque sólo el servidor CRNP debe analizar mensajes
 * SC_CALLBACK_REG
 *
 */
class CallbackReg
{
    // Definiciones útiles para el método setRegType
    public static final int ADD_CLIENT = 0;
    public static final int ADD_EVENTS = 1;
    public static final int REMOVE_EVENTS = 2;
    public static final int REMOVE_CLIENT = 3;

    public CallbackReg()
    {
        port = null;
        regType = null;
        regEvents = new Vector();
    }
```

```
/*
 * 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, regType no válido" +
            regTypeIn);
        regType = "ADD_CLIENT";
        break;
    }
}

public void addRegEvent(Event regEvent)
{
    regEvents.add(regEvent);
}


/*
 * convertToXml
 * ------------------
 * Construye un documento XML SC_CALLBACK_REG a partir de variables
 * de miembro. Depende de la capacidad del evento createXmlElement.
 */
public String convertToXml()
{
    Document document = null;
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.newDocument();
    } catch (ParserConfigurationException pce) {
        // No se puede crear el analizador con las opciones especificadas
        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);

        //
        // Ahora, convertir el documento en cadena.
        //
        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());
    }

    // variables de miembro privado
    private String port;
    private String regType;
    private Vector regEvents;
}

/*
 * clase RegReply
 * -----------
 * Esta clase guarda status_code y status_msg (ambas cadenas).
 * Sabe cómo analizar un elemento XML SC_REPLY en sus miembros.
 */
class RegReply
{

    /*
     * El constructor único toma un documento XML y lo analiza.
     */
    public RegReply(Document doc)
    {
        //
        // Convertir el documento en cadena.
        //
        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);
}

/*
 *  Accesores públicos
 */
public String getStatusCode()
{
    return (statusCode);
}

public String getStatusMsg()
{
    return (statusMsg);
}

/*
 * Imprime la información en una sola línea.
 */
public void print(PrintStream out)
{
    out.println(statusCode + ": " +
        (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Analiza el documento XML para recuperar statusCode y statusMsg.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Buscar el elemento SC_REPLY.
    //
    nl = doc.getElementsByTagName("SC_REPLY");
    if (nl.getLength() != 1) {
        System.out.println("Error al analizar: no se puede encontrar "
            + "nodo SC_REPLY.");
        return;
```

```
        }

        n = nl.item(0);

        // Recuperar el valor del atributo STATUS_CODE
        statusCode = ((Element)n).getAttribute("STATUS_CODE");

        //
        // Buscar el elemento SC_STATUS_MSG
        //
        nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
        if (nl.getLength() != 1) {
            System.out.println("Error al analizar: no se puede encontrar"
                + "nodo SC_STATUS_MSG.");
            return;
        }

        //
        // Obtener la sección TEXT, si la hubiera.
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Sin error, si no lo hay
            // se retorna discretamente.
            return;
        }

        // Recuperar el valor
        statusMsg = n.getNodeValue();
    }

    // variables de miembro privado
    private String statusCode;
    private String statusMsg;
}
```

Índice

B

C

D

E

J

K

L

M

N

O

P

R

S

T

U

V

W

X