Solaris OS용 Sun Cluster 데이터 서비스 개발 안내서

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

부품 번호: 819–2069
2005년 8월, 개정판 A

050816@12762

목차

머리말

Solaris OS용 Sun Cluster 데이터 서비스 개발 안내서에는 SPARC® 및 x86 기반 시스템에서의 Sun™ Cluster 데이터 서비스를 개발하기 위한 자원 관리 API 사용에 대한 정보가 들어 있습니다.

주 – 이 문서에서 "x86"이라는 용어는 Intel 마이크로프로세서 칩 32비트 제품군을 말하며 AMD에서 만든 마이크로프로세서 칩과 호환 가능합니다.

주 – Sun Cluster 소프트웨어는 SPARC 및 x86의 두 가지 플랫폼에서 실행됩니다. 이 설명서의 정보는 특정 장, 절, 주, 머리글로 표시된 항목, 그림, 표 또는 예에서 언급된 경우를 제외하고는 두 플랫폼 모두와 관련됩니다.

이 설명서의 대상

이 문서는 고급 개발자를 위해 작성되었기 때문에 Sun 소프트웨어 및 하드웨어에 대한 폭넓은 지식이 필요합니다. 이 문서에서는 개발자가 Solaris 운영 체제를 잘 알고 있다고 가정합니다.

이 책의 구성

Solaris OS용 Sun Cluster 데이터 서비스 개발 안내서는 다음 장과 부록으로 구성되어 있습니다.

1 장에서는 데이터 서비스를 개발하는 데 필요한 개념에 대해 개괄적으로 설명합니다.

2 장에서는 데이터 서비스 개발에 대한 자세한 정보를 제공합니다.

3 장에서는 자원 관리 API(RMAPI)를 구성하는 액세스 함수 및 콜백 메소드에 대한 참조를 제공합니다.

4 장에서는 자원 유형을 수정하기 위해 알아 두어야 할 문제점에 대해 설명합니다. 클러스터 관리자의 자원 업그레이드를 허용하는 방법에 대한 정보도 포함되어 있습니다.

5 장에서는 `in.named` 응용 프로그램용 샘플 Sun Cluster 데이터 서비스를 제공합니다.

6 장에서는 데이터 서비스 개발 라이브러리(DSDL)를 구성하는 응용 프로그램 프로그래밍 인터페이스에 대해 개괄적으로 설명합니다.

7 장에서는 자원 유형의 설계 및 구현에 사용되는 DSDL의 일반적인 용도에 대해 설명합니다.

8 장에서는 DSDL을 사용하여 구현된 샘플 자원 유형에 대해 설명합니다.

9 장에서는 SunPlex™ Agent Builder에 대해 설명합니다.

10 장에서는 일반 데이터 서비스를 만드는 방법에 대해 설명합니다.

11 장에서는 DSDL API 함수에 대해 설명합니다.

12 장에서는 CRNP(Cluster Reconfiguration Notification Protocol)에 대한 정보를 제공합니다. CRNP를 사용하여 페일오버와 확장 가능 응용 프로그램이 "클러스터를 인식"하도록 할 수 있습니다.

부록 A에서는 표준 자원 유형, 자원 및 자원 그룹 등록 정보에 대해 설명합니다.

부록 B에는 샘플 데이터 서비스의 각 메소드에 대한 전체 코드가 나와 있습니다.

부록 C에는 `SUNW.xfnts` 자원 유형의 각 메소드에 대한 전체 코드가 나와 있습니다.

부록 D에는 RGM(Resource Group Manager) 이름 및 값으로 유효한 문자에 대한 요구 사항이 나와 있습니다.

부록 E에는 고가용성(HA) 후보가 되기 위한 비클러스터 인식 응용 프로그램에 대한 요구 사항이 나와 있습니다.

부록 F에서는 CRNP용 문서 유형 정의(DTD)에 대해 설명합니다.

부록 G에는 12 장에 자세히 설명된 `CrnpClient.java` 응용 프로그램의 전체 내용이 나와 있습니다.

관련 문서

Sun Cluster 항목에 대한 정보는 다음 표에 나열된 설명서를 참조하십시오. Sun Cluster 설명서는 `http://docs.sun.com`에서 이용할 수 있습니다.

주제	문서
개요	**Solaris OS용 Sun Cluster 개요**
개념	**Solaris OS용 Sun Cluster 개념 안내서**
하드웨어 설치 및 관리	**Sun Cluster 3.0-3.1 Hardware Administration Manual for Solaris OS** 개별 하드웨어 관리 설명서
소프트웨어 설치	**Solaris OS용 Sun Cluster 소프트웨어 설치 안내서**
데이터 서비스 설치 및 관리	**Sun Cluster Data Services Planning and Administration Guide for Solaris OS** 개별 데이터 서비스 설명서
데이터 서비스 개발	**Solaris OS용 Sun Cluster 데이터 서비스 개발 안내서**
시스템 관리	**Solaris OS용 Sun Cluster 시스템 관리 안내서**
오류 메시지	**Sun Cluster Error Messages Guide for Solaris OS**
명령 및 함수 참조	**Sun Cluster Reference Manual for Solaris OS**

Sun Cluster 전체 설명서 목록은 `http://docs.sun.com`에서 해당 Sun Cluster 소프트웨어 릴리스의 릴리스 노트를 참조하십시오.

지원 받기

Sun Cluster 소프트웨어 설치 및 사용에 문제가 있으면 서비스 담당자에게 문의하십시오. 문의할 때 다음 정보가 필요합니다.

- 이름 및 전자 메일 주소

- 회사 이름, 주소 및 전화 번호
- 시스템 모델 및 일련 번호
- 운영 체제의 릴리스 번호(예: Solaris 10)
- Sun Cluster의 릴리스 번호(예: Sun Cluster 3.1)

다음 명령을 사용하여 서비스 담당자에게 제공할 시스템 정보를 수집합니다.

명령	기능
`prtconf -v`	시스템 메모리의 크기를 표시하고 주변 장치에 대한 정보를 보고합니다.
`psrinfo -v`	프로세서에 대한 정보를 표시합니다.
`showrev -p`	설치된 패치를 알려줍니다.
SPARC: `prtdiag -v`	시스템 진단 정보를 표시합니다.
`/usr/cluster/bin/scinstall -pv`	Sun Cluster 릴리스 및 패키지 버전 정보를 표시합니다.

`/var/adm/messages` 파일의 내용도 준비하십시오.

설명서, 지원 및 교육

Sun 기능	URL	설명
설명서	`http://www.sun.com/documentation/`	PDF 및 HTML 설명서를 다운로드하고 인쇄된 설명서를 주문할 수 있습니다.
지원 및 교육	`http://www.sun.com/supportraining/`	기술 지원 및 패치 다운로드는 물론 Sun 교육 과정에 대한 정보를 얻을 수 있습니다.

활자체 규약

다음 표는 이 책에서 사용된 활자체 변경 사항에 대하여 설명합니다.

표 P–1 활자체 규약

서체 또는 기호	의미	예
AaBbCc123	명령, 파일 및 디렉토리의 이름 등 컴퓨터 화면상에 출력되는 내용입니다.	`.login` 파일을 편집하십시오. `ls -a` 명령을 사용하여 모든 파일을 나열하십시오. `machine_name% you have mail.`
AaBbCc123	컴퓨터 화면상의 출력과는 달리 사용자가 직접 입력하는 내용입니다.	`machine_name%` **`su`** `Password:`
aabbcc123	명령줄 자리 표시자: 실제 이름이나 값으로 대체됩니다.	파일 삭제하려면 `rm` *filename*을 입력하십시오.
AaBbCc123	책 제목, 새로 나오는 단어나 용어, 강조 표시할 단어입니다.	**사용자 설명서**의 6장을 읽으십시오. **패치 분석**을 수행하십시오. 파일을 저장하지**마십시오**. [강조 표시된 일부 항목은 온라인에서 볼드로 표시됩니다.]

명령 예의 쉘 프롬프트

다음 표는 C 쉘, Bourne 쉘 및 Korn 쉘의 기본 시스템 프롬프트와 수퍼유저 프롬프트입니다.

표 P–2 쉘 프롬프트

쉘	프롬프트
C 쉘 프롬프트	`machine-name%`
C 쉘 수퍼유저 프롬프트	`machine-name#`

표 P–2 쉘 프롬프트 (계속)

쉘	프롬프트
Bourne 쉘 및 Korn 쉘 프롬프트	$
Bourne 쉘 및 Korn 쉘 수퍼유저 프롬프트	#

1장

자원 관리 개요

이 책에서는 Oracle®, Sun Java™ System Web Server(이전의 Sun ONE Web Server), DNS 등과 같은 소프트웨어 응용 프로그램의 자원 유형을 만들기 위한 지침을 제공합니다. 따라서 이 책은 자원 유형 개발자를 대상으로 합니다.

이 장에서는 데이터 서비스를 개발하기 위해 알아야 하는 개념을 개괄적으로 설명합니다. 이 장은 다음 내용으로 구성되어 있습니다.

주 – 이 문서에서 **자원 유형**과 **데이터 서비스**라는 용어는 서로 바꾸어 사용할 수 있습니다. **에이전트**라는 용어는 이 문서에서 거의 사용되지 않지만 **자원 유형** 및 **데이터 서비스**와 동일합니다.

Sun Cluster 응용 프로그램 환경

Sun Cluster 시스템에서는 응용 프로그램을 가용성과 확장성이 높은 자원으로 실행 및 관리할 수 있습니다. RGM(Resource Group Manager)은 고가용성 및 확장성을 위한 기법을 제공합니다. 이 기능에 대한 프로그래밍 인터페이스를 구성하는 요소는 다음과 같습니다.

- RGM이 클러스터에서 응용 프로그램을 제어할 수 있도록 개발자가 작성하는 콜백 메소드 집합
- 콜백 메소드를 작성하는 데 사용할 수 있는 저급 API 명령 및 함수 집합인 자원 관리 API(RMAPI). 이러한 API는 `libscha.so` 라이브러리에서 구현됩니다.

- 클러스터에서 프로세스를 모니터 및 재시작하기 위한 PMF(Process Monitor Facility)
- 저급 API 및 프로세스 관리 기능을 상위 수준에서 캡슐화하는 라이브러리 함수 집합인 DSDL(Data Service Development Library). DSDL은 콜백 메소드를 쉽게 작성할 수 있도록 일부 다른 기능을 추가합니다. 이 함수는 `libdsdev.so` 라이브러리에서 구현됩니다.

다음 그림은 이러한 요소의 상관 관계를 보여줍니다.

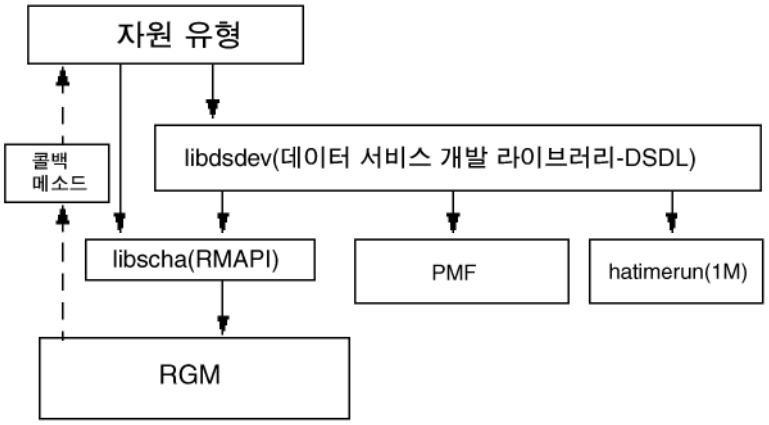


그림 1–1 Sun Cluster 응용 프로그램 환경의 프로그래밍 아키텍처

9 장에 설명된 SunPlex Agent Builder는 데이터 서비스를 만드는 과정을 자동화하는 Sun Cluster 패키지의 도구입니다. Agent Builder는 C 쉘(DSDL 함수를 사용하여 콜백 메소드 작성)이나 Korn 쉘(`ksh`)(저급 API 명령을 사용하여 콜백 메소드 작성)로 데이터 서비스 코드를 생성합니다.

RGM은 각 클러스터 노드에서 데몬으로 실행되거나 미리 구성된 정책에 따라 선택한 노드에서 자원을 자동으로 시작 및 중지합니다. RGM은 영향을 받은 노드에서 자원을 중지하고 다른 노드에서 자원을 시작함으로써 노드 실패 또는 재부트의 경우 자원의 가용성을 높여줍니다. 또한 RGM은 자원 실패를 감지하고 실패한 자원을 다른 노드로 재할당하거나 자원 성능의 다른 측면을 모니터할 수 있는 자원별 모니터를 자동으로 시작하고 중지합니다.

RGM은 한 번에 하나의 노드에서만 온라인 상태가 될 수 있는 페일오버 자원과 여러 노드에서 동시에 온라인 상태가 될 수 있는 확장 가능한 자원을 모두 지원합니다.

RGM 모델

이 절에서는 몇 가지 기본 용어를 소개하고 RGM 및 관련 인터페이스에 대해 좀더 자세히 설명합니다.

RGM은 상호 관련된 세 가지 주요 객체인 자원 유형, 자원 및 자원 그룹을 처리합니다. 아래 설명된 예는 이러한 객체의 개념을 소개합니다.

개발자는 기존 Oracle DBMS 응용 프로그램의 가용성을 높이는 `ha-oracle`이라는 자원 유형을 구현합니다. 최종 사용자는 각각 `ha-oracle` 유형의 자원인 마케팅, 엔지니어링 및 재무에 사용할 별개의 데이터베이스를 정의합니다. 클러스터 관리자는 이러한 자원을 여러 노드에서 실행하고 독립적으로 페일오버할 수 있도록 별개의 자원 그룹에 포함합니다. 개발자는 Oracle 데이터베이스가 필요한 고가용성 캘린더 서버를 구현하기 위해 `ha-calendar`라는 또 다른 자원 유형을 만듭니다. 클러스터 관리자는 재무 캘린더의 자원을 재무 데이터베이스 자원과 동일한 자원 그룹에 포함하여 두 자원이 모두 동일한 노드에서 실행되고 함께 페일오버되도록 합니다.

자원 유형

자원 유형은 다음으로 구성됩니다.

- 클러스터에서 실행되는 소프트웨어 응용 프로그램
- 응용 프로그램을 클러스터 자원으로 관리하기 위해 RGM에서 콜백 메소드로 사용하는 제어 프로그램
- 클러스터의 정적 구성 일부를 이루는 등록 정보 집합

RGM은 자원 유형 등록 정보를 사용하여 특정 유형의 자원을 관리합니다.

주 – 소프트웨어 응용 프로그램 외에 자원 유형은 네트워크 주소와 같은 다른 시스템 자원을 나타낼 수 있습니다.

개발자는 자원 유형 등록(RTR) 파일에서 자원 유형의 등록 정보를 지정하고 등록 정보 값을 설정합니다. RTR 파일은 34 페이지 "자원 및 자원 유형 등록 정보 설정" 및 `rt_reg`(4) 설명서 페이지에 설명된 형식을 따릅니다. 샘플 RTR 파일에 대한 설명은 76 페이지 "자원 유형 등록 파일 정의"를 참조하십시오.

223 페이지 "자원 유형 등록 정보"에는 자원 유형 등록 정보의 목록이 나와 있습니다.

클러스터 관리자는 클러스터에서 자원 유형 구현 및 기본 응용 프로그램을 설치 및 등록합니다. 등록 절차는 RTR 파일의 정보를 클러스터 구성으로 입력합니다. **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**에는 데이터 서비스를 등록하는 절차가 설명되어 있습니다.

자원

자원은 해당 자원 유형의 등록 정보와 값을 상속합니다. 또한 개발자는 RTR 파일에서 자원 등록 정보를 선언할 수 있습니다. 230 페이지 “자원 등록 정보”에는 자원 등록 정보의 목록이 나와 있습니다.

클러스터 관리자는 RTR 파일에서 등록 정보가 지정된 방법에 따라 특정 등록 정보의 값을 변경할 수 있습니다. 예를 들어, 등록 정보 정의는 허용 가능한 값의 범위를 지정하고 등록 정보의 조정 가능 시점을 조정 안 함, 항상, 작성 시(자원이 클러스터에 추가될 때) 또는 자원이 비활성화될 때로 지정할 수 있습니다. 이러한 사양 내에서 클러스터 관리자는 관리 명령을 사용하여 등록 정보를 변경할 수 있습니다.

클러스터 관리자는 각각 고유한 이름과 등록 정보 값 집합을 갖는 동일한 유형의 여러 자원을 작성하여 기본 응용 프로그램의 여러 인스턴스를 클러스터에서 실행할 수 있습니다. 각 인스턴스화는 클러스터 내에서 고유한 이름이 필요합니다.

자원 그룹

각 자원은 자원 그룹에서 구성되어야 합니다. RGM은 그룹의 모든 자원을 동일한 노드에서 함께 온라인 및 오프라인으로 전환합니다. RGM은 자원 그룹을 온라인 또는 오프라인 상태로 전환할 때 그룹의 개별 자원에서 콜백 메소드를 실행합니다.

자원 그룹이 현재 온라인 상태인 노드를 **기본** 또는 **기본 노드**라고 합니다. 자원 그룹은 각 기본 노드에 의해 **마스터**됩니다. 각 자원 그룹에는 자원 그룹의 모든 **잠재적 기본 노드** 또는 **마스터**를 식별하는 관련 `Nodelist` 등록 정보가 있습니다. `Nodelist` 등록 정보는 클러스터 관리자가 설정합니다.

또한 자원 그룹에는 등록 정보 집합이 있습니다. 이러한 등록 정보는 자원 그룹의 활성 상태를 반영하는 RGM이 설정하는 동적 등록 정보 및 클러스터 관리자가 설정할 수 있는 구성 등록 정보를 포함합니다.

RGM은 두 가지 유형의 자원 그룹,즉 페일오버 및 확장 가능 자원 그룹을 정의합니다. 페일오버 자원 그룹은 한 번에 하나의 노드에서만 온라인이 될 수 있는 반면 확장 가능 자원 그룹은 여러 노드에서 동시에 온라인이 될 수 있습니다. RGM은 각 유형의 자원 그룹을 작성하는 것을 지원하기 위해 등록 정보 집합을 제공합니다. 이러한 등록 정보에 대한 자세한 내용은 33 페이지 “클러스터에 데이터 서비스 전송” 및 42 페이지 “콜백 메소드 구현”을 참조하십시오.

245 페이지 “자원 그룹 등록 정보”에는 자원 그룹 등록 정보의 목록이 나와 있습니다.

RGM(Resource Group Manager)

RGM(Resource Group Manager)은 클러스터의 각 구성원 노드에서 실행되는 `rgmd` 데몬으로 구현됩니다. 모든 `rgmd` 프로세스는 서로 통신하며 함께 단일 클러스터 범주 기능으로 작동합니다.

RGM은 다음 함수를 지원합니다.

- 노드가 부트 또는 중단될 때마다 RGM은 관리되는 모든 자원 그룹을 적절한 마스터에서 자동으로 온라인 상태로 전환하여 가용성을 유지합니다.
- 특정 자원이 실패할 경우 해당 모니터 프로그램은 자원 그룹을 동일한 마스터에서 다시 시작하거나 새 마스터로 전환하도록 요청할 수 있습니다.
- 클러스터 관리자는 관리 명령을 실행하여 다음 작업 중 하나를 요청할 수 있습니다.
  - 자원 그룹의 마스터 변경
  - 자원 그룹 내의 특정 자원을 사용 가능 또는 불가능하게 설정
  - 자원 유형, 자원 또는 자원 그룹 작성, 삭제 또는 수정

RGM은 구성 변경을 활성화할 때마다 클러스터의 모든 구성원 노드에 이를 적용합니다. 이러한 종류의 작업을 **재구성**이라고 합니다. 개별 자원에서 상태 변경을 적용하기 위해 RGM은 해당 자원에서 자원 유형별 콜백 메소드를 호출합니다.

콜백 메소드

Sun Cluster 프레임워크는 콜백 기법을 사용하여 데이터 서비스와 RGM 간의 통신을 제공합니다. 이 프레임워크는 인자 및 반환 값을 비롯한 콜백 메소드 집합과 RGM이 각 메소드를 호출하는 환경을 정의합니다.

자원 유형 개발자는 개별 콜백 메소드 집합을 코딩하고 각 메소드를 RGM에서 호출할 수 있는 제어 프로그램으로 구현하여 데이터 서비스를 만듭니다. 즉, 데이터 서비스는 단일 실행 파일이 아니라 각각 RGM에서 직접 호출할 수 있는 여러 실행 스크립트(`ksh`) 또는 이진(C)으로 구성됩니다.

콜백 메소드는 RTR 파일을 통해 RGM에 등록됩니다. RTR 파일에서는 데이터 서비스에 대해 구현한 각 메소드의 프로그램을 식별할 수 있습니다. 클러스터 관리자가 클러스터에서 데이터 서비스를 등록하면 RGM은 콜백 프로그램의 ID를 비롯한 여러 정보를 제공하는 RTR 파일을 읽습니다.

자원 유형에 필요한 유일한 콜백 메소드는 시작 메소드(`Start` 또는 `Prenet_start`)와 중지 메소드(`Stop` 또는 `Postnet_stop`)입니다.

콜백 메소드는 다음 범주로 그룹화할 수 있습니다.

- 제어 및 초기화 메소드
  - `Start` 및 `Stop` 메소드는 온라인 또는 오프라인 상태로 전환 중인 그룹의 자원을 시작 및 중지합니다.
  - `Init`, `Fini` 및 `Boot` 메소드는 자원에 대해 초기화 및 종료 코드를 실행합니다.
- 관리 지원 메소드
  - `Validate` 메소드는 관리 작업에 의해 설정된 등록 정보를 확인합니다.
  - `Update` 메소드는 온라인 자원의 등록 정보 설정을 업데이트합니다.
- Net-relative 메소드

  `Prenet_start` 및 `Postnet_stop`은 동일한 자원 그룹의 네트워크 주소가 활성으로 구성되기 전이나 비활성으로 구성된 후에 특수한 시작 또는 종료 작업을 수행합니다.
- 모니터 제어 메소드
  - `Monitor_start` 및 `Monitor_stop`은 자원의 모니터를 시작 또는 중지합니다.
  - `Monitor_check`은 자원 그룹이 노드로 이동하기 전에 노드의 안정성을 평가합니다.

콜백 메소드에 대한 자세한 내용은 3 장 및 `rt_callbacks`(1HA) 설명서 페이지를 참조하십시오. 또한 샘플 데이터 서비스의 콜백 메소드에 대한 자세한 내용은 5 장 및 8 장을 참조하십시오.

프로그래밍 인터페이스

데이터 서비스 코드 작성을 위해 자원 관리 구조는 저급 또는 기본 API와 기본 API를 기반으로 작성된 고급 라이브러리, 그리고 개발자가 제공한 기본 입력에서 데이터 서비스를 자동으로 생성하는 SunPlex Agent Builder 도구를 제공합니다.

자원 관리 API

자원 관리 API(RMAPI)는 데이터 서비스가 시스템의 자원 유형, 자원 및 자원 그룹에 대한 정보에 액세스하고 로컬 재시작 또는 페일오버를 요청하며 자원 상태를 설정할 수 있게 하는 저급 함수 집합을 제공합니다. 이러한 함수는 `libscha.so` 라이브러리를 통해 액세스합니다. RMAPI는 쉘 명령과 C 함수의 두 가지 형태로 이러한 콜백 메소드를 제공합니다. RMAPI 함수에 대한 자세한 내용은 `scha_calls`(3HA) 설명서 페이지 및 3 장를 참조하십시오. 또한 샘플 데이터 서비스 콜백 메소드에서 이러한 함수를 사용하는 방법의 예는 5 장를 참조하십시오.

데이터 서비스 개발 라이브러리(DSDL)

RGM의 기본 **메소드 콜백 모델**을 유지하면서 높은 수준의 통합 프레임워크를 제공하는 데이터 서비스 개발 라이브러리(DSDL)는 RMAPI 위에 구축되었습니다. `libdsdev.so` 라이브러리는 DSDL 함수를 포함합니다. DSDL은 다음을 비롯하여 데이터 서비스 개발을 위한 다양한 기능을 통합합니다.

- `libscha.so`. 저급 자원 관리 API
- **PMF**. 프로세스와 해당 하위 항목을 모니터하고 중지되었을 경우 재시작하기 위한 방법을 제공하는 PMF(Process Monitor Facility). `pmfadm`(1M) 및 `rpc.pmfd`(1M) 설명서 페이지를 참조하십시오.
- `hatimerun`. 시간 제한 하에 프로그램을 실행하기 위한 기능. `hatimerun`(1M) 설명서 페이지를 참조하십시오.

대부분의 응용 프로그램에 대해 DSDL은 데이터 서비스를 작성하는 데 필요한 거의 또는 모든 기능을 제공합니다. 그러나 DSDL은 저급 API를 대체하는 것이 아니라 캡슐화 및 확장하는 것입니다. 실제로 많은 DSDL 함수가 `libscha.so` 함수를 호출합니다. 마찬가지로 DSDL을 사용하여 대부분의 데이터 서비스를 코딩하는 동안 `libscha.so` 함수를 직접 호출할 수 있습니다.

DSDL에 대한 자세한 내용은 6 장 및 `scha_calls`(3HA) 설명서 페이지를 참조하십시오.

SunPlex Agent Builder

Agent Builder는 데이터 서비스 작성을 자동화하는 도구입니다. 대상 응용 프로그램과 작성할 데이터 서비스에 대한 기본 정보를 입력합니다. Agent Builder는 소스 및 실행 코드(C 또는 Korn 쉘), 사용자 정의된 RTR 파일 및 Solaris 패키지가 포함된 데이터 서비스를 생성합니다.

대부분의 응용 프로그램에서는 Agent Builder를 사용하여 약간의 수동 변경만으로 완전한 데이터 서비스를 생성할 수 있습니다. 보다 세부적인 요구 사항(예: 추가 등록 정보에 대한 검증 추가)을 가진 응용 프로그램은 Agent Builder에서 수행할 수 없는 작업이 필요할 수 있습니다. 그러나 이러한 경우에도 Agent Builder를 사용하여 대부분의 코드를 생성하고 나머지를 수동으로 코딩할 수 있습니다. Agent Builder를 사용하면 최소한 Solaris 패키지를 자동으로 생성할 수 있습니다.

Resource Group Manager 관리 인터페이스

Sun Cluster는 클러스터 관리를 위한 그래픽 사용자 인터페이스(GUI)와 명령 집합을 모두 제공합니다.

SunPlex Manager

SunPlex Manager는 다음 작업을 수행할 수 있게 해주는 웹 기반 도구입니다.

- 클러스터 설치
- 클러스터 관리
- 자원 및 자원 그룹 구성 및 만들기
- Sun Cluster 소프트웨어로 데이터 서비스 구성

SunPlex Manager를 설치하는 방법과 SunPlex Manager를 사용하여 클러스터 소프트웨어를 설치하는 방법에 대한 지침은 **Solaris OS용 Sun Cluster 소프트웨어 설치 안내서**를 참조하십시오. SunPlex Manager는 대부분의 고유한 관리 작업에 대한 온라인 도움말을 제공합니다.

`scsetup` 유틸리티

`scsetup`(1M) 유틸리티를 사용하면 대부분의 Sun Cluster 관리 작업을 대화식으로 수행할 수 있습니다.

`scsetup` 유틸리티를 사용하여 다음과 같은 Sun Cluster 요소를 관리할 수 있습니다.

- 쿼럼
- 자원 그룹
- 데이터 서비스
- 클러스터 상호 연결
- 장치 그룹 및 볼륨
- 개인 호스트 이름
- 새 노드
- 기타 클러스터 등록 정보

`scsetup` 유틸리티를 사용하여 다음 작업도 수행할 수 있습니다.

- 자원 그룹 만들기
- 네트워크 자원을 자원 그룹에 추가
- 데이터 서비스 자원을 자원 그룹에 추가
- 자원 유형 등록
- 자온 그룹을 온라인 또는 오프라인 상태로 전환
- 자원 그룹 스위치오버
- 자원 활성화 또는 비활성화
- 자원 그룹 등록 정보 변경
- 자원 등록 정보 변경
- 자원 그룹에서 자원 제거
- 자원 그룹 제거
- 자원에서 `Stop_failed` 오류 플래그 지우기

관리 명령

RGM 객체 관리를 위한 Sun Cluster 명령은 `scrgadm`, `scswitch` 및 `scstat -g`입니다.

scrgadm 명령을 사용하면 RGM에 사용되는 자원 유형, 자원 그룹 및 자원 객체를 확인, 작성, 구성 및 삭제할 수 있습니다. 이 명령은 클러스터에 대한 관리 인터페이스의 일부이며 이 장의 나머지 부분에 설명된 응용 프로그램 인터페이스와 동일한 프로그래밍 컨텍스트에서 사용되지 않습니다. 그러나 scrgadm은 API가 작동하는 클러스터 구성을 생성하기 위한 도구입니다. 관리 인터페이스를 이해하는 것은 응용 프로그램 인터페이스를 이해하는 데 있어 기본이 됩니다. 이 명령으로 수행할 수 있는 관리 작업에 대한 자세한 내용은 scrgadm(1M) 설명서 페이지를 참조하십시오.

scswitch 명령은 지정된 노드에서 자원 그룹을 온라인 및 오프라인 상태로 전환하고 해당 모니터를 사용 가능 또는 사용 불가능하게 합니다. 이 명령으로 수행할 수 있는 관리 작업에 대한 자세한 내용은 scswitch(1M) 설명서 페이지를 참조하십시오.

scstat -g 명령은 모든 자원 그룹과 자원의 현재 동적 상태를 표시합니다. 이 명령으로 수행할 수 있는 관리 작업에 대한 자세한 내용은 scstat(1M) 설명서 페이지를 참조하십시오.

2장

데이터 서비스 개발

이 장에서는 응용 프로그램의 가용성을 높이거나 확장 가능하게 만드는 방법을 설명하고 데이터 서비스 개발에 대한 자세한 정보를 제공합니다.

이 장은 다음 내용으로 구성되어 있습니다.

응용 프로그램 적합성 분석

데이터 서비스 만들기의 첫 번째 단계는 대상 응용 프로그램이 고가용성 및 확장성에 대한 요구 사항을 충족시키는지 확인하는 것입니다. 응용 프로그램이 모든 요구 사항을 충족시키지 못할 경우 응용 프로그램 소스 코드를 수정하여 가용성을 높이거나 확장 가능하게 만들 수 있습니다.

다음 목록에는 확장 가능한 고가용성의 응용 프로그램을 만들기 위한 요구 사항이 요약되어 있습니다. 자세한 내용이 필요하거나 응용 프로그램 소스 코드를 수정해야 하는 경우 부록 B을 참조하십시오.

주 – 확장 가능한 서비스는 목록 뒤에 나오는 일부 추가 기준뿐만 아니라 고가용성에 대한 다음 조건을 모두 충족시켜야 합니다.

- 네트워크 인식(클라이언트-서버 모델) 및 네트워크 비인식(클라이언트 없는) 응용 프로그램은 모두 Sun Cluster 환경에서 가용성이 높거나 확장 가능할 수 있습니다. 그러나 Sun Cluster는 `telnet` 또는 `rlogin`을 통해 액세스하는 서버에서 응용 프로그램이 실행되는 시간 공유 환경에서는 향상된 가용성을 제공할 수 없습니다.
- 응용 프로그램은 충돌을 허용해야 합니다. 즉, 노드가 예기치 않게 사용 불능이 된 후 응용 프로그램을 시작하면 필요한 경우 디스크 데이터를 복구해야 합니다. 또한 충돌 후 복구 시간을 바인드해야 합니다. 디스크 복구 및 응용 프로그램 다시 시작 기능은 데이터 무결성 관련 문제이므로 충돌 허용은 고가용성 응용 프로그램을 만들기 위한 필수 조건입니다. 데이터 서비스의 연결 복구 기능은 필요 없습니다.
- 응용 프로그램은 해당 응용 프로그램이 실행되는 노드의 물리적 호스트 이름에 의존해서는 안 됩니다. 자세한 내용은 311 페이지 “호스트 이름”을 참조하십시오.
- 여러 IP 주소가 활성으로 구성된 환경에서 응용 프로그램이 제대로 작동해야 합니다. 예를 들어, 둘 이상의 공용 네트워크에 노드가 있는 다중 홈 호스트 환경 및 하나의 하드웨어 인터페이스에 여러 논리 인터페이스가 활성 상태로 구성되어 있는 노드 환경이 있습니다.
- 가용성을 높이려면 응용 프로그램 데이터가 클러스터 파일 시스템에 있어야 합니다. 310 페이지 “멀티호스트된 데이터”를 참조하십시오.

  응용 프로그램에서 데이터 위치에 하드 연결된 경로 이름을 사용하는 경우 응용 프로그램 소스 코드를 변경하지 않고 해당 경로를 클러스터 파일 시스템의 위치를 가리키는 심볼릭 링크로 변경할 수 있습니다. 자세한 내용은 310 페이지 “멀티호스트된 데이터 배치에 심볼릭 링크 사용”을 참조하십시오.
- 응용 프로그램 이진과 라이브러리는 각 노드나 클러스터 파일 시스템에 로컬로 위치할 수 있습니다. 클러스터 파일 시스템에 위치할 경우 한 번만 설치하면 된다는 장점과 롤링 업그레이드 사용 시 RGM의 제어 하에 응용 프로그램을 실행하는 동안 이진이 사용된다는 단점이 있습니다.
- 클라이언트에는 첫 번째 시도가 시간 초과된 경우 자동으로 조회를 다시 시도할 수 있는 용량이 있어야 합니다. 응용 프로그램과 프로토콜에서 이미 단일 서버 충돌 및 재부트를 처리한 경우에는 페일오버 또는 스위치오버되는 자원 그룹이 포함된 경우도 처리할 수 있습니다. 자세한 내용은 313 페이지 “클라이언트 재시도”를 참조하십시오.
- 응용 프로그램의 클러스터 파일 시스템에 UNIX® 도메인 소켓이나 명명된 파이프가 없어야 합니다.

확장 가능한 서비스는 다음 요구 사항도 충족시켜야 합니다.

- 응용 프로그램에 여러 인스턴스를 실행할 수 있는 기능이 있어야 합니다. 이 인스턴스는 모두 클러스터 파일 시스템의 동일한 응용 프로그램 데이터에서 작동합니다.
- 응용 프로그램에서는 여러 노드에서 동시에 액세스하기 위해 데이터 일관성을 제공해야 합니다.

- 응용 프로그램에서는 클러스터 파일 시스템 같이 전역적으로 볼 수 있는 기법이 있는 충분한 잠금을 구현해야 합니다.

확장 가능 서비스의 경우 응용 프로그램 특성에서 로드 균형 조정 정책도 결정합니다. 예를 들어, 인스턴스가 클라이언트 요청에 응답할 수 있도록 허용하는 로드 균형 조정 정책 `Lb_weighted`는 클라이언트 연결을 위해 서버의 메모리 내장 캐시를 사용하는 응용 프로그램에는 작동하지 않습니다. 이 경우 지정된 클라이언트의 트래픽을 응용 프로그램의 한 인스턴스로 제한하는 로드 균형 조정 정책을 지정해야 합니다. 로드 균형 조정 정책 `Lb_sticky` 및 `Lb_sticky_wild`는 클라이언트의 모든 요청을 동일한 응용 프로그램 인스턴스로 반복해서 전송하며 이 인스턴스에서 메모리 내장 캐시를 사용할 수 있습니다. 서로 다른 클라이언트에서 여러 클라이언트 요청이 올 경우 RGM은 서비스 인스턴스 간에 요청을 분산시킵니다. 확장 가능한 데이터 서비스의 로드 균형 조정 정책 설정에 대한 자세한 내용은 49 페이지 "페일오버 자원 구현"을 참조하십시오.

사용할 인터페이스 결정

Sun Cluster 개발자 지원 패키지(`SUNWscdev`)에서는 데이터 서비스 메소드를 코딩하기 위한 다음 두 집합의 인터페이스를 제공합니다.

- `libscha.so` 라이브러리에 있는 저급 함수 집합인 자원 관리 API(RMAPI)
- `libdsdev.so` 라이브러리에 있으며 RMAPI의 기능을 캡슐화하고 다른 추가 기능을 제공하는 고급 기능 집합인 데이터 서비스 개발 라이브러리(DSDL)

그리고 데이터 서비스 작성을 자동화하는 도구인 SunPlex Agent Builder도 Sun Cluster 개발자 지원 패키지에 포함되어 있습니다.

데이터 서비스를 개발하기 위해 권장되는 방법은 다음과 같습니다.

1. C 쉘 또는 Korn 쉘로 코드화할 것인지 결정합니다. Korn 쉘을 사용할 경우 C 인터페이스만 제공하는 DSDL을 사용할 수 없습니다.
2. Agent Builder를 실행하고, 요청된 정보를 지정한 다음 소스 코드와 실행 코드, RTR 파일 및 패키지를 포함하는 데이터 서비스를 생성합니다.
3. 생성된 데이터 서비스에 사용자 정의가 필요한 경우 생성된 소스 파일에 DSDL 코드를 추가할 수 있습니다. Agent Builder는 사용자 고유 코드를 추가할 수 있는 소스 파일의 특정 위치를 주석과 함께 표시합니다.
4. 코드에 대상 응용 프로그램을 지원하기 위한 추가 사용자 정의가 필요할 경우 기존 소스 코드에 RMAPI 기능을 추가할 수 있습니다.

실제로는 여러 가지 방법으로 데이터 서비스를 만들 수 있습니다. 예를 들어, 사용자 코드를 Agent Builder에서 생성된 코드의 특정 위치에 추가하는 대신 생성된 메소드 중 하나 또는 생성된 모니터 프로그램을 DSDL이나 RMAPI 함수를 사용하여 새로 작성한 프로그램으로 전부 바꿀 수 있습니다. 그러나 처리 방식과 무관하게 대부분의 경우 다음과 같은 이유로 인해 Agent Builder에서 시작하는 것이 좋습니다.

- 일반적인 측면에 국한되지만 Agent Builder에서 생성된 코드는 여러 데이터 서비스에서 테스트되었습니다.
- Agent Builder는 RTR 파일, `makefile`, 자원 패키지 및 데이터 서비스용 기타 지원 파일을 생성합니다. 데이터 서비스 코드를 사용하지 않더라도 이러한 기타 파일을 사용하면 많은 양의 작업을 줄일 수 있습니다.
- 생성된 코드를 수정할 수 있습니다.

주 – C 함수 집합과 스크립트에서 사용할 명령 집합을 제공하는 RMAPI와는 달리 DSDL에서는 C 함수 인터페이스만 제공합니다. 따라서 Agent Builder에 Korn 쉘(`ksh`) 출력을 지정한 경우 DSDL `ksh` 명령이 없기 때문에 생성된 소스 코드에서 RMAPI를 호출합니다.

데이터 서비스 작성을 위한 개발 환경 설정

데이터 서비스 개발을 시작하기 전에 Sun Cluster 개발 패키지(`SUNWscdev`)를 설치해야만 Sun Cluster 헤더 및 라이브러리 파일에 액세스할 수 있습니다. 모든 클러스터 노드에 이 패키지가 설치되어 있더라도 일반적으로 클러스터 노드가 아닌 별도의 비클러스터 개발 시스템에서 데이터 서비스를 개발합니다. 이런 일반적인 경우 `pkgadd` 명령을 사용하여 개발 시스템에 `SUNWscdev` 패키지를 설치해야 합니다.

코드를 컴파일 및 링크할 경우 특수한 옵션을 설정하여 헤더 및 라이브러리 파일을 식별해야 합니다.

주 – Solaris Operating System 및 Sun Cluster 제품에서 호환성 모드 컴파일 C++ 코드와 표준 모드 컴파일 C++ 코드를 혼합할 수 없습니다. 따라서 Sun Cluster에서 사용할 C++ 기반 데이터 서비스를 만들려면 해당 데이터 서비스를 다음과 같이 컴파일해야 합니다.

- Sun Cluster 3.0 이전 버전의 경우 호환성 모드를 사용합니다.
- Sun Cluster 3.1부터는 표준 모드를 사용합니다.

비클러스터 노드에서 개발을 마친 경우 테스트를 위해 완성된 데이터 서비스를 클러스터에 전송할 수 있습니다.

주 – Solaris 운영 체제를 사용하는 경우 Solaris 8 OS 이상 버전의 개발자 또는 전체 배포 소프트웨어 그룹을 사용 중인지 확인합니다.

이 절에서는 다음 작업을 수행하는 절차를 설명합니다.

- Sun Cluster 개발 패키지(SUNWscdev) 설치, 올바른 컴파일러 및 링커 옵션 설정
- 클러스터에 데이터 서비스 전송

▼ 개발 환경 설정 방법

다음 절차는 데이터 서비스 개발을 위해 SUNWscdev 패키지를 설치하고 컴파일러 및 링커 옵션을 설정하는 방법에 대해서 설명합니다.

단계 **1. 수퍼유저가 되거나 그에 상응하는 역할을 맡고**

2. 디렉토리를 원하는 CD-ROM 디렉토리로 변경합니다.

```
# cd cd-rom-directory
```

3. 현재 디렉토리에 SUNWscdev 패키지를 설치합니다.

- **영역 환경에서 Solaris 10 OS를 사용하는 경우 전역 영역의 전역 관리자로 다음 명령을 입력합니다.**

  ```
  # pkgadd -G -d . SUNWscdev
  ```

  SUNWscdev의 내용이 비전역 영역과 공유되는 전역 영역에 영향을 주지 않으면 SUNWscdev 패키지가 전역 영역에 추가됩니다.

- **다른 Solaris OS 버전을 사용하거나 비영역 환경에서 Solaris 10 OS를 사용하는 경우 다음 명령을 입력합니다.**

  ```
  # pkgadd -d . SUNWscdev
  ```

4. makefile에서 데이터 서비스 코드용으로 포함할 파일과 라이브러리 파일을 식별하는 컴파일러 및 링커 옵션을 지정합니다.

Sun Cluster 헤더 파일을 식별하는 -I 옵션, 개발 시스템의 컴파일 시 라이브러리 검색 경로를 지정하는 -L 옵션 및 클러스터의 런타임 링커에 대한 라이브러리 검색 경로를 지정하는 -R 옵션을 지정합니다.

```
# Makefile for sample data service
...

-I /usr/cluster/include

-L /usr/cluster/lib

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

클러스터에 데이터 서비스 전송

개발 시스템에서 데이터 서비스 개발을 완료한 경우 테스트를 위해 데이터 서비스를 클러스터에 전송해야 합니다. 전송 중 오류가 발생할 가능성을 줄이려면 데이터 서비스 코드와 RTR 파일을 하나의 패키지로 결합한 다음 이 패키지를 해당 클러스터의 모든 노드에 설치합니다.

주 – pkgadd를 사용하거나 데이터 서비스를 설치하는 다른 방법을 사용하여 데이터 서비스를 모든 클러스터 노드에 두어야 합니다. Agent Builder는 이 패키지를 자동으로 생성합니다.

자원 및 자원 유형 등록 정보 설정

Sun Cluster에서는 데이터 서비스의 정적 구성을 정의하기 위해 사용하는 자원 유형 등록 정보와 자원 등록 정보 집합을 제공합니다. 자원 유형 등록 정보는 각각의 콜백 메소드에 대한 경로뿐만 아니라 자원 유형, 자원 버전, API 버전 등을 지정합니다. 모든 자원 유형 등록 정보의 목록은 223 페이지 "자원 유형 등록 정보"를 참조하십시오.

Failover_mode, Thorough_probe_interval 및 메소드 시간 초과값과 같은 자원 등록 정보는 자원의 정적 구성을 정의합니다. Resource_state 및 Status와 같은 동적 자원 등록 정보는 관리 자원의 활성 상태를 반영합니다. 자원 등록 정보에 대한 자세한 내용은 230 페이지 "자원 등록 정보"를 참조하십시오.

데이터 서비스의 필수 구성 요소인 자원 유형 등록(RTR) 파일에서 자원 유형 및 자원 등록 정보를 선언합니다. 클러스터 관리자가 Sun Cluster에 데이터 서비스를 등록할 때 RTR 파일에 데이터 서비스의 초기 구성을 정의합니다.

Agent Builder에서 데이터 서비스에 유용하고 필요한 등록 정보 집합을 선언하기 때문에 Agent Builder를 사용하여 데이터 서비스용 RTR 파일을 생성하는 것이 좋습니다. 예를 들어, RTR 파일에 특정 등록 정보(예: Resource_type)를 선언해야 합니다. 그렇지 않으면 데이터 서비스 등록에 실패합니다. 필수는 아니지만 기타 등록 정보는 RTR 파일에 해당 등록 정보를 선언하지 않은 경우 클러스터 관리자가 사용할 수 없습니다. 반면 일부 등록 정보는 RGM에서 해당 등록 정보를 정의하고 기본값을 제공하기 때문에 선언 여부와 관계없이 사용할 수 있습니다. 이러한 복잡성을 피하려면 Agent Builder를 사용하여 올바른 RTR 파일을 생성합니다. 나중에 RTR 파일을 편집하여 필요한 경우 특정 값을 변경할 수 있습니다.

이 절의 나머지 부분에서는 Agent Builder에서 생성된 샘플 RTR 파일을 설명합니다.

자원 유형 등록 정보 선언

클러스터 관리자는 개발자가 RTR 파일에 선언한 자원 유형 등록 정보를 구성할 수 없습니다. 이러한 자원 유형 등록 정보는 자원 유형 영구 구성의 일부가 됩니다.

주 – 클러스터 관리자만 자원 유형 등록 정보 `Installed_nodes`를 구성할 수 있습니다. RTR 파일에 `Installed_nodes`를 선언할 수 없습니다.

자원 유형 선언 구문은 다음과 같습니다.

property-name = *value*;

주 – 자원 그룹, 자원 및 자원 유형의 등록 정보 이름은 대소문자를 구분하지 **않습니다**. 등록 정보 이름을 지정하는 경우 대문자와 소문자를 임의로 조합해서 사용할 수 있습니다.

다음은 샘플(`smpl`) 데이터 서비스용 RTR 파일의 자원 유형 선언입니다.

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

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

Init_nodes = RG_PRIMARIES;

RT_basedir=/opt/SUNWsmpl/bin;

Start           =    smpl_svc_start;
Stop            =    smpl_svc_stop;

Validate        =    smpl_validate;
Update          =    smpl_update;

Monitor_start   =    smpl_monitor_start;
Monitor_stop    =    smpl_monitor_stop;
Monitor_check   =    smpl_monitor_check;
```

정보 – RTR 파일의 첫 번째 항목으로 `Resource_type` 등록 정보를 선언해야 합니다. 그렇지 않으면 자원 유형 등록에 실패합니다.

자원 유형 선언의 첫 번째 집합에서는 다음과 같이 자원 유형에 대한 기본 정보를 제공합니다.

`Resource_type` 및 `Vendor_id`
자원 유형 이름을 제공합니다. 자원 유형 이름은 `Resource_type` 등록 정보(`smpl`)로만 지정하거나 샘플과 같이 `Vendor_id` 등록 정보를 접두어로 사용하고 "."으로 자원 유형과 구분하여 지정합니다(`SUNW.smpl`). `Vendor_id`를 사용하는 경우 자원 유형을 정의하는 회사의 주식 기호를 사용합니다. 자원 유형 이름은 클러스터에서 고유해야 합니다.

주 – 일반적으로 자원 유형 이름(*vendoridApplicationname*)이 패키지 이름으로 사용됩니다. Solaris 9 운영 체제부터는 Vendor ID와 Application Name의 조합이 9자를 넘을 수 있습니다. 그러나 이전 버전의 Solaris 운영 체제를 사용 중이면 RGM에서 이 제한을 적용하지 않는 경우에도 Vendor ID와 Application Name의 조합이 9자를 넘을 수 없습니다.

한편 Agent Builder는 명시적으로 자원 유형 이름에서 패키지 이름을 생성하기 때문에 9자 제한을 적용합니다.

`RT_description`
자원 유형을 간단하게 설명합니다.

`RT_version`
샘플 데이터 서비스의 버전을 식별합니다.

`API_version`
API의 버전을 식별합니다. 예를 들어 `API_version = 2`는 Sun Cluster 3.0부터, `API_version = 5`는 Sun Cluster 3.1 9/04부터 모든 버전의 Sun Cluster에 데이터 서비스를 설치할 수 있음을 나타냅니다. 그러나 `API_version = 5`는 또한 Sun Cluster 3.1 9/04 이전에 릴리스된 Sun Cluster의 모든 버전에 데이터 서비스를 설치할 수 없음을 나타냅니다. 이 등록 정보에 대해서는 223 페이지 "자원 유형 등록 정보"의 `API_version` 항목에서 자세히 설명합니다.

`Failover = TRUE`
동시에 여러 노드에서 온라인 상태가 될 수 있는 자원 그룹에서 데이터 서비스를 실행할 수 없음을 나타냅니다. 즉, 이 선언은 페일오버 데이터 서비스를 지정합니다. 이 등록 정보에 대해서는 223 페이지 "자원 유형 등록 정보"의 `Failover` 항목에서 자세히 설명합니다.

`Start`, `Stop` 및 `Validate`
RGM에서 호출한 각각의 콜백 메소드 프로그램에 대한 경로를 제공합니다. 이러한 경로는 `RT_basedir`에 지정된 디렉토리에 상대적인 경로입니다.

나머지 자원 유형 선언에서는 다음과 같은 구성 정보를 제공합니다.

`Init_nodes = RG_PRIMARIES`
RGM이 데이터 서비스를 마스터할 수 있는 노드에서만 `Init`, `Boot`, `Fini` 및 `Validate` 메소드를 호출하도록 지정합니다. `RG_PRIMARIES`에서 지정한 노드는 데이터 서비스가 설치된 모든 노드 의 서브 세트입니다. 데이터 서비스가 설치된 모든 노드에서 RGM이 이러한 메소드를 호출하도록 지정하려면 값을 `RT_INSTALLED_NODES`로 설정합니다.

`RT_basedir`
콜백 메소드 경로 등 상대 경로를 완료하려면 `/opt/SUNWsample/bin`을 디렉토리 경로로 지정합니다.

`Start`, `Stop` 및 `Validate`
RGM에서 호출한 각각의 콜백 메소드 프로그램에 대한 경로를 제공합니다. 이러한 경로는 `RT_basedir`에 지정된 디렉토리에 상대적인 경로입니다.

자원 등록 정보 선언

자원 유형 등록 정보와 마찬가지로 RTR 파일에 자원 등록 정보를 선언합니다. 일반적으로 RTR 파일의 자원 유형 선언 다음에 자원 등록 정보 선언이 나옵니다. 자원 선언 구문은 둥근 괄호로 묶은 속성 값 쌍의 집합입니다.

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

Sun Cluster에서 제공하는 **시스템 정의** 등록 정보라는 자원 등록 정보의 경우 RTR 파일에서 특정 속성을 변경할 수 있습니다. 예를 들어, Sun Cluster는 각 콜백 메소드에 대한 메소드 시간 초과 등록 정보의 기본값을 제공합니다. RTR 파일에서 다른 기본값을 지정할 수 있습니다.

Sun Cluster에서 제공하는 등록 정보 속성 집합을 사용하여 RTR 파일에 **확장** 등록 정보라는 새 자원 등록 정보를 정의할 수도 있습니다. 253 페이지 “자원 등록 정보 속성”에서는 자원 등록 정보를 변경하고 정의하기 위한 속성을 나열합니다. RTR 파일의 시스템 정의 등록 정보 선언 다음에는 확장 등록 정보 선언이 나옵니다.

시스템 정의 자원 등록 정보의 첫 번째 집합에서 다음과 같이 콜백 메소드의 시간 초과값을 지정합니다.

```
...

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

{
```

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

등록 정보(PROPERTY = *value*) 이름은 각 자원 등록 정보 선언의 첫 번째 속성이어야 합니다. RTR 파일의 등록 정보 속성에서 정의하는 한도 내에서 자원 등록 정보를 구성할 수 있습니다. 예를 들어, 해당 샘플의 각 메소드 시간 초과 기본값은 300초입니다. 클러스터 관리자가 이 값을 변경할 수 있지만, MIN 속성에서 지정한 허용 가능한 최소값은 60초입니다. 253 페이지 "자원 등록 정보 속성"은 자원 등록 정보 속성의 목록을 포함합니다.

자원 등록 정보의 다음 집합에서는 데이터 서비스에서 특정 용도가 있는 등록 정보를 정의합니다.

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

# The number of retries to be done within a certain period before concluding
# that the application cannot be successfully started on this node.
```

```
{
        PROPERTY = Retry_count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

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

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

이 자원 등록 정보 선언은 클러스터 관리자가 이 특성과 관련된 등록 정보 값을 변경할 수 있는 경우를 제한하는 TUNABLE 특성을 포함합니다. 예를 들어, AT_CREATION 값은 자원을 만들고 나중에 변경할 수 없는 경우에만 클러스터 관리자가 해당 값을 지정할 수 있음을 의미합니다.

이러한 등록 정보의 대부분은 변경할 이유가 없는 한 Agent Builder에서 생성한 기본값을 사용할 수 있습니다. 다음은 이 등록 정보에 대한 정보입니다. 자세한 내용은 230 페이지 “자원 등록 정보” 또는 r_properties(5) 설명서 페이지를 참조하십시오.

Failover_mode
: Start 또는 Stop 메소드가 실패할 경우 RGM에서 자원 그룹을 다시 배치하거나 노드를 중단할지를 나타냅니다.

`Thorough_probe_interval`, `Retry_count` 및 `Retry_interval`
오류 모니터에서 사용합니다. `Tunable`은 `ANYTIME`과 동일하므로 오류 모니터가 최적의 상태로 작동하지 않을 경우 클러스터 관리자가 조정할 수 있습니다.

`Network_resources_used`
데이터 서비스에 사용된 논리 호스트 이름 또는 공유 주소 자원 목록입니다. Agent Builder는 클러스터 관리자가 데이터 서비스 구성 시 자원 목록(있는 경우)을 지정할 수 있도록 이 등록 정보를 선언합니다.

`Scalable`
`FALSE`로 설정하면 이 자원에서 클러스터 네트워킹(공유 주소) 기능을 사용하지 않음을 나타냅니다. 이 등록 정보를 `FALSE`로 설정한 경우 페일오버 서비스를 나타내려면 자원 유형 등록 정보 `Failover`를 `TRUE`로 설정해야 합니다. 이 등록 정보를 사용하는 방법은 33 페이지 "클러스터에 데이터 서비스 전송" 및 42 페이지 "콜백 메소드 구현"을 참조하십시오.

`Load_balancing_policy` 및 `Load_balancing_weights`
자동으로 이 등록 정보를 선언하지만, 페일오버 자원 유형에서는 사용되지 않습니다.

`Port_list`
서버가 수신하는 포트 목록을 식별합니다. Agent Builder는 클러스터 관리자가 데이터 서비스 구성 시 포트 목록을 지정할 수 있도록 이 등록 정보를 선언합니다.

확장 등록 정보 선언

확장 등록 정보는 샘플 RTR 파일의 끝에 있습니다.

```
# Extension Properties
#

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

# The following two properties control restart of the fault monitor.
{
        PROPERTY = Monitor_retry_count;
        EXTENSION;
        INT;
        DEFAULT = 4;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Number of PMF restarts allowed for fault monitor.";
}
{
```

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

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

Agent Builder는 대부분의 데이터 서비스에 유용한 다음과 같은 확장 등록 정보를 만듭니다.

`Confdir_list`
: 많은 응용 프로그램에 유용한 정보인 응용 프로그램 구성 디렉토리에 대한 경로를 지정합니다. 클러스터 관리자는 데이터 서비스를 구성할 때 이 디렉토리의 위치를 지정할 수 있습니다.

`Monitor_retry_count`, `Monitor_retry_interval` 및 `Probe_timeout`
: 서버 데몬이 아니라 오류 모니터 자체의 재시작을 제어합니다.

`Child_mon_level`
: PMF에서 수행할 모니터링 수준을 설정합니다. 자세한 내용은 `pmfadm`(1M) 설명서 페이지를 참조하십시오.

`User added code` 주석으로 구분된 영역에 추가 확장 등록 정보를 만들 수 있습니다.

콜백 메소드 구현

이 절에서는 콜백 메소드 구현에 관련된 일반적인 정보를 제공합니다.

자원 및 자원 그룹 등록 정보 액세스

일반적으로 콜백 메소드는 자원 등록 정보 액세스해야 합니다. RMAPI에서는 자원의 시스템 정의 등록 정보와 확장 등록 정보를 액세스하기 위해 콜백 메소드에서 사용할 수 있는 쉘 명령과 C 함수를 모두 제공합니다. `scha_resource_get`(1HA) 및 `scha_resource_get`(3HA) 설명서 페이지를 참조하십시오.

DSDL에서는 시스템 정의 등록 정보를 액세스하기 위한 C 함수 세트(각 등록 정보마다 하나의 함수)와 확장 등록 정보를 액세스하기 위한 함수를 제공합니다. `scds_property_functions`(3HA) 및 `scds_get_ext_property`(3HA) 설명서 페이지를 참조하십시오.

`Status` 및 `Status_msg` 외에는 자원 등록 정보 설정을 위해 사용할 수 있는 API 함수가 없기 때문에 등록 정보 기법을 사용하여 데이터 서비스의 동적 상태 정보를 저장할 수 없습니다. 그보다는 전역 파일에 동적 상태 정보를 저장해야 합니다.

주 – 클러스터 관리자는 `scrgadm` 명령을 사용하거나 사용 가능한 그래픽 관리 명령 또는 인터페이스를 통해 특정한 자원 등록 정보를 설정할 수 있습니다. 그러나 클러스터 재구성 도중, 즉 RGM에서 메소드를 호출할 때 `scrgadm`이 실패하므로 콜백 메소드에서 `scrgadm`을 호출하지 마십시오.

메소드의 멱등원

일반적으로 RGM은 동일한 인자를 가진 동일한 자원에 연속해서 둘 이상의 메소드를 호출하지 않습니다. 그러나 `Start` 메소드가 실패할 경우 RGM은 자원이 한 번도 시작되지 않았더라도 자원에 대한 `Stop` 메소드를 호출할 수 있습니다. 마찬가지로 자원 데몬이 저절로 소멸된 경우에도 RGM은 이 자원 데몬에 대해 `Stop` 메소드를 실행할 수 있습니다. 동일한 시나리오가 `Monitor_start` 및 `Monitor_stop` 메소드에 적용됩니다.

이러한 이유로 `Stop` 및 `Monitor_stop` 메소드에 멱등원을 작성해야 합니다. 동일한 인자를 가진 동일한 자원에 대해 `Stop` 또는 `Monitor_stop`을 반복해서 호출할 경우 단일 호출과 동일한 결과가 만들어집니다.

멱등원에 함축된 한 가지 의미는 자원 또는 모니터가 이미 중지되고 아무런 작업을 수행하지 않더라도 `Stop` 및 `Monitor_stop`에서 0(성공)을 반환해야 한다는 것입니다.

주 – `Init`, `Fini`, `Boot` 및 `Update` 메소드도 멱등원이어야 합니다. `Start` 메소드는 멱등원일 필요가 없습니다.

일반 데이터 서비스

일반 데이터 서비스(GDS)는 단순 응용 프로그램을 Sun Cluster의 Resource Group Manager 프레임워크에 연결하여 가용성을 높이고 확장 가능하게 만드는 기법입니다. 이 기법에서는 일반적인 방법으로 응용 프로그램의 가용성을 높이거나 확장 가능하게 만들 때처럼 데이터 서비스를 코딩할 필요가 없습니다.

GDS 모델은 RGM 프레임워크와 상호 작용하기 위해 사전 컴파일된 자원 유형인 `SUNW.gds`를 사용합니다. 자세한 내용은 10 장을 참조하십시오.

응용 프로그램 제어

콜백 메소드를 사용하면 RGM에서 노드가 클러스터에 결합되거나 클러스터를 떠날 때마다 기본 자원(즉, 응용 프로그램)을 제어할 수 있습니다.

자원 시작 및 중지

자원 유형 구현에는 최소한 `Start` 메소드와 `Stop` 메소드가 필요합니다. RGM은 자원 그룹을 오프라인 및 온라인 상태로 전환하기 위해 적절한 시간에 적절한 노드에서 자원 유형의 메소드 프로그램을 호출합니다. 예를 들어, 클러스터 노드의 충돌 후 RGM은 해당 노드에서 마스터하는 모든 자원 그룹을 새 노드로 이동합니다. `Start` 메소드를 구현하여 RGM에 남은 호스트 노드에서 각 자원을 다시 시작하는 방법을 제공합니다.

로컬 노드에서 자원을 시작하고 사용할 수 있을 때까지 `Start` 메소드를 반환하면 안 됩니다. 긴 초기화 기간이 필요한 자원 유형의 `Start` 메소드에 설정된 시간 초과값이 충분히 긴지 확인합니다. 시간 초과값이 충분하도록 RTR 파일에서 `Start_timeout` 등록 정보에 대한 기본값과 최소값을 설정합니다.

RGM이 자원 그룹을 오프라인으로 만드는 상황에 `Stop` 메소드를 구현해야 합니다. 예를 들어, 자원 그룹이 노드 1에서 오프라인 상태가 되고 노드 2에서 다시 온라인 상태가 되는 것으로 가정합니다. 자원 그룹을 오프라인 상태로 만드는 동안 RGM은 해당 그룹의 자원에 대해 `Stop` 메소드를 호출하여 노드 1의 모든 활동을 중지합니다. 노드 1에서 모든 자원에 대해 `Stop` 메소드가 완료되면 RGM은 노드 2에서 자원 그룹을 다시 온라인 상태로 만듭니다.

해당 자원이 로컬 노드에서 모든 활동을 완전히 중지하고 종료될 때까지 Stop 메소드를 반환하면 안 됩니다. Stop 메소드의 가장 안전한 구현에서는 자원과 관련된 로컬 노드의 모든 프로세스를 종료합니다. 종료하는 데 긴 시간이 필요한 자원 유형은 Stop 메소드에 설정된 시간 초과값이 충분히 길어야 합니다. RTR 파일에 Stop_timeout 등록 정보를 설정합니다.

Stop 메소드가 실패하거나 시간 초과되면 자원 그룹은 클러스터 관리자의 개입이 필요한 오류 상태가 됩니다. 이런 상태를 방지하려면 Stop 및 Monitor_stop 메소드 구현에서 가능한 모든 오류 조건을 복구해야 합니다. 이상적으로는 이 메소드가 0 (성공) 오류 상태로 종료되어야 합니다. 이 상태는 로컬 노드에 있는 자원과 자원 모니터의 모든 활동을 성공적으로 중지한 상태입니다.

사용할 Start 및 Stop 메소드 결정

이 절에서는 Prenet_start 및 Postnet_stop 메소드 사용과 비교하여 Start 및 Stop 메소드 사용 시기에 대한 설명을 제공합니다. 사용할 올바른 메소드를 결정하려면 클라이언트와 데이터 서비스의 클라이언트-서버 네트워킹 프로토콜에 대해 모두 잘 알아야 합니다.

네트워크 주소 자원을 사용하는 서비스에서는 논리 호스트 이름 주소 구성과 관련된 특정 순서로 시작 또는 중지 단계를 수행해야 합니다. 선택적인 콜백 메소드 Prenet_start 및 Postnet_stop을 사용하면 동일한 자원 그룹의 네트워크 주소가 활성 또는 비활성으로 구성되기 전후에 자원 유형 구현에서 특수한 시작 및 종료 작업을 수행할 수 있습니다.

RGM은 데이터 서비스의 Prenet_start 메소드를 호출하기 전에 네트워크 주소를 연결하지만 네트워크 주소를 활성으로 구성하지 않는 메소드를 호출합니다. RGM은 데이터 서비스의 Postnet_stop 메소드를 호출한 후 네트워크 주소를 연결 해제하는 메소드를 호출합니다. RGM이 자원 그룹을 온라인 상태로 전환하는 순서는 다음과 같습니다.

1. 네트워크 주소를 연결합니다.
2. 데이터 서비스의 Prenet_start 메소드를 호출합니다(있는 경우).
3. 네트워크 주소를 활성으로 구성합니다.
4. 데이터 서비스의 Start 메소드를 호출합니다(있는 경우).

RGM이 자원 그룹을 오프라인 상태로 전환하는 순서는 그 반대입니다.

1. 데이터 서비스의 Stop 메소드를 호출합니다(있는 경우).
2. 네트워크 주소를 비활성으로 구성합니다.
3. 데이터 서비스의 Postnet_stop 메소드를 호출합니다(있는 경우).
4. 네트워크 주소를 연결 해제합니다.

Start, Stop, Prenet_start 또는 Postnet_stop 메소드 사용 여부를 결정할 때에는 먼저 서버측을 고려하십시오. 데이터 서비스 응용 프로그램 자원과 네트워크 주소 자원을 포함하는 자원 그룹을 온라인 상태로 전환할 경우 RGM에서는 데이터 서비스 자원 Start 메소드를 호출하기 전에 네트워크 주소를 활성으로 구성하는 메소드를 호출합니다. 따라서 데이터 서비스가 시작될 때 네트워크 주소를 활성으로 구성해야 하는 경우에는 Start 메소드를 사용하여 데이터 서비스를 시작합니다.

마찬가지로 데이터 서비스 자원과 네트워크 주소 자원을 모두 포함하는 자원 그룹을 오프라인 상태로 전환할 경우 RGM에서는 데이터 서비스 자원 Stop 메소드를 호출한 후 네트워크 주소를 비활성으로 구성하는 메소드를 호출합니다. 따라서 데이터 서비스가 중지될 때 네트워크 주소를 활성으로 구성해야 하는 경우에는 Stop 메소드를 사용하여 데이터 서비스를 중지합니다.

예를 들어, 데이터 서비스를 시작하거나 중지하려면 데이터 서비스의 관리 유틸리티나 라이브러리를 실행해야 할 수 있습니다. 경우에 따라 데이터 서비스에는 클라이언트-서버 네트워킹 인터페이스를 사용하여 관리를 수행하는 관리 유틸리티나 라이브러리가 있습니다. 즉, 관리 유틸리티에서 서버 데몬을 호출하기 때문에 관리 유틸리티나 라이브러리를 사용하려면 네트워크 주소가 활성 상태여야 합니다. 이 시나리오에서는 Start 및 Stop 메소드를 사용합니다.

데이터 서비스가 시작 및 중지될 때 네트워크 주소를 비활성으로 구성해야 하는 경우 Prenet_start 및 Postnet_stop 메소드를 사용하여 데이터 서비스를 시작 및 중지하십시오. 클러스터 재구성 후 네트워크 주소 또는 데이터 서비스 중 어떤 것이 먼저 온라인 상태로 전환되는지에 따라 클라이언트 소프트웨어에서 다르게 응답하도록 할지를 고려합니다(SCHA_GIVEOVER 인자를 사용한 scha_control() 또는 scswitch를 사용한 스위치오버). 예를 들어, 클라이언트 구현에서는 최소한의 재시도를 수행하고 데이터 서비스 포트가 사용 불가능한지 확인한 후 바로 중지할 수 있습니다.

데이터 서비스를 시작할 때 네트워크 주소를 활성으로 구성할 필요가 없을 경우 네트워크 인터페이스를 활성으로 구성하기 전에 데이터 서비스를 시작합니다. 이런 방식으로 데이터 서비스를 시작하면 네트워크 주소가 활성으로 구성된 직후 데이터 서비스가 클라이언트 요청에 응답할 수 있으므로 클라이언트가 재시도를 중지할 가능성이 줄어듭니다. 이 시나리오에서는 Start 메소드 대신 Prenet_start 메소드를 사용하여 데이터 서비스를 시작합니다.

Postnet_stop 메소드를 사용할 경우 데이터 서비스 자원은 여전히 네트워크가 주소가 비활성으로 구성되는 지점에 활성화되어 있습니다. 네트워크 주소가 비활성으로 구성된 후에만 Postnet_stop 메소드가 실행됩니다. 그 결과, 네트워크 주소가 응답하지 않는 경우를 제외하고 데이터 서비스의 TCP 또는 UDP 서비스 포트 또는 해당 RPC 프로그램 번호가 항상 네트워크의 클라이언트에 사용 가능한 것으로 표시됩니다.

주 – 클러스터에 RPC 서비스를 설치할 경우 서비스에서 100141, 100142, 100248 등의 프로그램 번호를 사용해서는 안 됩니다. 이 번호는 각각 Sun Cluster 데몬 rgmd_receptionist, fed, pmfd용으로 예약되어 있습니다. 설치하는 RPC 서비스에서 이 프로그램 번호 중 하나를 사용할 경우 해당 RPC 서비스의 프로그램 번호를 변경합니다.

Prenet_start 및 Postnet_stop 메소드 대신 Start 및 Stop 메소드를 사용하거나 둘 다 사용하도록 결정하는 경우 서버와 클라이언트의 요구 사항과 동작을 모두 고려해야 합니다.

Init, Fini 및 Boot 메소드

세 개의 선택적 메소드, Init, Fini 및 Boot를 사용하면 RGM에서 자원에 대한 초기화 및 종료 코드를 실행할 수 있습니다.

RGM은 다음 조건 중 하나의 결과로 자원이 관리 상태가 될 경우 Init 메소드를 실행하여 자원의 일회성 초기화를 수행합니다.

- 자원이 속한 자원 그룹이 관리 해제 상태에서 관리 상태로 전환되는 경우
- 이미 관리되는 자원 그룹에 자원이 만들어지는 경우

RGM은 다음 조건 중 하나의 결과로 자원이 관리 해제 상태가 될 경우 Fini 메소드를 실행하여 자원을 정리합니다.

- 자원이 속한 자원 그룹이 관리 해제 상태로 전환되는 경우
- 관리 자원 그룹에서 자원이 삭제되는 경우

정리는 멱등원이어야 합니다. 즉, 정리가 이미 완료된 경우 Fini가 성공적으로 종료됩니다.

RGM은 새로 클러스터에 결합된 노드, 즉 부트되거나 재부트된 노드에서 Boot 메소드를 실행합니다.

Boot 메소드는 일반적으로 Init와 동일한 초기화를 수행합니다. 이 초기화는 멱등원이어야 합니다. 즉, 로컬 노드에서 자원을 이미 초기화한 경우 Boot 및 Init가 성공적으로 종료됩니다.

자원 모니터링

일반적으로 검사한 자원이 제대로 작동하는지 확인하기 위해 자원에 대해 정기적인 오류 검사를 수행하도록 모니터를 구현합니다. 오류 검사가 실패할 경우 모니터에서 로컬로 재시작하거나 scha_control() RMAPI 함수 또는 scds_fm_action() DSDL 함수를 호출하여 영향을 받는 자원 그룹의 페일오버를 요청할 수 있습니다.

자원의 성능을 모니터하고 성능을 조정 또는 보고할 수도 있습니다. 자원 유형별 오류 모니터 작성은 선택 사항입니다. 오류 모니터를 작성하도록 선택하지 않더라도 자원 유형은 Sun Cluster에서 수행하는 클러스터의 기본 모니터링의 혜택을 받을 수 있습니다. Sun Cluster는 공용 네트워크에서 통신할 수 있기 위해 호스트 하드웨어의 오류, 호스트 운영 체제의 전체 오류 및 호스트의 오류를 검색합니다.

RGM에서 자원 모니터를 직접 호출하지는 않지만 자원 모니터를 자동으로 시작하는 기능을 제공합니다. 자원을 오프라인 상태로 전환할 경우 RGM은 자원을 중지하기 전에 Monitor_stop 메소드를 호출하여 로컬 노드에서 자원의 모니터를 중지합니다. 자원을 온라인 상태로 전환할 경우 RGM은 자원이 시작된 후 Monitor_start 메소드를 호출합니다.

scha_control() RMAPI 함수 및 scds_fm_action() DSDL 함수(scha_control() 호출)를 사용하면 자원 모니터에서 자원 그룹을 다른 노드로 페일오버하도록 요청할 수 있습니다. 온전성 검사의 하나로 scha_control()은 Monitor_check(정의된 경우)을 호출하여 요청한 노드가 자원을 포함하는 자원 그룹을 마스터할 수 있을 만큼 충분히 신뢰할 수 있는지 확인합니다. Monitor_check에서 노드를 신뢰할 수 없다고 역보고하거나 메소드가 시간 초과한 경우 RGM은 다른 노드를 찾아 페일오버 요청을 수행합니다. Monitor_check가 모든 노드에서 실패하면 페일오버가 취소됩니다.

자원 모니터는 자원 상태의 모니터 뷰를 반영하도록 Status 및 Status_msg 등록 정보를 설정할 수 있습니다. scha_resource_setstatus() RMAPI 함수, scha_resource_setstatus 명령 또는 scds_fm_action() DSDL 함수를 사용하여 이 등록 정보를 설정합니다.

주 – Status 및 Status_msg 등록 정보는 자원 모니터에 대한 특수 용도로 사용되지만 모든 프로그램에서 이 등록 정보를 설정할 수 있습니다.

RMAPI로 구현되는 오류 모니터의 예는 91 페이지 "오류 모니터 정의"를 참조하십시오. DSDL로 구현되는 오류 모니터의 예는 134 페이지 "SUNW.xfnts 오류 모니터"를 참조하십시오. Sun에서 제공하는 데이터 서비스에 내장된 오류 모니터에 대한 정보는 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**를 참조하십시오.

자원에 메시지 로깅 추가

다른 클러스터 메시지와 동일한 로그 파일에 상태 메시지를 기록하려면 일반 함수 scha_cluster_getlogfacility()를 사용하여 클러스터 메시지 기록에 사용할 기능 번호를 검색합니다.

일반 Solaris syslog() 함수와 함께 이 기능 번호를 사용하여 클러스터 로그에 메시지를 작성합니다. 일반 scha_cluster_get() 인터페이스를 통해 클러스터 로그 기능 정보를 액세스할 수도 있습니다.

프로세스 관리 제공

RMAPI 및 DSDL에서는 자원 모니터와 자원 제어 콜백 구현을 위한 프로세스 관리 기능을 제공합니다. RMAPI에서는 다음과 같은 기능을 정의합니다.

Process Monitor Facility(PMF): `pmfadm` 및 `rpc.pmfd`
: 프로세스 및 하위 프로세스를 모니터하고 프로세스가 소멸될 경우 재시작하는 방법을 제공합니다. 이 기능은 모니터되는 프로세스를 시작 및 제어하기 위한 `pmfadm` 명령과 `rpc.pmfd` 데몬으로 구성됩니다.

  DSDL에서는 PMF 기능을 구현하기 위해 `scds_pmf_` 이름이 앞에 나오는 함수 집합을 제공합니다. DSDL PMF 기능 개요 및 개별 함수 목록은 193 페이지 "PMF 함수"를 참조하십시오.

  명령 및 데몬에 대한 자세한 내용은 `pmfadm`(1M) 및 `rpc.pmfd`(1M) 설명서 페이지를 참조하십시오.

`halockrun`
: 파일 잠금을 유지하는 동안 자식 프로그램을 실행하기 위한 프로그램. 이 명령은 쉘 스크립트에 사용하기에 편리합니다.

  이 명령에 대한 자세한 내용은 `halockrun`(1M) 설명서 페이지를 참조하십시오.

`hatimerun`
: 시간 초과 제어 하에 자식 프로그램을 실행하기 위한 프로그램. 이 명령은 쉘 스크립트에 사용하기에 편리합니다.

  DSDL에서는 `hatimerun` 기능을 구현하기 위해 `scds_hatimerun()` 함수를 제공합니다.

  이 명령에 대한 자세한 내용은 `hatimerun`(1M) 설명서 페이지를 참조하십시오.

자원 관리 지원 제공

클러스터 관리자가 자원에 대해 수행할 수 있는 작업에는 자원 등록 정보 설정 및 변경이 포함됩니다. API는 사용자가 이러한 관리 작업에 연결되는 코드를 만들 수 있도록 `Validate` 및 `Update` 콜백 메소드를 정의합니다.

자원을 만들 때 및 클러스터 관리자가 자원이나 자원을 포함하는 그룹의 등록 정보를 업데이트할 때 RGM에서 선택적인 `Validate` 메소드를 호출합니다. RGM은 자원과 자원 그룹의 등록 정보 값을 `Validate` 메소드에 전달합니다. RGM은 자원 유형의 `Init_nodes` 등록 정보로 표시된 클러스터 노드 집합에서 `Validate`를 호출합니다. `Init_nodes`에 대한 자세한 내용은 223 페이지 "자원 유형 등록 정보" 또는 `rt_properties`(5) 설명서 페이지를 참조하십시오. RGM은 만들기 또는 업데이트를 적용하기 전에 `Validate`를 호출합니다. 노드의 메소드에서 실패 종료 코드가 발생하면 만들기 또는 업데이트가 실패합니다.

RGM은 클러스터 관리자가 자원 또는 자원 그룹 등록 정보를 변경하거나(RGM에서 등록 정보를 설정하는 경우 제외) 모니터가 `Status` 및 `Status_msg` 자원 등록 정보를 설정하는 경우에만 `Validate`를 호출합니다.

RGM은 선택적인 Update 메소드를 호출하여 실행 중인 자원에 등록 정보가 변경되었음을 알립니다. 클러스터 관리자가 자원이나 자원 그룹의 등록 정보 설정에 성공한 후 Update가 실행됩니다. RGM은 자원이 온라인인 노드에서 이 메소드를 호출합니다. 이 메소드에서는 API 액세스 함수를 사용하여 활성 자원에 영향을 미치는 등록 정보 값을 읽고 그에 따라 실행 중인 자원을 조정할 수 있습니다.

페일오버 자원 구현

페일오버 자원 그룹에는 내장된 자원 유형 LogicalHostname 및 SharedAddress와 같은 네트워크 주소와 페일오버 데이터 서비스에 대한 데이터 서비스 응용 프로그램 자원과 같은 페일오버 자원이 포함됩니다. 데이터 서비스가 페일오버 또는 스위치오버된 경우 하위 데이터 서비스 자원과 함께 네트워크 주소 자원이 클러스터 노드 사이 간을 이동합니다. RGM에서는 페일오버 자원의 구현을 지원하는 여러 개의 등록 정보를 제공합니다.

부울 자원 유형 등록 정보 Failover를 TRUE로 설정하여 한 번에 둘 이상의 노드에서 온라인 상태가 될 수 있는 자원 그룹에 자원이 구성되지 못하도록 제한합니다. 이 등록 정보의 기본값은 FALSE이므로 페일오버 자원의 RTR 파일에서 TRUE로 선언해야 합니다.

Scalable 자원 등록 정보는 자원에서 클러스터 공유 주소 기능을 사용하는지 확인합니다. 페일오버 자원의 경우 페일오버 자원에서 공유 주소를 사용하지 않기 때문에 Scalable을 FALSE로 설정합니다.

클러스터 관리자는 RG_mode 자원 그룹 등록 정보를 사용하여 자원 그룹을 페일오버 또는 확장 가능한 것으로 식별할 수 있습니다. RG_mode가 FAILOVER일 경우 RGM은 그룹의 Maximum_primaries 등록 정보를 1로 설정하고 자원 그룹을 단일 노드에서 마스터하도록 제한합니다. RGM을 사용하면 Failover 등록 정보가 True인 자원을 RG_mode가 Scalable인 자원 그룹에 만들 수 없습니다.

Implicit_network_dependencies 자원 그룹 등록 정보에서는 RGM이 그룹 내 모든 네트워크 주소 자원(LogicalHostname 및 SharedAddress)에서 비네트워크 주소 자원의 강력한 암시적 종속성을 적용하도록 지정합니다. 그 결과, 그룹의 네트워크 주소를 활성으로 구성할 때까지 그룹의 비네트워크 주소(데이터 서비스) 자원의 Start 메소드가 호출되지 않습니다. Implicit_network_dependencies 등록 정보 기본값은 TRUE로 설정됩니다.

확장 가능한 자원 구현

확장 가능한 자원은 동시에 둘 이상의 노드에서 온라인 상태가 될 수 있습니다. 확장 가능한 자원에는 Sun Cluster HA for Sun Java System Web Server(이전의 Sun Cluster HA for Sun ONE Web Server) 및 Sun Cluster HA for Apache 같은 데이터 서비스가 포함됩니다.

RGM에서는 확장 가능한 자원의 구현을 지원하는 여러 개의 등록 정보를 제공합니다.

부울 자원 유형 등록 정보 `Failover`를 `FALSE`로 설정하여 한 번에 둘 이상의 노드에서 온라인 상태가 될 수 있는 자원 그룹에 자원이 구성될 수 있도록 합니다.

`Scalable` 자원 등록 정보는 자원에서 클러스터 공유 주소 기능을 사용하는지 확인합니다. 확장 가능한 서비스는 공유 주소 자원을 사용하여 확장 가능한 서비스의 여러 인스턴스가 클라이언트에 단일 서비스로 나타나도록 하기 때문에 이 등록 정보를 `TRUE`로 설정합니다.

`RG_mode` 등록 정보를 사용하여 클러스터 관리자는 자원 그룹을 페일오버나 확장 가능으로 식별할 수 있습니다. `RG_mode`가 `SCALABLE`일 경우 RGM을 사용하면 `Maximum_primaries`가 1보다 큰 값을 가질 수 있습니다. 이것은 여러 노드에서 그룹을 동시에 마스터할 수 있음을 의미합니다RGM을 사용하면 `Failover` 등록 정보가 `FALSE`인 자원을 `RG_mode`가 `SCALABLE`인 자원 그룹에서 인스턴스화할 수 있습니다.

클러스터 관리자는 확장 가능한 서비스 자원을 포함하는 확장 가능한 자원 그룹을 만들고, 확장 가능한 자원이 종속된 공유 주소 자원을 포함하는 별도의 페일오버 자원 그룹을 만듭니다.

클러스터 관리자는 `RG_dependencies` 자원 그룹 등록 정보를 사용하여 노드에서 자원 그룹을 온라인 및 오프라인 상태로 전환하는 순서를 지정합니다. 확장 가능한 자원과 확장 가능한 자원이 종속되는 공유 주소 자원은 서로 다른 자원 그룹에 있기 때문에 확장 가능 서비스에는 이 순서 지정이 중요합니다. 확장 가능한 데이터 서비스를 시작하기 전에 먼저 해당 네트워크 주소(공유 주소) 자원을 활성으로 구성해야 합니다. 따라서 클러스터 관리자는 확장 가능한 서비스를 포함하는 자원 그룹의 `RG_dependencies` 등록 정보를 공유 주소 자원이 있는 자원 그룹을 포함하도록 설정해야 합니다.

자원의 RTR 파일에서 `Scalable` 등록 정보를 선언할 경우 RGM에서는 자원에 대해 다음과 같은 확장 가능 등록 정보 집합을 자동으로 만듭니다.

`Network_resources_used`
: 이 자원에서 사용한 공유 주소 자원을 식별합니다. 이 등록 정보 기본값은 빈 문자열로 설정되므로 클러스터 관리자는 자원을 만들 때 확장 가능한 서비스에서 사용하는 공유 주소의 실제 목록을 제공해야 합니다. `scsetup` 명령 및 SunPlex Manager는 확장 가능한 서비스에 필요한 자원과 그룹을 자동으로 설정하는 기능을 제공합니다.

Load_balancing_policy
자원의 로드 균형 조정 정책을 지정합니다. RTR 파일에서 정책을 명시적으로 설정하거나 기본값 LB_WEIGHTED를 사용할 수 있습니다. 어떤 경우든 RTR 파일에서 Load_balancing_policy에 대한 Tunable을 NONE 또는 FALSE로 설정하지 않는 한 클러스터 관리자는 자원을 만들 때 이 값을 변경할 수 있습니다. 올바른 값은 다음과 같습니다.

LB_WEIGHTED
Load_balancing_weights 등록 정보에 설정된 가중치에 따라 여러 노드에 로드를 분산합니다.

LB_STICKY
확장 가능 서비스의 해당 클라이언트(클라이언트 IP 주소로 식별)는 항상 같은 클러스터 노드로 전송됩니다.

LB_STICKY_WILD
고정된 와일드카드 서비스의 IP 주소로 연결되는 해당 클라이언트(클라이언트의 IP 주소로 식별)는 들어오는 포트 번호에 관계없이 항상 같은 클러스터 노드로 전송됩니다.

LB_STICKY 또는 LB_STICKY_WILD의 Load_balancing_policy가 있는 확장 가능한 서비스의 경우 서비스가 온라인 상태일 때 Load_balancing_weights 를 변경하면 기존 클라이언트 유사성이 재설정될 수 있습니다. 이 경우 클러스터의 다른 노드에서 이전에 클라이언트에 서비스를 제공했더라도 다른 노드에서 후속 클라이언트 요청을 서비스할 수 있습니다.

마찬가지로 클러스터에서 서비스의 새로운 인스턴스를 시작하면 기존 클라이언트 유사성이 재설정될 수 있습니다.

Load_balancing_weights
각 노드에 전송할 로드를 지정합니다. 형식은 *weight@node,weight@node*입니다. 여기서 *weight*는 지정한 *node*로 분산된 로드의 상대적인 비율을 반영하는 정수입니다. 노드로 분산되는 로드의 분수는 이 노드에 대한 가중치를 활성 인스턴스의 모든 가중치의 합으로 나눈 것입니다. 예를 들어 1@1,3@2는 노드 1에서 1/4의 로드를 받고 노드 2에서 3/4의 로드를 받도록 지정합니다.

Port_list
서버가 수신하고 있는 포트를 식별합니다. 이 등록 정보의 기본값은 빈 문자열입니다. RTR 파일에서 포트 목록을 제공할 수 있습니다. 그렇지 않으면 클러스터 관리자는 자원을 만들 때 실제 포트 목록을 제공해야 합니다.

클러스터 관리자가 확장 가능 또는 페일오버로 구성할 수 있는 데이터 서비스를 만들 수 있습니다. 그렇게 하려면 데이터 서비스의 RTR 파일에서 Failover 자원 유형 등록 정보와 Scalable 자원 유형을 FALSE로 선언합니다. 만들 때 조정 가능한 Scalable 등록 정보를 지정합니다.

Failover 등록 정보 값 FALSE를 사용하면 자원을 확장 가능한 자원 그룹에 구성할 수 있습니다. 클러스터 관리자는 자원을 만들 때 Scalable의 값을 TRUE로 변경함으로써 공유 주소를 사용 가능하게 하여 확장 가능한 서비스를 만들 수 있습니다.

한편 Failover를 FALSE로 설정하더라도 클러스터 관리자는 자원을 페일오버 자원 그룹으로 구성하여 페일오버 서비스를 구현할 수 있습니다. 클러스터 관리자는 FALSE인 Scalable의 값을 변경하지 않습니다. 이 시나리오를 지원하려면 Scalable 등록 정보에서 Validate 메소드를 선택해야 합니다. Scalable이 FALSE일 경우 자원이 페일오버 자원 그룹으로 구성되었는지 확인합니다.

확장 가능한 자원에 대한 추가 정보는 **Solaris OS용 Sun Cluster 개념 안내서**를 참조하십시오.

확장 가능 서비스에 대한 검증 검사

확장 가능한 등록 정보를 TRUE로 설정하여 자원을 만들거나 업데이트할 경우 RGM에서 다양한 자원 등록 정보를 검증합니다. 등록 정보가 제대로 구성되지 않은 경우 RGM은 시도된 업데이트나 만들기를 거부합니다. RGM에서 다음과 같은 검사를 수행합니다.

- Network_resources_used 등록 정보는 비워둘 수 없으며 기존 공유 주소 자원 이름을 포함해야 합니다. 확장 가능한 자원을 포함하는 자원 그룹의 Nodelist에 있는 모든 노드가 명명된 공유 주소 자원 각각의 NetIfList 등록 정보 또는 AuxNodeList 등록 정보에 나타나야 합니다.
- 확장 가능한 자원을 포함하는 자원 그룹의 RG_dependencies 등록 정보에는 확장 가능한 자원의 Network_resources_used 등록 정보에 나열된 모든 공유 주소 자원의 자원 그룹이 포함되어야 합니다.
- Port_list 등록 정보가 비어 있지 않고 포트-프로토콜 쌍의 목록을 포함해야 합니다. 각 포트 번호에 슬래시(/)를 추가하고 해당 포트에서 사용하는 프로토콜이 뒤에 와야 합니다. 예를 들면 다음과 같습니다.

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

  지정할 수 있는 프로토콜 값은 다음과 같습니다.

  - tcp, TCP IPv4의 경우
  - tcp6, TCP IPv6의 경우
  - udp, UDP IPv4의 경우
  - udp6, UDP IPv6의 경우

데이터 서비스 쓰기 및 테스트

TCP 연결 유지를 사용하여 서버 보호

서버측에서는 TCP 연결 유지를 사용하여 다운 또는 네트워크 분할된 클라이언트에 대해 시스템 자원이 낭비되지 않도록 합니다. 오랫동안 유지된 서버에서 이 자원이 정리되지 않을 경우 클라이언트 충돌 및 재부트 시 자원 낭비가 무한대로 증가합니다.

클라이언트-서버 통신에서 TCP 스트림을 사용할 경우 클라이언트와 서버 모두에서 TCP 연결 유지 기법을 사용해야 합니다. 비HA 단일 서버의 경우에도 마찬가지입니다.

다른 연결 지향 프로토콜에도 연결 유지 기법이 있을 수 있습니다.

클라이언트측에서 TCP 연결 유지를 사용하면 물리적 호스트 간에 네트워크 주소 자원이 페일오버되거나 스위치오버된 경우 클라이언트에게 이를 알릴 수 있습니다. 네트워크 주소 자원을 전송하면 TCP 연결이 끊어집니다. 그러나 클라이언트에서 연결 유지를 사용하지 않는 한, TCP 연결이 끊어지더라도 이러한 사실을 알 수 없습니다.

예를 들어, 클라이언트가 장기간 실행 중인 요청에 대한 서버의 응답을 기다리고 있고, 클라이언트의 요청 메시지는 이미 서버에 도착하여 TCP 계층에 인식된 것으로 가정합니다. 이 경우 클라이언트의 TCP 모듈은 요청을 계속 재전송할 필요가 없으며 클라이언트 응용 프로그램이 차단된 상태로 요청에 대한 응답을 기다립니다.

가능한 경우 클라이언트 응용 프로그램은 TCP 연결 유지 기법을 사용하는 것 외에도 응용 프로그램 수준에서 자체적으로 연결 유지를 주기적으로 수행해야 합니다. 모든 상황에서 TCP 연결 유지 기법이 완벽한 것은 아니기 때문입니다. 일반적으로 응용 프로그램 수준의 연결 유지를 사용하면 클라이언트-서버 프로토콜에서 null 작업을 지원하거나 최소한 상태 작업과 같은 효율적인 읽기 전용 작업을 지원해야 합니다.

HA 데이터 서비스 테스트

이 절에서는 HA 환경에서 데이터 서비스 구현을 테스트하는 방법을 제안합니다. 테스트 사례는 제안 사항이며 모든 경우를 다 적어 놓은 것은 아닙니다. 테스트 기반 Sun Cluster 구성에 액세스하여 테스트 작업이 작업 시스템에 영향을 미치지 않도록 해야 합니다.

자원 그룹이 물리적 호스트 간에 이동되는 모든 경우에 HA 데이터 서비스가 올바로 동작하는지 테스트합니다. 이러한 경우에는 시스템 충돌과 `scswitch` 명령 사용이 포함됩니다. 이러한 이벤트 발생 후에도 클라이언트 시스템을 계속 사용할 수 있는지 테스트합니다.

메소드의 멱등원을 테스트합니다. 예를 들어, 원래 메소드를 두 번 이상 호출하는 간단한 쉘 스크립트로 각 메소드를 임시로 교체합니다.

자원 간 종속성 조정

경우에 따라 하나의 클라이언트-서버 데이터 서비스에서 클라이언트에 대한 요청을 이행하면서 다른 클라이언트-서버 데이터 서비스에 대한 요청을 합니다. 예를 들어, A가 서비스를 제공하기 위해 B가 서비스를 제공해야 하는 경우 데이터 서비스 A는 데이터 서비스 B에 종속됩니다. Sun Cluster에서는 자원 그룹내에서 자원 종속성을 구성할 수 있도록 허용하여 이 요구 사항을 제공합니다. 종속성은 Sun Cluster에서 데이터 서비스를 시작 및 중지하는 순서에 영향을 미칩니다. 자세한 내용은 `scrgadm`(1M) 설명서 페이지를 참조하십시오.

자원 유형의 자원이 다른 유형의 자원에 종속될 경우 클러스터 관리자에게 자원과 자원 그룹을 적절하게 구성하도록 지시하거나, 올바로 구성하도록 스크립트나 도구를 제공해야 합니다. 종속 자원을 "종속 대상" 자원과 동일한 노드에서 실행해야 할 경우 두 자원을 모두 동일한 자원 그룹에 구성해야 합니다.

명시적인 자원 종속성을 사용할지 또는 HA 데이터 서비스의 고유한 코드에서 다른 데이터 서비스의 가용성에 대해 명시적인 자원 종속성을 생략하고 폴링할지 여부를 결정합니다. 종속 및 종속 대상 자원을 서로 다른 노드에서 실행할 수 있는 경우 이들을 별도의 자원 그룹에 구성합니다. 이 경우 전체 그룹에 자원 종속성을 구성할 수 없기 때문에 폴링이 필요합니다.

일부 데이터 서비스에서는 직접 데이터를 저장하지 않지만, 대신 다른 백엔드 데이터 서비스에 종속되어 모든 데이터를 저장합니다. 그런 데이터 서비스에서는 모든 읽기 및 업데이트 요청을 백엔드 데이터 서비스의 호출로 변환합니다. 예를 들어, 모든 데이터를 Oracle과 같은 SQL 데이터베이스에 보관하는 가상의 클라이언트-서버 약속 캘린더 서비스를 고려해 보십시오. 약속 캘린더 서비스에는 고유한 클라이언트-서버 네트워크 프로토콜이 있습니다. 예를 들어, ONC RPC 같은 RPC 사양 언어를 사용하여 해당 프로토콜을 정의했을 수 있습니다.

Sun Cluster 환경에서 HA-ORACLE을 사용하여 백엔드 Oracle 데이터베이스의 가용성을 높일 수 있습니다. 그런 다음 약속 캘린더 데몬을 시작 및 중지하기 위한 간단한 메소드를 작성할 수 있습니다. 클러스터 관리자가 약속 캘린더 자원 유형을 Sun Cluster에 등록합니다.

약속 캘린더 응용 프로그램을 Oracle 데이터베이스와 동일한 노드에서 실행해야 할 경우 클러스터 관리자가 HA-ORACLE 자원과 동일한 자원 그룹에 약속 캘린더 자원을 구성하고, 약속 캘린더 자원을 HA-ORACLE 자원에 종속되게 합니다. 이 종속성은 `scrgadm`의 `Resource_dependencies` 등록 정보 태그를 사용하여 지정합니다.

HA-ORACLE 자원을 약속 캘린더 자원과 다른 노드에서 실행할 수 있는 경우 클러스터 관리자는 두 개의 별도 자원 그룹에 이를 구성하고 Oracle 자원 그룹에 캘린더 자원 그룹의 자원 그룹 종속성을 구성할 수 있습니다. 그러나 자원 그룹 종속성은 자원 그룹을 모두 동일한 시간에 동일한 노드에서 시작 또는 중지할 경우에만 유효합니다. 따라서 캘린더 데이터 서비스 데몬을 시작한 후 Oracle 데이터베이스가 사용 가능하게 될 때까지 대기하도록 폴링할 수 있습니다. 캘린더 자원 유형의 `Start` 메소드에서는 대개 이 경우 성공을 반환합니다. 그러나 `Start` 메소드가 무기한 차단된 경우 이 메소드는 자원 그룹을 사용 중 상태로 만듭니다. 사용 중 상태에서는 그룹의 상태 변경(편집, 페일오버 또는 스위치오버)이 차단됩니다. 그러나 캘린더 자원의 `Start` 메소드가 시간 초과되거나 0 이외의 상태로 종료된 경우 Oracle 데이터베이스를 사용할 수 없는 동안에는 둘 이상의 노드 간에 자원 그룹이 핑퐁될 수 있습니다.

3장

자원 관리 API 참조

이 장에서는 자원 관리 API(RMAPI)를 구성하는 액세스 함수와 콜백 메소드를 나열하고 간략하게 설명합니다. 그러나 이 함수와 메소드에 대한 최종적인 참조서는 RMAPI 설명서 페이지입니다.

이 장은 다음 내용으로 구성되어 있습니다.

RMAPI 액세스 메소드

API에서는 자원 유형, 자원, 자원 그룹 등록 정보 및 기타 클러스터 정보를 액세스하는 함수를 제공합니다. 이 함수가 쉘 명령과 C 함수 형식 둘 다로 제공되므로 개발자는 제어 프로그램을 쉘 스크립트나 C 프로그램으로 구현할 수 있습니다.

RMAPI 쉘 명령

쉘 명령은 클러스터의 RGM에서 제어하는 서비스를 나타내는 자원 유형에 대한 콜백 메소드의 쉘 스크립트 구현에 사용됩니다. 이 명령을 사용하여 다음 작업을 수행할 수 있습니다.

- 자원, 자원 유형, 자원 그룹 및 클러스터에 대한 정보 액세스
- 모니터와 함께 사용하여 자원의 `Status` 및 `Status_msg` 등록 정보 설정
- 자원 그룹의 재시작이나 재배치 요청

주 – 이 절에서도 쉘 명령에 대한 간단한 설명을 제공하지만, 쉘 명령에 대한 최종적인 참조는 1HA 설명서 페이지에서 제공합니다. 다른 설명이 없는 한 모든 명령에는 동일한 이름의 설명서 페이지가 있습니다.

RMAPI 자원 명령

이 명령을 사용하여 자원에 대한 정보를 액세스하거나 자원의 `Status` 및 `Status_msg` 등록 정보를 설정할 수 있습니다.

`scha_resource_get`
RGM에서 제어하는 자원이나 자원 유형에 대한 정보를 액세스합니다. 이 명령은 `scha_resource_get()` C 함수와 동일한 정보를 제공합니다. 자세한 내용은 `scha_resource_get`(1HA) 설명서 페이지를 참조하십시오.

`scha_resource_setstatus`
RGM이 제어하는 자원의 `Status` 및 `Status_msg` 등록 정보를 설정합니다. 자원 모니터는 이 명령을 사용하여 모니터에서 인식한 자원 상태를 표시합니다. 이 명령은 `scha_resource_setstatus()` C 함수와 동일한 기능을 제공합니다. 이 명령에 대한 자세한 내용은 `scha_resource_setstatus`(1HA) 설명서 페이지를 참조하십시오.

주 – `scha_resource_setstatus()`가 자원 모니터에 특별히 사용되지만 모든 프로그램에서 이를 호출할 수 있습니다.

자원 유형 명령

`scha_resourcetype_get`
RGM에 등록된 자원 유형에 대한 정보를 액세스합니다. 이 명령은 `scha_resourcetype_get()` C 함수와 동일한 기능을 제공합니다. 이 명령에 대한 자세한 내용은 `scha_resourcetype_get`(1HA) 설명서 페이지를 참조하십시오.

자원 그룹 명령

이 명령을 사용하여 자원 그룹에 대한 정보를 액세스하거나 자원 그룹을 재시작할 수 있습니다.

`scha_resourcegroup_get`
RGM에서 제어하는 자원 그룹에 대한 정보를 액세스합니다. 이 명령은 `scha_resourcetype_get()` C 함수와 동일한 기능을 제공합니다. 이 명령에 대한 자세한 내용은 `scha_resourcegroup_get`(1HA) 설명서 페이지를 참조하십시오.

scha_control
RGM이 제어하는 자원 그룹의 재시작이나 다른 노드로의 재배치를 요청합니다. 이 명령은 scha_control() C 함수와 동일한 기능을 제공합니다. 이 명령에 대한 자세한 내용은 scha_control(1HA) 설명서 페이지를 참조하십시오.

클러스터 명령

scha_cluster_get
클러스터 이름, 노드 이름, ID, 상태, 자원 그룹 등의 클러스터에 대한 정보를 액세스합니다. 이 명령은 scha_cluster_get() C 함수와 동일한 정보를 제공합니다. 이 명령에 대한 자세한 내용은 scha_cluster_get(1HA) 설명서 페이지를 참조하십시오.

C 함수

C 함수는 클러스터의 RGM에서 제어하는 서비스를 나타내는 자원 유형에 대한 콜백 메소드의 C 프로그램 구현에 사용됩니다. 이 함수를 사용하여 다음 작업을 수행할 수 있습니다.

- 자원, 자원 유형, 자원 그룹 및 클러스터에 대한 정보 액세스
- 모니터와 함께 사용하여 자원의 Status 및 Status_msg 등록 정보 설정
- 자원 그룹의 재시작이나 재배치 요청
- 오류 코드를 적절한 오류 메시지로 변환

주 – 이 절에서도 C 함수에 대한 간단한 설명을 제공하지만, C 함수에 대한 최종적인 참조는 3HA 설명서 페이지에서 제공합니다. 다른 설명이 없는 한 모든 함수에는 동일한 이름의 설명서 페이지가 있습니다. C 함수의 출력 인자와 반환 코드에 대한 자세한 내용은 scha_calls(3HA) 설명서 페이지를 참조하십시오.

자원 함수

이 함수는 RGM에서 관리하는 자원에 대한 정보를 액세스하고 모니터에서 인식한 자원 상태를 표시합니다.

scha_resource_open(), scha_resource_get () 및 scha_resource_close()
이 함수는 RGM에서 관리하는 자원에 대한 정보를 액세스합니다.
scha_resource_open() 함수는 자원에 대한 액세스를 초기화하고 자원 정보를 액세스하는 scha_resource_get()에 대한 핸들을 반환합니다.
scha_resource_close() 함수는 이 핸들을 무효화하고 scha_resource_get() 반환 값에 할당된 메모리를 해제합니다.

scha_resource_open()에서 자원 핸들을 반환한 후 클러스터 재구성이나 관리 작업을 통해 자원을 변경할 수 있습니다. 그 결과, scha_resource_get()이 핸들을 통해 가져온 정보가 부정확할 수 있습니다. 자원에 대한 클러스터 재구성이나 관리 작업의 경우 RGM은 scha_err_seqid 오류 코드를

scha_resource_get()으로 반환하여 자원 정보가 변경되었을 수 있음을 나타냅니다. 이것은 치명적인 오류 메시지가 아닙니다. 함수가 성공적으로 반환됩니다. 메시지를 무시하고 반환된 정보를 적용하도록 선택하거나, 현재 핸들을 닫고 새 핸들을 열어서 자원 정보에 액세스할 수 있습니다.

하나의 설명서 페이지에서 세 함수를 모두 설명합니다. 개별 함수 scha_resource_open(3HA), scha_resource_get(3HA) 또는 scha_resource_close(3HA)를 통해 이 설명서 페이지에 액세스할 수 있습니다.

scha_resource_setstatus()
RGM이 제어하는 자원의 Status 및 Status_msg 등록 정보를 설정합니다. 자원의 모니터에서 이 함수를 사용하여 자원 상태를 나타냅니다.

주 – scha_resource_setstatus()가 자원 모니터에 특별히 사용되지만 모든 프로그램에서 이를 호출할 수 있습니다.

자원 유형 함수

이 함수는 RGM에 등록된 자원 유형에 대한 정보를 액세스합니다.

scha_resourcetype_open(), scha_resourcetype_get () 및 scha_resourcetype_close()
scha_resourcetype_open() 함수는 자원에 대한 액세스를 초기화하고 자원 유형 정보를 액세스하는 scha_resourcetype_get()에 대한 핸들을 반환합니다. scha_resourcetype_close() 함수는 이 핸들을 무효화하고 scha_resourcetype_get() 반환 값에 할당된 메모리를 해제합니다.

scha_resourcetype_open()에서 자원 유형 핸들을 반환한 후 클러스터 재구성이나 관리 작업을 통해 자원 유형을 변경할 수 있습니다. 그 결과, scha_resourcetype_get()이 핸들을 통해 가져온 정보가 부정확할 수 있습니다. 자원 유형에 대한 클러스터 재구성이나 관리 작업의 경우 RGM은 scha_err_seqid 오류 코드를 scha_resourcetype_get()으로 반환하여 자원 유형 정보가 변경되었을 수 있음을 나타냅니다. 이것은 치명적인 오류 메시지가 아닙니다. 함수가 성공적으로 반환됩니다. 메시지를 무시하고 반환된 정보를 적용하도록 선택하거나, 현재 핸들을 닫고 새 핸들을 열어서 자원 유형 정보에 액세스할 수 있습니다.

하나의 설명서 페이지에서 세 함수를 모두 설명합니다. 개별 함수 scha_resourcetype_open(3HA), scha_resourcetype_get(3HA) 또는 scha_resourcetype_close(3HA)를 통해 이 설명서 페이지에 액세스할 수 있습니다.

자원 그룹 함수

이 함수를 사용하여 자원 그룹에 대한 정보를 액세스하거나 자원 그룹을 다시 시작할 수 있습니다.

`scha_resourcegroup_open()`, `scha_resourcegroup_get ()` 및 `scha_resourcegroup_close()`
이 함수는 RGM에서 관리하는 자원 그룹에 대한 정보를 액세스합니다. `scha_resourcegroup_open()` 함수에서 자원 그룹에 대한 액세스를 초기화하고 자원 그룹 정보를 액세스하는 `scha_resourcegroup_get()`에 대한 핸들을 반환합니다. `scha_resourcegroup_close()` 함수는 이 핸들을 무효화하고 `scha_resourcegroup_get()` 반환 값에 할당된 메모리를 해제합니다.

`scha_resourcegroup_open()`에서 자원 그룹 핸들을 반환한 후 클러스터 재구성이나 관리 작업을 통해 자원 그룹을 변경할 수 있습니다. 그 결과, `scha_resourcegroup_get()`이 핸들을 통해 가져온 정보가 부정확할 수 있습니다. 자원 그룹에 대한 클러스터 재구성이나 관리 작업의 경우 RGM은 `scha_err_seqid` 오류 코드를 `scha_resourcegroup_get()`으로 반환하여 자원 그룹 정보가 변경되었을 수 있음을 나타냅니다. 이것은 치명적인 오류 메시지가 아닙니다. 함수가 성공적으로 반환됩니다. 메시지를 무시하고 반환된 정보를 적용하도록 선택하거나, 현재 핸들을 닫고 새 핸들을 열어서 자원 그룹 정보에 액세스할 수 있습니다.

하나의 설명서 페이지에서 세 함수를 모두 설명합니다. 개별 함수 `scha_resourcegroup_open`(3HA), `scha_resourcegroup_get`(3HA) 및 `scha_resourcegroup_close`(3HA)를 통해 이 설명서 페이지에 액세스할 수 있습니다.

`scha_control()`
RGM이 제어하는 자원 그룹의 재시작이나 다른 노드로의 재배치를 요청합니다. 이 함수에 대한 자세한 내용은 `scha_control`(3HA) 설명서 페이지를 참조하십시오.

클러스터 함수

이 함수는 클러스터에 대한 정보를 액세스하거나 반환합니다.

`scha_cluster_open()`, `scha_cluster_get ()` 및 `scha_cluster_close()`
이 함수는 클러스터 이름, 노드 이름, ID, 상태, 자원 그룹 등의 클러스터에 대한 정보를 액세스합니다.

`scha_cluster_open()`에서 클러스터 핸들을 반환한 후 클러스터 재구성이나 관리 작업을 통해 클러스터를 변경할 수 있습니다. 그 결과, `scha_cluster_get()`이 핸들을 통해 가져온 정보가 부정확할 수 있습니다. 클러스터에 대한 클러스터 재구성이나 관리 작업의 경우 RGM은 `scha_err_seqid` 오류 코드를 `scha_cluster_get()`으로 반환하여 클러스터 정보가 변경되었을 수 있음을 나타냅니다. 이것은 치명적인 오류 메시지가 아닙니다. 함수가 성공적으로 반환됩니다. 메시지를 무시하고 반환된 정보를 적용하도록 선택하거나, 현재 핸들을 닫고 새 핸들을 열어서 클러스터 정보에 액세스할 수 있습니다.

하나의 설명서 페이지에서 세 함수를 모두 설명합니다. 개별 함수 `scha_cluster_open`(3HA), `scha_cluster_get`(3HA) 및 `scha_cluster_close`(3HA)를 통해 이 설명서 페이지에 액세스할 수 있습니다.

`scha_cluster_getlogfacility()`
클러스터 로그로 사용 중인 시스템 로그 기능의 수를 반환합니다. `syslog()` Solaris 함수와 함께 반환된 값을 사용하여 이벤트와 상태 메시지를 클러스터 로그에

기록합니다. 이 함수에 대한 자세한 내용은 `scha_cluster_getlogfacility`(3HA) 설명서 페이지를 참조하십시오.

`scha_cluster_getnodename()`
함수가 호출된 클러스터 노드 이름을 반환합니다. 이 함수에 대한 자세한 내용은 `scha_cluster_getnodename`(3HA) 설명서 페이지를 참조하십시오.

유틸리티 함수

이 함수는 오류 코드를 오류 메시지로 변환합니다.

`scha_strerror()`
`scha_` 함수 중 하나에서 반환한 오류 코드를 해당 오류 메시지로 변환합니다. `logger` 명령과 함께 이 함수를 사용하여 메시지를 Solaris 시스템 로그(`syslog`)에 기록합니다. 이 함수에 대한 자세한 내용은 `scha_strerror`(3HA) 설명서 페이지를 참조하십시오.

RMAPI 콜백 메소드

콜백 메소드는 자원 유형을 구현하기 위해 API에서 제공하는 주요 요소입니다. 콜백 메소드를 사용하면 노드 부트나 충돌과 같이 클러스터 구성원이 변경된 경우 RGM에서 클러스터 자원을 제어할 수 있습니다.

주 – 클라이언트 프로그램에서 클러스터 시스템의 HA 서비스를 제어하기 때문에 콜백 메소드는 수퍼유저 또는 이와 동등한 역할 권한을 가진 RGM에서 실행됩니다. 제한된 파일 소유권과 권한을 사용하여 이 메소드를 설치 및 관리합니다. 특히 이러한 메소드에 `bin` 또는 `root`와 같은 권한이 있는 소유자를 지정하되 쓰기 가능하도록 설정하지 마십시오.

이 절에서는 콜백 메소드 인자와 종료 코드를 설명하고 다음 범주의 콜백 메소드에 대해 설명합니다.

- 제어 및 초기화 메소드
- 관리 지원 메소드
- Net-relative 메소드
- 모니터 제어 메소드

주 – 이 절에서도 메소드 실행 지점 및 자원에 대한 예상 효과를 포함하여 콜백 메소드에 대한 간단한 설명을 제공하지만, 콜백 메소드에 대한 최종적인 참조서는 `rt_callbacks`(1HA) 설명서 페이지입니다.

콜백 메소드에 제공할 수 있는 인자

RGM은 다음과 같이 콜백 메소드를 실행합니다.

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

메소드는 `Start`, `Stop` 또는 다른 콜백으로 등록된 프로그램의 경로 이름입니다. 자원 유형의 콜백 메소드가 해당 등록 파일에 선언됩니다.

모든 콜백 메소드 인자는 다음과 같이 플래그가 지정된 값으로 전달됩니다.

- `-R`은 자원 인스턴스의 이름을 나타냅니다.
- `-T`는 자원 유형을 나타냅니다.
- `-G`는 자원이 구성된 그룹을 나타냅니다.

액세스 함수와 함께 인자를 사용하여 자원에 대한 정보를 검색합니다.

`Validate` 메소드는 메소드가 호출되는 자원 및 자원 그룹의 등록 정보 값을 포함하는 추가 인자와 함께 호출됩니다.

자세한 내용은 `scha_calls`(3HA) 설명서 페이지를 참조하십시오.

콜백 메소드 종료 코드

모든 콜백 메소드에는 자원 상태에 대한 메소드 호출의 영향을 지정하는 동일한 종료 코드가 있습니다. 이러한 종료 코드에 대한 자세한 내용은 `scha_calls`(3HA) 설명서 페이지를 참조하십시오. 두 가지 주요 종료 코드 범주는 다음과 같습니다.

- `0` – 메소드 성공
- `0` 이외의 모든 값 – 메소드 실패

RGM은 시간 초과나 코어 덤프와 같은 콜백 메소드 실행의 비정상적인 실패도 처리합니다.

메소드 구현은 각 노드에서 `syslog()`를 사용하여 실패 정보를 출력해야 합니다. `stdout` 또는 `stderr`에 기록된 출력은 현재 로컬 노드의 콘솔에 표시되더라도 사용자에게 전달된다고 보장할 수 없습니다.

제어 및 초기화 콜백 메소드

기본 제어 및 초기화 콜백 메소드에서 자원을 시작 및 중지합니다. 다른 메소드는 자원에서 초기화 및 종료 코드를 실행합니다.

`Start`
자원을 포함하는 자원 그룹이 클러스터 노드에서 온라인 상태로 전환되면 RGM은 해당 노드에서 이 메소드를 실행합니다. 이 메소드는 해당 노드에서 자원을 활성화합니다.

활성화한 자원이 시작되고 로컬 노드에서 사용할 수 있을 때까지 `Start` 메소드를 종료하면 안 됩니다. 따라서 종료하기 전에 `Start` 메소드에서 자원을 폴링하여 자원을 시작했는지 확인해야 합니다. 또한 이 메소드에 충분한 시간 초과값을 설정해야 합니다. 예를 들어, 데이터베이스 데몬과 같은 특정 자원은 시작하는 데 시간이 더 걸리기 때문에 메소드에 대한 시간 초과값이 커야 합니다.

RGM에서 `Start` 메소드의 실패에 응답하는 방법은 `Failover_mode` 등록 정보의 설정에 따라 다릅니다.

자원의 `Start` 메소드에 대한 시간 초과값은 자원 유형 등록(RTR) 파일의 `Start_timeout` 등록 정보에 의해 설정됩니다.

`Stop`
자원을 포함하는 자원 그룹이 클러스터 노드에서 오프라인 상태로 전환되면 RGM은 해당 노드에서 이 필수 메소드를 실행합니다. 이 메소드는 자원이 활성화된 경우 자원을 비활성화합니다.

메소드에서 제어하는 자원이 로컬 노드에서 모든 활동을 완전히 중지하고 모든 파일 설명자를 닫을 때까지 `Stop` 메소드를 종료하면 안 됩니다. 그렇지 않으면 RGM에서 실제로 활성 상태인 자원을 중지된 것으로 가정하기 때문에 데이터가 손상될 수 있습니다. 데이터 손상을 방지하는 가장 안전한 방법은 자원과 관련된 로컬 노드에서 모든 프로세스를 종료하는 것입니다.

종료하기 전에 `Stop` 메소드에서 자원을 폴링하여 중지되었는지 확인해야 합니다. 또한 이 메소드에 충분한 시간 초과값을 설정해야 합니다. 예를 들어, 데이터베이스 데몬과 같은 특정 자원은 중지하는 데 시간이 더 걸리기 때문에 메소드에 대한 시간 초과값이 커야 합니다.

RGM에서 `Stop` 메소드의 실패에 응답하는 방법은 `Failover_mode` 등록 정보의 설정에 따라 다릅니다. 230 페이지 “자원 등록 정보”를 참조하십시오.

자원의 `Stop` 메소드에 대한 시간 초과값은 RTR 파일의 `Stop_timeout` 등록 정보에 의해 설정됩니다.

`Init`
RGM은 자원이 관리 상태가 될 경우, 즉 자원 그룹이 관리 해제에서 관리 상태로 전환되거나 이미 관리 상태인 자원 그룹에 자원이 만들어질 경우 자원의 일회성 초기화를 수행하기 위해 이 선택적인 메소드를 실행합니다. 이 메소드는 `Init_nodes` 자원 등록 정보로 식별된 노드에서 호출됩니다.

`Fini`
RGM은 자원이 관리 해제 상태가 될 경우, 즉 자원 그룹이 관리 해제 상태로 전환되거나 관리 상태인 자원 그룹에서 자원이 삭제될 경우 정리를 위해 이 선택적인 메소드를 실행합니다. 이 메소드는 `Init_nodes` 자원 등록 정보로 식별된 노드에서 호출됩니다.

Boot
RGM은 자원을 포함하는 자원 그룹이 RGM에서 이미 관리되고 있을 때 클러스터에 결합된 노드에서 자원을 초기화하기 위해 Init와 유사한 이 선택적인 메소드를 실행합니다. 이 메소드는 Init_nodes 자원 등록 정보로 식별된 노드에서 실행됩니다. 부트 또는 재부트 결과로 노드가 클러스터에 결합되거나 재결합될 경우 Boot 메소드가 호출됩니다.

주 – Init, Fini 또는 Boot 메소드가 실패하면 syslog() 함수에서 오류 메시지를 생성하지만 RGM의 자원 관리에는 달리 영향을 주지 않습니다.

관리 지원 메소드

자원의 관리 작업에는 자원 등록 정보의 설정 및 변경이 포함됩니다. Validate 및 Update 콜백 메소드를 사용하면 자원 유형 구현에서 이러한 관리 작업을 수행할 수 있습니다.

Validate
RGM은 자원을 만들 때 및 클러스터 관리자가 자원이나 자원을 포함하는 자원 그룹의 등록 정보를 업데이트할 때 이 선택적인 메소드를 호출합니다. 이 메소드는 자원 유형의 Init_nodes 등록 정보로 식별된 클러스터 노드 집합에서 호출됩니다. Validate는 만들기나 업데이트가 적용되기 전에 호출됩니다. 노드에 메소드의 실패 종료 코드가 있으면 만들기나 업데이트가 취소됩니다.

Validate는 클러스터 관리자가 자원 또는 자원 그룹 등록 정보를 변경하는 경우에만 호출되며 RGM에서 등록 정보를 설정하는 경우나 모니터에서 Status 및 Status_msg 자원 등록 정보를 설정하는 경우에는 호출되지 않습니다.

Update
RGM은 실행 중인 자원에 등록 정보가 변경되었음을 알리기 위해 이 선택적인 메소드를 실행합니다. RGM은 관리 작업이 자원이나 자원 그룹의 등록 정보를 성공적으로 설정한 후 Update를 실행합니다. 이 메소드는 자원이 온라인 상태인 노드에서 호출되며 API 액세스 함수를 사용하여 활성 자원에 영향을 미치는 등록 정보 값을 읽고 실행 중인 자원을 적절하게 조정합니다.

주 – Update 메소드가 실패하면 syslog() 함수에서 오류 메시지를 생성하지만 RGM의 자원 관리에는 달리 영향을 주지 않습니다.

Net-Relative 콜백 메소드

네트워크 주소 자원을 사용하는 서비스는 네트워크 주소 구성에 따라 특정 순서로 시작 또는 중지 단계를 수행해야 합니다. 선택적 콜백 메소드인 Prenet_start 및 Postnet_stop을 사용하면 자원 유형 구현에서 관련된 네트워크 주소를 구성하거나 구성 해제하기 전후에 특수한 시작 및 종료 작업을 수행할 수 있습니다.

Prenet_start
동일한 자원 그룹의 네트워크 주소를 구성하기 전에 특수한 시작 작업을 수행하기 위해 이 선택적인 메소드를 호출합니다.

Postnet_stop
동일한 자원 그룹의 네트워크 주소를 비활성으로 구성한 후 특수한 종료 작업을 수행하기 위해 이 선택적인 메소드를 호출합니다.

모니터 제어 콜백 메소드

자원 유형 구현에는 선택적으로 자원의 성능을 모니터하거나, 자원 상태를 보고하거나, 자원 실패에 대한 작업을 수행하는 프로그램이 포함될 수 있습니다. Monitor_start, Monitor_stop 및 Monitor_check 메소드는 자원 유형 구현에서 자원 모니터의 구현을 지원합니다.

Monitor_start
자원을 시작한 후 자원에 대한 모니터를 시작하기 위해 이 선택적인 메소드를 호출합니다.

Monitor_stop
자원을 중지하기 전에 자원의 모니터를 중지하기 위해 이 선택적인 메소드를 호출합니다.

Monitor_check
자원 그룹을 노드에 재배치하기 전에 노드의 신뢰성을 액세스하기 이해 이 선택적인 메소드를 호출합니다. 다른 메소드와 동시에 실행할 때 충돌하지 않도록 Monitor_check 메소드를 구현해야 합니다.

4장

자원 유형 수정

이 장에서는 자원 유형을 수정하기 위해 알아 두어야 할 문제점에 대해 설명합니다. 클러스터 관리자의 자원 업그레이드를 허용하는 방법에 대한 정보도 포함되어 있습니다.

이 장은 다음 내용으로 구성되어 있습니다.

자원 유형 수정 개요

클러스터 관리자는 다음 작업을 수행할 수 있어야 합니다.

- 기존 자원 유형의 새 버전 설치 및 등록
- 지정된 자원 유형의 여러 버전 등록 허용
- 자원을 삭제하고 다시 만들지 않아도 기존 자원을 새 버전의 자원 유형으로 업그레이드

업그레이드하려는 자원 유형을 **업그레이드 인식** 자원 유형이라고 합니다. 변경할 수 있는 기존 자원 유형의 요소는 다음과 같습니다.

- 자원 유형 등록 정보의 속성
- 표준 및 확장 등록 정보를 포함하는 선언된 자원 등록 정보 세트
- `default`, `min`, `max`, `arraymin`, `arraymax` 또는 `tunability`와 같은 자원 등록 정보의 속성

- 선언된 메소드 세트
- 메소드 또는 모니터 구현

주 – 응용 프로그램 코드를 수정할 때 반드시 자원 유형을 수정해야 하는 것은 아닙니다.

클러스터 관리자가 자원 유형을 업그레이드할 수 있게 해주는 도구를 제공하기 위한 요구 사항을 알아야 합니다. 이 장에서는 이러한 도구를 설정하기 위해 알아야 하는 사항을 설명합니다.

자원 유형 등록(RTR) 파일의 내용 설정

자원 유형 이름

vendor-id, *resource-type*, *rt-version* 등 RTR 파일에 지정된 세 가지 등록 정보가 자원 유형 이름을 구성합니다. `scrgadm` 명령은 다음과 같이 점과 콜론 분리자를 삽입하여 자원 유형의 이름을 만듭니다.

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

vendor-id 접두어는 여러 공급업체에서 제공하는 동일한 이름의 등록 파일 두 개를 구분하는 역할을 합니다. *vendor-id*를 고유하게 설정하려면 자원 유형을 만들 때 회사의 주식 기호를 사용합니다. *rt-version*은 동일한 기본 자원 유형의 등록된 여러 버전(업그레이드)을 구분합니다.

다음 명령을 입력하여 정규화된 자원 유형 이름을 얻을 수 있습니다.

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

Sun Cluster 3.1 이전에 등록된 자원 유형 이름은 계속해서 다음 구문을 사용합니다.

vendor-id.*resource-type*

자원 유형 이름의 형식에 대해서는 306 페이지 “자원 유형 이름 형식”에서 설명합니다.

#$upgrade 및 #$upgrade_from 지시어 지정

수정 중인 자원 유형이 업그레이드를 인식하게 하려면 RTR 파일에 `#$upgrade` 지시어가 포함되어야 하며, 지원할 자원 유형의 각 이전 버전에 대해 `#$upgrade` 지시어 뒤에 0개 이상의 `#$upgrade_from` 지시어를 추가합니다.

#$upgrade 및 #$upgrade_from 지시어는 RTR 파일에서 자원 유형 등록 정보 선언과 자원 선언 부분 사이에 있어야 합니다. rt_reg(4) 설명서 페이지를 참조하십시오.

예 4–1 RTR 파일의 #$upgrade_from 지시어

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$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
```

#$upgrade_from 지시어의 형식은 다음과 같습니다.

```
#$upgrade_from version tunability
```

version
RT_version입니다. 자원 유형에 버전이 없거나 이전에 RTR 파일에서 정의한 버전이 아닌 경우 빈 문자열(“”)을 지정합니다.

tunability
클러스터 관리자가 지정한 RT_version을 업그레이드할 수 있는 조건 또는 시기입니다.

#$upgrade_from 지시어에 사용할 수 있는 조정 기능 값은 다음과 같습니다.

ANYTIME
클러스터 관리자의 자원 업그레이드 가능 시기에 대한 제한이 없는 경우에 사용합니다. 업그레이드 중에 자원이 완전히 온라인 상태가 될 수 있습니다.

WHEN_UNMONITORED
새 자원 유형 버전의 메소드가 다음과 같은 경우에 사용합니다.

- Update, Stop, Monitor_check 및 Postnet_stop 메소드가 이전 자원 유형 버전의 시작 메소드(Prenet_stop 및 Start)와 호환되는 경우
- Fini 메소드가 이전 버전의 Init 메소드와 호환되는 경우

클러스터 관리자는 업그레이드 전에 자원 모니터 프로그램만 중지하면 됩니다.

WHEN_OFFLINE
새 자원 유형 버전의 Update, Stop, Monitor_check 또는 Postnet_stop 메소드가 다음과 같은 경우에 사용합니다.

- 이전 버전의 Init 메소드와 호환되는 경우
- 이전 자원 유형 버전의 시작 메소드(Prenet_stop 및 Start)와 호환되지 않는 경우

클러스터 관리자가 업그레이드 전에 자원을 오프라인 상태로 전환해야 합니다.

WHEN_DISABLED
WHEN_OFFLINE과 유사합니다. 그러나 클러스터 관리자가 업그레이드 전에 자원을 비활성화해야 합니다.

WHEN_UNMANAGED
새 자원 유형 버전의 Fini 메소드가 이전 버전의 Init 메소드와 호환되지 않는 경우에 사용합니다. 클러스터 관리자가 업그레이드 전에 기존 자원 그룹을 관리 해제 상태로 전환해야 합니다.

자원 유형 버전이 #$upgrade_from 지시어 목록에 나타나지 않는 경우 RGM은 기본적으로 WHEN_UNMANAGED 조정 기능 옵션을 해당 버전에 지정합니다.

AT_CREATION
기존 자원을 새 버전의 자원 유형으로 업그레이드할 수 없도록 차단하려면 사용합니다. 클러스터 관리자가 자원을 삭제하고 다시 만들어야 합니다.

RTR 파일에서 RT_version 변경

RTR 파일 내용이 변경될 때마다 RTR 파일에서 RT_version 등록 정보만 변경하면 됩니다. 이 버전의 자원 유형이 최신 버전임을 나타내는 등록 정보 값을 선택합니다.

RTR 파일의 RT_version 문자열에 다음 문자를 포함하지 **마십시오**. 이러한 문자를 사용하면 자원 유형의 등록에 실패합니다.

- 스페이스
- 탭
- 슬래시(/)
- 백슬래시(\)
- 별표(*)
- 물음표(?)
- 쉼표(,)
- 세미콜론(;)
- 왼쪽 대괄호([)
- 오른족 대괄호(])

Sun Cluster 3.0에서 선택 사항이었던 RT_Version 등록 정보는 Sun Cluster 3.1부터 필수입니다.

이전 버전 Sun Cluster의 자원 유형 이름

Sun Cluster 3.0의 자원 유형 이름은 다음과 같은 버전 접미어를 포함하지 않습니다.

vendor-id . resource-type

원래 Sun Cluster 3.0에서 등록된 자원 유형은 클러스터 관리자가 클러스터링 소프트웨어를 Sun Cluster 3.1 이후 릴리스로 업그레이드한 후에도 이 구문에 따른 이름을 사용합니다. 마찬가지로 Sun Cluster 3.1 이상을 실행하는 클러스터에 RTR 파일을 등록한 경우 RTR 파일에서 #$upgrade 지시어가 누락된 자원 유형에는 버전 접미어 없이 Sun Cluster 3.0 형식 이름이 제공됩니다.

클러스터 관리자는 Sun Cluster 3.0의 #$upgrade 지시어 또는 #$upgrade_from 지시어를 사용하여 RTR 파일을 등록할 수 있습니다. 그러나 Sun Cluster 3.0에서 기존 자원을 새 자원 유형으로 업그레이드할 수는 없습니다.

클러스터 관리자에 의한 업그레이드 시 수행되는 작업

클러스터 관리자가 수행해야 하는 작업 또는 자원 유형 업그레이드 시 수행되는 작업은 다음과 같습니다.

- 기존 자원 등록 정보 속성이 새 버전의 자원 유형 검증 조건을 만족하지 않으면 클러스터 관리자가 유효한 값을 제공해야 합니다. 클러스터 관리자는 다음 조건에서 이 작업을 수행해야 합니다.
  - 새 버전의 자원 유형에 기본값이 없으며 이전 버전에서 선언되지 않은 등록 정보를 사용하는 경우
  - 새 버전에서 해당 값이 선언되지 않았거나 유효하지 않은 등록 정보를 기존 자원이 사용하는 경우. 선언된 등록 정보 중에서 새 버전의 자원 유형에서 선언되지 않은 등록 정보는 자원에서 삭제됩니다.
- 지원되지 않는 버전의 자원 유형에서 업그레이드를 시도하면 이 시도는 실패합니다.
- 업그레이드 후에는 자원이 새 버전의 자원 유형에서 모든 등록 정보의 자원 등록 정보 속성을 상속합니다.
- RTR 파일에서 자원 유형의 기본값을 변경하면 기존 자원이 새 기본값을 상속합니다. 새 기본값은 등록 정보가 `AT_CREATION` 또는 `WHEN_DISABLED`에만 조정 가능으로 선언된 경우에도 상속됩니다. 클러스터 관리자가 만드는 동일한 유형의 등록 정보도 이 기본값을 상속합니다. 그러나 클러스터 관리자가 등록 정보에 새 기본값을 지정하면 새 기본값이 RTR 파일에 지정된 기본값보다 우선합니다.

주 – Sun Cluster 3.0에서 만든 자원은 이후 버전의 Sun Cluster로 업그레이드 시 자원 유형에서 새 기본 자원 등록 정보 속성을 상속하지 않습니다. 이 제한은 Sun Cluster 3.0 클러스터에서 업그레이드하는 Sun Cluster 3.1 클러스터에만 적용됩니다. 클러스터 관리자는 등록 정보 값을 지정하여 기본값을 무시함으로써 이 제한을 해결할 수 있습니다.

자원 유형 모니터 코드 구현

클러스터 관리자는 Sun Cluster 3.0에서 업그레이드 인식 자원 유형을 등록할 수 있습니다. 그러나 Sun Cluster는 버전 접미어 없이 자원 유형 이름을 기록합니다. Sun Cluster 3.0과 Sun Cluster 3.1에서 모두 제대로 실행하려면 이 자원 유형의 모니터 코드에서 다음 두 가지 이름 지정 규약을 처리할 수 있어야 합니다.

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

자원 유형 이름의 형식에 대해서는 306 페이지 “자원 유형 이름 형식”에서 설명합니다.

클러스터 관리자는 동일한 버전의 자원 유형을 두 개의 다른 이름으로 두 번 등록할 수 없습니다. 모니터 코드에서 올바른 이름을 결정할 수 있게 하려면 모니터 코드에서 다음 명령을 호출합니다.

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

그런 다음 출력 값을 `vers`와 비교합니다. 특정 `vers` 값에 대해 이러한 명령 중 하나만 성공합니다.

설치 요구 사항 및 패키지화 결정

자원 유형 패키지의 설치 요구 사항과 패키지화를 결정하는 경우 다음 두 가지 요구 사항을 명심하십시오.

- 새 자원 유형이 등록되면 디스크에서 해당 RTR 파일에 액세스할 수 있어야 합니다.
- 새 유형의 자원을 만들면 새 유형의 선언된 모든 메소드 경로 이름과 모니터 프로그램이 디스크에 있고 실행 가능해야 합니다. 자원을 사용하는 한 이전 메소드와 모니터 프로그램은 제자리에 있어야 합니다.

사용할 올바른 패키지화를 결정하려면 다음 질문을 고려해 보십시오.

- RTR 파일이 변경됩니까?
- 등록 정보의 기본값이나 조정 기능이 변경됩니까?
- 등록 정보의 `min` 또는 `max` 값이 변경됩니까?
- 업그레이드에서 등록 정보를 추가 또는 삭제합니까?
- 모니터 코드가 변경됩니까?
- 메소드 코드가 변경됩니까?
- 새 메소드, 모니터 코드 또는 둘 다가 이전 버전과 호환됩니까?

이러한 질문의 대답은 새 자원 유형에 사용할 올바른 패키지화를 결정하는 데 도움이 될 것입니다.

RTR 파일 변경 전 주의 사항

자원 유형을 수정할 때 반드시 새 메소드 또는 모니터 코드를 만들어야 하는 것은 아닙니다. 예를 들어, 자원 등록 정보의 기본값이나 조정 기능만 변경할 수 있습니다. 이 인스턴스에서는 메소드 코드를 변경하지 않기 때문에 읽을 수 있는 RTR 파일에 유효한 새 경로 이름만 있으면 됩니다.

이전 자원 유형을 다시 등록할 필요가 없는 경우 새 RTR 파일이 이전 버전을 덮어쓸 수 있습니다. 그렇지 않으면 새 RTR 파일을 새 경로에 저장합니다.

업그레이드 시 등록 정보의 기본값이나 조정 기능이 변경되면 새 버전의 자원 유형에 대해 Validate 메소드를 사용하여 기존 등록 정보 속성이 새 자원 유형에 유효한지 확인합니다. 유효하지 않으면 클러스터 관리자는 기존 자원의 등록 정보를 올바른 값으로 변경할 수 있습니다. 업그레이드 시 등록 정보의 min, max 또는 type 속성이 변경되는 경우 클러스터 관리자가 자원 유형을 업그레이드하면 scrgadm 명령이 자동으로 이러한 제약 조건을 검증합니다.

업그레이드 시 새 등록 정보가 추가되거나 이전 등록 정보가 삭제되는 경우 콜백 메소드 또는 모니터 코드를 변경해야 할 수도 있습니다.

모니터 코드 변경

자원 유형의 모니터 코드만 변경하는 경우 패키지 설치가 모니터 이진을 덮어쓸 수 있습니다.

메소드 코드 변경

자원 유형에서 메소드 코드만 변경하는 경우 새 메소드 코드가 이전 메소드 코드와 호환되는지 여부를 결정해야 합니다. 이 질문의 대답에 따라 새 메소드 코드를 새 경로에 저장해야 하는지 또는 이전 메소드를 덮어쓸 수 있는지가 결정됩니다.

새 Stop, Postnet_stop 및 Fini 메소드(선언된 경우)를 이전 버전의 Start, Prenet_stop 또는 Init 메소드로 초기화 또는 시작된 자원에 적용할 수 있는 경우 이전 메소드를 새 메소드로 덮어쓸 수 있습니다.

등록 정보에 새 기본값을 적용하면 Stop, Postnet_stop 또는 Fini와 같은 메소드가 실패하는 경우 클러스터 관리자는 자원 유형 업그레이드 시 자원 상태를 적절하게 제한해야 합니다.

Type_version 등록 정보의 조정 기능을 제한하여 클러스터 관리자가 업그레이드 시 자원 상태를 제한할 수 있게 합니다.

패키지화하는 한 가지 방법은 패키지에서 지원되는 이전 버전의 모든 자원 유형을 포함하는 것입니다. 이 방법을 사용하면 이전 메소드 경로를 덮어쓰거나 삭제하지 않아도 새 버전의 패키지가 이전 버전의 패키지를 대체할 수 있습니다. 지원할 이전 버전의 수를 결정해야 합니다.

사용할 패키지화 체계 결정

다음 표에는 새 자원 유형에 사용할 패키지화 체계가 요약되어 있습니다.

표 4–1 사용할 패키지화 체계 결정

변경 유형	조정 가능 값	패키지화 체계
RTR 파일에서만 등록 정보를 변경합니다.	ANYTIME	새 RTR 파일만 배달합니다.
메소드를 업데이트합니다.	ANYTIME	이전 메소드와 다른 경로에 업데이트된 메소드를 저장합니다.
새 모니터 프로그램을 설치합니다.	WHEN_UNMONITORED	이전 버전의 모니터만 덮어씁니다.
메소드를 업데이트합니다. 새 Update 및 Stop 메소드는 이전 Start 메소드와 호환되지 않습니다.	WHEN_OFFLINE	이전 메소드와 다른 경로에 업데이트된 메소드를 저장합니다.
메소드를 업데이트하고 RTR 파일에 새 등록 정보를 추가합니다. 새 메소드에는 새로운 등록 정보가 필요합니다. 포함하는 자원 그룹이 온라인 상태를 유지할 수 있게 하지만 노드의 자원 그룹이 오프라인 상태에서 온라인 상태로 이동할 경우 자원이 온라인 상태가 되는 것을 방지하기 위한 것입니다.	WHEN_DISABLED	이전 버전의 메소드를 덮어씁니다.
메소드를 업데이트하고 RTR 파일에 새 등록 정보를 추가합니다. 새 메소드에는 새 등록 정보가 필요하지 않습니다.	ANYTIME	이전 버전의 메소드를 덮어씁니다.
메소드를 업데이트합니다. 새로운 Fini 메소드는 이전 Init 메소드와 호환되지 않습니다.	WHEN_UNMANAGED	이전 메소드와 다른 경로에 업데이트된 메소드를 저장합니다.
메소드를 업데이트합니다. RTR 파일이 변경되지 않습니다.	해당 없음. RTR 파일이 변경되지 않습니다.	이전 버전의 메소드를 덮어씁니다. RTR 파일을 변경하지 않았기 때문에 자원을 등록하거나 업그레이드할 필요가 없습니다.

수정된 자원 유형을 제공하기 위한 설명서

클러스터 관리자에게 자원 유형을 업그레이드하는 방법을 알려주는 지침은 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**의 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**의 "Upgrading a Resource Type"에 포함되어 있습니다. 클러스터 관리자가 수정되는 자원 유형을 업그레이드할 수 있게 하려면 이 절에 설명된 추가 정보와 함께 이러한 지침을 참조하십시오.

일반적으로 새 자원 유형을 만드는 경우 다음 정보가 포함된 설명서를 제공해야 합니다.

- 추가, 변경 또는 삭제하는 등록 정보를 설명합니다.
- 등록 정보가 새 요구 사항을 준수하도록 만드는 방법을 설명합니다.
- 자원에 대한 조정 기능 제약 조건을 설명합니다.
- 새 기본 등록 정보 속성을 설명합니다.
- 클러스터 관리자에게 필요한 경우 기존 자원 등록 정보를 적절한 값으로 설정할 수 있음을 알립니다.

업그레이드 설치 전 수행할 작업과 관련된 정보

클러스터 관리자에게 노드에서 업그레이드 패키지를 설치하기 전에 수행해야 하는 작업을 다음과 같이 설명합니다.

- 업그레이드 패키지가 기존 메소드를 덮어쓰는 경우 클러스터 관리자에게 비클러스터 모드에서 노드를 재부트하도록 지시합니다.
- 업그레이드 패키지가 모니터 코드만 업데이트하고 메소드 코드를 변경되지 않은 상태로 유지하는 경우 클러스터 관리자에게 해당 노드를 클러스터 모드로 실행하도록 지시합니다. 클러스터 관리자에게 모든 자원 유형의 모니터링을 해제하도록 지시합니다.
- 업그레이드 패키지가 RTR 파일만 업데이트하고 모니터 코드를 변경되지 않은 상태로 유지하는 경우 클러스터 관리자에게 해당 노드를 클러스터 모드로 실행하도록 지시합니다. 또한 모든 자원 유형에 대해 모니터링을 설정하도록 지시합니다.

자원 업그레이드 시와 관련된 정보

클러스터 관리자에게 자원을 새 버전의 자원 유형으로 업그레이드할 수 있는 경우를 설명합니다. 클러스터 관리자가 자원 유형을 업그레이드할 수 있는 조건은 RTR 파일의 각 자원 버전에 대한 `#$upgrade_from` 지시어의 조정 기능에 따라 다음과 같이 결정됩니다.

- 언제든지(ANYTIME)
- 자원 모니터링이 해제된 경우에만(WHEN_UNMONITORED)
- 자원이 오프라인 상태인 경우에만(WHEN_OFFLINE)
- 자원이 비활성화된 경우에만(WHEN_DISABLED)
- 자원 그룹이 관리 해제된 경우에만(WHEN_UNMANAGED)

예 4–2 #$upgrade_from이 클러스터 관리자의 업그레이드 가능 시기를 정의하는 방법

이 예에서는 클러스터 관리자가 자원을 새 버전의 자원 유형으로 업그레이드할 수 있는 조건에 #$upgrade_from 지시어의 조정 기능이 미치는 영향을 보여줍니다.

```
#$upgrade_from   "1.1"   WHEN_OFFLINE
#$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
```

버전	클러스터 관리자가 자원을 업그레이드할 수 있는 시기
1.1, 1.2 또는 1.3	자원이 오프라인 상태인 경우에만
2.0	자원 모니터링이 해제된 경우에만
2.1	언제든지
다른 모든 버전	자원 그룹이 관리 해제된 경우에만

자원 등록 정보 변경과 관련된 정보

클러스터 관리자가 업그레이드 시 기존 자원의 등록 정보를 수정해야 하는 자원 유형 변경을 설명합니다. 수행할 수 있는 변경은 다음과 같습니다.

- 변경한 기존 자원 유형 등록 정보의 기본 설정
- 새로 도입한 자원 유형의 새 확장 등록 정보
- 취소한 자원 유형의 기존 등록 정보
- 자원 유형에 대해 선언한 표준 등록 정보 집합 변경
- min, max, arraymin, arraymax, default 및 tunability와 같은 변경한 자원 등록 정보 속성
- 선언한 메소드 세트 변경
- 변경한 메소드 또는 오류 모니터 구현

5장

샘플 데이터 서비스

이 장에서는 `in.named` 응용 프로그램의 샘플 Sun Cluster 데이터 서비스인 HA-DNS에 대해 설명합니다. `in.named` 데몬은 도메인 이름 서비스(DNS)의 Solaris 구현입니다. 샘플 데이터 서비스는 자원 관리 API를 사용하여 응용 프로그램의 가용성을 높이는 방법을 보여줍니다.

자원 관리 API는 쉘 스크립트 인터페이스와 C 프로그램 인터페이스를 지원합니다. 이 장의 샘플 응용 프로그램은 쉘 스크립트 인터페이스를 사용하여 작성되었습니다.

이 장은 다음 내용으로 구성되어 있습니다.

샘플 데이터 서비스 개요

샘플 데이터 서비스는 관리 작업, 응용 프로그램 실패 또는 노드 실패와 같은 클러스터 이벤트에 응답하여 클러스터의 노드 사이에서 DNS 응용 프로그램을 시작, 중지, 재시작 및 전환합니다.

응용 프로그램 재시작은 PMF(Process Monitor Facility)에 의해 관리됩니다. 응용 프로그램 중지 수가 실패 시간 창에서 실패 횟수를 초과할 경우 오류 모니터는 응용 프로그램 자원을 포함하는 자원 그룹을 다른 노드로 페일오버합니다.

샘플 데이터 서비스는 nslookup 명령을 사용하여 응용 프로그램 상태가 정상인지 확인하는 PROBE 메소드의 형태로 오류 모니터링 기능을 제공합니다. 정지된 DNS 서비스가 감지된 경우 검사는 DNS 응용 프로그램을 로컬로 재시작하여 문제를 해결하려고 시도합니다. 문제가 해결되지 않고 서비스 문제가 지속적으로 감지되는 경우 검사는 해당 서비스를 클러스터의 다른 노드로 페일오버합니다.

특히 샘플 데이터 서비스는 다음 요소를 포함합니다.

- 데이터 서비스의 정적 등록 정보를 정의하는 자원 유형 등록 파일
- HA-DNS 데이터 서비스를 포함하는 자원 그룹이 온라인 상태가 되었을 때 in.named 데몬을 시작하기 위해 RGM에 의해 실행되는 Start 콜백 메소드
- HA-DNS를 포함하는 자원 그룹이 오프라인 상태가 되었을 때 in.named 데몬을 중지하기 위해 RGM에 의해 실행되는 Stop 콜백 메소드
- DNS 서버가 실행 중인지 확인하여 서비스의 가용성을 검사하기 위한 오류 모니터 오류 모니터는 사용자 정의 PROBE 메소드에 의해 구현되며 Monitor_start 및 Monitor_stop 콜백 메소드로 시작 및 중지합니다.
- 서비스의 구성 디렉토리에 액세스할 수 있는지 검증하기 위해 RGM에 의해 실행되는 Validate 콜백 메소드
- 시스템 관리자가 자원 등록 정보의 값을 변경할 때 오류 모니터를 재시작하기 위해 RGM에 의해 실행되는 Update 콜백 메소드

자원 유형 등록 파일 정의

이 예의 자원 유형 등록(RTR) 파일은 DNS 자원 유형의 정적 구성을 정의합니다. 이 유형의 자원은 RTR 파일에 정의된 등록 정보를 상속합니다.

클러스터 관리자가 HA-DNS 데이터 서비스를 등록할 때 RGM은 RTR 파일의 정보를 읽습니다.

RTR 파일 개요

RTR 파일은 잘 정의된 형식을 따릅니다. RTR 파일에서는 지원 유형 등록 정보가 가장 먼저 정의되고 시스템 정의 자원 등록 정보가 그 다음에 정의되며 마지막으로 확장 등록 정보가 정의됩니다. 자세한 내용은 rt_reg(4) 설명서 페이지 및 34 페이지 “자원 및 자원 유형 등록 정보 설정”을 참조하십시오.

이 절에서는 샘플 RTR 파일의 특정 등록 정보에 대해 설명하며 샘플 RTR 파일의 여러 부분을 보여줍니다. 샘플 RTR 파일 내용의 전체 목록은 255 페이지 “자원 유형 등록(RTR) 파일 목록”을 참조하십시오.

샘플 RTR 파일의 자원 유형 등록 정보

샘플 RTR 파일은 다음 목록에서처럼 맨 앞에 주석이 있고 HA-DNS 구성을 정의하는 자원 유형 등록 정보가 그 뒤에 옵니다.

주 – 자원 그룹, 자원 및 자원 유형의 등록 정보 이름은 대소문자를 구분하지 **않습니다**. 등록 정보 이름을 지정하는 경우 대문자와 소문자를 임의로 조합해서 사용할 수 있습니다.

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

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

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

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

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

Start         =   dns_svc_start;
Stop          =   dns_svc_stop;

Validate      =   dns_validate;
Update        =   dns_update;

Monitor_start =   dns_monitor_start;
Monitor_stop  =   dns_monitor_stop;
Monitor_check =   dns_monitor_check;
```

정보 – RTR 파일의 첫 번째 항목으로 `Resource_type` 등록 정보를 선언해야 합니다. 그렇지 않으면 자원 유형 등록에 실패합니다.

이러한 등록 정보에 대한 몇 가지 정보는 다음과 같습니다.

- 자원 유형 이름은 `Resource_type` 등록 정보로만 지정하거나(`sample`) *vendor-id*를 접두어로 사용하고 (.) 뒤에 자원 유형 등록 정보를 추가하여 지정할 수 있습니다(`SUNW.sample`).

*vendor-id*를 사용하는 경우 자원 유형을 정의하는 회사의 주식 기호를 사용하십시오. 자원 유형 이름은 클러스터에서 고유해야 합니다.

- `RT_version` 등록 정보는 공급업체가 지정한 대로 샘플 데이터 서비스의 버전을 식별합니다.
- `API_version` 등록 정보는 Sun Cluster 버전을 식별합니다. 예를 들어 `API_version = 2`는 Sun Cluster 3.0부터, `API_version = 5`는 Sun Cluster 3.1 9/04부터 모든 버전의 Sun Cluster에 데이터 서비스를 설치할 수 있음을 나타냅니다. 그러나 `API_version = 5`는 또한 3.1 9/04 이전에 릴리스된 Sun Cluster의 모든 버전에 데이터 서비스를 설치할 수 없음을 나타냅니다. 이 등록 정보에 대해서는 223 페이지 "자원 유형 등록 정보"의 `API_version` 항목에서 자세히 설명합니다.
- `Failover = TRUE`는 동시에 여러 노드에서 온라인 상태가 될 수 있는 자원 그룹에서 데이터 서비스를 실행할 수 없음을 나타냅니다.
- `RT_basedir`은 콜백 메소드 경로와 같은 상대 경로를 완성하기 위한 디렉토리 경로로서 `/opt/SUNWsample/bin`을 가리킵니다.
- `Start`, `Stop` 및 `Validate`는 RGM에 의해 실행되는 각각의 콜백 메소드 프로그램에 대한 경로를 제공합니다. 이러한 경로는 `RT_basedir`에 지정된 디렉토리에 상대적인 경로입니다.
- `Pkglist`는 샘플 데이터 서비스 설치를 포함하는 패키지로서 `SUNWsample`을 식별합니다.

`Single_instance`, `Init_nodes` 및 `Installed_nodes`와 같이 이 RTR 파일에 지정되지 않은 자원 유형 등록 정보는 기본값으로 설정됩니다. 223 페이지 "자원 유형 등록 정보"에는 자원 유형 등록 정보의 전체 목록이 기본값과 함께 나와 있습니다.

클러스터 관리자는 RTR 파일의 자원 유형 등록 정보 값을 변경할 수 없습니다.

샘플 RTR 파일의 자원 등록 정보

일반적으로 자원 등록 정보는 RTR 파일에서 자원 유형 등록 정보 뒤에 선언됩니다. 자원 등록 정보는 Sun Cluster 소프트웨어가 제공하는 시스템 정의 등록 정보와 개발자가 정의하는 확장 등록 정보를 포함합니다. 어떠한 유형이든 최소값, 최대값 및 기본값과 같이 Sun Cluster 소프트웨어가 제공하는 여러 등록 정보 속성을 지정할 수 있습니다.

RTR 파일의 시스템 정의 등록 정보

다음 목록은 샘플 RTR 파일의 시스템 정의 등록 정보를 보여줍니다.

```
# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.

# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.
```

```
# The MIN value for all method timeouts is set to 60 seconds. This
# prevents administrators from setting shorter timeouts, which do not
# improve switchover/failover performance, and can lead to undesired
# RGM actions (false failovers, node reboot, or moving the resource group
# to ERROR_STOP_FAILED state, requiring operator intervention). Setting
# too-short method timeouts leads to a *decrease* in overall availability
# of the data service.
{
   PROPERTY = Start_timeout;
   MIN=60;
   DEFAULT=300;
}

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

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

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

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

Sun Cluster 소프트웨어가 시스템 정의 등록 정보를 제공하기는 하지만 자원 등록 정보 속성을 사용하여 다른 기본값을 설정할 수 있습니다. 자원 등록 정보에 적용할 수 있는 전체 속성 목록은 253 페이지 "자원 등록 정보 속성"을 참조하십시오.

샘플 RTR 파일의 시스템 정의 자원 등록 정보에 대한 다음 사항에 주의하십시오.

- Sun Cluster는 모든 시간 초과에 대해 최소값(1초) 및 기본값(3600초, 즉 1시간)을 제공합니다. 샘플 RTR 파일은 최소 시간 초과값을 60초로, 기본값을 300초로 변경합니다. 클러스터 관리자는 이 기본값을 적용하거나 시간 초과값을 60초 이상의 다른 값으로 변경할 수 있습니다. Sun Cluster의 경우 최대값에는 제한이 없습니다.
- `Thorough_probe_interval`, `Retry_count` 및 `Retry_interval`의 `TUNABLE` 속성은 `ANYTIME`으로 설정됩니다. 이 설정은 데이터 서비스가 실행 중인 경우에도 클러스터 관리자가 이러한 등록 정보의 값을 변경할 수 있다는 것을 의미합니다. 이러한 등록 정보는 샘플 데이터 서비스에 의해 구현되는 오류 모니터에서 사용합니다. 관리 작업에 의해 이러한 자원 등록 정보나 다른 자원 등록 정보가 변경되면 샘플 데이터 서비스는 `Update` 메소드를 구현하여 오류 모니터를 중지했다가 다시 시작합니다. 104 페이지 "`Update` 메소드 작동 방식"을 참조하십시오.
- 자원 등록 정보는 다음과 같이 분류됩니다.
  - **필수적**. 클러스터 관리자가 자원을 만들 때 반드시 값을 지정해야 합니다.
  - **선택적**. 클러스터 관리자가 값을 지정하지 않을 경우 시스템에서 기본값을 제공합니다.
  - **조건적**. 등록 정보가 RTR 파일에 선언된 경우에만 RGM에서 등록 정보를 만듭니다.

  샘플 데이터 서비스의 오류 모니터에서 `Thorough_probe_interval`, `Retry_count`, `Retry_interval` 및 `Network_resources_used` 조건적 등록 정보를 사용하기 때문에 개발자는 RTR 파일에서 이러한 등록 정보를 선언해야 합니다. 등록 정보 분류 방법에 대한 자세한 내용은 `r_properties`(5) man page or 230 페이지 "자원 등록 정보"를 참조하십시오.

RTR 파일의 확장 등록 정보

샘플 RTR 파일의 끝에는 다음과 같은 확장 등록 정보가 있습니다.

```
# Extension Properties

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

# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 120;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

샘플 RTR 파일은 두 개의 확장 등록 정보인 `Confdir` 및 `Probe_timeout`을 정의합니다. `Confdir` 등록 정보는 DNS 구성 디렉토리의 경로를 지정합니다. 이 디렉토리는 DNS가 성공적으로 작동하는 데 필요한 `in.named` 파일을 포함합니다. 샘플 데이터 서비스의 `Start` 및 `Validate` 메소드는 DNS를 시작하기 전에 이 등록 정보를 사용하여 구성 디렉토리와 `in.named` 파일에 액세스할 수 있는지 확인합니다.

데이터 서비스가 구성되면 `Validate` 메소드는 새 디렉토리에 액세스할 수 있는지 확인합니다.

샘플 데이터 서비스의 `PROBE` 메소드는 Sun Cluster 콜백 메소드가 아니라 사용자 정의 메소드입니다. 따라서 Sun Cluster는 이 메소드에 대한 `Probe_timeout` 등록 정보를 제공하지 않습니다. 개발자는 클러스터 관리자가 `Probe_timeout` 값을 구성할 수 있도록 RTR 파일에 확장 등록 정보를 정의해야 합니다.

모든 메소드에 공통 기능 제공

이 절에서는 샘플 데이터 서비스의 모든 콜백 메소드에 사용되는 다음 기능에 대해 설명합니다.

- 82 페이지 "명령 인터프리터 식별 및 경로 내보내기"

명령 인터프리터 식별 및 경로 내보내기

쉘 스크립트의 첫 번째 줄은 명령 인터프리터를 식별해야 합니다. 샘플 데이터 서비스의 각 메소드 스크립트는 다음과 같이 명령 인터프리터를 식별합니다.

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

샘플 응용 프로그램의 모든 메소드 스크립트에서는 사용자의 PATH 설정을 사용하는 대신 Sun Cluster 이진 및 라이브러리에 대한 경로를 내보냅니다.

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

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

PMF_TAG 및 SYSLOG_TAG 변수 선언

Validate를 제외한 모든 메소드 스크립트는 pmfadm을 사용하여 데이터 서비스 또는 모니터를 시작하거나 중지하여 자원 이름을 전달합니다. 각 스크립트는 pmfadm에 전달할 수 있는 PMF_TAG 변수를 정의하여 데이터 서비스나 모니터를 식별합니다.

마찬가지로 각 메소드 스크립트는 logger 명령을 사용하여 시스템 로그에 메시지를 기록합니다. 각 스크립트는 -t 옵션과 함께 logger에 전달할 수 있는 SYSLOG_TAG 변수를 정의하여 메시지가 기록되는 자원의 자원 유형, 자원 이름 및 자원 그룹을 식별합니다.

모든 메소드는 다음 샘플 코드에 표시된 것처럼 동일한 방법으로 SYSLOG_TAG를 정의합니다. dns_probe, dns_svc_start , dns_svc_stop 및 dns_monitor_check 메소드는 다음과 같이 PMF_TAG를 정의합니다(dns_svc_stop 메소드에서 pmfadm 및 logger를 사용한 경우).

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

PMF_TAG=$RESOURCE_NAME.named

SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
```

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

dns_monitor_start, dns_monitor_stop 및 dns_update 메소드는 다음과 같이 PMF_TAG를 정의합니다(dns_monitor_stop 메소드에서 pmfadm을 사용한 경우).

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

PMF_TAG=$RESOURCE_NAME.monitor
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME
...
# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
```

함수 인자 구문 분석

RGM은 Validate를 제외한 모든 콜백 메소드를 다음과 같이 실행합니다.

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

메소드 이름은 콜백 메소드를 구현하는 프로그램의 경로 이름입니다. 데이터 서비스는 RTR 파일의 각 메소드에 대한 경로 이름을 지정합니다. 이러한 경로 이름은 마찬가지로 RTR 파일에 있는 RT_basedir 등록 정보에 지정된 디렉토리에 상대적입니다. 예를 들어, 샘플 데이터 서비스의 RTR 파일에서 기본 디렉토리 및 메소드 이름은 다음과 같이 지정됩니다.

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

모든 콜백 메소드 인자는 다음과 같이 플래그가 지정된 값으로 전달됩니다. -R 인자는 자원 인스턴스의 이름을 나타냅니다. -T 인자는 자원 유형을 나타냅니다. -G 인자는 자원이 구성되는 그룹을 나타냅니다. 콜백 메소드에 대한 자세한 내용은 rt_callbacks(1HA) 설명서 페이지를 참조하십시오.

주 – Validate 메소드는 추가 인자, 즉 메소드가 호출되는 자원 및 자원 그룹의 등록 정보 값과 함께 호출됩니다. 자세한 내용은 100 페이지 "등록 정보 업데이트 처리"를 참조하십시오.

각 콜백 메소드에는 전달되는 인자를 구문 분석하기 위한 함수가 필요합니다. 모든 콜백에 동일한 인자가 전달되므로 데이터 서비스는 응용 프로그램의 모든 콜백에서 사용되는 단일 구문 분석 함수를 제공합니다.

다음은 샘플 응용 프로그램의 콜백 메소드에 사용되는 parse_args() 함수를 보여줍니다.

```
##############################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
      typeset opt

      while getopts 'R:G:T:' opt
      do
              case "$opt" in
              R)
                   # Name of the DNS resource.
                   RESOURCE_NAME=$OPTARG
                   ;;
              G)
                   # Name of the resource group in which the resource is
                   # configured.
                   RESOURCEGROUP_NAME=$OPTARG
                   ;;
              T)
                   # Name of the resource type.
                   RESOURCETYPE_NAME=$OPTARG
                   ;;
              *)
                   logger -p ${SYSLOG_FACILITY}.err \
                   -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                   "ERROR: Option $OPTARG unknown"
                   exit 1
                       ;;
              esac
    done
}
```

주 – 샘플 응용 프로그램의 PROBE 메소드는 Sun Cluster 콜백 메소드가 아니라 사용자 정의 메소드이지만 콜백 메소드와 동일한 인자와 함께 호출됩니다. 따라서 이 메소드는 다른 콜백 메소드에 사용되는 것과 동일한 구문 분석 함수를 포함합니다.

구문 분석 함수는 다음과 같이 MAIN에서 호출됩니다.

```
parse_args "$@"
```

오류 메시지 생성

콜백 메소드는 syslog() 함수를 사용하여 오류 메시지를 최종 사용자에게 출력해야 합니다. 샘플 데이터 서비스의 모든 콜백 메소드는 다음과 같이 scha_cluster_get 명령을 사용하여 클러스터 로그에 사용되는 syslog() 함수 번호를 검색합니다.

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

이 값은 쉘 변수 SYSLOG_FACILITY에 저장되며 logger 명령 기능으로 사용되어 메시지를 클러스터 로그에 기록할 수 있습니다. 예를 들어, 샘플 데이터 서비스의 Start 메소드는 다음과 같이 syslog() 함수를 검색하고 데이터 서비스가 시작되었다는 메시지를 기록합니다.

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

자세한 내용은 scha_cluster_get(1HA) 설명서 페이지를 참조하십시오.

등록 정보에 대한 정보 얻기

대부분의 콜백 메소드는 데이터 서비스의 자원 및 자원 유형 등록 정보에 대한 정보를 얻어야 합니다. API는 이러한 정보를 얻을 수 있도록 scha_resource_get() 함수를 제공합니다.

시스템 정의 등록 정보와 확장 등록 정보를 사용할 수 있습니다. 시스템 정의 등록 정보는 미리 정의되어 있는 반면 확장 등록 정보는 RTR 파일에서 개발자가 직접 정의합니다.

scha_resource_get()을 사용하여 시스템 정의 등록 정보의 값을 얻을 때는 -O 옵션과 함께 등록 정보의 이름을 지정합니다. 이 경우 명령은 등록 정보의 **값**만 반환합니다. 예를 들어, 샘플 데이터 서비스에서 Monitor_start 메소드는 검사 프로그램을 찾아야만 시작할 수 있습니다. 검사 프로그램은 RT_basedir 등록 정보에서 가리키는 데이터 서비스의 기본 디렉토리에 있으므로 Monitor_start 메소드는 다음과 같이 RT_basedir의 값을 검색하여 RT_BASEDIR 변수에 저장합니다.

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

확장 등록 정보의 경우 -O 옵션을 사용하여 확장 등록 정보라고 지정하고 등록 정보의 이름을 마지막 인자로 제공해야 합니다. 이때 명령은 등록 정보의 **유형**과 **값**을 모두 반환합니다. 예를 들어, 샘플 데이터 서비스에서 검사 프로그램은 다음과 같이 Probe_timeout 확장 등록 정보의 유형과 값을 검색한 다음 awk 명령을 사용하여 값만 PROBE_TIMEOUT 쉘 변수에 저장합니다.

```
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}'`
```

데이터 서비스 제어

데이터 서비스는 클러스터의 응용 프로그램 데몬을 활성화하기 위한 `Start` 또는 `Prenet_start` 메소드와 클러스터의 응용 프로그램 데몬을 중지하기 위한 `Stop` 또는 `Postnet_stop` 메소드를 제공해야 합니다. 샘플 데이터 서비스는 `Start` 및 `Stop` 메소드를 구현합니다. `Prenet_start` 및 `Postnet_stop`을 대신 사용할 수 있는 경우에 대한 자세한 내용은 44 페이지 "사용할 Start 및 Stop 메소드 결정"을 참조하십시오.

Start 메소드 작동 방식

데이터 서비스 자원을 포함하는 자원 그룹이 노드에서 온라인 상태가 되거나 자원 그룹이 이미 온라인 상태에서 자원이 활성화되는 경우 RGM은 해당 클러스터 노드에서 `Start` 메소드를 실행합니다. 샘플 응용 프로그램의 `Start` 메소드는 해당 노드에서 `in.named`(DNS) 데몬을 활성화합니다.

이 절에서는 샘플 응용 프로그램에 사용된 `Start` 메소드의 주요 부분에 대해 설명합니다. `parse_args()` 함수, `syslog()` 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

`Start` 메소드의 전체 목록은 258 페이지 "Start 메소드 코드 목록"을 참조하십시오.

Start 메소드의 기능

샘플 데이터 서비스의 `Start` 메소드는 DNS를 시작하기 전에 구성 디렉토리와 구성 파일(`named.conf`)이 액세스 및 사용 가능한지 확인합니다. `named.conf`의 정보는 DNS 작업을 성공적으로 수행하는 데 필수적입니다.

이 콜백 메소드는 PMF(`pmfadm`)를 사용하여 DNS 데몬(`in.named`)을 시작합니다. DNS가 중단되거나 시작하는 데 실패할 경우 PMF는 지정된 간격 동안 지정된 횟수만큼 DNS 데몬을 시작합니다. 재시도 횟수와 간격은 데이터 서비스 RTR 파일의 등록 정보에서 지정합니다.

구성 확인

DNS를 실행하려면 구성 디렉토리의 `named.conf` 파일에 있는 정보가 필요합니다. 따라서 `Start` 메소드는 DNS를 시작하기 전에 디렉토리와 파일을 액세스할 수 있는지 확인하기 위해 몇 가지 온전성 검사를 수행합니다.

`Confdir` 확장 등록 정보는 구성 디렉토리의 경로를 제공합니다. 이 등록 정보 자체는 RTR 파일에 정의되어 있습니다. 그러나 데이터 서비스를 구성할 때 클러스터 관리자가 실제 위치를 지정합니다.

샘플 데이터 서비스에서 Start 메소드는 scha_resource_get() 함수를 사용하여 구성 디렉토리의 위치를 검색합니다.

주 – Confdir이 확장 등록 정보이므로 scha_resource_get()은 유형과 값을 모두 반환합니다. awk 명령은 값만 검색하여 쉘 변수인 CONFIG_DIR에 저장합니다.

```
# find the value of Confdir set by the cluster administrator at the time of
# adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the "type" as well as the "value" for the
# extension properties. Get only the value of the extension property
CONFIG_DIR=`echo $config_info | awk '{print $2}'`
```

Start 메소드는 CONFIG_DIR 값을 사용하여 디렉토리에 액세스할 수 있는지 확인합니다. 액세스할 수 없는 경우 Start는 오류 메시지를 기록하고 오류 상태로 종료합니다. 88 페이지 "Start 종료 상태"를 참조하십시오.

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

응용 프로그램 데몬을 시작하기 전에 이 메소드는 named.conf 파일이 존재하는지 확인하기 위해 최종 검사를 수행합니다. 파일이 없는 경우 Start는 오류 메시지를 기록하고 오류 상태로 종료합니다.

```
# Change to the $CONFIG_DIR directory in case there are relative
# pathnames in the data files.
cd $CONFIG_DIR

# Check that the named.conf file is present in the $CONFIG_DIR directory
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} File $CONFIG_DIR/named.conf is missing or empty"
   exit 1
fi
```

응용 프로그램 시작

이 메소드는 프로세스 관리자 기능(pmfadm)을 사용하여 응용 프로그램을 시작합니다. pmfadm 명령을 사용하면 지정된 시간 프레임 동안 응용 프로그램을 재시작할 횟수를 설정할 수 있습니다. RTR 파일에는 응용 프로그램 재시작을 시도하는 횟수를 지정하는 Retry_count 등록 정보와 시도 기간을 지정하는 Retry_interval 등록 정보가 포함되어 있습니다.

Start 메소드는 scha_resource_get() 함수를 사용하여 Retry_count 및 Retry_interval 값을 검색하고 이러한 값을 쉘 변수에 저장합니다. Start 메소드는 -n과 -t 옵션을 사용하여 이러한 값을 pmfadm으로 전달합니다.

```
# Get the value for retry count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME`
# Get the value for retry interval from the RTR file. This value is in seconds
# and must be converted to minutes for passing to pmfadm. Note that the
# conversion rounds up; for example, 50 seconds rounds up to 1 minute.
((RETRY_INTRVAL=`scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME` / 60))

# Start the in.named daemon under the control of PMF. Let it crash and restart
# up to $RETRY_COUNT times in a period of $RETRY_INTERVAL; if it crashes
# more often than that, PMF will cease trying to restart it.
# If there is a process already registered under the tag
# <$PMF_TAG>, then PMF sends out an alert message that the
# process is already running.
pmfadm -c $PMF_TAAG -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Log a message indicating that HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.err \
         -t [$SYSLOG_TAG] \
         "${ARGV0} HA-DNS successfully started"
fi
exit 0
```

Start 종료 상태

Start 메소드는 기본 응용 프로그램이 실제로 실행 중이고 사용 가능할 때까지 성공 상태로 종료해서는 안 되며 특히 종속된 다른 데이터 서비스가 있는 경우에는 더욱 그러합니다. 성공을 확인하는 한 가지 방법은 Start 메소드를 종료하기 전에 응용 프로그램을 검사하여 실행 중인지 확인하는 것입니다. 데이터베이스와 같은 복잡한 응용 프로그램의 경우 응용 프로그램이 충돌 복구를 초기화 및 수행할 수 있을 만큼 RTR 파일의 Start_timeout 등록 정보 값을 충분히 높게 설정해야 합니다.

주 – 샘플 데이터 서비스의 응용 프로그램 자원(DNS)은 신속하게 시작되므로 샘플 데이터 서비스는 성공 상태로 종료하기 전에 DNS가 실행 중인지 확인하기 위해 폴링을 수행하지 않습니다.

이 메소드가 DNS를 시작하지 못하고 실패 상태로 종료할 경우 RGM은 대응 방법을 결정하는 Failover_mode 등록 정보를 검사합니다. 샘플 데이터 서비스는 Failover_mode 등록 정보를 명시적으로 설정하지 않으므로 클러스터 관리자가 기본값을 무시하고 다른 값을 지정하지 않은 경우 이 등록 정보는 기본값 NONE을 가집니다. 이 경우 RGM은 데이터 서비스의 상태를 설정하는 것 외에 다른 조치를 취하지 않습니다. 동일한 노드에서 재시작하거나 다른 노드로 페일오버하려면 클러스터 관리자가 해당 작업을 시작해야 합니다.

Stop 메소드 작동 방식

RGM은 HA-DNS 자원을 포함하는 자원 그룹이 노드에서 오프라인 상태가 되거나 자원 그룹이 온라인 상태에서 자원이 비활성화된 경우 해당 클러스터 노드에서 Stop 메소드를 실행합니다. 이 메소드는 해당 노드에서 in.named(DNS) 데몬을 중지합니다.

이 절에서는 샘플 응용 프로그램에 사용된 Stop 메소드의 주요 부분에 대해 설명합니다. parse_args() 함수, syslog() 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

Stop 메소드의 전체 목록은 261 페이지 "Stop 메소드 코드 목록"을 참조하십시오.

Stop 메소드의 기능

데이터 서비스 중지를 시도할 때는 기본적으로 두 가지 사항을 고려해야 합니다. 첫 번째는 순차적 종료를 수행하는 것입니다. 순차적 종료를 수행하는 최선의 방법은 pmfadm을 통해 SIGTERM 신호를 보내는 것입니다.

두 번째 고려 사항은 데이터 서비스를 Stop_failed 상태가 되지 않도록 실제로 중지하는 것입니다. 이를 수행하는 가장 좋은 방법은 pmfadm을 통해 SIGKILL 신호를 보내는 것입니다.

샘플 데이터 서비스의 Stop 메소드는 이러한 사항을 모두 고려합니다. 이 메소드는 SIGTERM 신호를 보낸 다음 이 신호가 데이터 서비스를 중지하는 데 실패할 경우 SIGKILL 신호를 보냅니다.

DNS 중지를 시도하기 전에 Stop 메소드는 프로세스가 실제로 실행 중인지 확인합니다. 프로세스가 실행 중이면 Stop은 PMF(pmfadm)를 사용하여 이를 중지합니다.

이 Stop 메소드는 멱등성이 보장됩니다. 먼저 Start 메소드 호출을 통해 데이터 서비스를 시작하지 않은 채 RGM에서 Stop 메소드를 두 번 호출해서는 안 되지만 자원이 시작된 적이 없거나 저절로 중지된 경우에도 해당 자원에서 Stop 메소드를 호출할 수 있습니다. 따라서 이 Stop 메소드는 DNS가 실행 중이 아닌 경우에도 성공 상태로 종료합니다.

응용 프로그램 중지

Stop 메소드는 데이터 서비스를 중지하기 위한 2계층 방식,즉 pmfadm을 통해 SIGTERM 신호를 사용하는 순차적이거나 완만한 방식과 SIGKILL 신호를 사용하는 돌발적이거나 하드한 방식을 제공합니다. Stop 메소드는 Stop_timeout 값(Stop 메소드가 반환되어야 하는 시간)을 가져옵니다. 그런 다음 Stop 메소드는 다음 샘플 코드와 같이 이 시간의 80%를 완만하게 중지하는 데 할당하고 15%를 돌발적으로 중지하는 데 할당합니다(5%는 예약됨).

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

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

Stop 메소드는 pmfadm -q를 사용하여 DNS 데몬이 실행 중인지 확인합니다. DNS 데몬이 실행 중이면 Stop은 먼저 DNS 프로세스를 종료하기 위해 pmfadm -s를 사용하여 TERM 신호를 보냅니다. 시간 초과값의 80%가 지난 후에도 이 신호가 프로세스를 종료하지 못할 경우 Stop은 SIGKILL 신호를 보냅니다. 이 신호도 시간 초과값의 15% 이내에 프로세스를 종료하지 못하면 이 메소드는 오류 메시지를 기록하고 오류 상태로 종료합니다.

pmfadm이 프로세스를 종료할 경우 이 메소드는 프로세스가 중지되었다는 메시지를 기록하고 성공 상태로 종료합니다.

DNS 프로세스가 실행 중이 아니면 이 메소드는 DNS 프로세스가 실행 중이 아니라는 메시지를 기록하고 성공 상태로 종료합니다. 다음 코드 샘플은 Stop에서 pmfadm을 사용하여 DNS 프로세스를 중지하는 방법을 보여줍니다.

```
# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG; then
   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $RESOURCE_NAME.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err \
          -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"

      # Since the data service did not stop with a SIGTERM signal, use
      # SIGKILL now and wait for another 15% of the total timeout value.
      pmfadm -s $PMF_TAG -w $HARD_TIMEOUT KILL
      if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err \
          -t [$SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESSFUL"
         exit 1
      fi
fi
else
   # The data service is not running as of now. Log a message and
   # exit success.
   logger -p ${SYSLOG_FACILITY}.err \
           -t [$SYSLOG_TAG] \
           "HA-DNS is not started"

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service resource in STOP_FAILED State.
   exit 0
fi

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

Stop 종료 상태

Stop 메소드는 기본 응용 프로그램이 실제로 중지될 때까지 성공 상태로 종료해서는 안 되며 특히 종속된 다른 데이터 서비스가 있는 경우에는 더욱 그러합니다. 그렇지 않을 경우 데이터가 손상될 수 있습니다.

데이터베이스와 같은 복잡한 응용 프로그램의 경우 중지 도중에 응용 프로그램을 정리할 수 있을 만큼 RTR 파일의 Stop_timeout 등록 정보 값을 충분히 높게 설정해야 합니다.

이 메소드가 DNS를 중지하지 못하고 실패 상태로 종료할 경우 RGM은 대응 방법을 결정하는 Failover_mode 등록 정보를 검사합니다. 샘플 데이터 서비스는 Failover_mode 등록 정보를 명시적으로 설정하지 않으므로 클러스터 관리자가 기본값을 무시하고 다른 값을 지정하지 않은 경우 이 등록 정보는 기본값 NONE을 가집니다. 이 경우 RGM은 데이터 서비스의 상태를 Stop_failed로 설정하는 것 외에 다른 조치를 취하지 않습니다. 클러스터 관리자가 응용 프로그램을 강제로 중지하고 Stop_failed 상태를 지워야 합니다.

오류 모니터 정의

샘플 응용 프로그램은 DNS 자원(in.named)의 안정성을 모니터하기 위해 기본 오류 모니터를 구현합니다. 오류 모니터는 다음 요소로 구성됩니다.

- dns_probe - nslookup을 사용하여 샘플 데이터 서비스가 제어하는 DNS 자원이 실행 중인지 확인하는 사용자 정의 프로그램입니다. DNS가 실행 중이 아닐 경우 이 메소드는 DNS를 로컬로 다시 시작하기 위해 시도하거나 재시작 시도 횟수에 따라 데이터 서비스를 다른 노드에 재할당하도록 RGM에 요청합니다.
- dns_monitor_start - dns_probe를 시작하는 콜백 메소드입니다. 모니터링이 사용 가능한 경우 샘플 데이터 서비스가 온라인 상태가 되면 RGM에서 dns_monitor_start를 자동으로 호출합니다.
- dns_monitor_stop - dns_probe를 중지하는 콜백 메소드입니다. 샘플 데이터 서비스를 오프라인 상태로 전환하기 전에 RGM에서 자동으로 dns_monitor_stop을 호출합니다.
- dns_monitor_check - PROBE 프로그램이 데이터 서비스를 새 노드로 페일오버할때 구성 디렉토리를 사용할 수 있는지 확인하기 위해 Validate 메소드를 호출하는 콜백 메소드입니다.

검사 프로그램 작동 방식

dns_probe 프로그램은 샘플 데이터 서비스가 제어하는 DNS 자원이 실행 중인지 확인하는 지속적으로 실행되는 프로세스를 구현합니다. dns_probe는 샘플 데이터 서비스가 온라인 상태가 된 후 RGM에서 자동으로 실행하는 dns_monitor_start 메소드에 의해 시작됩니다. 데이터 서비스는 RGM에서 샘플 데이터 서비스를 오프라인 상태로 전환하기 전에 호출하는 dns_monitor_stop 메소드에 의해 중지됩니다.

이 절에서는 샘플 응용 프로그램에 사용된 PROBE 메소드의 주요 부분에 대해 설명합니다. parse_args() 함수, syslog() 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

PROBE 메소드의 전체 목록은 264 페이지 "PROBE 프로그램 코드 목록"을 참조하십시오.

검사 프로그램의 기능

검사는 무한 루프로 실행되며 nslookup을 사용하여 적절한 DNS 자원이 실행 중인지 확인합니다. DNS가 실행 중인 경우 검사는 Thorough_probe_interval 시스템 정의 등록 정보에 의해 설정된 지정된 간격 동안 일시 정지했다가 다시 검사를 수행합니다. DNS가 실행 중이 아닌 경우 이 프로그램은 DNS를 로컬로 다시 시작하기 위해 시도하거나 재시작 시도 횟수에 따라 데이터 서비스를 다른 노드에 재할당하도록 RGM에 요청합니다.

등록 정보 값 얻기

이 프로그램에는 다음 등록 정보 값이 필요합니다.

- Thorough_probe_interval – 검사가 일시 정지하는 기간 설정
- Probe_timeout – 검사를 수행하는 nslookup 명령에서 검사의 시간 초과값 적용
- Network_resources_used – DNS가 실행 중인 IP 주소 검색
- Retry_count 및 Retry_interval – 재시작 시도 횟수 및 이러한 시도가 계산되는 기간 지정
- RT_basedir – PROBE 프로그램과 gettime 유틸리티가 포함된 디렉토리 검색

scha_resource_get() 함수는 다음과 같이 이러한 등록 정보의 값을 검색하여 쉘 변수에 저장합니다.

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

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

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

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

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

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

주 – Thorough_probe_interval과 같은 시스템 정의 등록 정보의 경우 scha_resource_get() 함수는 값만 반환합니다. Probe_timeout과 같은 확장 등록 정보의 경우 scha_resource_get() 함수는 유형과 값을 반환합니다. 값만 검색하려면 awk 명령을 사용합니다.

서비스의 안정성 검사

검사 자체는 nslookup 명령의 무한 while 루프입니다. while 루프 앞에서 nslookup 응답을 저장하기 위한 임시 파일이 설정됩니다. probefail 및 retries 변수는 0으로 초기화됩니다.

```
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0
```

while 루프는 다음 작업을 수행합니다.

- 검사의 일지 정지 간격을 설정합니다.
- hatimerun을 사용하여 nslookup을 시작하고 Probe_timeout 값을 전달하며 대상 호스트를 식별합니다.
- nslookup 반환 코드의 성공 또는 실패에 기초하여 probefail 변수를 설정합니다.
- probefail이 1 (실패)로 설정된 경우 다른 DNS 서버가 아닌 샘플 데이터 서비스에서 nslookup에 대한 응답이 오는지 확인합니다.

다음은 while 루프 코드입니다.

```
while :
do
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep
   # for a duration of THOROUGH_PROBE_INTERVAL.
   sleep $PROBE_INTERVAL

   # Run an nslookup command of the IP address on which DNS is serving.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
   > $DNSPROBEFILE 2>&1

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

   # Make sure that the reply to nslookup comes from the HA-DNS
   # server and not from another nameserver mentioned in the
   # /etc/resolv.conf file.
   if [ $probefail -eq 0 ]; then
# Get the name of the server that replied to the nslookup query.
```

```
   SERVER=` awk ' $1=="Server:" { print $2 }' \
   $DNSPROBEFILE | awk -F. ' { print $1 } ' `
   if [ -z "$SERVER" ]; then
      probefail=1
      else
         if [ $SERVER != $DNS_HOST ]; then
            probefail=1
         fi
   fi
fi
```

재시작 및 페일오버 비교

`probefail` 변수가 0(성공)이 아닌 다른 값일 경우 `nslookup` 명령이 시간 초과되었거나 샘플 서비스의 DNS가 아닌 다른 서버에서 응답을 보낸 것입니다. 이러한 경우 DNS 서버는 예상대로 작동하지 않으며 오류 모니터는 `decide_restart_or_failover()` 함수를 호출하여 데이터 서비스를 로컬로 다시 시작할 것인지 아니면 데이터 서비스를 다른 노드로 재할당하도록 RGM에 요청할 것인지 결정합니다. `probefail` 변수가 0일 경우 검사에 성공했다는 메시지가 생성됩니다.

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

`decide_restart_or_failover()` 함수는 시간 창(`Retry_interval`)과 실패 횟수(`Retry_count`)를 사용하여 DNS를 로컬로 재시작할 것인지 아니면 데이터 서비스를 다른 노드로 재할당하도록 RGM에 요청할 것인지 결정합니다. 이 함수는 다음 조건부 논리를 구현합니다. 264 페이지 "PROBE 프로그램 코드 목록"의 `decide_restart_or_failover()` 코드 목록에 코드가 들어 있습니다.

- 첫 번째 실패인 경우 데이터 서비스를 다시 시작합니다. 오류 메시지를 기록하고 `retries` 변수에서 카운터를 증가시킵니다.
- 첫 번째 실패가 아니지만 창이 초과된 경우 데이터 서비스를 다시 시작합니다. 오류 메시지를 기록하고 카운터를 재설정한 다음 창을 진행합니다.
- 시간이 여전히 창 안에 있고 재시도 카운터가 초과된 경우 다른 노드로 페일오버합니다. 페일오버에 실패한 경우 오류를 기록하고 상태 1(실패)로 검사 프로그램을 종료합니다.
- 시간이 여전히 창 안에 있지만 재시도 카운터가 초과되지 않은 경우 데이터 서비스를 다시 시작합니다. 오류 메시지를 기록하고 `retries` 변수에서 카운터를 증가시킵니다.

시간 간격 동안에 재시작 횟수가 한도에 도달할 경우 이 함수는 데이터 서비스를 다른 노드로 재할당하도록 RGM에 요청합니다. 재시작 횟수가 한도 이내이거나 간격이 초과되어 카운트가 다시 시작된 경우 이 함수는 동일한 노드에서 DNS의 재시작을 시도합니다. 이 함수와 관련하여 다음 사항에 주의하십시오.

- 재시작 간의 시간을 추적하기 위해 gettime 유틸리티가 사용됩니다. 이 유틸리티는 RT_basedir 디렉토리에 있는 C 프로그램입니다.
- Retry_count 및 Retry_interval 시스템 정의 자원 등록 정보는 재시작 시도 횟수와 이러한 시도가 계산되는 간격을 결정합니다. 이러한 등록 정보의 기본값은 RTR 파일에서 5분(300초) 동안 두 번 시도를 수행하는 것이지만 클러스터 관리자가 값을 변경할 수 있습니다.
- 동일한 노드에서 데이터 서비스를 다시 시작하기 위해 restart_service() 함수가 호출됩니다. 이 함수에 대한 자세한 내용은 다음 절, 95 페이지 "데이터 서비스 재시작"을 참조하십시오.
- scha_control() API 함수는 GIVEOVER 옵션과 함께 사용되어 샘플 데이터 서비스를 포함하는 자원 그룹을 오프라인 상태로 전환하고 다른 노드에서 다시 온라인 상태로 전환합니다.

데이터 서비스 재시작

동일한 노드에서 데이터 서비스를 재시작하기 위해 decide_restart_or_failover()에 의해 restart_service() 함수가 호출됩니다. 이 함수는 다음 논리를 실행합니다.

- 데이터 서비스가 여전히 PMF에 등록되어 있는지 확인합니다. 데이터 서비스가 여전히 등록되어 있는 경우 다음 작업을 수행합니다.
  - 데이터 서비스의 Stop 메소드 이름과 Stop_timeout 값을 얻습니다.
  - hatimerun을 사용하여 데이터 서비스에 대한 Stop 메소드를 시작함으로써 Stop_timeout 값을 전달합니다.
  - 데이터 서비스가 성공적으로 중지된 경우 데이터 서비스의 Start 메소드 이름과 Start_timeout 값을 얻습니다.
  - hatimerun을 사용하여 데이터 서비스에 대한 Start 메소드를 시작함으로써 Start_timeout 값을 전달합니다.
- 데이터 서비스가 더 이상 PMF에 등록되지 않은 경우 데이터 서비스가 PMF에서 허용되는 최대 재시도 횟수를 초과했다는 것을 의미합니다. 따라서 scha_control() 함수가 GIVEOVER 옵션과 함께 호출되어 데이터 서비스를 다른 노드로 페일오버합니다.

```
function restart_service
{

        # To restart the data service, first verify that the
        # data service itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the data service is still registered under
                # PMF, first stop the data service and start it back up again.

                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT \
```

```
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
        STOP_METHOD=`scha_resource_get -O STOP \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
        hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                -T $RESOURCETYPE_NAME

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

        # Obtain the START method name and the START_TIMEOUT value for
        # this resource.
        START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
        START_METHOD=`scha_resource_get -O START \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
        hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
                -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                -T $RESOURCETYPE_NAME

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


    else
            # The absence of the TAG for the dataservice
            # implies that the data service has already
            # exceeded the maximum retries allowed under PMF.
            # Therefore, do not attempt to restart the
            # data service again, but try to failover
            # to another node in the cluster.
            scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                    -R $RESOURCE_NAME
    fi

    return 0
}
```

검사 종료 상태

로컬 재시작 시도에 실패하고 다른 노드로 페일오버하려는 시도 역시 실패한 경우 샘플 데이터 서비스의 PROBE 프로그램은 실패 상태로 종료하고 페일오버 시도가 실패했습니다라는 메시지를 기록합니다.

Monitor_start 메소드 작동 방식

RGM은 샘플 데이터 서비스가 온라인 상태가 된 후 Monitor_start 메소드를 호출하여 dns_probe 메소드를 시작합니다.

이 절에서는 샘플 응용 프로그램에 사용된 Monitor_start 메소드의 주요 부분에 대해 설명합니다. parse_args() 함수, syslog() 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

Monitor_start 메소드의 전체 목록은 269 페이지 "Monitor_start 메소드 코드 목록"을 참조하십시오.

Monitor_start 메소드의 기능

이 메소드는 PMF(pmfadm)를 사용하여 검사를 시작합니다.

검사 시작

Monitor_start 메소드는 RT_basedir 등록 정보 값을 가져와서 PROBE 프로그램에 대한 전체 경로 이름을 구성합니다. 이 메소드는 pmfadm의 무한 재시도 옵션(-n -1, -t -1)을 사용하여 검사를 시작하며 이는 검사를 시작하는 데 실패할 경우 PMF에서 시간 제한 없이 검사 시작을 무한대로 시도한다는 것을 의미합니다.

```
# Find where the probe program resides by obtaining the value of the
# RT_basedir property of the resource.
RT_BASEDIR=‘scha_resource_get -O RT_basedir -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAME‘

# Start the probe for the data service under PMF. Use the infinite retries
# option to start the probe. Pass the resource name, type, and group to the
# probe program.
pmfadm -c $RESOURCE_NAME.monitor -n -1 -t -1 \
   $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME
```

Monitor_stop 메소드 작동 방식

RGM은 샘플 데이터 서비스가 오프라인 상태가 되었을 때 Monitor_stop 메소드를 호출하여 dns_probe의 실행을 중지합니다.

이 절에서는 샘플 응용 프로그램에 사용된 Monitor_stop 메소드의 주요 부분에 대해 설명합니다. parse_args() 함수, syslog() 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

Monitor_stop 메소드의 전체 목록은 271 페이지 "Monitor_stop 메소드 코드 목록"을 참조하십시오.

Monitor_stop 메소드의 기능

이 메소드는 PMF(pmfadm)를 사용하여 검사가 실행 중인지 확인하고 검사가 실행 중일 경우 이를 중지합니다.

모니터 중지

Monitor_stop 메소드는 pmfadm -q를 사용하여 검사가 실행 중인지 확인하고 검사가 실행 중일 경우 pmfadm -s를 사용하여 이를 중지합니다. 검사가 이미 중지된 경우에도 이 메소드는 성공적으로 종료하므로 메소드의 멱등성이 보장됩니다.

주의 – TERM과 같은 마스크 가능한 신호가 아니라 KILL 신호를 pmfadm과 함께 사용하여 검사를 중지해야 합니다. 그렇지 않을 경우 Monitor_stop 메소드가 무한대로 정지되고 결국 시간이 초과될 수 있습니다. 이 문제의 원인은 PROBE 메소드가 데이터 서비스를 재시작하거나 페일오버해야 할 경우 scha_control() 을 호출하기 때문입니다. scha_control()이 데이터 서비스를 오프라인 상태로 전환하는 과정의 일부로 Monitor_stop을 호출할 때 Monitor_stop이 마스크 가능한 신호를 사용하면 Monitor_stop은 scha_control()이 완료되기를 기다리면서 일시 정지되고 scha_control()은 Monitor_stop이 완료되기를 기다리면서 일시 정지됩니다.

```
# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG; then
   pmfadm -s $PMF_TAG KILL
   if [ $? -ne 0 ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Could not stop monitor for resource " \
            $RESOURCE_NAME
           exit 1
   else
         # could successfully stop the monitor. Log a message.
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} Monitor for resource " $RESOURCE_NAME \
            " successfully stopped"
   fi
fi
exit 0
```

Monitor_stop 종료 상태

Monitor_stop 메소드는 PROBE 메소드를 중지할 수 없는 경우 오류 메시지를 기록합니다. RGM은 기본 노드에서 샘플 데이터 서비스를 MONITOR_FAILED 상태로 만들며 이로 인해 기본 노드에 패닉이 발생할 수 있습니다.

Monitor_stop은 검사가 중지되기 전까지 종료해서는 안 됩니다.

Monitor_check 메소드 작동 방식

PROBE 메소드가 데이터 서비스를 포함하는 자원 그룹을 새 노드로 페일오버하려고 시도할 때마다 RGM은 Monitor_check 메소드를 호출합니다.

이 절에서는 샘플 응용 프로그램에 사용된 Monitor_check 메소드의 주요 부분에 대해 설명합니다. parse_args() 함수, syslog() 함수와 같이 모든 콜백 메소드에 공통된 기능은 다루지 않습니다. 공통된 기능에 대해서는 81 페이지 "모든 메소드에 공통 기능 제공"에서 설명합니다.

Monitor_check 메소드의 전체 목록은 272 페이지 "Monitor_check 메소드 코드 목록"을 참조하십시오.

동시에 실행되는 다른 메소드와 충돌하지 않도록 Monitor_check 메소드를 구현해야 합니다.

Monitor_check 메소드는 새 노드에서 DNS 구성 디렉토리를 사용할 수 있는지 확인하기 위해 Validate 메소드를 호출합니다. Confdir 확장 등록 정보는 DNS 구성 디렉토리를 가리킵니다. 따라서 Monitor_check은 다음에 나온 것처럼 Validate 메소드의 경로 및 이름과 Confdir의 값을 얻은 다음 이 값을 Validate에 전달합니다.

```
# Obtain the full path for the Validate method from
# the RT_basedir property of the resource type.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
   -G $RESOURCEGROUP_NAM?

# Obtain the name of the Validate method for this resource.
VALIDATE_METHOD=`scha_resource_get -O Validate \
   -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?

# Obtain the value of the Confdir property in order to start the
# data service. Use the resource name and the resource group entered to
# obtain the Confdir value set at the time of adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
 -G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the type as well as the value for extension
# properties. Use awk to get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Call the validate method so that the dataservice can be failed over
# successfully to the new node.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
   -T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR
```

샘플 응용 프로그램에서 데이터 서비스를 호스트하는 데 특정 노드가 적합한지 확인하는 방법에 대해서는 100 페이지 "Validate 메소드 작동 방식"을 참조하십시오.

등록 정보 업데이트 처리

샘플 데이터 서비스는 클러스터 관리자에 의한 등록 정보 업데이트를 처리하기 위해 `Validate` 및 `Update` 메소드를 구현합니다.

`Validate` 메소드 작동 방식

RGM은 자원이 만들어질 때와 관리 작업을 통해 자원이나 자원을 포함하는 그룹의 등록 정보가 업데이트될 때 `Validate` 메소드를 호출합니다. RGM은 작성 또는 업데이트가 적용되기 전에 `Validate`를 호출하며 임의의 노드에서 이 메소드가 실패 종료 코드를 반환할 경우 작성 또는 업데이트가 취소됩니다.

RGM에서 등록 정보를 설정하거나 모니터가 `Status` 및 `Status_msg` 자원 등록 정보를 설정할 때가 아니라 클러스터 관리자가 자원 또는 자원 그룹 등록 정보를 변경하는 경우에만 `Validate`가 호출됩니다.

주 – `Monitor_check` 메소드는 `PROBE` 메소드가 데이터 서비스를 새 노드로 페일오버할 때마다 `Validate` 메소드를 명시적으로 호출합니다.

`Validate` 메소드의 기능

RGM은 업데이트 중인 등록 정보와 값을 포함하여 다른 메소드로 전달되는 항목에 대한 추가 인자와 함께 `Validate`를 호출합니다. 따라서 샘플 데이터 서비스의 이 메소드는 추가 인자를 처리하기 위해 다른 `parse_args()` 함수를 구현해야 합니다.

샘플 데이터 서비스의 `Validate` 메소드는 단일 등록 정보, 즉 `Confdir` 확장 등록 정보를 확인합니다. 이 등록 정보는 DNS의 성공적인 작업에 중요한 DNS 구성 디렉토리를 가리킵니다.

주 – DNS가 실행 중인 경우 구성 디렉토리를 변경할 수 없기 때문에 `Confdir` 등록 정보는 RTR 파일에서 `TUNABLE = AT_CREATION`으로 선언됩니다. 따라서 `Validate` 메소드는 업데이트 결과로 `Confdir` 등록 정보를 확인하기 위해 호출되지 않으며 데이터 서비스 자원을 만드는 중에만 호출됩니다.

RGM이 `Validate`로 전달하는 등록 정보 중 하나가 `Confdir`인 경우 `parse_args()` 함수는 해당 값을 검색하여 저장합니다. 그런 다음 `Validate`는 `Confdir`의 새 값이 가리키는 디렉토리가 액세스 가능한지와 `named.conf` 파일이 해당 디렉토리에 있으며 데이터를 포함하는지를 확인합니다.

parse_args() 함수가 RGM에서 전달한 명령줄 인자에서 Confdir의 값을 검색할 수 없는 경우에도 Validate는 Confdir 등록 정보를 검증하려고 시도합니다. Validate는 scha_resource_get()을 사용하여 정적 구성에서 Confdir의 값을 얻습니다. 그런 다음 구성 디렉토리가 액세스 가능한지와 비어 있지 않은 named.conf 파일을 포함하는지 확인하기 위해 동일한 검사를 수행합니다.

Validate가 실패 상태로 종료할 경우 Confdir뿐 아니라 모든 등록 정보의 업데이트 또는 작성에 실패합니다.

Validate 메소드 구문 분석 함수

RGM은 다른 콜백 메소드와 달리 여러 인자 집합을 Validate 메소드에 전달하므로 Validate는 다른 메소드에는 없는 인자를 구문 분석하기 위한 함수를 필요로 합니다. Validate 및 다른 콜백 메소드에 전달되는 인자에 대한 자세한 내용은 rt_callbacks(1HA) 설명서 페이지를 참조하십시오. 다음은 Validate parse_args() 함수를 보여주는 코드 샘플입니다.

```
#############################################################################
# Parse Validate arguments.
#
function parse_args # [args...]
{

   typeset opt
   while getopts 'cur:x:g:R:T:G:' opt
   do
         case "$opt" in
         R)
                  # Name of the DNS resource.
                  RESOURCE_NAME=$OPTARG
                  ;;
         G)
                  # Name of the resource group in which the resource is
                  # configured.
                  RESOURCEGROUP_NAME=$OPTARG
                  ;;
         T)
                  # Name of the resource type.
                  RESOURCETYPE_NAME=$OPTARG
                  ;;
         r)
                  # The method is not accessing any system defined
                  # properties so this is a no-op
                  ;;
         g)
                  # The method is not accessing any resource group
                  # properties, so this is a no-op
                  ;;
         c)
                  # Indicates the Validate method is being called while
                  # creating the resource, so this flag is a no-op.
                  ;;
```

```
        u)
                # Indicates the updating of a property when the
                # resource already exists. If the update is to the
                # Confdir property then Confdir should appear in the
                # command-line arguments. If it does not, the method must
                # look for it specifically using scha_resource_get.
                UPDATE_PROPERTY=1
                ;;
        x)
                # Extension property list. Separate the property and
                # value pairs using "=" as the separator.
                PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                VAL=`echo $OPTARG | awk -F= '{print $2}'`
                # If the Confdir extension property is found on the
                # command line, note its value.
                if [ $PROPERTY == "Confdir" ]; then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                fi
                ;;
        *)
                logger -p ${SYSLOG_FACILITY}.err \
                -t [$SYSLOG_TAG] \
                "ERROR: Option $OPTARG unknown"
                exit 1
                ;;
        esac
    done
}
```

다른 메소드의 `parse_args()` 함수와 마찬가지로 이 함수는 자원 이름을 캡처하는 `R` 플래그, 자원 그룹 이름을 캡처하는 `G` 플래그 및 RGM이 전달한 자원 유형을 캡처하는 `T` 플래그를 제공합니다.

자원을 업데이트하는 동안 확장 등록 정보를 검증하기 위해 이 메소드가 호출되므로 시스템 정의 등록 정보를 나타내는 `r` 플래그, 자원 그룹 등록 정보를 나타내는 `g` 플래그 및 자원 작성 도중 수행되는 검증을 나타내는 `c` 플래그는 무시됩니다.

`u` 플래그는 `UPDATE_PROPERTY` 쉘 변수의 값을 `1(TRUE)`로 설정하며 `x` 플래그는 업데이트할 등록 정보의 이름과 값을 캡처합니다. 업데이트되는 등록 정보 중 하나가 `Confdir`인 경우 `CONFDIR` 쉘 변수에 해당 값이 저장되며 `CONFDIR_FOUND` 변수가 `1(TRUE)`로 설정됩니다.

Confdir 검증

`MAIN` 함수에서 `Validate`는 먼저 `CONFDIR` 변수를 빈 문자열로 설정하고 `UPDATE_PROPERTY` 및 `CONFDIR_FOUND`를 `0`으로 설정합니다.

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

이어서 `Validate`는 `parse_args()`를 호출하여 RGM에서 전달한 인자를 구문 분석합니다.

```
parse_args "$@"
```

그런 다음 Validate는 등록 정보 업데이트의 결과로 Validate가 호출되는지와 Confdir 확장 등록 정보가 명령줄에 있었는지를 검사합니다. Validate는 Confdir 등록 정보에 값이 있는지 확인한 다음, 값이 없을 경우 오류 메시지와 함께 실패 상태로 종료합니다.

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

# Verify that the Confdir property has a value. If not there is a failure
# and exit with status 1
if [[ -z $CONFDIR ]]; then
         logger -p ${SYSLOG_FACILITY}.err \
            "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
         exit 1
fi
```

주 – 특히 앞의 코드는 Validate가 업데이트 결과로 호출되는지($UPDATE_PROPERTY == 1) 및 명령줄에서 등록 정보가 발견되지 **않았는지**(CONFDIR_FOUND == 0) 확인하여 이에 해당하는 경우 scha_resource_get() 을 사용하여 Confdir의 기존 값을 검색합니다. Confdir이 명령줄에서 발견된 경우(CONFDIR_FOUND == 1) scha_resource_get() 이 아닌 parse_args() 함수에서 CONFDIR 값을 얻습니다.

그런 다음 Validate 메소드는 CONFDIR 값을 사용하여 디렉토리에 액세스할 수 있는지 확인합니다. 디렉토리에 액세스할 수 없는 경우 Validate는 오류 메시지를 기록하고 오류 상태로 종료합니다.

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

Confdir 등록 정보의 업데이트를 검증하기 전에 Validate는 named.conf 파일이 존재하는지 확인하는 최종 검사를 수행하여파일이 없는 경우 이 메소드는 오류 메시지를 기록하고 오류 상태로 종료합니다.

```
# Check that the named.conf file is present in the Confdir directory
if [ ! -s $CONFDIR/named.conf ]; then
         logger -p ${SYSLOG_FACILITY}.err \
            -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
         exit 1
fi
```

최종 검사에 통과한 경우 Validate는 성공을 나타내는 메시지를 기록하고 성공 상태로 종료합니다.

```
# Log a message indicating that the Validate method was successful.
logger -p ${SYSLOG_FACILITY}.err \
   -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

Validate 종료 상태

Validate가 성공(0) 상태로 종료할 경우 새 값으로 Confdir이 작성됩니다. Validate가 실패(1) 상태로 종료할 경우 Confdir 및 다른 등록 정보가 만들어지지 않으며 그 이유를 나타내는 메시지가 생성됩니다.

Update 메소드 작동 방식

RGM은 해당 등록 정보가 변경되었음을 실행 중인 자원에 알리기 위해 Update 메소드를 실행합니다. RGM은 클러스터 관리자가 자원 또는 자원 그룹의 등록 정보를 설정하는 데 성공한 후에 Update를 실행합니다. 이 메소드는 자원이 온라인 상태인 노드에서 호출됩니다.

Update 메소드의 기능

Update 메소드는 등록 정보를 업데이트하지 않으며 이는 RGM에 의해 수행됩니다. Update 메소드는 실행 중인 프로세스에 업데이트가 발생했음을 알립니다. 샘플 데이터 서비스에서 등록 정보 업데이트의 영향을 받는 유일한 프로세스는 오류 모니터이며, 따라서 오류 모니터 프로세스는 Update 메소드에 의해 중지 및 재시작됩니다.

Update 메소드는 오류 모니터가 실행 중인지 확인한 다음 pmfadm 명령을 사용하여 이를 중지해야 합니다. 이 메소드는 오류 모니터를 구현하는 검사 프로그램의 위치를 검색한 다음 pmfadm 명령을 사용하여 오류 모니터를 재시작합니다.

Update를 사용하여 모니터 중지

Update 메소드는 pmfadm -q를 사용하여 모니터가 실행 중인지 확인하고 모니터가 실행 중일 경우 pmfadm -s TERM을 사용하여 이를 중지합니다. 모니터가 성공적으로 종료하면 해당 결과에 대한 메시지가 클러스터 관리자에게 보내집니다. 모니터를 중지할 수 없는 경우 Update는 실패 상태로 종료하고 오류 메시지를 클러스터 관리자에게 보냅니다.

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

# Kill the monitor that is running already
```

```
pmfadm -s $PMF_TAG TERM
    if [ $? -ne 0 ]; then
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$SYSLOG_TAG] \
                 "${ARGV0} Could not stop the monitor"
       exit 1
    else
    # could successfully stop DNS. Log a message.
       logger -p ${SYSLOG_FACILITY}.err \
              -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                 "Monitor for HA-DNS successfully stopped"
    fi
```

모니터 다시 시작

모니터를 다시 시작하려면 Update 메소드에서 검사 프로그램을 구현하는 스크립트를 찾아야 합니다. 검사 프로그램은 RT_basedir 등록 정보에서 가리키는 데이터 서비스의 기본 디렉토리에 있으므로 Update는 다음과 같이 RT_basedir의 값을 검색하여 RT_BASEDIR 변수에 저장합니다.

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

그런 다음 Update는 pmfadm과 함께 RT_BASEDIR을 사용하여 dns_probe 프로그램을 재시작합니다. 성공할 경우 Update는 성공 상태로 종료하고 해당 결과에 대한 메시지를 클러스터 관리자에게 보냅니다. pmfadm이 검사 프로그램을 시작할 수 없는 경우 Update는 실패 상태로 종료하고 오류 메시지를 기록합니다.

Update 종료 상태

Update 메소드가 실패하면 자원이 “업데이트 실패” 상태가 됩니다. 이 상태는 자원의 RGM 관리에 영향을 주지 않지만 syslog() 함수를 통해 업데이트 작업이 실패했음을 관리 도구에 표시합니다.

6장

데이터 서비스 개발 라이브러리(DSDL)

이 장에서는 데이터 서비스 개발 라이브러리(DSDL)를 구성하는 응용 프로그램 프로그래밍 인터페이스에 대해 개괄적으로 설명합니다. DSDL은 `libdsdev.so` 라이브러리에서 구현되며 Sun Cluster 패키지에 포함되어 있습니다.

이 장은 다음 내용으로 구성되어 있습니다.

DSDL 개요

DSDL API는 자원 관리 API(RMAPI) 위에 놓이므로 RMAPI를 대체하는 것이 아니라 RMAPI 기능을 캡슐화하고 확장합니다. DSDL은 특정 Sun Cluster 통합 문제를 해결하는 미리 지정된 솔루션을 제공함으로써 데이터 서비스 개발을 단순화합니다. 결과적으로 응용 프로그램 시작, 종료 및 모니터 절차를 Sun Cluster와 통합하는 데 많은 시간을 낭비할 필요가 없기 때문에 응용 프로그램 고유의 고가용성 및 확장성 문제에 개발 시간의 대부분을 투자할 수 있습니다.

구성 등록 정보 관리

모든 콜백 메소드는 구성 등록 정보에 액세스해야 합니다. DSDL은 다음과 같은 방법으로 등록 정보에 대한 액세스를 지원합니다.

- 환경 초기화
- 등록 정보 값을 검색하기 위한 일반 함수 집합 제공

각 콜백 메소드의 시작 부분에서 호출해야 하는 `scds_initialize()` 함수는 다음 작업을 수행합니다.

- 명령줄 구문 분석 함수를 작성할 필요가 없도록 RGM이 콜백 메소드에 전달하는 명령줄 인자(`argc` 및 `argv[]`)를 검사하고 처리합니다.
- 다른 DSDL 함수에 사용되는 내부 데이터 구조를 설정합니다. 예를 들어, RGM에서 등록 정보 값을 검색하는 일반 함수는 해당 값을 이러한 구조에 저장합니다. 마찬가지로 RGM에서 검색된 값보다 우선하는 명령줄 값도 이러한 데이터 구조에 저장됩니다.
- 로깅 환경을 초기화하고 오류 모니터 검사 설정을 검증합니다.

주 – `Validate` 메소드의 경우 `scds_initialize()`는 명령줄에서 전달되는 등록 정보 값을 구문 분석하므로 `Validate`에 대한 구문 분석 함수를 작성할 필요가 없습니다.

DSDL은 자원 유형, 자원 및 자원 그룹 등록 정보와 일반적으로 사용되는 확장 등록 정보를 검색하는 함수 집합을 제공합니다. 이러한 함수는 다음 규칙을 사용하여 등록 정보에 대한 액세스를 표준화합니다.

- 각 함수는 `scds_initialize()`에서 반환한 핸들 인자만 사용합니다.
- 각 함수는 특정 등록 정보에 해당합니다. 함수의 반환 값 유형은 검색되는 등록 정보 값 유형과 일치합니다.
- `scds_initialize()`에서 값이 미리 계산되었으므로 함수는 오류를 반환하지 않습니다. 명령줄에서 새 값이 전달되지 않을 경우 함수는 RGM에서 값을 검색합니다.

데이터 서비스 시작 및 중지

`Start` 메소드는 클러스터 노드에서 데이터 서비스를 시작하는 데 필요한 작업을 수행합니다. 일반적으로 여기에는 자원 등록 정보 검색, 응용 프로그램 고유의 실행 파일 및 구성 파일 찾기, 올바른 명령줄 인자를 사용하여 응용 프로그램 시작 등이 포함됩니다.

scds_initialize() 함수는 자원 구성을 검색합니다. Start 메소드는 시작할 응용 프로그램의 구성 디렉토리와 파일을 식별하는 Confdir_list와 같은 특정 등록 정보의 값을 검색하기 위해 등록 정보 일반 함수를 사용할 수 있습니다.

Start 메소드는 scds_pmf_start()를 호출하여 PMF(Process Monitor Facility)의 제어 하에 응용 프로그램을 시작할 수 있습니다. PMF를 사용하면 프로세스에 적용할 모니터 수준을 지정하고 실패가 발생할 경우 프로세스를 재시작할 수 있습니다. DSDL에서 구현되는 Start 메소드의 예는 126 페이지 "xfnts_start 메소드"를 참조하십시오.

응용 프로그램이 실행 중이 아닐 때 노드에서 호출된 경우에도 Stop 메소드가 성공 상태로 종료되도록 Stop 메소드는 멱등원이어야 합니다. Stop 메소드가 실패하면 중지하려는 자원이 STOP_FAILED 상태로 설정되어 클러스터의 하드웨어가 재부트될 수 있습니다.

자원이 STOP_FAILED 상태가 되지 않도록 Stop 메소드는 자원을 중지하기 위해 노력해야 합니다. scds_pmf_stop() 함수는 자원을 중지하기 위해 단계별로 시도합니다. 이 함수는 먼저 SIGTERM 신호를 사용하여 자원 중지를 시도하며 이 시도가 실패할 경우 SIGKILL 신호를 사용합니다. 자세한 내용은 scds_pmf_stop(3HA) 설명서 페이지를 참조하십시오.

오류 모니터 구현

DSDL은 미리 지정된 모델을 제공하여 오류 모니터를 구현하는 복잡한 작업을 단순화합니다. Monitor_start 메소드는 노드에서 자원이 시작될 때 PMF의 제어 하에서 오류 모니터를 시작합니다. 오류 모니터는 자원이 노드에서 실행되는 동안 루프 상태로 실행됩니다. DSDL 오류 모니터의 고급 논리는 다음과 같습니다.

- scds_fm_sleep() 함수는 Thorough_probe_interval 등록 정보를 사용하여 검사 작업 간의 시간 간격을 결정합니다. 이 간격 동안 PMF에서 응용 프로그램 프로세스 실패를 확인하면 자원이 재시작됩니다.
- 검사 자체는 0(실패 없음)에서 100(완전한 실패)까지의 실패 심각도를 나타내는 값을 반환합니다.
- 검사 반환값이 scds_action() 함수로 전달됩니다. 이 함수는 Retry_interval 등록 정보의 간격 내에 누적된 실패 기록을 유지 관리합니다.
- scds_action() 함수는 실패가 발생할 경우 수행할 작업을 다음과 같이 결정합니다.
  - 누적 실패가 100 미만이면 아무 작업도 수행하지 않습니다.
  - 누적 실패가 100(완전한 실패)에 도달하면 데이터 서비스를 재시작합니다. Retry_interval을 초과하면 기록을 재설정합니다.
  - 재시작 횟수가 Retry_interval에 지정된 시간 내에서 Retry_count 등록 정보 값을 초과할 경우 데이터 서비스를 페일오버합니다.

네트워크 주소 정보 액세스

DSDL은 자원 및 자원 그룹에 대한 네트워크 주소 정보를 반환하는 일반 함수를 제공합니다. 예를 들어 `scds_get_netaddr_list()`는 자원이 사용하는 네트워크 주소 자원을 검색하여 오류 모니터가 응용 프로그램을 검사할 수 있게 합니다.

DSDL은 또한 TCP 기반 모니터를 위한 함수 집합을 제공합니다. 일반적으로 이러한 함수는 서비스에 대한 간단한 소켓 연결을 설정하고 데이터를 읽어 서비스에 기록한 다음 서비스와의 연결을 끊습니다. 검사 결과는 DSDL `scds_fm_action()` 함수로 전달되어 수행할 작업을 결정하는 데 사용될 수 있습니다.

TCP 기반 오류 모니터의 예는 139 페이지 "`xfnts_validate` 메소드"를 참조하십시오.

자원 유형 구현 디버깅

DSDL은 데이터 서비스의 디버깅을 지원하는 기능을 기본으로 제공합니다.

DSDL 유틸리티 `scds_syslog_debug()`는 디버깅 문을 자원 유형 구현에 추가하기 위한 기본 프레임워크를 제공합니다. 디버깅 **수준**(1-9 사이의 숫자)은 각 클러스터 노드의 각 자원 유형 구현에 대해 동적으로 설정할 수 있습니다. 모든 자원 유형 콜백 메소드는 1-9 사이의 정수만 포함하는 `/var/cluster/rgm/rt/`*rtname*`/loglevel`이라는 파일을 읽습니다. DSDL 함수 `scds_initialize()`는 이 파일을 읽고 내부적으로 디버그 수준을 지정한 수준으로 설정합니다. 기본 디버그 수준은 0이며 이 수준은 데이터 서비스가 디버깅 메시지를 기록하지 않도록 지정합니다.

`scds_syslog_debug()` 함수는 `scha_cluster_getlogfacility()` 함수에서 반환된 기능을 `LOG_DEBUG`의 우선 순위에 따라 사용합니다. `/etc/syslog.conf` 파일에서 이러한 디버그 메시지를 구성할 수 있습니다.

`scds_syslog()` 함수를 사용하면 일부 디버깅 메시지를 자원 유형의 일반 작업에 대한 정보 메시지(`LOG_INFO` 우선 순위)로 만들 수 있습니다. 8 장의 샘플 DSDL 응용 프로그램에서 `scds_syslog_debug()` 및 `scds_syslog()` 함수가 자유롭게 사용된 것을 볼 수 있습니다.

가용성이 높은 로컬 파일 시스템 만들기

HAStoragePlus 자원 유형을 사용하여 Sun Cluster 환경 내에서 로컬 파일 시스템의 가용성을 높일 수 있습니다. 로컬 파일 시스템 분할 영역은 전역 디스크 그룹에 위치해야 합니다. 또한 유사성 스위치오버를 사용할 수 있어야 하고 Sun Cluster 환경을 페일오버가 가능하도록 구성해야 합니다. 이 설정은 멀티 호스트 디스크에 직접 연결된 모든 호스트에서 멀티 호스트 디스크에 있는 모든 파일 시스템에 액세스할 수 있게 합니다. 입출력 작업이 많은 일부 데이터 서비스의 경우 고가용성 로컬 파일 시스템을 사용하는 것이 적극 권장됩니다. **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**의 "Enabling Highly Available Local File Systems"에는 HAStoragePlus 자원 유형 구성에 대한 정보가 포함되어 있습니다.

7장

자원 유형 정의

이 장에서는 자원 유형을 디자인 및 구현하는 데 있어 DSDL(Data Service Development Library)의 일반적인 용도를 살펴봅니다. 또한 자원 유형을 디자인하여 자원 구성을 검증하고 자원을 시작, 정지 및 모니터링하는 것에 초점을 맞추어 설명합니다. 마지막으로 DSDL을 사용하여 자원 유형 콜백 메소드를 구현하는 방법에 대해 설명합니다.

자세한 내용은 `rt_callbacks`(1HA) 설명서 페이지를 참조하십시오.

이러한 작업을 수행하려면 자원의 등록 정보 설정에 액세스해야 합니다. DSDL 유틸리티 `scds_initialize()`는 이러한 자원 등록 정보에 액세스하기 위한 일관된 방법을 제공합니다. 각 콜백 메소드의 시작 부분에서 호출되도록 설계된 이 유틸리티 함수는 클러스터 프레임워크에서 자원의 모든 등록 정보를 검색하여 `scds_get`*name*`()` 함수 계열에 사용할 수 있게 합니다.

이 장은 다음 내용으로 구성되어 있습니다.

자원 유형 등록(RTR) 파일

자원 유형 등록(RTR) 파일은 Sun Cluster에 대해 자원 유형의 세부 정보를 지정합니다. 이러한 세부 정보에는 다음 정보가 포함됩니다.

- 구현에 필요한 등록 정보
- 이러한 등록 정보의 데이터 유형과 기본값
- 자원 유형 구현을 위한 콜백 메소드의 파일 시스템 경로
- 시스템 정의 등록 정보의 다양한 설정

DSDL과 함께 제공되는 샘플 RTR 파일은 대부분의 자원 유형 구현에 충분합니다. 개발자는 자원 유형 이름 및 자원 유형 콜백 메소드의 경로 이름과 같은 몇 가지 기본 요소만 편집하면 됩니다. 자원 유형을 구현하기 위해 새 등록 정보가 필요한 경우 자원 유형 구현의 RTR 파일에서 이를 확장 등록 정보로 선언한 다음 `scds_get_ext_property()` 유틸리티를 사용하여 새 등록 정보에 액세스할 수 있습니다.

`Validate` 메소드

자원 유형 구현의 `Validate` 콜백 메소드의 목적은 제안된 자원 설정(자원의 제안된 등록 정보 설정에 지정된 대로)이 자원 유형에 허용되는지 확인하는 것입니다.

자원 유형 구현의 `Validate` 메소드는 다음 두 조건 중 하나에서 RGM에 의해 호출됩니다.

- 자원 유형의 새 자원을 만들 때
- 자원 또는 자원 그룹의 등록 정보를 업데이트할 때

이러한 두 시나리오는 자원의 `Validate` 메소드에 전달되는 `-c`(작성) 또는 `-u`(업데이트) 명령줄 옵션이 있는지에 따라 구별할 수 있습니다.

`Validate` 메소드는 자원 유형 등록 정보 `Init_nodes`의 값으로 정의되는 노드 집합의 각 노드에서 호출됩니다. `Init_nodes`가 `RG_PRIMARIES` 로 설정된 경우 `Validate`는 자원을 포함하는 자원 그룹을 호스팅할 수 있는(즉, 이러한 그룹의 기본이 될 수 있는) 각 노드에서 호출됩니다. `Init_nodes`가 `RT_INSTALLED_NODES`로 설정된 경우에는 자원 유 형 소프트웨어가 설치된 각 노드(일반적으로 클러스터의 모든 노드)에서 `Validate`가 호출됩니다.

`Init_nodes`의 기본값은 `RG_PRIMARIES`입니다(`rt_reg`(4) 설명서 페이지 참조). `Validate` 메소드가 호출되는 시점에 RGM은 아직 자원을 만들지 않았거나(작성 콜백의 경우) 업데이트 중인 등록 정보의 업데이트된 값을 아직 적용하지 않았습니다(업데이트 콜백의 경우).

주 – HAStoragePlus 자원 유형에서 관리하는 로컬 파일 시스템을 사용 중이면 scds_hasp_check() 함수를 사용하여 해당 자원 유형의 상태를 확인합니다. 자원에 대해 정의된 Resource_dependencies 또는 Resource_dependencies_weak 시스템 등록 정보를 사용하여 자원이 의존하는 모든 SUNW.HAStoragePlus 자원의 상태(온라인 또는 기타)에서 이 정보를 가져옵니다. scds_hasp_check () 함수에서 반환된 전체 상태 코드 목록은 scds_hasp_check(3HA) 설명서 페이지를 참조하십시오.

DSDL 함수 scds_initialize()는 다음과 같은 방법으로 이러한 상황을 처리합니다.

- 자원 작성의 경우 scds_initialize()는 명령줄에서 전달된 대로 제안된 자원 등록 정보의 구문을 분석합니다. 따라서 자원이 이미 시스템에서 생성된 것처럼 자원 등록 정보의 제안된 값을 사용할 수 있습니다.
- 자원 또는 자원 그룹 업데이트의 경우 클러스터 관리자가 업데이트 중인 등록 정보의 제안된 값은 명령줄에서 읽어오고 값이 업데이트되지 않는 나머지 등록 정보는 자원 관리 API를 사용하여 Sun Cluster에서 읽어옵니다. DSDL을 사용 중이면 이러한 작업에 신경 쓸 필요가 없습니다. 자원의 모든 등록 정보를 사용할 수 있는 것처럼 자원을 검증할 수 있습니다.

자원 등록 정보의 검증을 구현하는 함수를 svc_validate()라고 하고 이 함수가 scds_get_*name*() 계열의 함수를 사용하여 검증할 등록 정보를 찾는다고 가정합니다. 허용되는 자원 설정이 이 함수의 반환 코드 0으로 표시된다고 가정하면 자원 유형의 Validate 메소드는 다음 코드 단편과 같이 나타날 수 있습니다.

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

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

검증 함수는 또한 자원 검증이 실패한 이유를 기록해야 합니다. 이에 대한 세부 정보는 제외하고(8 장에서 Validation 함수에 대한 실제적인 처리 방법 설명) svc_validate() 함수를 다음과 같이 구현할 수 있습니다.

```
int
svc_validate(scds_handle_t handle)
{
   scha_str_array_t *confdirs;
   struct stat    statbuf;
   confdirs = scds_get_confdir_list(handle);
   if (stat(confdirs->str_array[0], &statbuf) == -1) {
```

```
   return (1);   /* Invalid resource property setting */
   }
   return (0);   /* Acceptable setting */
}
```

따라서 개발자는 `svc_validate()` 함수의 구현만 신경 쓰면 됩니다.

Start 메소드

자원 유형 구현의 `Start` 콜백 메소드는 자원을 시작하기 위해 선택된 클러스터 노드에서 RGM에 의해 호출됩니다. 명령줄에 자원 그룹 이름, 자원 이름 및 자원 유형 이름이 전달됩니다. `Start` 메소드는 클러스터 노드에서 데이터 서비스 자원을 시작하는 데 필요한 작업을 수행합니다. 일반적으로 여기에는 자원 등록 정보 검색, 응용 프로그램 특정 실행 파일 및/또는 구성 파일 찾기, 해당 명령줄 인자로 응용 프로그램 시작 등의 작업이 포함됩니다.

DSDL의 경우 자원 구성은 이미 `scds_initialize()` 유틸리티에 의해 검색됩니다. 응용 프로그램에 대한 시작 작업은 `svc_start()` 함수에 포함될 수 있습니다. 다른 함수 `svc_wait()`를 호출하여 응용 프로그램이 실제로 시작되었는지 확인할 수 있습니다. `Start` 메소드에 대한 단순화된 코드는 다음과 같이 나타납니다.

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

   if (scds_initialize(&handle, argc, argv)!= SCHA_ERR_NOERR) {
   return (1);   /* Initialization Error */
   }
   if (svc_validate(handle) != 0) {
   return (1);   /* Invalid settings */
   }
   if (svc_start(handle) != 0) {
   return (1);   /* Start failed */
   }
   return (svc_wait(handle));
}
```

이 시작 메소드 구현에서 `svc_validate()`를 호출하여 자원 구성을 검증합니다. 실패할 경우 자원 구성과 응용 프로그램 구성이 일치하지 않거나 해당 클러스터 노드에 시스템과 관련된 문제가 있는 것입니다. 예를 들어, 자원에 필요한 클러스터 파일 시스템을 현재 해당 클러스터 노드에서 사용하지 못할 수 있습니다. 이러한 경우 해당 클러스터 노드에서 자원을 시작하려는 시도는 아무 소용이 없으며 RGM으로 하여금 다른 노드에서 자원을 시작하도록 하는 것이 좋습니다.

그러나 위에서는 `svc_validate()`가 응용 프로그램에 절대적으로 필요한 클러스터 노드의 자원만 검사하도록 제한되어 있으며 그렇지 않을 경우 모든 클러스터 노드에서 자원이 시작되지 않아 `START_FAILED` 상태가 될 수 있다고 가정합니다. 이 상태에 대한 자세한 내용은 `scswitch`(1M) 설명서 페이지 및 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**를 참조하십시오.

`svc_start()` 함수는 노드에서 자원 시작에 성공한 경우 0을 반환해야 하며 문제가 발생한 경우에는 0이 아닌 값을 반환해야 합니다. 이 함수가 실패할 경우 RGM은 다른 클러스터 노드에서 해당 자원을 시작하려고 합니다.

DSDL을 가능한 많이 활용하기 위해 `svc_start()` 함수는 `scds_pmf_start()` 유틸리티를 호출하여 PMF(Process Management Facility)에서 응용 프로그램을 시작할 수 있습니다. 이 유틸리티는 또한 PMF의 실패 콜백 작업 기능을 사용하여 프로세스 실패를 감지합니다. 자세한 내용은 `pmfadm`(1M) 설명서 페이지에서 `-a` 작업 인자에 대한 설명을 참조하십시오.

Stop 메소드

자원 유형 구현의 `Stop` 콜백 메소드는 응용 프로그램을 중지하기 위해 클러스터 노드에서 RGM에 의해 호출됩니다. `Stop` 메소드의 콜백 의미는 다음을 요구합니다.

- `Start` 메소드가 노드에서 성공적으로 완료되지 않은 경우에도 RGM에 의해 `Stop` 메소드가 호출될 수 있으므로 `Stop` 메소드는 **멱등원**이어야 합니다. 따라서 `Stop` 메소드는 응용 프로그램이 클러스터 노드에서 실행 중이 아니며 이에 대해 수행할 작업이 없는 경우에도 성공(0을 반환하고 종료)해야 합니다.
- 자원 유형의 `Stop` 메소드가 클러스터 노드에서 실패(0이 아닌 값을 반환하고 종료)할 경우 중지하려는 자원은 `STOP_FAILED` 상태가 됩니다. 자원의 `Failover_mode` 설정에 따라 이 상태로 인해 RGM에서 클러스터 노드의 하드 재부트를 수행할 수 있습니다.

  따라서 이 메소드가 응용 프로그램을 실제로 중지하도록 `Stop` 메소드를 디자인해야 합니다. `SIGKILL`을 사용하여 응용 프로그램을 종료하는 데 실패할 경우 응용 프로그램을 강제로 중단해야 할 수도 있습니다.

  또한 프레임워크에서 `Stop_timeout` 등록 정보의 만료를 중지 실패로 간주하고 자원을 `STOP_FAILED` 상태로 설정하기 때문에 이 메소드가 적절한 시간에 응용 프로그램을 중지하도록 해야 합니다.

DSDL 유틸리티 `scds_pmf_stop()`은 `SIGTERM`을 통해 응용 프로그램을 유연하게 중지하려고 시도한 다음 프로세스에 `SIGKILL`을 전달하므로 대부분의 응용 프로그램을 중지할 수 있습니다. 이 함수는 응용 프로그램이 `scds_pmf_start()`를 통해 PMF에서 시작되었다고 가정합니다. 이 유틸리티에 대한 자세한 내용은 193 페이지 "PMF 함수"를 참조하십시오.

응용 프로그램을 중지하는 응용 프로그램 특정 함수를 `svc_stop()`이라고 가정할 경우 `Stop` 메소드를 다음과 같이 구현합니다.

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

앞의 `svc_stop()` 함수 구현에 `scds_pmf_stop()` 함수를 포함할지 여부는 여기서 다루지 않습니다. 응용 프로그램이 `Start` 메소드를 통해 PMF에서 시작되었는지에 따라 `scds_pmf_stop()` 함수의 포함 여부가 결정될 것입니다.

`svc_validate()` 메소드는 `Stop` 메소드 구현에 사용되지 않습니다. 이는 시스템에 문제가 있는 경우에도 `Stop` 메소드가 해당 노드에서 응용 프로그램의 중지를 시도해야 하기 때문입니다.

Monitor_start 메소드

RGM은 자원에 대한 오류 모니터를 시작하기 위해 `Monitor_start` 메소드를 호출합니다. 오류 모니터는 자원이 관리하는 응용 프로그램의 상태를 모니터합니다. 자원 유형 구현은 일반적으로 백그라운드에서 실행되는 별도의 데몬으로 오류 모니터를 구현합니다. `Monitor_start` 콜백 메소드는 이 데몬을 적절한 인자로 시작하는 데 사용됩니다.

모니터 데몬 자체에 오류가 발생할 수 있으므로(예를 들어, 모니터 데몬이 중지되어 응용 프로그램이 모니터되지 않을 수 있음) PMF를 사용하여 모니터 데몬을 시작해야 합니다. DSDL 유틸리티 `scds_pmf_start()`는 오류 모니터를 시작하기 위한 지원을 기본으로 제공합니다. 이 유틸리티는 모니터 데몬 프로그램의 자원 유형 콜백 메소드 구현 위치로 `RT_basedir`에 대한 상대 경로 이름을 사용하고 데몬의 무제한 재시작을 방지하기 위해 DSDL에서 관리하는 `Monitor_retry_interval` 및 `Monitor_retry_count` 확장 등록 정보를 사용합니다.

또한 모니터 데몬이 RGM에 의해 직접 호출되지 않더라도 모든 콜백 메소드에 정의된 명령줄 구문(즉, `-R` *resource* `-G` *resource-group* `-T` *resource-type*)이 모니터 데몬에 동일하게 적용됩니다. 마지막으로 이 유틸리티는 모니터 데몬 구현 자체에서 `scds_initialize()` 유틸리티를 사용하여 고유한 환경을 설정할 수 있게 합니다. 작업을 위한 노력의 대부분은 모니터 데몬 자체를 디자인하는 데 필요합니다.

Monitor_stop 메소드

RGM은 `Monitor_start` 메소드를 통해 시작된 오류 모니터 데몬을 중지하기 위해 `Monitor_stop` 메소드를 호출합니다. 이 콜백 메소드의 실패는 `Stop` 메소드의 실패와 동일한 방식으로 처리됩니다. 따라서 `Monitor_stop` 메소드는 멱등원이어야 하며 `Stop` 메소드와 같이 견고해야 합니다.

scds_pmf_start() 유틸리티를 사용하여 오류 모니터 데몬을 시작한 경우 scds_pmf_stop() 유틸리티를 사용하여 오류 모니터 데몬을 중지합니다.

Monitor_check 메소드

RGM은 클러스터 노드가 자원을 마스터할 수 있는지를 확인하기 위해 지정한 자원의 노드에서 자원에 대한 Monitor_check 콜백 메소드를 실행합니다. 즉, RGM은 이 메소드를 실행하여 자원에서 관리 중인 응용 프로그램을 노드에서 성공적으로 실행할 수 있는 지를 확인합니다.

이 경우 일반적으로 응용 프로그램에 필요한 모든 시스템 자원을 클러스터 노드에서 사용할 수 있는지도 확인해야 합니다. 114 페이지 "Validate 메소드"에 설명된 것처럼 개발자가 구현하는 svc_validate() 함수의 목적은 최소한 이러한 점을 확인하는 데 있습니다.

자원 유형 구현에 의해 관리되는 특정 응용 프로그램에 따라 Monitor_check 메소드를 작성하여 몇 가지 추가 작업을 수행할 수 있습니다. Monitor_check 메소드는 동시에 실행되는 다른 메소드와 충돌하지 않도록 구현해야 합니다. DSDL을 사용 중이면 Monitor_check 메소드에서 자원 등록 정보의 응용 프로그램 특정 검증을 구현하는 svc_validate() 함수를 호출해야 합니다.

Update 메소드

RGM은 클러스터 관리자에 의한 모든 변경 사항을 활성 자원의 구성에 적용하기 위해 자원 유형 구현의 Update 메소드를 호출합니다. Update 메소드는 자원이 현재 온라인 상태인 노드(있을 경우)에서만 호출됩니다.

RGM은 Validate 메소드를 실행하기 전에 자원 유형의 Update 메소드를 실행하므로 이제 막 변경된 자원 구성은 자원 유형 구현에 허용될 수 있습니다. Validate 메소드는 자원 또는 자원 그룹 등록 정보가 변경되기 전에 호출되며 제안된 변경 사항이 Validate 메소드에 의해 거부될 수 있습니다. 활성(온라인) 자원이 새 설정을 인식할 수 있도록 변경 사항이 적용된 후에 Update 메소드가 호출됩니다.

동적으로 업데이트할 수 있기를 원하는 등록 정보를 신중하게 결정하고 RTR 파일에서 TUNABLE = ANYTIME 설정으로 이러한 등록 정보에 표시해야 합니다. 일반적으로 오류 모니터 데몬이 사용하는 자원 유형 구현의 모든 등록 정보를 동적으로 업데이트할 수 있어야 한다는 것을 지정할 수 있습니다. 그러나 Update 메소드는 적어도 모니터 데몬을 재시작해야 합니다.

사용할 수 있는 등록 정보는 다음과 같습니다.

- Thorough_probe_interval
- Retry_count
- Retry_interval
- Monitor_retry_count
- Monitor_retry_interval
- Probe_timeout

이러한 등록 정보는 오류 모니터 데몬이 서비스 상태를 검사하는 방법, 상태를 검사하는 빈도, 오류를 추적하기 위해 사용하는 기록 간격, PMF에 의해 오류 모니터 데몬에 설정되는 재시작 임계값 등에 영향을 미칩니다. 이러한 등록 정보의 업데이트를 구현하기 위해 scds_pmf_restart () 유틸리티가 DSDL에서 제공됩니다.

자원 등록 정보를 동적으로 업데이트할 수 있어야 하지만 해당 등록 정보의 수정으로 인해 실행 중인 응용 프로그램이 영향을 받을 수 있는 경우 해당 등록 정보에 대한 업데이트가 실행 중인 임의의 응용 프로그램 인스턴스에 정확하게 적용되도록 적절한 작업을 구현해야 합니다. 현재로서는 DSDL을 사용하여 이런 방법으로 자원 등록 정보를 동적으로 업데이트할 수 없습니다. Validate와 마찬가지로 명령줄에서는 수정된 등록 정보를 Update로 전달할 수 없습니다.

Init, Fini 및 Boot 메소드에 대한 설명

이러한 메소드는 자원 관리 API 사양에 의해 정의되는 **일회성 작업** 메소드입니다. DSDL에 포함된 샘플 구현은 이러한 메소드의 사용을 보여주지 않습니다. 그러나 이러한 메소드가 필요한 경우 DSDL의 모든 기능을 이러한 메소드에서도 사용할 수 있습니다. 일반적으로 Init 및 Boot 메소드는 자원 유형 구현에서 **일회성 작업**을 구현한다는 점에서 동일합니다. Fini 메소드는 일반적으로 Init 또는 Boot 메소드의 작업을 **실행 취소**하는 작업을 수행합니다.

오류 모니터 데몬 디자인

DSDL을 사용한 자원 유형 구현에는 일반적으로 다음 작업을 담당하는 오류 모니터 데몬이 있습니다.

- 관리 중인 응용 프로그램의 상태를 정기적으로 모니터합니다. 모니터 데몬의 이 특정 작업은 대체로 응용 프로그램과 자원 유형별로 크게 다를 수 있습니다. DSDL에는 간단한 TCP 기반 서비스에 대해 상태 검사를 수행하는 몇 가지 유틸리티 기능이 기본으로 제공됩니다. 이러한 유틸리티를 사용하여 HTTP, NNTP, IMAP 및 POP3 같은 ASCII 기반 프로토콜을 사용하는 응용 프로그램을 구현할 수 있습니다.
- Retry_interval 및 Retry_count 자원 등록 정보를 사용하여 응용 프로그램에서 발생한 문제를 추적합니다. 응용 프로그램이 완전히 실패하면 오류 모니터는 PMF 작업 스크립트가 서비스를 재시작해야 하는지 또는 응용 프로그램 실패가 신속하게

누적되어 페일오버를 수행해야 하는지를 확인해야 합니다. DSDL 유틸리티 `scds_fm_action()` 및 `scds_fm_sleep()`는 이 기법의 구현을 지원하기 위한 목적으로 사용됩니다.

- 일반적으로 응용 프로그램을 재시작하거나 포함하는 자원 그룹의 페일오버를 시도하여 적절한 작업을 수행합니다. DSDL 유틸리티 `scds_fm_action()`은 이 알고리즘을 구현하며 이러한 목적을 위해 경과한 `Retry_interval` 초에서 현재까지 누적된 검사 실패를 계산합니다.
- 클러스터 관리 GUI 뿐만 아니라 `scstat` 명령에서 응용 프로그램 상태를 사용할 수 있도록 자원 상태를 업데이트합니다.

DSDL 유틸리티는 오류 모니터 데몬의 주 루프가 이 장의 끝에 나오는 의사 코드로 표시될 수 있도록 설계되었습니다.

DSDL을 사용하여 오류 모니터를 구현하는 경우 다음을 명심하십시오.

- PMF를 통한 프로세스 중지 알림이 동기식이므로 `scds_fm_sleep()`가 응용 프로그램 프로세스 중지를 신속하게 감지합니다. 따라서 오류 감지 시간이 대폭 줄어들기 때문에 서비스의 가용성이 향상됩니다. 그렇지 않으면 오류 모니터가 서비스 상태를 검사하고 응용 프로그램 프로세스 중지를 확인하기 위해 너무 자주 실행될 수 있습니다.
- RGM이 `scha_control` API를 통해 서비스를 페일오버하려는 시도를 거부할 경우 `scds_fm_action()`은 현재 실패 기록을 **재설정**(삭제)합니다. 그 이유는 현재 실패 기록이 이미 `Retry_count`를 초과한 상태이기 때문입니다. 모니터 데몬이 다음 반복 시에 작동되어 데몬의 상태 검사를 성공적으로 완료할 수 없으면 이 모니터 데몬은 `scha_control()` 함수를 다시 호출합니다. 마지막 반복에서 거부를 일으켰던 상황이 여전히 유효하기 때문에 이러한 호출 시도가 다시 거부될 수 있습니다. 기록을 재설정하면 오류 모니터는 다음 반복 시에 최소한 현재 상황을 로컬로(예를 들어, 응용 프로그램 재시작을 통해) 수정하려고 시도합니다.
- 상황 자체가 수정되지 않을 경우 곧바로 `scha_control()`을 시도하는 것이 일반적이기 때문에 `scds_fm_action()`은 재시작 실패의 경우 응용 프로그램 실패 기록을 재설정하지 **않습니다**.
- `scds_fm_action()` 유틸리티는 실패 기록에 따라 자원 상태를 `SCHA_RSSTATUS_OK`, `SCHA_RSSTATUS_DEGRADED` 또는 `SCHA_RSSTATUS_FAULTED`로 업데이트합니다. 결과적으로 이 상태를 클러스터 시스템 관리에 사용할 수 있습니다.

대부분의 경우 응용 프로그램 특정 상태 검사 작업은 별도의 독립 실행형 유틸리티(예: `svc_probe()`)에서 구현되고 다음과 같은 일반적인 주 루프와 통합될 수 있습니다.

```
for (;;) {
   /* sleep for a duration of thorough_probe_interval between
   *  successive probes.
   */
   (void) scds_fm_sleep(scds_handle,
   scds_get_rs_thorough_probe_interval(scds_handle));
   /* Now probe all ipaddress we use. Loop over
   * 1. All net resources we use.
   * 2. All ipaddresses in a given resource.
   * For each of the ipaddress that is probed,
```

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

8장

샘플 DSDL 자원 유형 구현

이 장에서는 DSDL(Data Service Development Library)을 사용하여 구현되는 샘플 자원 유형 `SUNW.xfnts`에 대해 설명합니다. 이 데이터 서비스는 C로 작성되며 기본 응용 프로그램은 TCP/IP 기반 서비스인 X Font Server입니다. 부록 C에는 `SUNW.xfnts` 자원 유형의 각 메소드에 대한 전체 코드가 나와 있습니다.

이 장은 다음 내용으로 구성되어 있습니다.

X Font Server

X Font Server는 클라이언트에 글꼴 파일을 제공하는 TCP/IP 기반 서비스입니다. 클라이언트는 글꼴 집합을 요청하기 위해 서버에 연결하며 서버는 디스크에서 글꼴 파일을 읽어 클라이언트에 제공합니다. X Font Server 데몬은 `/usr/openwin/bin/xfs`에 있는 서버 이진으로 구성됩니다. 데몬은 대개 `inetd`에서 시작됩니다. 그러나 현재 샘플의 경우 `/etc/inetd.conf` 파일에서 해당 항목이 `fsadmin -d` 명령 등에 의해 비활성화되어 데몬이 Sun Cluster 소프트웨어의 제어만 받는 것으로 가정합니다.

X Font Server 구성 파일

기본적으로 X Font Server는 `/usr/openwin/lib/X11/fontserver.cfg` 파일에서 구성 정보를 읽습니다. 이 파일의 카탈로그 항목에는 데몬에서 제공할 수 있는 글꼴 디렉토리 목록이 포함되어 있습니다. 클러스터 관리자는 클러스터 파일 시스템에서 글꼴 디렉토리를 찾을 수 있습니다. 이 위치는 시스템에서 글꼴 데이터베이스의 단일 복사본을 유지 관리하여 Sun Cluster에서의 X Font Server 사용을 최적화합니다. 클러스터 관리자가 이 위치를 변경하려면 글꼴 데이터베이스의 새 경로에 따라 `fontserver.cfg`를 편집해야 합니다.

클러스터 관리자는 또한 구성이 용이하도록 구성 파일 자체를 클러스터 파일 시스템에 저장할 수 있습니다. `xfs` 데몬은 이 파일의 기본 제공 위치를 무시하는 명령줄 인자를 제공합니다. `SUNW.xfnts` 자원 유형은 다음 명령을 사용하여 Sun Cluster 소프트웨어의 제어 하에서 데몬을 시작합니다.

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

`SUNW.xfnts` 자원 유형 구현에서 `Confdir_list` 등록 정보를 사용하여 `fontserver.cfg` 구성 파일의 위치를 관리할 수 있습니다.

TCP 포트 번호

`xfs` 서버 데몬이 수신하는 TCP 포트 번호는 일반적으로 "fs" 포트(대개 `/etc/services` 파일에서 `7100`으로 정의됨)입니다. 그러나 클러스터 관리자는 `xfs` 명령에 `-port` 옵션을 사용하여 기본 설정을 무시할 수 있습니다.

`SUNW.xfnts` 자원 유형의 `Port_list` 등록 정보를 사용하여 기본값을 설정하고 클러스터 관리자가 `xfs` 명령에 `-port` 옵션을 사용 가능하도록 지원할 수 있습니다. 이 등록 정보의 기본값은 RTR 파일에서 `7100/tcp`로 정의합니다. `SUNW.xfnts` `Start` 메소드에서 `Port_list`를 `xfs` 명령줄의 `-port` 옵션에 전달합니다. 결과적으로 이 자원 유형의 사용자는 포트 번호가 기본적으로 `7100/tcp`이므로 이를 지정할 필요가 없습니다. 클러스터 관리자는 자원 유형을 구성할 때 `Port_list` 등록 정보에 다른 값을 지정할 수 있습니다.

SUNW.xfnts RTR 파일

이 절에서는 `SUNW.xfnts` RTR 파일의 여러 주요 등록 정보에 대해 설명합니다. 이 파일의 각 등록 정보에 대한 용도는 이 절에서 설명되지 않습니다. 이러한 설명은 34 페이지 "자원 및 자원 유형 등록 정보 설정"을 참조하십시오.

`Confdir_list` 확장 등록 정보는 다음과 같이 구성 디렉토리 또는 디렉토리 목록을 식별합니다.

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

Confdir_list 등록 정보는 기본값을 지정하지 않습니다. 클러스터 관리자가 자원을 만들 때 디렉토리를 지정해야 합니다. 조정 기능이 AT_CREATION으로 제한되므로 이 값은 나중에 변경할 수 없습니다.

Port_list 등록 정보는 다음과 같이 서버 데몬이 수신하는 포트를 식별합니다.

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

이 등록 정보에서 기본값을 선언하므로 클러스터 관리자는 자원을 만들 때 새 값을 지정할지 아니면 기본값을 그대로 사용할지 선택할 수 있습니다. 조정 기능이 AT_CREATION으로 제한되므로 이 값은 나중에 변경할 수 없습니다.

함수 및 콜백 메소드의 이름 지정 규칙

다음 규칙을 알면 다양한 샘플 코드를 식별할 수 있습니다.

- RMAPI 함수는 scha_로 시작합니다.
- DSDL 함수는 scds_로 시작합니다.
- 콜백 메소드는 xfnts_로 시작합니다.
- 사용자가 작성한 함수는 svc_로 시작합니다.

scds_initialize() 함수

DSDL은 각 콜백 메소드가 메소드의 시작 부분에서 scds_initialize() 함수를 호출할 것을 요구합니다. 이 함수는 다음 작업을 수행합니다.

- 해당 프레임워크에서 데이터 서비스 메소드로 전달하는 명령줄 인자(argc 및 argv)를 검사하여 처리합니다. 이 메소드는 추가 명령줄 인자를 처리할 필요가 없습니다.
- DSDL의 다른 함수에 사용되는 내부 데이터 구조를 설정합니다.

- 로깅 환경을 초기화합니다.
- 오류 모니터 검사 설정을 검증합니다.

`scds_close()` 함수를 사용하여 `scds_initialize()`에서 할당한 자원을 해제합니다.

xfnts_start 메소드

RGM은 데이터 서비스 자원을 포함하는 자원 그룹이 클러스터 노드에서 온라인 상태이거나 자원이 사용 가능할 때 해당 노드에서 `Start` 메소드를 실행합니다. `SUNW.xfnts` 샘플 자원 유형에서 `xfnts_start` 메소드는 해당 노드에서 `xfs` 데몬을 활성화합니다.

`xfnts_start` 메소드는 `scds_pmf_start()`를 호출하여 PMF에서 데몬을 시작합니다. PMF는 자동 실패 알림 및 재시작 기능과 오류 모니터와의 통합을 제공합니다.

주 – `xfnts_start`에서 첫 번째 호출은 몇 가지 필수 **관리 작업** 기능을 수행하는 `scds_initialize()`입니다. 자세한 내용은 125 페이지 "scds_initialize() 함수" 및 `scds_initialize`(3HA) 설명서 페이지를 참조하십시오.

X Font Server 시작 전 서비스 검증

`xfnts_start` 메소드는 X Font Server를 시작하기 전에 다음과 같이 `svc_validate()`를 호출하여 `xfs` 데몬을 지원하기 위한 적절한 구성이 있는지 확인합니다.

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

자세한 내용은 139 페이지 "xfnts_validate 메소드"를 참조하십시오.

svc_start()를 사용하여 서비스 시작

`xfnts_start` 메소드는 `xfnts.c` 파일에 정의된 `svc_start()` 메소드를 호출하여 `xfs` 데몬을 시작합니다. 이 절에서는 `svc_start()`에 대해 설명합니다.

`xfs` 데몬을 시작하는 명령은 다음과 같습니다.

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

Confdir_list 확장 등록 정보는 *config-directory*를 식별하고 Port_list 시스템 등록 정보는 *port-number*를 식별합니다. 클러스터 관리자는 데이터 서비스를 구성할 때 이러한 등록 정보에 특정 값을 제공합니다.

xfnts_start 메소드는 다음과 같이 이러한 등록 정보를 문자열 배열로 선언하며 scds_get_ext_confdir_list() 및 scds_get_port_list() 함수를 사용하여 클러스터 관리자가 설정한 값을 얻습니다. 이러한 함수에 대한 자세한 내용은 scds_property_functions(3HA) 설명서 페이지를 참조하십시오.

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

   /* get the configuration directory from the confdir_list property */
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

   /* obtain the port to be used by XFS from the Port_list property */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }
```

confdirs 변수가 배열의 첫 번째 요소(0)를 가리킨다는 것에 주의합니다.

xfnts_start 메소드는 다음과 같이 sprintf()를 사용하여 xfs에 대한 명령줄을 생성합니다.

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

데몬이 생성한 메시지를 삭제하기 위해 출력이 /dev/null로 리디렉션된다는 것에 주의합니다.

xfnts_start 메소드는 다음과 같이 xfs 명령줄을 scds_pmf_start()로 전달하여 PMF의 제어 하에서 데이터 서비스를 시작합니다.

```
scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

   if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
```

```
}
```

scds_pmf_start() 호출에 대해 다음 사항에 주의합니다.

- SCDS_PMF_TYPE_SVC 인자는 시작할 프로그램을 데이터 서비스 응용 프로그램으로 식별합니다. 이 메소드는 오류 모니터나 다른 유형의 응용 프로그램을 시작할 수도 있습니다.
- SCDS_PMF_SINGLE_INSTANCE 인자는 이를 단일 인스턴스 자원으로 식별합니다.
- cmd 인자는 이전에 생성된 명령줄입니다.
- 마지막 인자인 -1은 자식 모니터 수준을 지정합니다. -1 값은 PMF가 원래 프로세스뿐만 아니라 모든 자식을 모니터하도록 지정합니다.

반환 전에 svc_pmf_start()는 다음과 같이 portlist 구조에 할당된 메모리를 해제합니다.

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

svc_start()에서 반환

svc_start()가 성공적으로 반환된 경우에도 기본 응용 프로그램을 시작하는 데 실패했을 수 있습니다. 따라서 svc_start()에서 성공 메시지를 반환하려면 먼저 해당 응용 프로그램이 실행 중인지 검사해야 합니다. 또한 검사 시 응용 프로그램을 시작하는 데 시간이 걸리기 때문에 즉시 사용하지 못할 수도 있다는 것을 고려해야 합니다. svc_start() 메소드는 다음과 같이 xfnts.c 파일에 정의된 svc_wait()를 호출하여 응용 프로그램이 실행 중인지 확인합니다.

```
/* Wait for the service to start up fully */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Calling svc_wait to verify that service has started.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "Successfully started the service.");
   } else {
      scds_syslog(LOG_ERR, "Failed to start the service.");
   }
```

svc_wait() 함수는 다음과 같이 scds_get_netaddr_list()를 호출하여 응용 프로그램을 검사하는 데 필요한 네트워크 주소 자원을 얻습니다.

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

```
   }

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

그런 다음 `svc_wait()` 함수는 다음과 같이 `Start_timeout` 및 `Stop_timeout` 값을 얻습니다.

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

서버를 시작하는 데 걸리는 시간을 나타내기 위해 `svc_wait()`는 `scds_svc_wait()`를 호출하고 `Start_timeout` 값의 3%에 해당하는 시간 초과값을 전달합니다. 그런 다음 `svc_wait()` 함수는 `svc_probe()` 함수를 호출하여 응용 프로그램이 시작되었는지 확인합니다. `svc_probe()` 메소드는 지정된 포트에서 서버에 대한 간단한 소켓 연결을 설정합니다. 포트에 연결하는 데 실패할 경우 `svc_probe()`는 완전한 실패를 나타내는 값 100을 반환합니다. 연결이 설정되었지만 포트에 대한 연결을 끊는 데 실패한 경우 `svc_probe()`는 값 50을 반환합니다.

`svc_probe()`의 실패 또는 부분 실패 시 `svc_wait()`는 시간 초과값이 5인 `scds_svc_wait()`를 호출합니다. `scds_svc_wait()` 메소드는 검사 빈도를 5초 간격으로 제한합니다. 또한 이 메소드는 서비스 시작을 시도한 횟수를 계산합니다. 시도 횟수가 자원의 `Retry_interval` 등록 정보에서 지정한 기간 내에 자원의 `Retry_count` 등록 정보 값을 초과하면 `scds_svc_wait()` 함수가 실패를 반환합니다. 이러한 경우 `svc_start()` 함수도 실패를 반환합니다.

```
#define    SVC_CONNECT_TIMEOUT_PCT    95
#define    SVC_WAIT_PCT       3
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

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

       /* Call scds_svc_wait() so that if service fails too
```

```
    if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
        != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR, "Service failed to start.");
        return (1);
    }

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

주 – 종료하기 전에 `xfnts_start` 메소드는 `scds_close()`를 호출하여 `scds_initialize()`에 의해 할당된 자원을 재생 이용합니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 `scds_close`(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_stop 메소드

`xfnts_start` 메소드가 `scds_pmf_start()`를 사용하여 PMF에서 서비스를 시작하므로 `xfnts_stop`은 `scds_pmf_stop()`을 사용하여 서비스를 중지합니다.

주 – `xfnts_stop`에서 첫 번째 호출은 몇 가지 필수 **관리 작업** 기능을 수행하는 `scds_initialize()`입니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 `scds_initialize`(3HA) 설명서 페이지를 참조하십시오.

`xfnts_stop` 메소드는 다음과 같이 `xfnts.c` 파일에 정의된 `svc_stop()` 메소드를 호출합니다.

```
scds_syslog(LOG_ERR, "Issuing a stop request.");
    err = scds_pmf_stop(scds_handle,
        SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
        scds_get_rs_stop_timeout(scds_handle));

    if (err != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "Failed to stop HA-XFS.");
        return (1);
    }

    scds_syslog(LOG_INFO,
        "Successfully stopped HA-XFS.");
    return (SCHA_ERR_NOERR); /* Successfully stopped */
```

`svc_stop()`의 `scds_pmf_stop()` 함수 호출에 대해 다음 사항에 주의합니다.

- SCDS_PMF_TYPE_SVC 인자는 중지할 프로그램을 데이터 서비스 응용 프로그램으로 식별합니다. 이 메소드는 오류 모니터나 다른 유형의 응용 프로그램을 중지할 수도 있습니다.
- SCDS_PMF_SINGLE_INSTANCE 인자는 신호를 식별합니다.
- SIGTERM 인자는 자원 인스턴스를 중지하는 데 사용할 신호를 식별합니다. 이 신호가 인스턴스를 중지하는 데 실패할 경우 scds_pmf_stop() 은 인스턴스를 중지하기 위해 SIGKILL을 보내며 이 신호도 실패할 경우 시간 초과 오류를 반환합니다. 자세한 내용은 scds_pmf_stop(3HA) 설명서 페이지를 참조하십시오.
- 시간 초과값은 자원의 Stop_timeout 등록 정보 값입니다.

주 – 종료하기 전에 xfnts_stop 메소드는 scds_close() 를 호출하여 scds_initialize() 에 의해 할당된 자원을 재생 이용합니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 scds_close(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_monitor_start 메소드

RGM은 노드에서 Monitor_start 메소드를 호출하여 해당 노드에서 자원이 시작된 후 오류 모니터를 시작합니다. xfnts_monitor_start 메소드는 scds_pmf_start() 를 사용하여 PMF에서 모니터 데몬을 시작합니다.

주 – xfnts_monitor_start에서 첫 번째 호출은 몇 가지 필수 **관리 작업** 기능을 수행하는 scds_initialize() 입니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 scds_initialize(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_monitor_start 메소드는 다음과 같이 xfnts.c 파일에 정의된 mon_start 메소드를 호출합니다.

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling Monitor_start method for resource <%s>.",
      scds_get_resource_name(scds_handle));

    /* Call scds_pmf_start and pass the name of the probe. */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
```

```
        return (1);
    }

    scds_syslog(LOG_INFO,
        "Started the fault monitor.");

    return (SCHA_ERR_NOERR); /* Successfully started Monitor */
}
```

svc_mon_start() 의 scds_pmf_start() 함수 호출에 대해 다음 사항에 주의합니다.

- SCDS_PMF_TYPE_MON 인자는 시작할 프로그램을 오류 모니터로 식별합니다. 이 메소드는 데이터 서비스나 다른 유형의 응용 프로그램을 시작할 수도 있습니다.
- SCDS_PMF_SINGLE_INSTANCE 인자는 이를 단일 인스턴스 자원으로 식별합니다.
- xfnts_probe 인자는 시작할 모니터 데몬을 식별합니다. 모니터 데몬은 다른 콜백 프로그램과 동일한 디렉토리에 있는 것으로 가정됩니다.
- 마지막 인자인 0은 자식 모니터 수준을 지정합니다. 이 경우 PMF는 모니터 데몬만 모니터합니다.

주 – 종료하기 전에 xfnts_monitor_start 메소드는 scds_close() 를 호출하여 scds_initialize() 에 의해 할당된 자원을 재생 이용합니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 scds_close(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_monitor_stop 메소드

xfnts_monitor_start 메소드가 scds_pmf_start() 를 사용하여 PMF에서 모니터 데몬을 시작하므로 xfnts_monitor_stop은 scds_pmf_stop() 을 사용하여 모니터 데몬을 중지합니다.

주 – xfnts_monitor_stop에서 첫 번째 호출은 몇 가지 필수 **관리 작업** 기능을 수행하는 scds_initialize() 입니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 scds_initialize(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_monitor_stop 메소드는 다음과 같이 xfnts.c 파일에 정의된 mon_stop() 메소드를 호출합니다.

```
scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");
```

```
   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully stopped monitor */
}
```

svc_mon_stop()의 scds_pmf_stop() 함수 호출에 대해 다음 사항에 주의합니다.

- SCDS_PMF_TYPE_MON 인자는 중지할 프로그램을 오류 모니터로 식별합니다. 이 메소드는 데이터 서비스나 다른 유형의 응용 프로그램을 중지할 수도 있습니다.
- SCDS_PMF_SINGLE_INSTANCE 인자는 이를 단일 인스턴스 자원으로 식별합니다.
- SIGKILL 인자는 자원 인스턴스를 중지하는 데 사용할 신호를 식별합니다. 이 신호가 인스턴스를 중지하는 데 실패할 경우 scds_pmf_stop()은 시간 초과 오류를 반환합니다. 자세한 내용은 scds_pmf_stop(3HA) 설명서 페이지를 참조하십시오.
- 시간 초과값은 자원의 Monitor_stop_timeout 등록 정보 값입니다.

주 – 종료하기 전에 xfnts_monitor_stop 메소드는 scds_close()를 호출하여 scds_initialize()에 의해 할당된 자원을 재생 이용합니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 scds_close(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_monitor_check 메소드

RGM은 오류 모니터가 자원이 포함된 자원 그룹을 다른 노드로 페일오버할 때마다 Monitor_check 메소드를 호출합니다. xfnts_monitor_check 메소드는 svc_validate() 메소드를 호출하여 xfs 데몬을 지원하기 위한 적절한 구성이 있는지 확인합니다. 자세한 내용은 139 페이지 “xfnts_validate 메소드”를 참조하십시오. xfnts_monitor_check의 코드는 다음과 같습니다.

```
/* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
```

```
        return (1);
    }

    rc =  svc_validate(scds_handle);
    scds_syslog_debug(DBG_LEVEL_HIGH,
        "monitor_check method "
        "was called and returned <%d>.", rc);

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

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

SUNW.xfnts 오류 모니터

RGM은 PROBE 메소드를 직접 호출하지 않고 Monitor_start 메소드를 호출하여 노드에서 자원이 시작된 후 모니터를 시작합니다. xfnts_monitor_start 메소드는 PMF의 제어 하에서 오류 모니터를 시작합니다. xfnts_monitor_stop 메소드는 오류 모니터를 중지합니다.

SUNW.xfnts 오류 모니터는 다음 작업을 수행합니다.

- xfs와 같은 간단한 TCP 기반 서비스를 검사하도록 특수하게 설계된 유틸리티를 사용하여 xfs 서버 데몬의 상태를 정기적으로 모니터합니다.
- Retry_count 및 Retry_interval 등록 정보를 사용하여 시간 창 내에서 응용 프로그램에 발생하는 문제를 추적하고 완전한 응용 프로그램 실패가 발생할 경우 데이터 서비스를 다시 시작할지 아니면 페일오버할지 결정합니다. scds_fm_action() 및 scds_fm_sleep() 함수는 이 추적 및 결정 기법을 기본적으로 지원합니다.
- scds_fm_action()을 사용하여 페일오버 또는 재시작 결정을 구현합니다.
- 자원 상태를 업데이트하여 관리 도구 및 그래픽 사용자 인터페이스(GUI)에서 사용할 수 있게 만듭니다.

xfonts_probe 주 루프

xfonts_probe 메소드는 루프를 구현합니다. 루프를 구현하기 전에 xfonts_probe는 다음 작업을 수행합니다.

- 다음과 같이 xfnts 자원의 네트워크 주소 자원을 검사합니다.

```
/* Get the ip addresses available for this resource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
```

```
            "No network address resource in resource group.");
        scds_close(&scds_handle);
        return (1);
    }

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

- scds_fm_sleep()를 호출하고 Thorough_probe_interval 값을 시간 초과값으로 전달합니다. 다음과 같이 검사 작업 사이에 Thorough_probe_interval 값 동안 검사가 일시 정지됩니다.

```
timeout = scds_get_ext_probe_timeout(scds_handle);

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

xfnts_probe 메소드는 다음과 같이 루프를 구현합니다.

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

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

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

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

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

svc_probe() 함수는 검사 논리를 구현합니다. svc_probe() 반환값이 scds_fm_action()으로 전달되고, 여기서 응용 프로그램을 다시 시작할지, 자원 그룹을 페일오버할지, 아니면 아무 작업도 수행하지 않을지 결정합니다.

svc_probe() 함수

svc_probe() 함수는 scds_fm_tcp_connect()를 호출하여 지정된 포트에 대한 간단한 소켓 연결을 설정합니다. 연결에 실패할 경우 svc_probe()는 완전한 실패를 나타내는 값 100을 반환합니다. 연결에 성공했지만 연결을 끊는 데 실패한 경우 svc_probe()는 부분적 실패를 나타내는 값 50을 반환합니다. 연결과 연결 끊기에 모두 성공한 경우 svc_probe()는 성공을 나타내는 값 0을 반환합니다.

svc_probe()의 코드는 다음과 같습니다.

```
int svc_probe(scds_handle_t scds_handle,
char *hostname, int port, int timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
   char   testcmd[2048];
   int    time_used, time_remaining;
   time_t      connect_timeout;


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

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

   /*
```

```
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "Failed to connect to port <%d> of resource <%s>.",
       port, scds_get_resource_name(scds_handle));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

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

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

time_used = (int)(t2 - t1);

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

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
       "%d seconds during connect operation and exceeded the "
       "timeout by %d seconds. Attempting disconnect with timeout"
       " %d ",
       connect_timeout,
       abs(time_used),
       SVC_DISCONNECT_TIMEOUT_SECONDS);

   time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
}

/*
 * Return partial failure in case of disconnection failure.
 * Reason: The connect call is successful, which means
 * the application is alive. A disconnection failure
```

```
         * could happen due to a hung application or heavy load.
         * If it is the later case, don't declare the application
         * as dead by returning complete failure. Instead, declare
         * it as partial failure. If this situation persists, the
         * disconnect call will fail again and the application will be
         * restarted.
         */
        rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
        if (rc != SCHA_ERR_NOERR) {
           scds_syslog(LOG_ERR,
               "Failed to disconnect to port %d of resource %s.",
               port, scds_get_resource_name(scds_handle));
           /* this is a partial failure */
           return (SCDS_PROBE_COMPLETE_FAILURE/2);
        }

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

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

        /*
         * The connection and disconnection to port is successful,
         * Run the fsinfo command to perform a full check of
         * server health.
         * Redirect stdout, otherwise the output from fsinfo
         * ends up on the console.
         */
        (void) sprintf(testcmd,
            "/usr/openwin/bin/fsinfo -server %s:%d > /dev/null",
            hostname, port);
        scds_syslog_debug(DBG_LEVEL_HIGH,
            "Checking the server status with %s.", testcmd);
        if (scds_timerun(scds_handle, testcmd, time_remaining,
           SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

           scds_syslog(LOG_ERR,
              "Failed to check server status with command <%s>",
              testcmd);
           return (SCDS_PROBE_COMPLETE_FAILURE/2);
        }
        return (0);
     }
```

작업이 완료되면 svc_probe()는 성공(0), 부분적 실패(50) 또는 완전한 실패(100)를 나타내는 값을 반환합니다. xfnts_probe 메소드는 이 값을 scds_fm_action()에 전달합니다.

오류 모니터 작업 결정

xfnts_probe 메소드는 scds_fm_action()을 호출하여 수행할 작업을 결정합니다. scds_fm_action()의 논리는 다음과 같습니다.

- Retry_interval 등록 정보 값 내에서 누적 실패 기록을 유지 관리합니다.
- 누적 실패가 100(완전한 실패)에 도달하면 데이터 서비스를 재시작합니다. Retry_interval을 초과하면 기록을 재설정합니다.
- 다시 시작 횟수가 Retry_interval에 지정된 시간 내에서 Retry_count 등록 정보 값을 초과할 경우 데이터 서비스를 페일오버합니다.

예를 들어, 검사가 xfs 서버에 연결을 설정하지만 연결을 끊는 데 실패한다고 가정합니다. 이는 서버가 실행 중이지만 정지되거나 일시적으로 로드 상태가 될 수 있음을 나타냅니다. 연결을 끊는 데 실패하면 scds_fm_action()으로 부분적 실패 값(50)을 보냅니다. 이 값은 데이터 서비스를 재시작하기 위한 임계값보다 작지만 실패 기록에서 유지 관리됩니다.

다음 검사 중에 서버가 연결 끊기에 다시 실패하면 scds_fm_action()에서 유지 관리하는 실패 기록에 값 50이 추가됩니다. 이제 누적 실패 값이 100이므로 scds_fm_action()이 데이터 서비스를 재시작합니다.

xfnts_validate 메소드

자원을 만들 때 및 클러스터 관리자가 자원이나 자원을 포함한 그룹의 등록 정보를 업데이트할 때 RGM은 Validate 메소드를 호출합니다. RGM은 만들기나 업데이트가 적용되기 전에 Validate를 호출하며 임의의 노드에서 이 메소드가 실패 종료 코드를 반환할 경우 만들기 또는 업데이트가 취소됩니다.

RGM은 클러스터 관리자가 자원 또는 자원 그룹 등록 정보를 변경하거나 모니터가 Status 및 Status_msg 자원 등록 정보를 설정하는 경우에만 Validate를 호출합니다. RGM에서 등록 정보를 설정하는 경우에는 Validate를 호출하지 않습니다.

주 – Monitor_check 메소드는 PROBE 메소드가 데이터 서비스를 새 노드로 페일오버할 때마다 Validate 메소드를 명시적으로 호출합니다.

RGM은 업데이트 중인 등록 정보와 값을 포함하여 다른 메소드로 전달되는 항목에 대한 추가 인자와 함께 Validate를 호출합니다. xfnts_validate의 시작 부분에 있는 scds_initialize() 호출은 RGM이 xfnts_validate로 전달하는 모든 인자를 구문 분석하고 scds_handle 인자에 정보를 저장합니다. xfnts_validate가 호출하는 서브루틴은 이 정보를 사용합니다.

xfnts_validate 메소드는 다음 조건을 확인하는 svc_validate()를 호출합니다.

- Confdir_list 등록 정보가 자원에 대해 설정되었으며 단일 디렉토리를 정의합니다.

```
scha_str_array_t *confdirs;
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

- Confdir_list로 지정된 디렉토리에 fontserver.cfg 파일이 들어 있습니다.

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

   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suppress lint error because errno.h prototype
       * is missing void arg
       */
      scds_syslog(LOG_ERR,
          "Failed to access file <%s> : <%s>",
          xfnts_conf, strerror(errno));   /*lint !e746 */
      return (1);
   }
```

- 클러스터 노드에서 서버 데몬 이진을 액세스할 수 있습니다.

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
      scds_syslog(LOG_ERR,
          "Cannot access XFS binary : <%s> ", strerror(errno));
      return (1);
   }
```

- Port_list 등록 정보가 단일 포트를 지정합니다.

```
scds_port_list_t   *portlist;
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list: %s.",
         scds_error_string(err));
      return (1); /* Validation Failure */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
```

```
        scds_syslog(LOG_ERR,
            "Property Port_list must have only one value.");
        scds_free_port_list(portlist);
        return (1); /* Validation Failure */
    }
#endif
```

- 데이터 서비스를 포함하는 자원 그룹에 최소한 하나 이상의 네트워크 주소 자원도 포함되어 있습니다.

```
scds_net_resource_list_t *snrlp;
    if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
        != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "No network address resource in resource group: %s.",
           scds_error_string(err));
        return (1); /* Validation Failure */
    }

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

반환되기 전에 `svc_validate()`는 할당된 모든 자원을 해제합니다.

```
finished:
    scds_free_net_list(snrlp);
    scds_free_port_list(portlist);

    return (rc); /* return result of validation */
```

주 – 종료하기 전에 `xfnts_validate` 메소드는 `scds_close()`를 호출하여 `scds_initialize()`에 의해 할당된 자원을 재생 이용합니다. 자세한 내용은 125 페이지 “scds_initialize() 함수” 및 `scds_close`(3HA) 설명서 페이지를 참조하십시오.

---

xfnts_update 메소드

RGM은 `Update` 메소드를 호출하여 실행 중인 자원에 해당 등록 정보가 변경된 것을 알립니다. `xfnts` 데이터 서비스에 대한 변경할 수 있는 등록 정보만 오류 모니터와 관련됩니다. 따라서 등록 정보가 업데이트될 때마다 `xfnts_update` 메소드는 `scds_pmf_restart_fm()`을 호출하여 오류 모니터를 재시작합니다.

```
  /* check if the Fault monitor is already running and if so stop
   * and restart it. The second parameter to scds_pmf_restart_fm()
```

```
* uniquely identifies the instance of the fault monitor that needs
* to be restarted.
*/

scds_syslog(LOG_INFO, "Restarting the fault monitor.");
result = scds_pmf_restart_fm(scds_handle, 0);
if (result != SCHA_ERR_NOERR) {
   scds_syslog(LOG_ERR,
       "Failed to restart fault monitor.");
   /* Free up all the memory allocated by scds_initialize */
   scds_close(&scds_handle);
   return (1);
}

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

주 – `scds_pmf_restart_fm()`의 두 번째 인자는 여러 인스턴스가 있을 경우 재시작할 오류 모니터의 인스턴스를 고유하게 식별합니다. 이 예의 값 `0`은 오류 모니터의 인스턴스가 한 개밖에 없음을 나타냅니다.

9장

SunPlex Agent Builder

이 장에서는 자원 유형이나 데이터 서비스의 작성이 RGM(Resource Group Manager)의 제어 하에 실행되도록 자동화하는 도구인 SunPlex Agent Builder와 Agent Builder용 Cluster Agent 모듈에 대해 설명합니다. **자원 유형**은 클러스터 환경에서 RGM의 제어하에 응용 프로그램을 실행시키는 응용 프로그램 래퍼입니다.

이 장은 다음 내용으로 구성되어 있습니다.

Agent Builder 개요

Agent Builder에서는 응용 프로그램과 작성할 자원 유형에 대한 정보를 지정하기 위한 그래픽 사용자 인터페이스(GUI)를 제공합니다. Agent Builder는 네트워크 인식 응용 프로그램과 네트워크 비인식 응용 프로그램을 지원합니다. **네트워크 인식 응용 프로그램**은 네트워크를 사용하여 클라이언트와 통신합니다. **네트워크 비인식 응용 프로그램**은 독립 실행형 응용 프로그램입니다.

주 – Agent Builder의 GUI 버전에 액세스할 수 없는 경우 명령줄 인터페이스를 통해 Agent Builder에 액세스할 수 있습니다. 160 페이지 “Agent Builder 명령줄 버전 사용 방법”을 참조하십시오.

지정한 정보에 따라 Agent Builder는 다음과 같은 소프트웨어를 생성합니다.

- 자원 유형 메소드 콜백에 해당하는 페일오버 또는 확장 가능한 자원 유형의 C, Korn 쉘(ksh) 또는 일반 데이터 서비스(GDS) 소스 파일 집합. 이러한 파일은 네트워크 인식(클라이언트-서버 모델) 응용 프로그램과 네트워크 비인식(클라이언트 가 없는 모델) 응용 프로그램에 모두 사용됩니다.
- 사용자 정의 자원 유형 등록(RTR) 파일(C 쉘 또는 Korn 쉘 소스 코드를 생성할 경우)
- 각 파일의 사용 방법을 설명하는 사용자 정의 설명서 페이지 및 자원 유형의 인스턴스(자원)를 시작, 중지 및 제거하는 사용자 정의 유틸리티 스크립트
- 이진 파일(C 소스 코드 생성 시), RTR 파일(C 쉘 또는 Korn 쉘 소스 코드 생성 시) 및 유틸리티 스크립트를 포함하는 Solaris 패키지

또한 Agent Builder를 사용하여 PMF(Process Monitor Facility)에서 모니터해야 하고 개별적으로 다시 시작해야 하는 다중 독립 프로세스 트리를 가진 응용 프로그램에 대한 자원 유형을 생성할 수 있습니다.

Agent Builder를 사용하기 전에 알아야 할 사항

Agent Builder를 사용하기 전에 다중 독립 프로세스 트리를 가진 자원 유형을 만드는 방법을 알아야 합니다.

Agent Builder에서는 다중 독립 프로세스 트리를 가진 응용 프로그램에 대한 자원 유형을 만들 수 있습니다. 이러한 프로세스 트리는 PMF에서 개별적으로 모니터하고 시작한다는 점에서 독립적입니다. PMF는 고유한 태그를 사용하여 각 프로세스 트리를 시작합니다.

주 – 지정하는 생성된 소스 코드가 C 또는 GDS인 경우에만 Agent Builder를 사용하여 다중 독립 프로세스 트리를 가진 자원 유형을 만들 수 있습니다. Korn 쉘에 대해서는 Agent Builder를 사용하여 이러한 자원 유형을 만들 수 없습니다. Korn 쉘에 대해 이러한 자원 유형을 만들려면 수동으로 코드를 작성해야 합니다.

다중 독립 프로세스 트리를 가진 기본 응용 프로그램의 경우 단일 명령줄을 지정하여 응용 프로그램을 시작할 수 없습니다. 그보다는 텍스트 파일을 만들어 각각의 줄이 명령에 대한 전체 경로를 지정하여 응용 프로그램 프로세스 트리 중 하나를 시작해야 합니다. 이 파일의 모든 줄은 공백이 아니어야 합니다. Agent Builder Configure 화면의 Start Command 텍스트 필드에서 이 텍스트 파일을 지정합니다.

이 파일에 실행 권한을 부여하지 않으면 Agent Builder에서 이 파일을 구별할 수 있습니다. 이 파일의 용도는 여러 개의 명령이 포함된 단순 실행 가능 스크립트에서 여러 프로세스 트리를 시작하기 위한 것입니다. 텍스트 파일에 실행 권한을 부여하면 해당 자원은 클러스터에 아무 문제나 오류 없이 나타나지만 모든 명령이 하나의 PMF 태그로 시작됩니다. 따라서 PMF에서 개별적으로 프로세스 트리를 모니터하거나 재시작할 수 없습니다.

Agent Builder 사용

이 절에서는 Agent Builder 사용 전에 완료해야 할 작업을 포함하여 Agent Builder 사용 방법에 대해 설명합니다. 또한 이 절에서는 자원 유형 코드를 생성한 후의 Agent Builder 이용 방법에 대해서도 설명합니다.

이 장은 다음 내용으로 구성되어 있습니다.

응용 프로그램 분석

Agent Builder를 사용하려면 먼저 응용 프로그램이 가용성 또는 확장성이 뛰어난 응용 프로그램이 되기 위한 기준을 충족시키는지 확인해야 합니다. Agent Builder에서는 응용 프로그램의 런타임 특성만을 토대로 분석을 하므로 이러한 분석을 수행할 수 없습니다. 이에 대한 자세한 내용은 29 페이지 “응용 프로그램 적합성 분석”을 참조하십시오.

Agent Builder는 개발자 응용 프로그램에 대해 항상 전체 자원 유형을 생성할 수 있는 것은 아니지만 대부분의 경우 적어도 부분적인 솔루션을 제공합니다. 예를 들어, 보다 복잡한 응용 프로그램에는 Agent Builder에서 기본적으로 생성하지 않는 추가 코드가 필요 할 수 있습니다. 이러한 코드에는 추가 등록 정보에 대한 검증 확인을 추가하는 코드, Agent Builder에서 제공하지 않는 매개 변수를 조정하는 코드 등이 포함됩니다. 이런 경우 생성된 소스 코드나 RTR 파일을 변경해야 합니다. Agent Builder는 이런 종류의 융통성을 제공할 수 있도록 설계되었습니다.

Agent Builder는 생성된 소스 코드의 특정 지점에 사용자 정의 자원 유형 코드를 추가할 수 있는 주석을 배치합니다. 소스 코드를 변경한 후 Agent Builder에서 생성한 makefile을 사용하여 소스 코드를 다시 컴파일하고 자원 유형 패키지를 다시 생성할 수 있습니다.

Agent Builder에서 생성한 코드를 사용하지 않고 전체 자원 유형 코드를 작성한 경우라도 Agent Builder에서 제공한 makefile 및 구조를 이용하여 자원 유형에 대한 Solaris 패키지를 만들 수 있습니다.

Agent Builder 설치 및 구성

Agent Builder에는 특수 설치가 필요하지 않습니다. Agent Builder는 기본적으로 표준 Sun Cluster 소프트웨어 설치 중에 설치되는 SUNWscdev 패키지에 포함되어 있습니다. 자세한 내용은 **Solaris OS용 Sun Cluster 소프트웨어 설치 안내서**를 참조하십시오.

Agent Builder를 사용하기 전에 다음 요구 사항을 확인하십시오.

- Java 런타임 환경이 $PATH 변수에 포함되어 있는지 확인합니다. Agent Builder가 Java Development Kit 버전 1.3.1 이상에 종속되어 있는지 확인합니다. Java Development Kit가 $PATH 변수에 포함되어 있지 않은 경우 Agent Builder 명령(scdsbuilder)이 오류 메시지를 반환하고 표시하는지 확인합니다.
- Solaris 8 OS 버전 이상의 Developer System Support 소프트웨어 그룹을 설치했는지 확인합니다.
- cc 컴파일러가 $PATH 변수에 포함되어 있는지 확인합니다. Agent Builder가 $PATH 변수에서 cc의 첫 번째 항목을 사용하여 자원 유형에 대한 C 이진 코드를 생성하는 데 사용되는 컴파일러를 식별하는지 확인합니다. cc가 $PATH에 포함되지 않은 경우 Agent Builder는 해당 옵션을 비활성화하여 C 코드를 생성합니다. 151 페이지 "Create 화면 사용"을 참조하십시오.

주 – 표준 cc 컴파일러가 아닌 다른 컴파일러를 Agent Builder와 함께 사용할 수 있습니다. 다른 컴파일러를 사용하려면 cc에서 gcc 등 다른 컴파일러로의 심볼릭 링크를 $PATH에 만듭니다. 또는 makefile(현재 CC=cc)의 컴파일러 지정을 다른 컴파일러의 전체 경로로 변경합니다. 예를 들어, Agent Builder에서 생성한 makefile에서 CC=cc를 CC=*pathname*/gcc로 변경합니다. 이 경우 Agent Builder를 직접 실행할 수 없으며 데이터 서비스 코드와 패키지를 생성하려면 make 및 make pkg 명령을 사용해야 합니다.

Agent Builder 화면

Agent Builder는 단계별로 해당 화면이 있는 2단계 마법사입니다. Agent Builder에서는 새로운 자원 유형 작성 프로세스를 안내하는 다음 두 개의 화면을 제공합니다.

1. **Create 화면.** 이 화면에서 만들려는 자원 유형에 대한 기본 정보(예: 이름, 생성된 파일의 작업 디렉토리 등)를 제공합니다. 작업 디렉토리는 자원 유형 템플리트를 만들고 구성하는 위치입니다. 또한 다음 정보를 지정합니다.
   - 만들려는 자원의 종류(확장 가능 또는 페일오버)
   - 기본 응용 프로그램이 네트워크를 인식하는지 여부(즉, 네트워크를 사용하여 클라이언트와 통신하는지 여부)
   - 생성할 코드 유형(C, Korn 쉘(ksh) 또는 GDS)

   GDS에 대한 자세한 내용은 10 장를 참조하십시오. 이 화면에서 모든 정보를 제공하고 Create를 선택하여 해당 출력을 생성해야 합니다. 그런 다음 구성 화면을 표시할 수 있습니다.

2. **Configure 화면.** 이 화면에서 UNIX 쉘에 전달될 수 있는 전체 명령줄을 지정하여 기본 응용 프로그램을 시작해야 합니다. 또는 응용 프로그램을 중지 및 검사하기 위한 명령을 제공할 수 있습니다. 이러한 명령을 지정하지 않으면 생성된 출력은 신호를 사용하여 응용 프로그램을 중지하고 기본 검사 기법을 제공합니다. 검사 명령에 대한 자세한 내용은 153 페이지 "Configure 화면 사용"을 참조하십시오. Configure 화면을 사용하여 시작, 중지, 검사 명령의 각 시간 초과값도 변경할 수 있습니다.

Agent Builder 시작

주 – Agent Builder의 GUI 버전에 액세스할 수 없는 경우 명령줄 인터페이스를 통해 Agent Builder에 액세스할 수 있습니다. 160 페이지 "Agent Builder 명령줄 버전 사용 방법"을 참조하십시오.

기존 자원 유형의 작업 디렉토리에서 Agent Builder를 시작하면 Agent Builder에서 Create 및 Configure 화면을 기존 자원 유형 값으로 초기화합니다.

다음 명령을 입력하여 Agent Builder를 시작합니다.

```
% /usr/cluster/bin/scdsbuilder
```

Create 화면이 나타납니다.

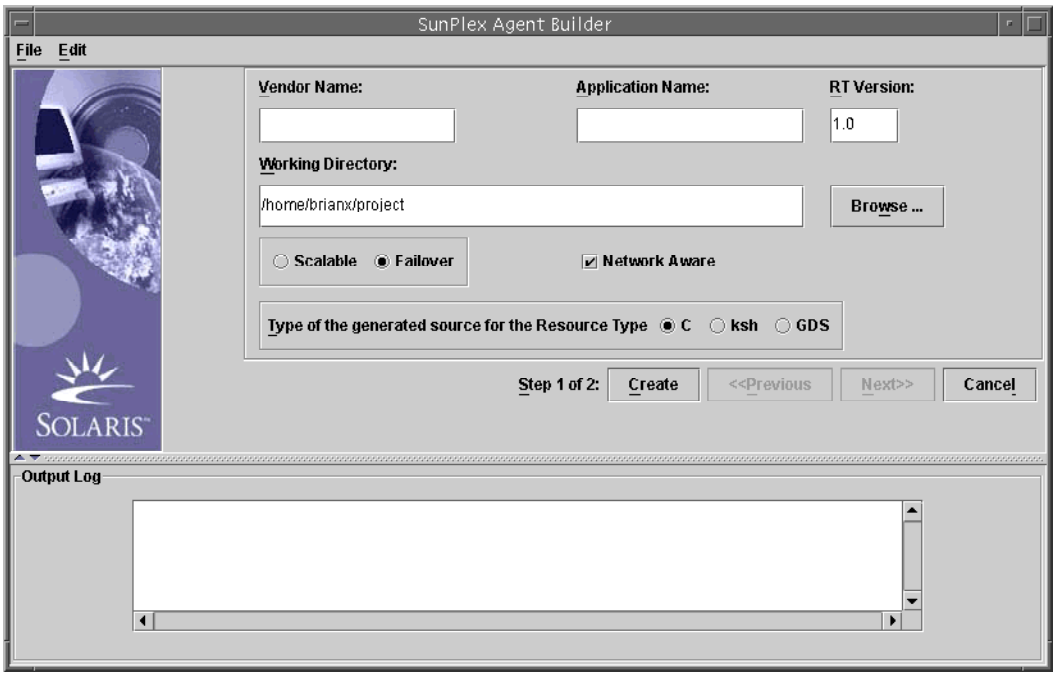


그림 **9–1** Agent Builder의 Create 화면

Agent Builder 탐색

다음 작업을 수행하여 Create 및 Configure 화면에 대한 정보를 입력합니다.

- 필드에 직접 정보 입력
- 디렉토리 구조를 찾아 파일이나 디렉토리 선택
- 상호 배타적인 라디오 버튼 세트 중 하나 선택(예: Scalable 또는 Failover 선택)
- Network Aware 확인란을 선택하여 기본 응용 프로그램을 네트워크 인식으로 식별하거나 이 확인란을 비워두어 네트워크 비인식 응용 프로그램으로 식별

각 화면 아래 있는 버튼을 사용하면 작업을 완료하거나, 다음이나 이전 화면으로 이동하거나, Agent Builder를 종료할 수 있습니다. Agent Builder는 이러한 버튼을 필요에 따라 강조표시하거나 비활성화합니다.

예를 들어, Create 화면에서 필드를 입력하고 원하는 옵션을 선택한 경우 화면 아래쪽에 있는 Create를 누릅니다. 이전 화면이 없고 이 단계를 완료하기 전에는 다음 단계로 이동할 수 없기 때문에 Previous 및 Next는 비활성화됩니다.

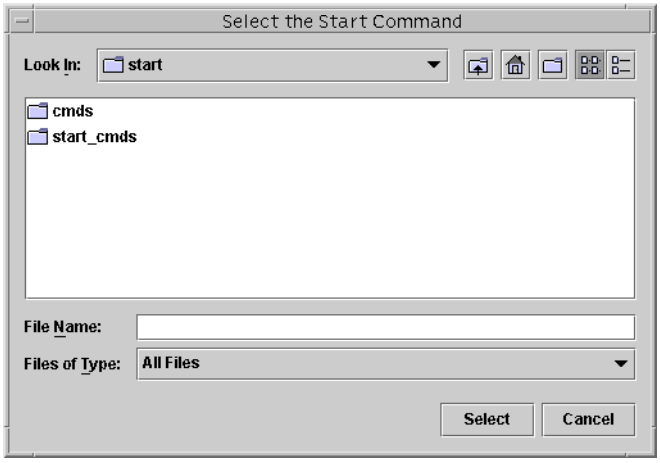
Agent Builder는 화면 아래쪽에 있는 Output Log 영역에 진행 메시지를 표시합니다. Agent Builder가 완료되면 성공 메시지 또는 경고 메시지를 표시합니다. Next가 강조 표시됩니다. 마지막 화면인 경우에는 Cancel만 강조 표시됩니다.

언제든지 Cancel을 눌러 Agent Builder를 종료할 수 있습니다.

Browse 명령

특정 Agent Builder 필드를 사용하여 정보를 입력하거나 Browse를 눌러 디렉토리 구조를 찾아보고 파일이나 디렉토리를 선택할 수 있습니다.

Browse를 누르면 다음 화면과 유사한 화면이 나타납니다.



폴더를 두 번 눌러 폴더를 엽니다. 커서를 파일로 이동하면 파일 이름이 File Name 상자에 표시됩니다. 원하는 파일을 찾아서 커서를 이동한 다음 Select를 누릅니다.

주 – 디렉토리를 찾으려면 커서를 원하는 디렉토리로 이동하고 Open을 누릅니다. 디렉토리에 하위 디렉토리가 없으면 Agent Builder에서 Browse 창을 닫고 커서를 이동한 디렉토리 이름을 해당 필드에 넣습니다. 디렉토리에 하위 디렉토리가 있는 경우 Close를 눌러 찾아보기 창을 닫고 이전 화면을 다시 표시합니다. Agent Builder에서 커서를 이동한 디렉토리 이름을 해당 필드에 넣습니다.

Browse 화면의 오른쪽 위에 있는 아이콘은 다음 작업을 수행합니다.

아이콘	목적
	디렉토리 트리에서 한 수준 위로 이동합니다.
	홈 폴더로 되돌아갑니다.
	현재 선택된 폴더에 새 폴더를 만듭니다.
	서로 다른 뷰 간을 전환하기 위한 이 아이콘은 나중에 사용하기 위해 예약되어 있습니다.

Agent Builder 메뉴

Agent Builder는 File 및 Edit 드롭다운 메뉴를 제공합니다.

Agent Builder File 메뉴

File 메뉴에는 다음 두 가지 옵션이 있습니다.

- **Load Resource Type.** 기존 자원 유형을 로드합니다. Agent Builder는 기존 자원 유형에 대한 작업 디렉토리를 선택하는 Browse 화면을 제공합니다. Agent Builder를 시작한 디렉토리에 자원 유형이 있으면 Agent Builder는 자동으로 이 자원 유형을 로드합니다. Load Resource Type을 사용하면 임의의 디렉토리에서 Agent Builder를 시작하고 새 자원 유형을 만들기 위한 템플리트로 사용할 기존 자원 유형을 선택할 수 있습니다. 158 페이지 “Agent Builder를 사용하여 작성한 코드 재사용”을 참조하십시오.

- **Exit.** Agent Builder를 종료합니다. Create 화면이나 Configure 화면에서 Cancel을 눌러 종료할 수도 있습니다.

Agent Builder Edit 메뉴

Edit 메뉴에는 다음 두 가지 옵션이 있습니다.

- **Clear Output Log.** 출력 로그에서 정보를 지웁니다. Create나 Configure를 선택할 때마다 Agent Builder는 상태 메시지를 출력 로그에 추가합니다. Agent Builder에서 소스 코드를 변경하고 출력을 재생성하는 반복 과정에서 상태 메시지를 분리하려면 매번 사용하기 전에 로그 파일을 저장하고 지울 수 있습니다.
- **Save Log File.** 로그 출력을 파일로 저장합니다. Agent Builder는 디렉토리를 선택하고 파일 이름을 지정할 수 있는 Browse 화면을 제공합니다.

Create 화면 사용

자원 유형을 만드는 첫 번째 단계는 Agent Builder를 시작할 때 나타나는 Create 화면에 정보를 입력하는 것입니다. 다음 그림에서는 필드에 정보를 입력한 후의 Create 화면을 보여줍니다.

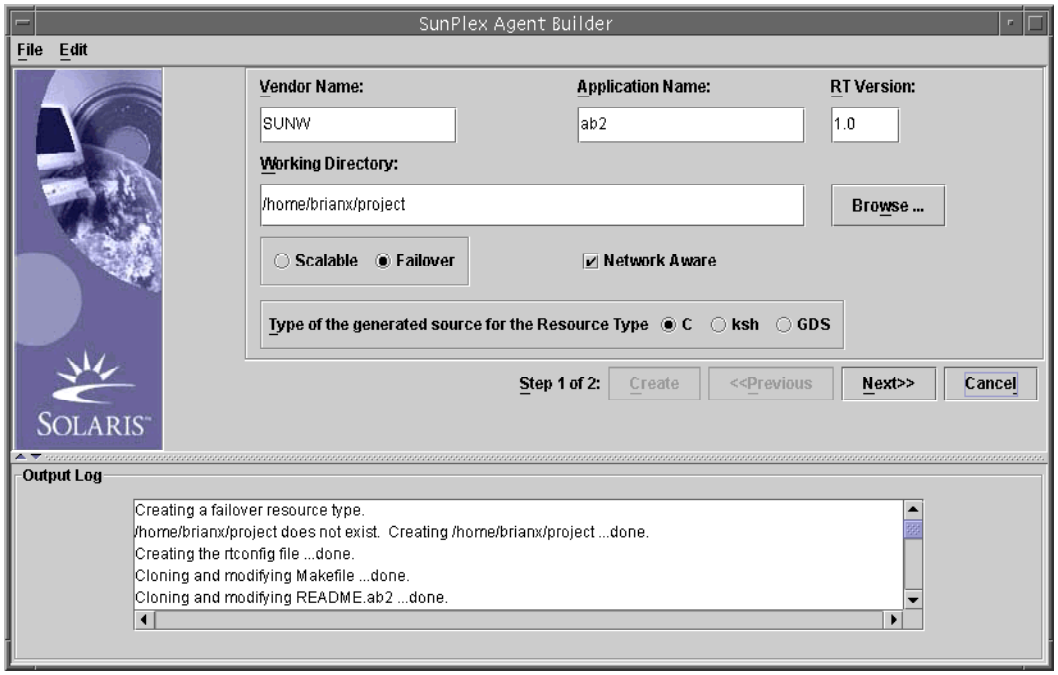


그림 **9–2** 정보를 입력한 후의 Agent Builder Create 화면

Create 화면에는 다음과 같은 필드, 라디오 버튼 및 확인란이 포함되어 있습니다.

- **Vendor Name.** 자원 유형의 공급업체를 식별하는 이름입니다. 대개 공급업체의 주식 기호를 지정하지만 해당 공급업체를 고유하게 식별하는 모든 이름이 유효합니다. 영숫자만 사용하십시오.
- **Application Name.** 자원 유형의 이름입니다. 영숫자만 사용하십시오.

주 – Vendor Name과 Application Name을 함께 사용하여 자원 유형의 전체 이름을 구성합니다. Solaris 9 운영 체제부터는 Vendor Name과 Application Name의 조합이 9자를 넘을 수 있습니다. 그러나 이전 버전의 Solaris 운영 체제를 사용 중이면 Vendor Name과 Application Name의 조합이 9자를 넘을 수 없습니다.

- **RT Version.** 생성된 자원 유형의 버전입니다. RT Version은 기본 자원 유형이 동일한 여러 등록된 버전이나 업그레이드를 구분합니다.

  다음 문자는 RT Version 필드에 사용할 수 없습니다.

  - 공백
  - Tab
  - 슬래시(/)
  - 백슬래시(\)
  - 별표(*)
  - 물음표(?)
  - 쉼표(,)
  - 세미콜론(;)
  - 여는 대괄호([)
  - 닫는 대괄호(])
- **Working Directory.** Agent Builder가 대상 자원 유형에 대해 만든 모든 파일을 포함하기 위해 만든 디렉토리 구조의 디렉토리입니다. 하나의 작업 디렉토리에 하나의 자원 유형만 만들 수 있습니다. Agent Builder는 이 필드를 Agent Builder를 시작한 디렉토리 경로로 초기화하지만 다른 이름을 입력하거나 Browse를 사용하여 다른 디렉토리를 찾을 수 있습니다.

  Agent Builder는 작업 디렉토리에 자원 유형 이름을 가진 하위 디렉토리를 만듭니다. 예를 들어, `SUNW`가 공급업체 이름이고 `ftp`가 응용 프로그램 이름일 경우 Agent Builder는 이 하위 디렉토리 이름을 `SUNWftp`로 지정합니다.

  Agent Builder는 대상 자원 유형의 모든 디렉토리와 파일을 이 하위 디렉토리에 넣습니다. 161 페이지 “Agent Builder에서 만드는 디렉토리 구조”를 참조하십시오.
- **Scalable 또는 Failover.** 대상 자원 유형이 페일오버인지 또는 확장 가능인지를 지정합니다.
- **Network Aware.** 기본 응용 프로그램이 네트워크를 인식하는지, 즉 네트워크를 사용하여 해당 클라이언트와 통신하는지 여부를 지정합니다. 네트워크 인식을 지정하려면 Network Aware 확인란을 선택하고 네트워크 비인식을 지정하려면 이 확인란을 선택하지 않습니다.
- **C, ksh.** 생성된 소스 코드의 언어를 지정합니다. 이러한 옵션은 상호 배타적이지만 Agent Builder에서는 Korn 쉘 생성 코드를 사용하여 자원 유형을 만든 다음 동일한 정보를 재사용하여 C 생성 코드를 만들 수 있습니다. 158 페이지 “Agent Builder를

사용하여 작성한 코드 재사용"을 참조하십시오.

- **GDS.** 해당 서비스가 일반 데이터 서비스가 되도록 지정합니다. 일반 데이터 서비스 작성 및 구성에 대한 자세한 내용은 10 장를 참조하십시오.

주 – cc 컴파일러가 $PATH 변수에 없는 경우 Agent Builder에서 C 라디오 버튼을 비활성화하고 ksh 라디오 버튼을 선택하도록 허용합니다. 다른 컴파일러를 지정하려면 146 페이지 "Agent Builder 설치 및 구성" 마지막 부분의 설명을 참조하십시오.

필수 정보를 지정한 후 Create을 누릅니다. 화면 아래쪽의 Output Log 영역에 Agent Builder가 수행 중인 작업이 표시됩니다. Edit 메뉴에서 Save Output Log를 선택하여 출력 로그의 정보를 저장할 수 있습니다.

작업이 완료되면 Agent Builder는 성공 메시지 또는 경고 메시지를 표시합니다.

- Agent Builder에서 이 단계를 완료할 수 없는 경우 출력 로그에서 자세한 내용을 확인하십시오.
- Agent Builder가 성공적으로 완료되면 Next를 눌러 자원 유형 생성을 완료할 수 있는 Configure 화면을 표시합니다.

주 – 전체 자원 유형 생성은 2단계의 프로세스이며, 첫 번째 단계(Create)를 완료한 다음 지정한 정보나 Agent Builder에서 완료한 작업을 잃어버리지 않고 Agent Builder를 종료할 수 있습니다. 158 페이지 "Agent Builder를 사용하여 작성한 코드 재사용"을 참조하십시오.

Configure 화면 사용

Agent Builder에서 자원 유형 생성을 완료한 후 Create 화면에서 Next를 누르면 다음 그림과 같은 Configure 화면이 나타납니다. 자원 유형을 만들기 전에는 Configure 화면에 액세스할 수 없습니다.

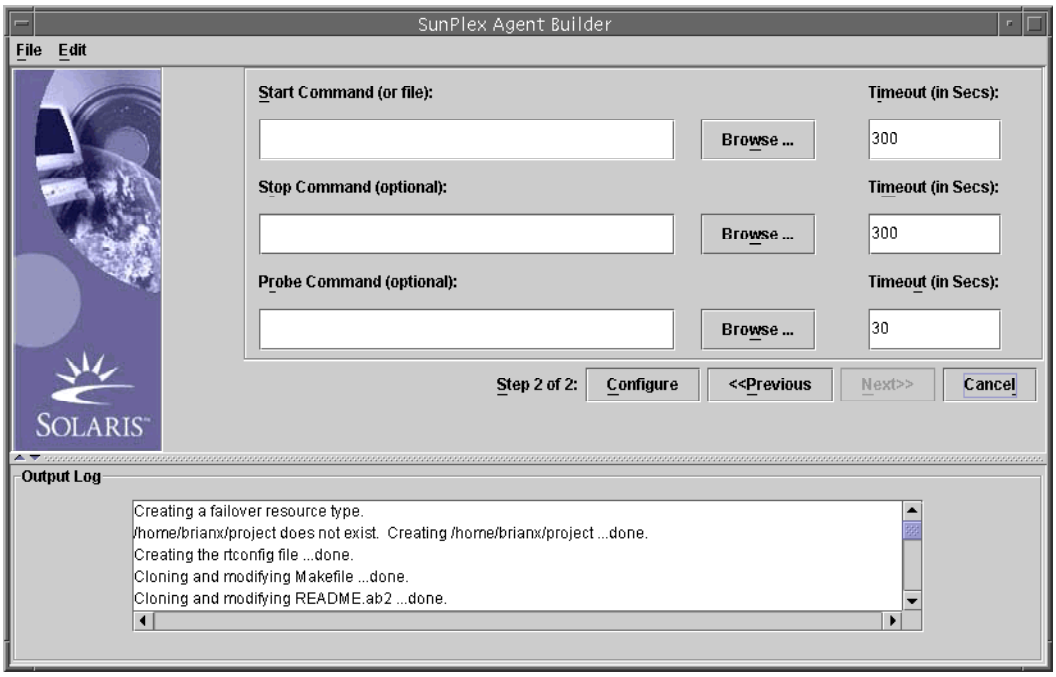


그림 **9–3** Agent Builder의 Configure 화면

Configure 화면에는 다음과 같은 필드가 있습니다.

- **Start Command.** 기본 응용 프로그램을 시작하기 위해 UNIX 쉘에 전달할 수 있는 전체 명령줄입니다. 시작 명령을 지정해야 합니다. 제공된 필드에 명령을 입력하거나 Browse 버튼을 사용하여 응용 프로그램을 시작하는 명령이 포함된 파일을 찾을 수 있습니다.

  전체 명령줄에는 호스트 이름, 포트 번호, 구성 파일 경로 등 응용 프로그램을 시작하는 데 필요한 모든 것이 포함되어야 합니다. 156 페이지 “등록 정보 변수 사용”에 설명된 등록 정보 변수를 지정할 수도 있습니다. Korn 쉘 기반 응용 프로그램에서 명령줄에 호스트 이름을 지정해야 하는 경우 Agent Builder가 정의하는 `$hostnames` 변수를 사용할 수 있습니다. 156 페이지 “Agent Builder Korn 쉘 기반 `$hostnames` 변수 사용”을 참조하십시오.

  명령을 큰따옴표(“”)로 묶지 마십시오.

주 – 기본 응용 프로그램에 다중 독립 프로세스 트리가 있고 각 프로세스가 PMF(Process Monitor Facility) 제어의 고유한 태그로 시작할 경우 단일 명령을 지정할 수 없습니다. 그 대신 각 프로세스 트리를 시작하는 개별 명령이 포함된 텍스트 파일을 만들고 Start Command 텍스트 필드에서 이 파일의 경로를 지정해야 합니다. 144 페이지 “Agent Builder를 사용하기 전에 알아야 할 사항”을 참조하십시오. 이 절에는 이 파일이 올바로 작동하는 데 필요한 일부 특수 문자가 나열되어 있습니다.

- **Stop Command.** 기본 응용 프로그램을 중지하기 위해 UNIX 쉘에 전달할 수 있는 전체 명령줄입니다. 제공된 필드에 명령을 입력하거나 Browse 버튼을 사용하여 응용 프로그램을 중지하는 명령이 포함된 파일을 찾을 수 있습니다. 156 페이지 “등록 정보 변수 사용”에 설명된 등록 정보 변수를 지정할 수도 있습니다. Korn 쉘 기반 응용 프로그램에서 명령줄에 호스트 이름을 지정해야 하는 경우 Agent Builder가 정의하는 `$hostnames` 변수를 사용할 수 있습니다. 156 페이지 “Agent Builder Korn 쉘 기반 $hostnames 변수 사용”을 참조하십시오.

  이 명령은 선택 사항입니다. 중지 명령을 지정하지 않은 경우에는 생성된 코드가 `Stop` 메소드에서 다음과 같은 신호를 사용하여 응용 프로그램을 중지합니다.

  - `Stop` 메소드는 `SIGTERM`을 보내 응용 프로그램을 중지하고 시간 초과값의 80% 동안 대기한 다음 응용 프로그램을 종료합니다.
  - `SIGTERM` 신호가 실패한 경우 `Stop` 메소드는 `SIGKILL`을 보내 응용 프로그램을 중지하고 시간 초과값의 15% 동안 대기한 다음 응용 프로그램을 종료합니다.
  - `SIGKILL`이 실패한 경우 `Stop` 메소드가 제대로 종료되지 않습니다. 시간 초과값의 나머지 5%는 오버헤드로 간주됩니다.

주의 – 응용 프로그램이 완전히 중지되기 전에 중지 명령이 반환되지 않도록 합니다.

- **Probe Command.** 주기적으로 실행하여 응용 프로그램의 상태를 확인하고 `0`(성공)과 `100`(완전한 실패) 사이의 종료 상태를 반환할 수 있는 명령입니다. 이 명령은 선택 사항입니다. 명령에 대한 전체 경로를 입력하거나 Browse를 사용하여 응용 프로그램을 검사하기 위한 명령이 포함된 파일을 찾을 수 있습니다.

  일반적으로 기본 응용 프로그램의 단순 클라이언트를 지정합니다. 검사 명령을 지정하지 않으면 생성된 코드는 간단하게 자원에 사용되는 포트에 연결하거나 연결을 끊습니다. 연결 및 연결 끊기에 성공하면 생성된 코드에서 응용 프로그램 상태를 선언합니다. 156 페이지 “등록 정보 변수 사용”에 설명된 등록 정보 변수를 지정할 수도 있습니다. Korn 쉘 기반 응용 프로그램에서 검사 명령줄에 호스트 이름을 지정해야 하는 경우 Agent Builder가 정의하는 `$hostnames` 변수를 사용할 수 있습니다. 156 페이지 “Agent Builder Korn 쉘 기반 $hostnames 변수 사용”을 참조하십시오.

  명령을 큰따옴표(“”)로 묶지 마십시오.

- **Timeout.** 각 명령에 대한 시간 초과값(초)입니다. 새 값을 지정하거나 Agent Builder에서 제공하는 기본값을 적용할 수 있습니다. 기본값은 시작 및 중지의 경우 300초, 검사의 경우 30초입니다.

Agent Builder Korn 쉘 기반 $hostnames 변수 사용

많은 응용 프로그램, 특히 네트워크 인식 응용 프로그램의 경우 응용 프로그램에서 수신하는 호스트 이름과 서비스 고객 요청을 명령줄의 응용 프로그램에 전달해야 합니다. 대부분의 경우 호스트 이름은 Configure 화면에서 대상 자원 유형에 대한 시작, 중지 및 검사 명령에 지정해야 하는 인자입니다. 그러나 응용 프로그램이 수신하는 호스트 이름은 클러스터에 특정합니다. 호스트 이름은 자원이 클러스터에서 실행될 때 결정되고 Agent Builder에서 자원 유형 코드를 생성할 때는 결정될 수 없습니다.

이 문제를 해결하기 위해 Agent Builder는 개발자가 명령줄에서 시작, 중지 및 검사 명령에 지정할 수 있는 $hostnames 변수를 제공합니다.

주 – $hostnames 변수는 Korn 쉘 기반 서비스에서만 지원됩니다. $hostnames 변수는 C 기반 및 GDS 기반 서비스에서는 지원되지 않습니다.

다음 예와 같이 $hostnames 변수를 실제 호스트 이름과 같이 정확하게 지정합니다.

```
% /opt/network_aware/echo_server -p port-no -l $hostnames
```

대상 자원 유형의 자원이 클러스터에서 실행되는 경우 해당 자원에 대해 구성된 LogicalHostname 또는 SharedAddress 호스트 이름이 $hostnames 변수 값으로 대체됩니다. 호스트 이름은 자원의 Network_resources_used 자원 등록 정보에서 해당 자원에 대해 구성됩니다.

멀티 호스트 이름을 가진 Network_resources_used 등록 정보를 구성한 경우 $hostnames 변수에는 모든 호스트 이름이 쉼표로 구분되어 포함됩니다.

등록 정보 변수 사용

등록 정보 변수를 사용하여 RGM 프레임워크에서 선택한 Sun Cluster 자원 유형, 자원 및 자원 그룹 등록 정보 값을 검색할 수도 있습니다. Agent Builder는 등록 정보 변수에 대한 시작, 검사 또는 중지 명령 문자열을 검색하여 Agent Builder에서 명령을 실행하기 전에 이러한 변수를 해당 값으로 대체합니다.

주 – 등록 정보 변수는 Korn 쉘 기반 서비스에는 지원되지 않습니다.

등록 정보 변수 목록

이 절에는 사용할 수 있는 등록 정보 변수가 나열되어 있습니다. Sun Cluster 자원 유형, 자원 및 자원 그룹 등록 정보에 대해서는 부록 A에서 설명합니다.

자원 등록 정보 변수

- HOSTNAMES
- RS_CHEAP_PROBE_INTERVAL
- RS_MONITOR_START_TIMEOUT
- RS_MONITOR_STOP_TIMEOUT
- RS_NAME
- RS_NUM_RESTARTS
- RS_RESOURCE_DEPENDENCIES
- RS_RESOURCE_DEPENDENCIES_WEAK
- RS_RETRY_COUNT
- RS_RETRY_INTERVAL
- RS_SCALABLE
- RS_START_TIMEOUT
- RS_STOP_TIMEOUT
- RS_THOROUGH_PROBE_INTERVAL
- SCHA_STATUS

자원 유형 등록 정보 변수

- RT_API_VERSION
- RT_BASEDIR
- RT_FAILOVER
- RT_INSTALLED_NODES
- RT_NAME
- RT_RT_VERSION
- RT_SINGLE_INSTANCE

자원 그룹 등록 정보 변수

- RG_DESIRED_PRIMARIES
- RG_GLOBAL_RESOURCES_USED
- RG_IMPLICIT_NETWORK_DEPENDENCIES
- RG_MAXIMUM_PRIMARIES
- RG_NAME
- RG_NODELIST
- RG_NUM_RESTARTS
- RG_PATHPREFIX
- RG_PINGPONG_INTERVAL
- RG_RESOURCE_LIST

등록 정보 변수 구문

이 예에 표시된 것처럼 등록 정보 이름 앞에 퍼센트 기호(%)를 포함하여 등록 정보 변수를 나타냅니다.

```
/opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

앞의 예에서 Agent Builder는 이러한 등록 정보 변수를 해석하고 다음 값을 사용하여 `echo_server` 스크립트를 시작할 수 있습니다.

```
/opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

Agent Builder에서 등록 정보 변수를 대체하는 방법

Agent Builder는 등록 정보 변수 유형을 다음과 같이 해석합니다.

- integer는 실제 값(예: `300`)으로 대체됩니다.
- 부울 값은 문자열 `TRUE` 또는 `FALSE`로 대체됩니다.
- string은 실제 문자열(예: `phys-node-1`)로 대체됩니다.
- 문자열 목록은 목록의 모든 구성원으로 대체됩니다. 각 문자열은 쉼표로 구분됩니다(예: `phys-node-1,phys-node-2,phys-node-3`).
- 정수 목록은 목록의 모든 구성원으로 대체됩니다. 각 정수는 쉼표로 구분됩니다(예: `1,2,3`).
- 열거 유형은 문자열 형식의 해당 값으로 대체됩니다.

Agent Builder를 사용하여 작성한 코드 재사용

Agent Builder를 사용하여 다음과 같은 방법으로 완료된 작업을 재사용할 수 있습니다.

- Agent Builder를 사용하여 만든 기존 자원 유형을 복제할 수 있습니다.
- Agent Builder에서 생성한 소스 코드를 편집한 다음 코드를 다시 컴파일하여 새로운 패키지를 만들 수 있습니다.

▼ 기존 자원 유형 복제 방법

Agent Builder에서 생성한 기존 자원 유형을 복제하려면 다음 절차를 수행합니다.

단계 **1. 다음 중 하나의 방법을 사용하여 기존 자원 유형을 Agent Builder로 로드합니다.**

- Agent Builder에서 만든 기존 자원 유형의 작업 디렉토리에서 Agent Builder를 시작합니다. 작업 디렉토리에 `rtconfig` 파일이 들어 있는지 확인합니다. Agent Builder는 Create 및 Configure 화면에서 해당 자원 유형에 대한 값을 로드합니다.
- File 드롭다운 메뉴에서 Load Resource Type 옵션을 사용합니다.

2. **Create 화면에서 작업 디렉토리를 변경합니다.**

   Browse를 사용하여 디렉토리를 선택해야 합니다. 새 디렉토리 이름을 입력하는 것만으로는 충분하지 않습니다. 디렉토리를 선택한 후 Agent Builder에서 Create 버튼을 다시 사용할 수 있습니다.

3. **기존 자원 유형을 변경합니다.**

   자원 유형에 대해 생성된 코드 유형을 변경할 수도 있습니다. 예를 들어, 처음에 Korn 쉘 버전의 자원 유형을 만들었지만 시간이 지나면서 C 버전이 필요한 경우 다음을 수행할 수 있습니다.

   - 기존 Korn 쉘 자원 유형을 로드합니다.
   - 출력 언어를 C로 변경합니다.
   - Create를 눌러 Agent Builder에서 C 버전의 자원 유형을 만들도록 합니다.

4. **복제된 자원 유형을 만듭니다.**

   a. **Create를 눌러 자원 유형을 만듭니다.**

   b. **Next를 눌러 Configure 화면을 표시합니다.**

   c. **Configure를 눌러 자원 유형을 구성한 다음 Cancel을 눌러 마칩니다.**

생성된 소스 코드 편집

Agent Builder는 자원 유형 생성 프로세스를 간소화하기 위해 지정할 수 있는 정보 양을 제한하며, 이에 따라 생성된 자원 유형의 범위도 제한됩니다. 따라서 고급 기능을 추가하려면 생성된 소스 코드나 RTR 파일을 수정해야 합니다. 이러한 기능에는 추가 등록 정보에 대한 검증 확인을 추가하는 코드, Agent Builder에서 제공하지 않는 매개변수를 조정하는 코드 등이 포함됩니다.

소스 파일은 *install-directory*/*rt-name*/`src` 디렉토리에 있습니다. Agent Builder는 소스 코드에 개발자가 코드를 추가할 수 있는 주석을 포함합니다. 주석의 형식은 다음과 같습니다(C 코드의 경우).

```
/* User added code -- BEGIN vvvvvvvvvvvvvvv */
/* User added code -- END   ^^^^^^^^^^^^^^^ */
```

주 – 이러한 주석은 파운드 기호(#)로 주석의 시작을 나타낸다는 점을 제외하고 Korn 쉘 소스 코드에서와 동일합니다.

예를 들어, *rt-name*`.h`는 다른 프로그램에서 사용하는 모든 유틸리티 함수를 선언합니다. 선언 목록 끝에는 개발자가 코드에 추가했을 수 있는 추가 함수를 선언할 수 있는 주석이 있습니다.

또한 Agent Builder는 *install-directory*/*rt-name*/`src` 디렉토리에 적절한 대상과 함께 makefile을 생성합니다. `make` 명령을 사용하여 소스 코드를 다시 컴파일하고 `make pkg` 명령을 사용하여 자원 유형 패키지를 다시 생성합니다.

RTR 파일은 *install-directory*/*rt-name*/etc 디렉토리에 있습니다. 표준 텍스트 편집기를 사용하여 RTR 파일을 편집할 수 있습니다. RTR 파일에 대한 자세한 내용은 34 페이지 "자원 및 자원 유형 등록 정보 설정"을 참조하십시오. 등록 정보에 대한 자세한 내용은 부록 A를 참조하십시오.

▼ Agent Builder 명령줄 버전 사용 방법

Agent Builder 명령줄 버전은 그래픽 사용자 인터페이스(GUI)와 동일한 기본 프로세스를 따릅니다. 그러나 GUI에 정보를 입력하는 대신 scdscreate 및 scdsconfig 명령에 인자를 전달합니다. 자세한 내용은 scdscreate(1HA) 및 scdsconfig(1HA) 설명서 페이지를 참조하십시오.

Agent Builder 명령줄 버전을 사용하려면 다음 단계를 수행합니다.

단계

1. **scdscreate를 사용하여 가용성이 높거나 확장 가능한 응용 프로그램을 만들기 위한 Sun Cluster 자원 유형 템플리트를 만듭니다.**

2. **scdsconfigure를 사용하여 scdscreate로 만든 자원 유형 템플리트를 구성합니다.**

   등록 정보 변수를 지정할 수 있습니다. 등록 정보 변수에 대해서는 156 페이지 "등록 정보 변수 사용"에서 설명합니다.

3. **디렉토리를 작업 디렉토리의 pkg 하위 디렉토리로 변경합니다.**

4. **pkgadd 명령을 사용하여 scdscreate로 만든 패키지를 설치합니다.**

   - **영역 환경에서 Solaris 10 OS를 실행 중인 경우 전역 영역의 전역 관리자로 다음 명령을 입력합니다.**

     # **pkgadd -G -d .** *package-name*

     패키지 내용이 비전역 영역과 공유되는 전역 영역에 영향을 주지 않는 경우 지정한 패키지가 전역 영역에 추가됩니다.

   - **Solaris OS의 다른 버전이나 비영역 환경에서 실행 중인 Solaris 10 OS의 경우 다음 명령을 입력합니다.**

     # **pkgadd -d .** *package-name*

5. **(옵션) 생성된 소스 코드를 편집합니다.**

6. **시작 스크립트를 실행합니다.**

Agent Builder에서 만드는 디렉토리 구조

Agent Builder에서 대상 자원 유형에 대해 생성한 모든 파일을 보관하는 디렉토리 구조를 만듭니다. Create 화면에서 작업 디렉토리를 지정합니다. 개발한 추가 자원 유형에 대해 별도의 설치 디렉토리를 지정해야 합니다. Agent Builder는 공급업체 이름과 자원 유형 이름을 결합한 이름의 하위 디렉토리를 작업 디렉토리에 만듭니다. 예를 들어, `SUNW`를 공급업체 이름으로 지정하고 `ftp`라는 자원 유형을 만든 경우 Agent Builder는 작업 디렉토리에 `SUNWftp`라는 디렉토리를 만듭니다.

이 하위 디렉토리에서 Agent Builder는 다음 표에 나열된 디렉토리를 만들어 채웁니다.

디렉토리 이름	내용
`bin`	C 출력의 경우 소스 파일에서 컴파일된 이진 파일을 포함합니다. Korn 쉘 출력의 경우 `src` 디렉토리와 동일한 파일을 포함합니다.
`etc`	RTR 파일을 포함합니다. Agent Builder에서 공급업체 이름과 응용 프로그램 이름을 마침표(.)로 구분해서 연결하여) RTR 파일 이름을 생성합니다. 예를 들어, 공급업체 이름이 `SUNW`이고 자원 유형 이름이 `ftp`일 경우 RTR 파일 이름은 `SUNW.ftp`입니다.
`man`	`start`, `stop` 및 `remove` 유틸리티 스크립트에 대한 사용자 정의 설명서 페이지를 포함합니다. 예를 들어 `startftp`(1M), `stopftp`(1M), `removeftp`(1M) 등이 있습니다. 이 설명서 페이지를 보려면 `man -M` 옵션을 사용하여 경로를 지정합니다. 예를 들면 다음과 같습니다. **`%` `man -M`** *install-directory*`/SUNWftp/man removeftp`
`pkg`	생성된 데이터 서비스를 비롯한 최종 Solaris 패키지를 포함합니다.
`src`	Agent Builder에서 생성한 소스 파일을 포함합니다.
`util`	Agent Builder에서 생성한 `start`, `stop` 및 `remove` 유틸리티 스크립트를 포함합니다. 163 페이지 “Sun Agent Builder에서 만드는 유틸리티 스크립트 및 설명서 페이지”를 참조하십시오. Agent Builder에서 이 스크립트 이름에 응용 프로그램 이름을 추가합니다(예: `startftp`, `stopftp` 및 `removeftp`).

Agent Builder 출력

소스 및 이진 파일

RGM(Resource Group Manager)은 자원 그룹과 궁극적으로 클러스터의 자원을 관리합니다. RGM은 콜백 모델에서 작동합니다. 노드 실패와 같은 특정 이벤트가 발생할 경우 RGM은 영향을 받는 노드에서 실행 중인 각 자원에 대해 자원 유형의 메소드를 호출합니다. 예를 들어, RGM은 영향을 받는 노드에서 실행 중인 자원을 중지하기 위해 `Stop` 메소드를 호출한 다음 자원의 `Start` 메소드를 호출하여 다른 노드에서 자원을 시작합니다. 이 모델에 대한 자세한 내용은 21 페이지 "RGM 모델", 23 페이지 "콜백 메소드" 및 `rt_callbacks`(1HA) 설명서 페이지를 참조하십시오.

이 모델을 지원하기 위해 Agent Builder는 8개의 실행 가능 C 프로그램 또는 Korn 쉘 스크립트를 *install-directory*`/`*rt-name*`/bin` 디렉토리에 생성합니다. 이러한 프로그램 또는 쉘 스크립트는 콜백 메소드 역할을 합니다.

주 – 엄격하게 말하자면 오류 모니터를 구현하는 *rt-name*`_probe` 프로그램은 콜백 프로그램이 아닙니다. RGM은 *rt-name*`_probe`를 직접 호출하지 않고 *rt-name*`_monitor_start` 및 *rt-name*`_monitor_stop`을 호출합니다. 이러한 메소드는 *rt-name* `_probe`를 호출하여 오류 모니터를 시작하고 중지합니다.

Agent Builder에서 생성하는 8개 메소드는 다음과 같습니다.

- *rt-name*`_monitor_check`
- *rt-name*`_monitor_start`
- *rt-name*`_monitor_stop`
- *rt-name*`_probe`
- *rt-name*`_svc_start`
- *rt-name*`_svc_stop`
- *rt-name*`_update`
- *rt-name*`_validate`

각 메소드에 대한 자세한 내용은 `rt_callbacks`(1HA) 설명서 페이지를 참조하십시오.

install-directory`/`*rt-name*`/src` 디렉토리(C 출력)에서 Agent Builder는 다음 파일을 생성합니다.

- 헤더 파일(*rt-name*`.h`)
- 모든 메소드에 공통된 코드를 포함하는 소스 파일(*rt-name*`c`)
- 공통 코드의 객체 파일(*rt-name*`.o`)
- 각 메소드의 소스 파일(`*.c`)
- 각 메소드의 객체 파일(`*.o`)

Agent Builder는 *rt-name*.o 파일을 각 메소드 o 파일에 연결하여 *install-directory*/*rt-name*/bin 디렉토리에 실행 가능 파일을 만듭니다.

Korn 쉘 출력의 경우 *install-directory*/*rt-name*/bin 및 *install-directory*/*rt-name*/src 디렉토리가 동일합니다. 각 디렉토리에는 일곱 개의 콜백 메소드와 Probe 메소드에 해당하는 여덟 개의 실행 스크립트가 포함되어 있습니다.

주 – Korn 쉘 출력에는 시간을 가져오고 검사하기 위해 특정 콜백 메소드에 필요한 2개의 컴파일된 유틸리티 프로그램인 gettime 및 gethostnames가 포함됩니다.

소스 코드를 편집하고 make 명령을 실행하여 코드를 다시 컴파일한 다음 작업을 마치면 make pkg 명령을 실행하여 새 패키지를 생성할 수 있습니다. 소스 코드 변경을 지원하기 위해 Agent Builder는 소스 코드의 올바른 위치에 코드를 추가할 수 있는 주석을 포함합니다. 159 페이지 “생성된 소스 코드 편집”을 참조하십시오.

Sun Agent Builder에서 만드는 유틸리티 스크립트 및 설명서 페이지

자원 유형을 생성하고 클러스터에 해당 패키지를 설치한 경우 일반적으로 관리 명령이나 SunPlex Manager를 사용하여 클러스터에서 실행 중인 자원 유형의 인스턴스(자원)를 가져와야 합니다. 그러나 편의상 Agent Builder는 대상 자원 유형의 자원을 중지 및 제거하기 위한 스크립트뿐만 아니라 이러한 용도의 사용자 정의 유틸리티 스크립트를 생성합니다. *install-directory*/*rt-name*/util 디렉토리에 있는 이 3개의 스크립트는 다음 작업을 수행합니다.

- **시작 스크립트.**자원 유형을 등록하고 필요한 자원 그룹 및 자원을 만듭니다. 또한 응용 프로그램이 네트워크의 클라이언트와 통신할 수 있도록 하는 네트워크 주소 자원(LogicalHostname 또는 SharedAddress)을 만듭니다.
- **중지 스크립트.** 자원을 중지합니다.
- **제거 스크립트.** 시작 스크립트 작업을 실행 취소합니다. 즉, 이 스크립트는 자원, 자원 그룹 및 대상 자원 그룹을 중지하고 제거합니다.

주 – 이러한 스트립트는 내부 규칙을 사용하여 자원 및 자원 그룹의 이름을 지정하기 때문에 해당하는 시작 스크립트에서 시작한 자원에만 제거 스크립트를 사용할 수 있습니다.

Agent Builder는 응용 프로그램 이름을 스크립트 이름에 추가하여 이 스크립트 이름을 지정합니다. 예를 들어, 응용 프로그램 이름이 ftp일 경우 스크립트 이름은 startftp, stopftp 및 removeftp입니다.

Agent Builder는 각 유틸리티 스크립트의 *install-directory*/*rt-name*/man/man1m 디렉토리에 설명서 페이지를 제공합니다. 스크립트에 전달해야 하는 인자가 이 설명서 페이지에 나와 있으므로 스크립트를 시작하기 전에 이 설명서 페이지를 읽어보십시오.

이 설명서 페이지를 보려면 man 명령에 -M 옵션을 사용하여 이 man 디렉토리 경로를 지정합니다. 예를 들어, SUNW가 공급업체이고 ftp가 응용 프로그램 이름일 경우 다음 명령을 입력하여 startftp(1M) 설명서 페이지를 볼 수 있습니다.

```
% man -M install-directory/SUNWftp/man startftp
```

클러스터 관리자도 설명서 페이지 유틸리티 스크립트를 사용할 수 있습니다. Agent Builder에서 생성한 패키지를 클러스터에 설치한 경우 유틸리티 스크립트의 설명서 페이지는 /opt/*rt-name*/man 디렉토리에 있습니다. 예를 들어, 다음 명령을 입력하여 startftp(1M) 설명서 페이지를 볼 수 있습니다.

```
% man -M /opt/SUNWftp/man startftp
```

Agent Builder에서 만드는 지원 파일

Agent Builder는 pkginfo, postinstall, postremove 및 preremove와 같은 지원 파일을 *install-directory*/*rt-name*/etc 디렉토리에 저장합니다. 이 디렉토리에는 대상 자원 유형에 사용 가능한 자원 및 자원 유형 등록 정보를 선언하고 자원을 클러스터에 등록할 때 등록 정보값을 초기화하는 자원 유형 등록(RTR) 파일도 포함됩니다. 자세한 내용은 34 페이지 "자원 및 자원 유형 등록 정보 설정"을 참조하십시오. RTR 파일 이름은 *vendor-name*.*resource-type-name*(예: SUNW.ftp)으로 지정됩니다.

표준 텍스트 편집기를 사용하여 이 파일을 편집하고 소스 코드를 다시 컴파일하지 않고 변경할 수 있습니다. 그러나 make pkg 명령을 사용하여 패키지를 다시 만들어야 합니다.

Agent Builder에서 만드는 패키지 디렉토리

install-directory/*rt-name*/pkg 디렉토리에는 Solaris 패키지가 포함됩니다. 패키지 이름은 공급업체 이름과 응용 프로그램 이름을 결합한 이름(예: SUNWftp)입니다. *install-directory*/*rt-name*/src 디렉토리의 makefile에서는 새 패키지 작성을 지원합니다. 예를 들어, 소스 파일을 변경하고 코드를 다시 컴파일하거나 패키지 유틸리티 스크립트를 변경한 경우 make pkg 명령을 사용하여 새 패키지를 만듭니다.

클러스터에서 패키지를 제거한 경우 둘 이상의 노드에서 동시에 명령을 실행하면 pkgrm 명령이 실패할 수 있습니다. 다음 두 가지 방법 중 하나로 이 문제를 해결할 수 있습니다.

- 노드에서 pkgrm 명령을 실행하기 전에 클러스터의 한 노드에서 remove *rt-name* 스크립트를 실행합니다.
- 필요한 모든 정리 작업을 담당하는 클러스터의 한 노드에서 pkgrm 명령을 실행한 다음 필요한 경우 나머지 노드에서 동시에 pkgrm 명령을 실행합니다.

여러 노드에서 동시에 실행하여 pkgrm이 실패한 경우 한 노드에서 명령을 다시 실행한 다음 나머지 노드에서 실행합니다.

rtconfig 파일

작업 디렉토리에 C 또는 Korn 쉘 소스 코드를 생성하면 Agent Builder에서 `rtconfig`라는 구성 파일을 생성합니다. 이 파일에는 개발자가 Create 화면과 Configure 화면에서 지정한 정보가 포함됩니다. 기존 자원 유형의 작업 디렉토리에서 Agent Builder를 시작하면 Agent Builder는 `rtconfig` 파일을 읽고 개발자가 기존 자원 유형에 대해 제공한 정보를 Create 및 Configure 화면에 채웁니다. File 드롭다운 메뉴에서 Load Resource Type을 선택하여 기존 자원 유형을 로드하는 경우에도 Agent Builder는 이와 유사하게 작동합니다. 기존 자원 유형을 복제할 경우 이 기능이 유용합니다. 158 페이지 "Agent Builder를 사용하여 작성한 코드 재사용"을 참조하십시오.

Agent Builder용 Cluster Agent 모듈

Agent Builder용 Cluster Agent 모듈은 NetBeans™ 모듈입니다. 이 모듈은 Sun Java Studio(이전의 Sun ONE Studio) 제품을 통해 Sun Cluster 소프트웨어용 자원 유형을 만들 수 있도록 하는 GUI를 제공합니다.

주 – Sun Java Studio 설명서에는 Sun Java Studio 제품의 설정, 설치 및 사용 방법에 대한 정보가 들어 있습니다. 이 설명서는 `http://wwws.sun.com/software/sundev/jde/documentation/index.html` 웹 사이트에서 이용할 수 있습니다.

▼ Cluster Agent 모듈 설치 및 설정 방법

Cluster Agent 모듈은 Sun Cluster 소프트웨어를 설치할 때 설치됩니다. Sun Cluster 설치 도구는 Cluster Agent 모듈 파일 `scdsbuilder.jar`을 `/usr/cluster/lib/scdsbuilder`에 저장합니다. Sun Java Studio 소프트웨어와 함께 Cluster Agent 모듈을 사용하려면 이 파일에 대한 심볼릭 링크를 만들어야 합니다.

주 – Cluster Agent 모듈을 실행하려는 시스템에 Sun Cluster, Sun Java Studio 제품 및 Java 1.4가 설치되어 사용할 수 있어야 합니다.

단계 **1. 모든 사용자 또는 개발자 자신만 Cluster Agent 모듈을 사용하도록 허용합니다.**

- 모든 사용자가 사용할 수 있게 하려면 수퍼유저가 되거나 그에 상응하는 역할이 되어 전역 모듈 디렉토리에 심볼릭 링크를 만듭니다.

```
# cd /opt/s1studio/ee/modules
# ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

주 – /opt/s1studio/ee 이외의 디렉토리에 Sun Java Studio 소프트웨어를 설치한 경우 이 디렉토리 경로를 개발자가 사용한 경로로 대체합니다.

- 개발자만 사용할 수 있게 하려면 modules 하위 디렉토리에 심볼릭 링크를 만듭니다.

```
% cd ~your-home-dir/ffjuser40ee/modules
% ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
```

2. Sun Java Studio 소프트웨어를 중지했다가 다시 시작합니다.

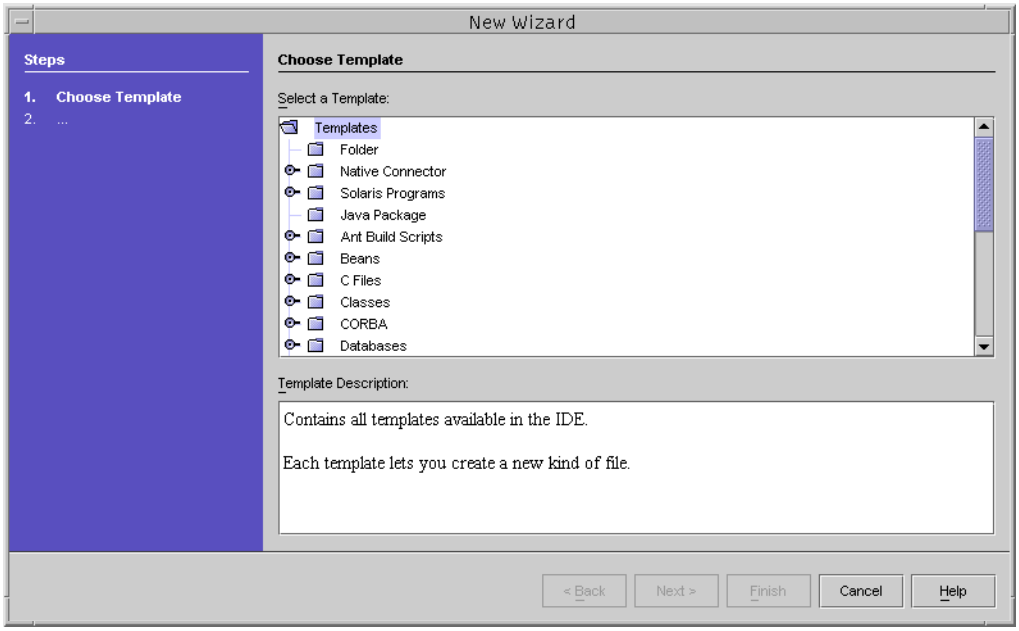
▼ Cluster Agent 모듈 시작 방법

다음은 Sun Java Studio 소프트웨어에서 Cluster Agent 모듈을 시작하는 방법을 설명하는 단계입니다.

주 – Sun Java Studio 설명서에는 Sun Java Studio 제품의 설정, 설치 및 사용 방법에 대한 정보가 들어 있습니다. 이 설명서는 http://wwws.sun.com/software/sundev/jde/documentation/index.html 웹 사이트에서 이용할 수 있습니다.

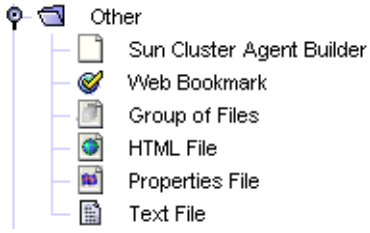
단계 1. Sun Java Studio의 File 메뉴에서 New를 선택하거나 도구 모음에서 다음 아이콘을 누릅니다.

새로 만들기 마법사 화면이 나타납니다.

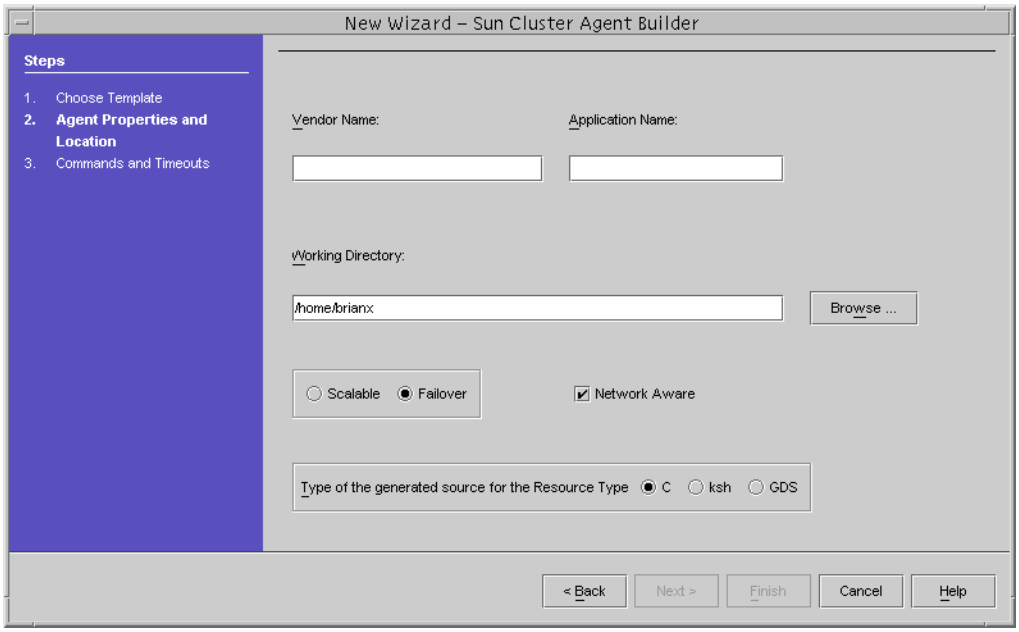


2. **Select a Template 창에서 필요한 경우 아래로 스크롤하여 Other 폴더 옆에 있는 키를 누릅니다.**

Other 폴더가 열립니다.



3. **Other 폴더에서 Sun Cluster Agent Builder를 선택하고 Next를 누릅니다.**
   Sun Java Studio용 Cluster Agent 모듈이 시작됩니다. 첫 번째 새로 만들기 마법사 - Sun Cluster Agent Builder 화면이 나타납니다.



Cluster Agent 모듈 사용

Agent Builder 소프트웨어를 사용하는 것과 마찬가지로 Cluster Agent 모듈을 사용합니다. 인터페이스는 동일합니다. 예를 들어, 다음 그림에서는 Agent Builder 소프트웨어의 Create 화면과 Cluster Agent 모듈의 첫 번째 새로 만들기 마법사 - Sun Cluster Agent Builder 화면에 동일한 필드와 선택 항목이 들어 있다는 것을 보여줍니다.

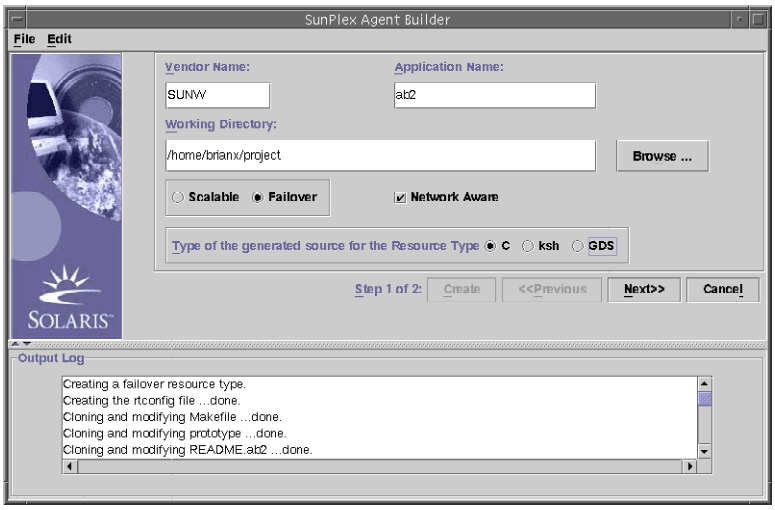


그림 9–4 Agent Builder 소프트웨어의 Create 화면

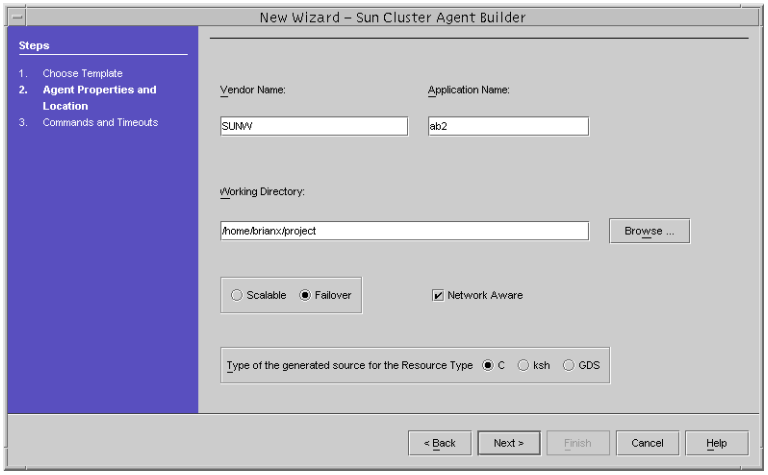


그림 9–5 Cluster Agent 모듈의 새로 만들기 마법사 - Sun Cluster Agent Builder 화면

Cluster Agent 모듈과 Agent Builder의 차이점

Cluster Agent 모듈과 Agent Builder는 여러 가지 면에서 비슷하지만 약간의 차이점이 있습니다.

- Cluster Agent 모듈에서는 두 번째 새로 만들기 마법사 - Sun Cluster Agent Builder 화면에서 종료를 누른 경우에만 자원 유형이 작성 및 구성됩니다. 첫 번째 새로 만들기 마법사 - Sun Cluster Agent Builder 화면에서 다음을 누르면 자원 유형이 만들어지지 **않습니다**.

  Agent Builder의 Create 화면에서 Create를 누르면 즉시 자원 유형이 생성되고 Configure 화면에서 Configure를 누르면 즉시 자원 유형이 구성됩니다.
- Agent Builder의 Output Log 영역에 나타나는 정보는 Sun Java Studio 제품의 별도 창에 표시됩니다.

10장

일반 데이터 서비스

이 장은 일반 데이터 서비스(GDS)에 대한 정보를 제공하며 GDS를 사용하는 서비스를 작성하는 방법에 대해 설명합니다. SunPlex Agent Builder 또는 Sun Cluster 관리 명령을 사용하여 이 서비스를 만듭니다.

이 장은 다음 내용으로 구성되어 있습니다.

일반 데이터 서비스 개념

GDS는 간단한 네트워크 인식 및 네트워크 비인식 응용 프로그램을 Sun Cluster RGM(Resource Group Management) 프레임워크에 연결하여 가용성을 높이거나 확장 가능하게 만드는 기법입니다. 이 기법에서는 일반적인 방법으로 응용 프로그램의 가용성을 높이거나 확장 가능하게 만들 때처럼 데이터 서비스를 코딩할 필요가 없습니다.

GDS는 사전에 컴파일된 하나의 데이터 서비스입니다. 사전 컴파일된 데이터 서비스와 해당 구성 요소, 콜백 메소드(`rt_callbacks`) 구현 및 자원 유형 등록 파일(`rt_reg`)은 수정할 수 없습니다.

이 절은 다음 내용으로 구성되어 있습니다.

- 사전 컴파일된 자원 유형
- GDS 사용의 장점 및 단점
- GDS를 사용하는 서비스를 만드는 방법
- GDS의 이벤트 기록 방법
- 필수 GDS 등록 정보

- 선택적 GDS 등록 정보

사전 컴파일된 자원 유형

일반 데이터 서비스 자원 유형 SUNW.gds는 SUNWscgds 패키지에 포함되어 있습니다. scinstall 유틸리티는 클러스터 설치 중에 이 패키지를 설치합니다. scinstall(1M) 설명서 페이지를 참조하십시오. SUNWscgds 패키지에는 다음과 같은 파일이 포함되어 있습니다.

```
# 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
```

GDS 사용의 장점 및 단점

GDS를 사용할 경우 Agent Builder 소스 코드(scdscreate(1HA) 설명서 페이지 참조) 또는 Sun Cluster 관리 명령을 사용할 때와 비교해서 다음과 같은 장점이 있습니다.

- GDS는 쉽게 사용할 수 있습니다.
- GDS와 GDS의 메소드는 사전에 컴파일되기 때문에 수정할 수 없습니다.
- Agent Builder를 사용하여 응용 프로그램을 위한 스크립트를 생성할 수 있습니다. 이러한 스크립트는 여러 클러스터에서 재사용할 수 있도록 Solaris 패키지로 보관됩니다.

GDS를 사용하면 많은 장점이 있지만 다음과 같은 인스턴스에서는 이 기법을 사용하지 **않는** 것이 좋습니다.

- 확장 등록 정보를 추가하거나 기본값을 변경하는 경우처럼 사전 컴파일된 자원 유형으로 사용할 수 있는 것보다 많은 컨트롤이 필요한 경우
- 특별한 기능을 추가하기 위해 소스 코드를 수정해야 하는 경우

GDS를 사용하는 서비스를 만드는 방법

다음 두 가지 방법으로 GDS를 사용하는 서비스를 만들 수 있습니다.

- Agent Builder
- Sun Cluster 관리 명령

GDS 및 Agent Builder

Agent Builder를 사용하고 생성된 소스 코드의 유형으로 GDS를 선택합니다. 지정된 응용 프로그램을 위한 자원을 구성하는 스크립트 세트를 만드는 데는 사용자 입력이 사용됩니다.

GDS 및 Sun Cluster 관리 명령

이 방법은 SUNWscgds에 있는 사전 컴파일된 데이터 서비스 코드를 사용합니다. 그러나 클러스터 관리자는 Sun Cluster 관리 명령을 사용하여 자원을 만들고 구성해야 합니다. scrgadm(1M) 및 scswitch(1M) 설명서 페이지를 참조하십시오.

GDS 기반 서비스 구성에 사용할 방법 선택

올바른 scrgadm 및 scswitch 명령을 실행하려면 많은 입력이 필요합니다. 예를 들어, 184 페이지 "Sun Cluster 관리 명령을 사용하여 GDS를 사용하는 고가용성 서비스를 만드는 방법" 및 185 페이지 "Sun Cluster 관리 명령을 사용하여 GDS를 사용하는 확장 가능 서비스를 만드는 방법"을 참조하십시오.

Agent Builder를 통해 GDS를 사용하면 scrgadm 및 scswitch 명령을 대신 실행하는 스크립트가 생성되기 때문에 간단하게 처리할 수 있습니다.

GDS에서 이벤트를 기록하는 방법

GDS를 사용하면 GDS에서 GDS가 시작하는 스크립트로 전달되는 관련 정보를 기록할 수 있습니다. 이 정보에는 시작, 검사 및 중지 메소드의 상태와 등록 정보 변수가 포함됩니다. 이 정보를 사용하여 스크립트에서 문제 또는 오류를 진단하거나 다른 목적으로 적용할 수 있습니다.

177 페이지 "Log_level 등록 정보"에 설명된 Log_level 등록 정보를 사용하여 GDS에서 기록할 메시지의 수준 또는 유형을 지정합니다. NONE, INFO 또는 ERR을 지정할 수 있습니다.

GDS 로그 파일

다음 두 개의 GDS 로그 파일은 /var/cluster/logs/DS/*resource-group-name*/*resource-name* 디렉토리에 있습니다.

- start_stop_log.txt에는 자원 시작 및 중지 메소드에 의해 생성된 메시지가 있습니다.
- probe_log.txt에는 자원 모니터에 의해 생성된 메시지가 있습니다.

다음 예에서는 start_stop_log.txt에 포함된 정보의 유형을 보여줍니다.

```
10/20/2005 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
10/20/2005 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

다음 예에서는 probe_log.txt에 포함된 정보의 유형을 보여줍니다.

```
10/20/2005 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
10/20/2005 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2005 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

필수 GDS 등록 정보

응용 프로그램이 네트워크 비인식 응용 프로그램인 경우 Start_command 확장 등록 정보와 Port_list 등록 정보를 모두 제공해야 합니다. 응용 프로그램이 네트워크 인식 응용 프로그램인 경우 Start_command 확장 등록 정보만 제공해야 합니다.

Start_command 확장 등록 정보

Start_command 확장 등록 정보에서 지정한 시작 명령이 응용 프로그램을 시작합니다. 이 명령은 쉘에 직접 전달되어 응용 프로그램을 시작할 수 있도록 인자가 지정된 UNIX 명령이어야 합니다.

Port_list 등록 정보

Port_list 등록 정보는 응용 프로그램이 수신하는 포트 목록을 식별합니다. Port_list 등록 정보는 Agent Builder에 의해 작성된 시작 스크립트에서 지정하거나 Sun Cluster 관리 명령을 사용하는 경우 scrgadm 명령으로 지정해야 합니다.

선택적 GDS 등록 정보

선택적 GDS 등록 정보는 **시스템 정의 등록 정보**와 **확장 등록 정보**를 모두 포함합니다. 시스템 정의 등록 정보는 Sun Cluster에서 제공하는 표준 등록 정보 세트입니다. RTR 파일에 정의된 등록 정보를 확장 등록 정보라고 합니다. 선택적 GDS 등록 정보는 다음과 같습니다.

- Network_resources_used 등록 정보
- Stop_command 확장 등록 정보
- Probe_command 확장 등록 정보
- Start_timeout 등록 정보
- Stop_timeout 등록 정보
- Probe_timeout 확장 등록 정보
- Child_mon_level 확장 등록 정보(관리 명령에만 사용됨)
- Failover_enabled 확장 등록 정보

- Stop_signal 확장 등록 정보
- Log_level 확장 등록 정보

Network_resources_used 등록 정보

이 등록 정보의 기본값은 null입니다. 응용 프로그램이 하나 이상의 주소에 바인드되어야 하는 경우에는 이 등록 정보를 지정해야 합니다. 이 등록 정보가 생략되거나 Null로 지정되면 응용 프로그램이 모든 주소에서 수신하는 것으로 처리됩니다.

GDS 자원을 구성하기 전에 LogicalHostname 또는 SharedAddress 자원이 이미 구성되어 있어야 합니다. LogicalHostname 또는 SharedAddress 자원을 구성하는 방법은 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**를 참조하십시오.

값을 지정하려면 자원 이름을 하나 이상 지정하십시오. 각 자원 이름은 한 개 이상의 LogicalHostname 자원이나 한 개 이상의 SharedAddress 자원을 포함할 수 있습니다. 자세한 내용은 r_properties(5) 설명서 페이지를 참조하십시오.

Stop_command 등록 정보

중지 명령은 응용 프로그램을 중지하고 응용 프로그램이 완전히 중지된 후에 반환되어야 합니다. 이 명령은 쉘에 직접 전달되어 응용 프로그램을 중지할 수 있는 전체 UNIX 명령이어야 합니다.

Stop_command 확장 등록 정보를 입력하면 GDS 중지 메소드는 중지 시간 초과값의 80% 상태에서 중지 명령을 시작합니다. 중지 명령의 시작 결과와 관계없이 GDS 중지 메소드는 중지 시간 초과값의 15% 상태에서 SIGKILL을 전송합니다. 남은 5%의 시간은 작업 관리 오버헤드를 위해 예약됩니다.

중지 명령을 생략하면 GDS가 Stop_signal에 지정된 신호를 사용하여 응용 프로그램을 중지합니다.

Probe_command 등록 정보

지정된 응용 프로그램의 상태를 검사 명령으로 정기적으로 검사합니다. 이 명령은 쉘에 직접 전달되어 응용 프로그램을 검사할 수 있도록 인자가 지정된 UNIX 명령이어야 합니다. 검사 명령을 실행하면 응용 프로그램이 올바로 실행 중인 경우 종료 상태 0을 반환합니다.

검사 명령의 종료 상태는 응용 프로그램의 오류 심각도를 확인하는 데 사용됩니다. **검사 상태**라고 불리는 이 종료 상태는 0(성공)과 100(완전한 실패) 사이의 정수여야 합니다. 검사 상태는 특수 값 201일 수도 있습니다. 이 상태에서는 Failover_enabled가 FALSE로 설정되어 있지 않은 경우 응용 프로그램이 즉시 페일오버됩니다. GDS 검사 알고리즘은 검사 상태를 사용하여 응용 프로그램을 로컬로 재시작할 것인지 아니면 페일오버할 것인지를 결정합니다. 자세한 내용은 scds_fm_action(3HA) 설명서 페이지를 참조하십시오. 종료 상태가 201이면 응용 프로그램이 즉시 페일오버됩니다.

중지 명령을 생략하면 GDS가 간단한 자체 검사를 제공합니다. 이 검사는 `Network_resources_used` 등록 정보나 `scds_get_netaddr_list()` 함수 결과로부터 파생된 IP 주소 집합에서 응용 프로그램에 연결합니다. 자세한 내용은 `scds_get_netaddr_list`(3HA) 설명서 페이지를 참조하십시오. 연결에 성공하면 즉시 연결을 끊습니다. 연결과 연결 끊기가 모두 성공하면 응용 프로그램이 안정적으로 실행되고 있다고 판단할 수 있습니다.

주 – GDS와 함께 제공되는 검사는 전체 기능을 제공하는 응용 프로그램 특정 검사의 간단한 대안으로만 사용됩니다.

`Start_timeout` 등록 정보

이 등록 정보는 시작 명령에 대한 시작 시간 초과를 지정합니다. 자세한 내용은 174 페이지 "`Start_command` 확장 등록 정보"를 참조하십시오. `Start_timeout`의 기본값은 300초입니다.

`Stop_timeout` 등록 정보

이 등록 정보는 중지 명령에 대한 중지 시간 초과를 지정합니다. 자세한 내용은 175 페이지 "`Stop_command` 등록 정보"를 참조하십시오. `Stop_timeout`의 기본값은 300초입니다.

`Probe_timeout` 등록 정보

이 등록 정보는 검사 명령에 대한 시간 초과 값을 지정합니다. 자세한 내용은 175 페이지 "`Probe_command` 등록 정보"를 참조하십시오. `Probe_timeout`의 기본값은 30초입니다.

`Child_mon_level` 등록 정보

주 – Sun Cluster 관리 명령을 사용하는 경우 `Child_mon_level` 등록 정보를 사용할 수 있습니다. Agent Builder를 사용하는 경우에는 이 등록 정보를 사용할 수 없습니다.

이 등록 정보는 PMF(Process Monitor Facility)를 통해 모니터되는 프로세스를 제어합니다. 이 등록 정보는 분기된 자식 프로세스에 대한 모니터 범위를 나타냅니다. 이 등록 정보는 `pmfadm` 명령의 `-C` 인자처럼 작동합니다. `pmfadm`(1M) 설명서 페이지를 참조하십시오.

이 등록 정보를 생략하거나 기본값 `-1`로 설정하면 `pmfadm` 명령에서 `-C` 옵션을 생략한 것과 동일한 기능을 합니다. 즉, 모든 자식과 하위 프로세스가 모니터됩니다.

Failover_enabled 등록 정보

이 부울 확장 등록 정보는 자원의 페일오버 동작을 제어합니다. 이 확장 등록 정보를 TRUE로 설정하면 Retry_interval에 지정된 시간(초) 내에 재시작 횟수가 Retry_count를 초과할 경우 응용 프로그램이 페일오버됩니다.

이 등록 정보를 FALSE로 설정하면 Retry_interval 시간(초) 내에 재시작 횟수가 Retry_count를 초과해도 응용 프로그램이 재시작되거나 다른 노드로 페일오버되지 않습니다.

이 등록 정보를 사용하여 응용 프로그램 자원이 자원 그룹의 페일오버를 시작하는 것을 방지할 수 있습니다. 이 등록 정보의 기본값은 TRUE입니다.

Stop_signal 등록 정보

GDS는 이 정수 확장 등록 정보 값을 사용하여 PMF를 통해 응용 프로그램을 중지하는 데 사용된 신호를 확인합니다. 지정할 수 있는 정수 값 목록은 signal(3HEAD) 설명서 페이지를 참조하십시오. 기본값은 15(SIGTERM)입니다.

Log_level 등록 정보

이 등록 정보는 GDS가 기록하는 진단 메시지의 수준 또는 유형을 지정합니다. 이 등록 정보에 대해 NONE, INFO 또는 ERR을 지정할 수 있습니다. NONE을 지정하면 GDS가 진단 메시지를 기록하지 않습니다. INFO를 지정하면 정보 메시지만 기록됩니다. ERR을 지정하면 오류 메시지만 기록됩니다. 기본적으로 GDS는 진단 메시지를 기록하지 않습니다(NONE).

Agent Builder를 사용하여 GDS를 사용하는 서비스 만들기

Agent Builder를 사용하여 GDS를 사용하는 서비스를 만들 수 있습니다. Agent Builder에 대한 자세한 내용은 9 장를 참조하십시오.

GDS 기반 스크립트 만들기 및 구성

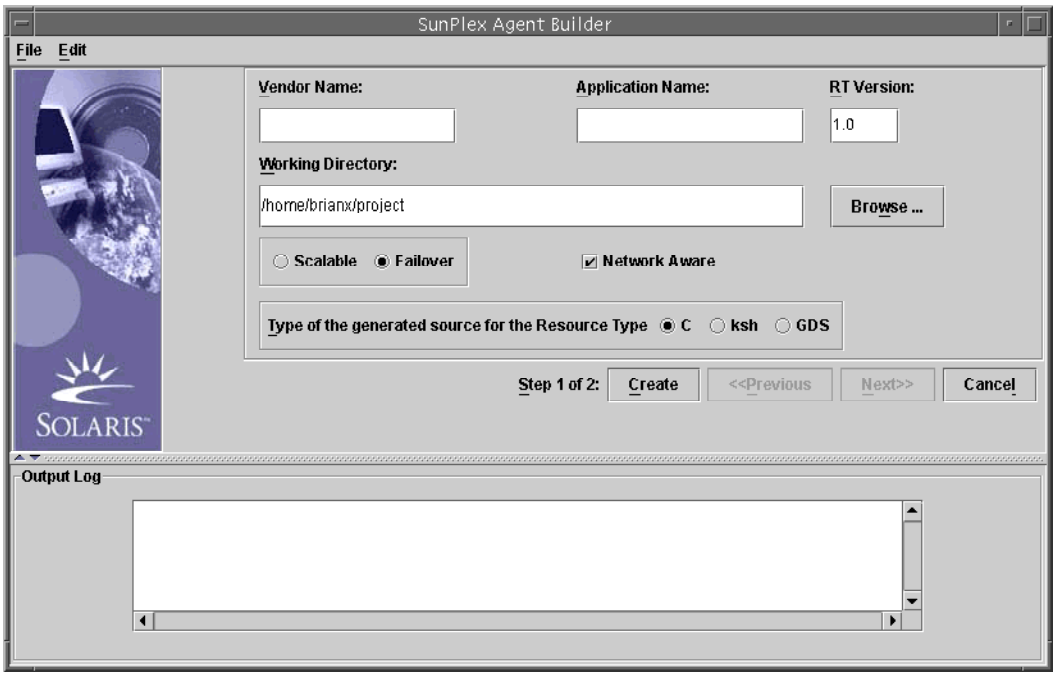
▼ Agent Builder를 시작하고 스크립트를 만드는 방법

단계 1. **수퍼유저가 되거나 동등한 역할을 맡습니다.**

2. **Agent Builder를 시작합니다.**

```
# /usr/cluster/bin/scdsbuilder
```

3. **Agent Builder Create 화면이 나타납니다.**



4. **Vendor Name을 입력합니다.**

5. **Application Name을 입력합니다.**

주 – Solaris 9 운영 체제부터는 Vendor Name과 Application Name의 조합이 9자를 넘을 수 있습니다. 그러나 Solaris 운영 체제의 이전 버전을 사용 중이면 Vendor Name과 Application Name의 조합이 9자를 넘으면 안 됩니다. 이 조합은 패키지 이름으로 스크립트에 사용됩니다.

6. **작업 디렉토리로 이동합니다.**

   경로를 입력하는 대신 Browse 드롭다운 메뉴를 사용하여 디렉토리를 선택할 수 있습니다.

7. **데이터 서비스가 확장 가능인지 아니면 페일오버인지 선택합니다.**

   Network Aware는 GDS를 만들 때 기본값이기 때문에 선택하지 않아도 됩니다.

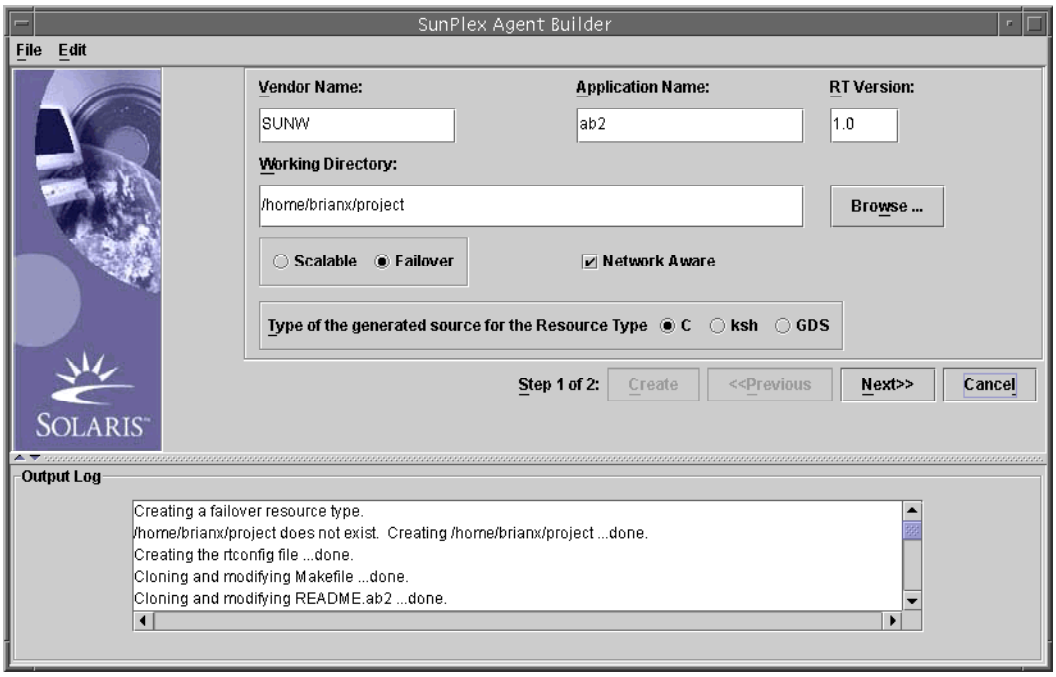
8. **GDS를 선택합니다.**

9. **(옵션) RT Version을 표시된 기본값과 다르게 변경합니다.**

주 – RT 버전 필드에는 공백, 탭, 슬래시(/), 백슬래시(\), 별표(*), 물음표(?), 쉼표(,), 세미콜론(;), 여는 대괄호([) 또는 닫는 대괄호(]) 문자를 사용할 수 없습니다.

10. **만들기를 누릅니다.**

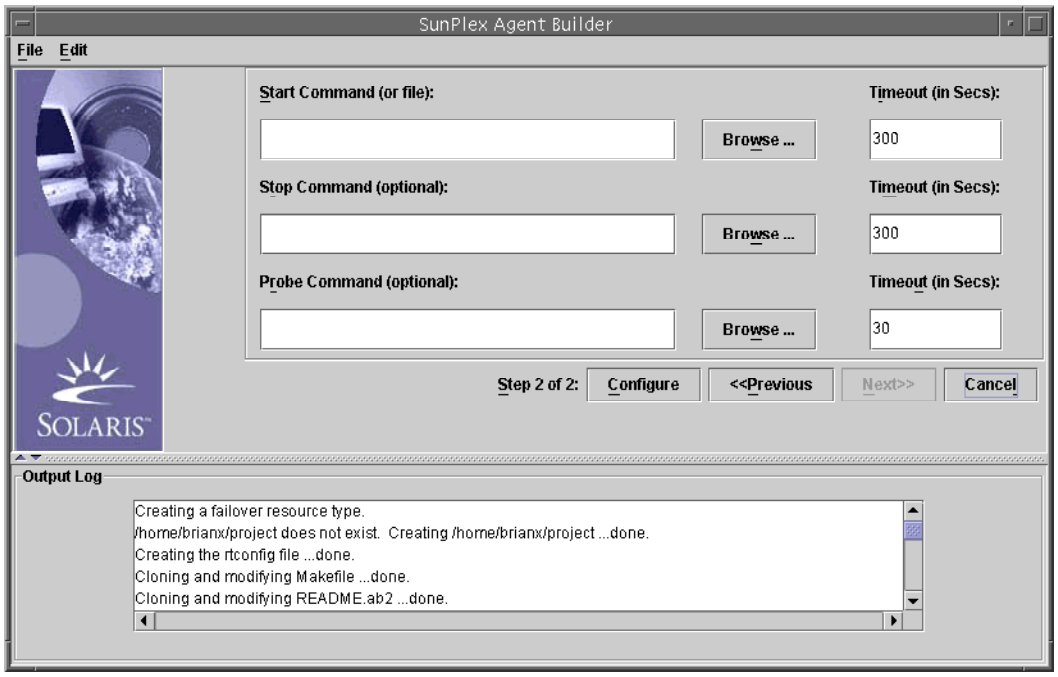
    Agent Builder가 스크립트를 만듭니다. Output Log 영역에 결과가 표시됩니다.



Create 버튼이 비활성화됩니다. 스크립트를 구성할 수 있습니다.

11. 다음을 누릅니다.

Configure 화면이 나타납니다.



▼ 스크립트 구성 방법

스크립트를 만든 후 새 서비스를 구성해야 합니다.

단계

1. **시작 명령의 위치를 입력하거나 Browse를 눌러 시작 명령을 찾습니다.**
   등록 정보 변수를 지정할 수 있습니다. 등록 정보 변수에 대한 자세한 내용은 156 페이지 "등록 정보 변수 사용"을 참조하십시오.

2. **(옵션) 중지 명령의 위치를 입력하거나 Browse를 눌러 중지 명령을 찾습니다.**
   등록 정보 변수를 지정할 수 있습니다. 등록 정보 변수에 대한 자세한 내용은 156 페이지 "등록 정보 변수 사용"을 참조하십시오.

3. **(옵션) 검사 명령의 위치를 입력하거나 Browse를 눌러 검사 명령을 찾습니다.**
   등록 정보 변수를 지정할 수 있습니다. 등록 정보 변수에 대한 자세한 내용은 156 페이지 "등록 정보 변수 사용"을 참조하십시오.

4. **(옵션) 시작, 중지 및 검사 명령에 대한 시간 초과값을 지정합니다.**

5. **Configure를 누릅니다.**
   Agent Builder가 스크립트를 구성합니다.

주 – Agent Builder는 공급업체 이름과 응용 프로그램 이름을 연결하여 패키지 이름을 만듭니다.

스크립트의 패키지는 생성되어 다음 디렉토리에 저장됩니다:

working-dir/*vendor-name-application*/pkg

예를 들어, /export/wdir/NETapp/pkg와 같은 위치가 됩니다.

6. **클러스터의 각 노드에서 수퍼유저가 되거나 이와 대등한 역할을 사용합니다.**

7. **완성된 패키지를 클러스터의 각 노드에 설치합니다.**

   - **영역 환경에서 Solaris 10 OS를 사용하는 경우 전역 영역의 전역 관리자로 다음 명령을 입력합니다.**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -G -d . NETapp
     ```

     패키지 내용이 비전역 영역과 공유되는 전역 영역의 어떤 부분에도 영향을 주지 않을 경우 지정한 패키지가 전역 영역에 추가됩니다.

     pkgadd가 설치하는 파일은 다음과 같습니다.

     ```
     /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
     ```

   - **Solaris OS의 다른 모든 버전이나 비영역 환경에서 Solaris 10 OS를 사용하는 경우 다음 명령을 입력합니다.**

     ```
     # cd /export/wdir/NETapp/pkg
     # pkgadd -d . NETapp
     ```

     pkgadd가 설치하는 파일은 다음과 같습니다.

     ```
     /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
```

주 – 설명서 페이지와 스크립트 이름은 이전에 Create 화면에서 입력한 응용 프로그램 이름을 스크립트 이름 뒤에 추가한 것과 같습니다(예 `startapp`).

8. **클러스터의 한 노드에서 자원을 구성하고 응용 프로그램을 시작합니다.**

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

`startapp` 스크립트의 인자는 자원의 유형에 따라 달라집니다. (failover 또는 scalable).

주 – 입력해야 하는 명령줄을 확인하려면 사용자 정의 설명서 페이지를 확인하거나 인자 없이 `startapp` 스크립트를 실행하여 사용 방법을 표시하십시오.

설명서 페이지를 보려면 설명서 페이지에 대한 경로를 지정해야 합니다. 예를 들어, `startapp`(1M) 설명서 페이지를 보려면 다음을 입력합니다.

```
# man -M /opt/NETapp/man startapp
```

사용 방법을 표시하려면 다음을 입력합니다.

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be
 specified. For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup-adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

Agent Builder의 출력

Agent Builder는 패키지를 만들 때 제공한 입력을 기반으로 세 개의 스크립트와 하나의 구성 파일을 생성합니다. 구성 파일은 자원 그룹과 자원 유형의 이름을 지정합니다.

스크립트는 다음과 같습니다.

- **시작 스크립트.** 자원을 구성하고 RGM의 제어를 받는 응용 프로그램을 시작합니다.
- **중지 스크립트.** 응용 프로그램을 중지하고 자원과 자원 그룹의 사용을 해제합니다.

- **제거 스크립트.** 시작 스크립트에서 만든 자원과 자원 그룹을 제거합니다.

이 스크립트는 Agent Builder가 GDS 기반이 아닌 데이터 서비스를 위해 생성한 유틸리티 스크립트와 인터페이스 및 동작이 같습니다. 스크립트는 여러 클러스터에서 재사용할 수 있도록 Solaris 패키지로 보관됩니다.

일반적으로 `scrgadm` 명령에 대한 입력으로 지정되는 자원 그룹이나 기타 인자의 이름을 원하는 대로 지정하여 구성 파일을 사용자 정의할 수 있습니다. 스크립트를 사용자 정의하지 않으면 Agent Builder가 `scrgadm` 인자에 기본값을 제공합니다.

Sun Cluster 관리 명령을 사용하여 GDS를 사용하는 서비스 만들기

이 절에서는 GDS에 인자를 입력하는 방법을 설명합니다. 기존 Sun Cluster 관리 명령(예: `scrgadm` 및 `scswitch`)을 사용하여 GDS를 유지 관리합니다.

스크립트에서 해당 기능을 제공하는 경우 이 절에 있는 저급 관리 명령을 사용할 필요가 없습니다. 그러나 GDS 기반 자원에 대한 세부적인 제어가 필요한 경우에는 저급 관리 명령을 사용할 수 있습니다. 이러한 명령은 스크립트에 의해 실행됩니다.

▼ Sun Cluster 관리 명령을 사용하여 GDS를 사용하는 고가용성 서비스를 만드는 방법

단계 **1. 수퍼유저가 되거나 동등한 역할을 맡습니다.**

2. 자원 유형 `SUNW.gds`를 등록하십시오.

```
# scrgadm -a -t SUNW.gds
```

3. `LogicalHostname` 자원과 페일오버 서비스 자체를 포함하는 자원 그룹을 만듭니다.

```
# scrgadm -a -g haapp_rg
```

4. `LogicalHostname` 자원을 위한 자원을 만듭니다.

```
# scrgadm -a -L -g haapp_rs -l hhead
```

5. 페일오버 서비스 자체를 위한 자원을 만듭니다.

```
# scrgadm -a -j haapp_rs -g haapp_rg -t SUNW.gds \
        -y Scalable=false -y Start_timeout=120 \
        -y Stop_timeout=120 -x Probe_timeout=120 \
```

```
-y Port_list="2222/tcp" \
-x Start_command="/export/ha/appctl/start" \
-x Stop_command="/export/ha/appctl/stop" \
-x Probe_command="/export/app/bin/probe" \
-x Child_mon_level=0 -y Network_resources_used=hhead \
-x Failover_enabled=TRUE -x Stop_signal=9
```

6. 자원 그룹 **haapp_rg**를 온라인 상태로 전환합니다.

```
# scswitch -Z -g haapp_rg
```

▼ Sun Cluster 관리 명령을 사용하여 GDS를 사용하는 확장 가능 서비스를 만드는 방법

단계 1. 수퍼유저가 되거나 동등한 역할을 맡습니다.

2. 자원 유형 **SUNW.gds**를 등록하십시오.

```
# scrgadm -a -t SUNW.gds
```

3. **SharedAddress** 자원을 위한 자원 그룹을 만듭니다.

```
# scrgadm -a -g sa_rg
```

4. **sa_rg**에 **SharedAddress** 자원을 만듭니다.

```
# scrgadm -a -S -g sa_rg -l hhead
```

5. 확장 가능 서비스를 위한 자원 그룹을 만듭니다.

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

6. 확장 가능한 서비스를 위한 자원을 만듭니다.

```
# scrgadm -a -j app_rs -g app_rg -t SUNW.gds \
       -y Scalable=TRUE -y Start_timeout=120 \
       -y Stop_timeout=120 -x Probe_timeout=120 \
       -y Port_list="2222/tcp" \
       -x Start_command="/export/app/bin/start" \
       -x Stop_command="/export/app/bin/stop" \
       -x Probe_command="/export/app/bin/probe" \
       -x Child_mon_level=0 -y Network_resource_used=hhead \
       -x Failover_enabled=TRUE -x Stop_signal=9
```

7. 네트워크 자원이 포함된 자원 그룹을 온라인 상태로 전환합니다.

```
 # scswitch -Z -g sa_rg
```

8. 자원 그룹 **app_rg**를 온라인 상태로 전환합니다.

```
# scswitch -Z -g app_rg
```

Agent Builder에 대한 명령줄 인터페이스

Agent Builder는 GUI와 동일한 기능을 제공하는 명령줄 인터페이스를 통합합니다. 이 인터페이스는 `scdscreate` 및 `scdsconfig` 명령으로 구성됩니다. `scdscreate`(1HA) 및 `scdsconfig`(1HA) 설명서 페이지를 참조하십시오.

▼ Agent Builder의 명령줄 버전을 사용하여 GDS를 사용하는 서비스를 만드는 방법

이 절에서는 명령줄 인터페이스를 사용하여 177 페이지 "Agent Builder를 사용하여 GDS를 사용하는 서비스 만들기"에 표시된 것과 동일한 단계를 수행하는 방법을 설명합니다.

단계 **1. 수퍼유저가 되거나 동등한 역할을 맡습니다.**

2. 서비스를 만듭니다.

- 페일오버 서비스의 경우에는 다음을 입력합니다.

  # scdscreate -g -V NET -T app -d/export/*wdir*

- 확장 가능 서비스의 경우에는 다음을 입력합니다.

  # scdscreate -g -s -V NET -T app -d/export/*wdir*

주 – `-d` 인자는 선택 사항입니다. 이 인자를 지정하지 않으면 현재 디렉토리가 작업 디렉토리가 됩니다.

---

3. 서비스를 구성합니다.

```
# scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" \
-m "/export/app/bin/probe" -d /export/wdir
```

등록 정보 변수를 지정할 수 있습니다. 등록 정보 변수에 대한 자세한 내용은 156 페이지 "등록 정보 변수 사용"을 참조하십시오.

주 – `start` 명령만 필수입니다. 다른 옵션과 인자는 모두 선택 사항입니다.

---

4. 완성된 패키지를 클러스터의 각 노드에 설치합니다.

- **영역 환경에서 Solaris 10 OS를 사용하는 경우 전역 영역의 전역 관리자로 다음 명령을 입력합니다.**

  ```
  # cd /export/wdir/NETapp/pkg
  # pkgadd -G -d . NETapp
  ```

  패키지 내용이 비전역 영역과 공유되는 전역 영역의 어떤 부분에도 영향을 주지 않을 경우 지정한 패키지가 전역 영역에 추가됩니다.

  pkgadd가 설치하는 파일은 다음과 같습니다.

  ```
  /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
  ```

- **Solaris OS의 다른 모든 버전이나 비영역 환경에서 Solaris 10 OS를 사용하는 경우 다음 명령을 입력합니다.**

  ```
  # cd /export/wdir/NETapp/pkg
  # pkgadd -d . NETapp
  ```

  pkgadd가 설치하는 파일은 다음과 같습니다.

  ```
  /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
  ```

주 – 설명서 페이지와 스크립트 이름은 이전에 Create 화면에서 입력한 응용 프로그램 이름을 스크립트 이름 뒤에 추가한 것과 같습니다(예 `startapp`).

---

5. 클러스터의 한 노드에서 자원을 구성하고 응용 프로그램을 시작합니다.

```
# /opt/NETapp/util/startapp -h logicalhostname -p port-and-protocol-list
```

`startapp` 스크립트의 인자는 자원 유형에 따라 달라집니다. (failover 또는 scalable).

주 – 입력해야 하는 명령줄을 확인하려면 사용자 정의 설명서 페이지를 확인하거나 인자 없이 `startapp` 스크립트를 실행하여 사용 방법을 표시하십시오.

설명서 페이지를 보려면 설명서 페이지에 대한 경로를 지정해야 합니다. 예를 들어, `startapp`(1M) 설명서 페이지를 보려면 다음을 입력합니다.

```
# man -M /opt/NETapp/man startapp
```

사용 방법을 표시하려면 다음을 입력합니다.

```
# /opt/NETapp/util/startapp
 The resource name of LogicalHostname or SharedAddress must be specified.
 For failover services:
 Usage: startapp -h logicalhostname
           -p port-and-protocol-list
           [-n ipmpgroup/adapter-list]
 For scalable services:
 Usage: startapp -h shared-address-name
           -p port-and-protocol-list
           [-l load-balancing-policy]
           [-n ipmpgroup/adapter-list]
           [-w load-balancing-weights]
```

11장

DSDL API 함수

이 장에서는 데이터 서비스 개발 라이브러리(DSDL) API 함수를 나열하고 이에 대해 간략하게 설명합니다. 각 DSDL 함수에 대한 전체 설명은 개별 3HA 설명서 페이지에 나와 있습니다. DSDL은 C 인터페이스만 제공하며 스크립트 기반 DSDL 인터페이스는 사용할 수 없습니다.

이 장은 다음 내용으로 구성되어 있습니다.

일반적 용도의 함수

이 절의 함수는 다양한 기능을 제공합니다. 이러한 함수를 사용하여 다음 작업을 수행할 수 있습니다.

- DSDL 환경 초기화
- 자원, 자원 유형 및 자원 그룹 이름과 확장 등록 정보 값 검색
- 자원 그룹 페일오버와 재시작 및 자원 재시작
- 오류 문자열을 오류 메시지로 변환
- 시간 초과 이전에 명령 실행

초기화 함수

다음 함수는 호출 메소드를 초기화합니다.

- `scds_initialize`(3HA) – 자원을 할당하고 DSDL 환경을 초기화합니다.

- scds_close(3HA) – scds_initialize()가 할당한 자원을 비웁니다.

검색 함수

다음 함수는 자원, 자원 유형, 자원 그룹 및 확장 등록 정보에 대한 정보를 검색합니다.

- scds_get_resource_type_name(3HA) – 호출 프로그램에 대한 자원 유형 이름을 검색합니다.
- scds_get_resource_name(3HA) – 호출 프로그램에 대한 자원 이름을 검색합니다.
- scds_get_resource_group_name(3HA) – 호출 프로그램에 대한 자원 그룹 이름을 검색합니다.
- scds_get_ext_property(3HA) – 지정한 확장 등록 정보의 값을 검색합니다.
- scds_free_ext_property(3HA) – scds_get_ext_property()가 할당한 메모리를 비웁니다.

다음 함수는 자원에 사용되는 SUNW.HAStoragePlus 자원에 대한 상태 정보를 검색합니다.

scds_hasp_check(3HA) – 자원에 사용되는 SUNW.HAStoragePlus 자원에 대한 상태 정보를 검색합니다. 자원에 대하여 정의된 Resource_dependencies 또는 Resource_dependencies_weak 시스템 등록 정보를 사용하여 자원이 의존하는 모든 SUNW.HAStoragePlus 자원의 상태(온라인 또는 오프라인)로부터 이 정보를 얻습니다. 자세한 내용은 SUNW.HAStoragePlus(5) 설명서 페이지를 참조하십시오.

페일오버 및 재시작 함수

다음 함수는 자원이나 자원 그룹을 페일오버하거나 재시작합니다.

- scds_failover_rg(3HA) – 자원 그룹을 페일오버합니다.
- scds_restart_rg(3HA) – 자원 그룹을 재시작합니다.
- scds_restart_resource(3HA) – 자원을 재시작합니다.

실행 함수

다음 함수는 시간 초과 시 명령을 실행하고 오류 코드를 오류 메시지로 변환합니다.

- scds_timerun(3HA) – 시간 초과 값에 도달하면 명령을 실행합니다.
- scds_error_string(3HA) – 오류 코드를 오류 문자열로 변환합니다.

등록 정보 함수

이러한 함수는 일반적으로 사용되는 일부 확장 등록 정보를 비롯하여 관련 자원, 자원 그룹 및 자원 유형의 특정 등록 정보에 액세스하기 위한 일반 API를 제공합니다. DSDL은 명령줄 인자를 구문 분석하는 `scds_initialize()` 함수를 제공합니다. 그런 다음 DSDL은 관련 자원 유형, 자원 및 자원 그룹의 다양한 등록 정보를 **캐싱**합니다.

`scds_property_functions`(3HA) 설명서 페이지에서는 다음을 포함하여 이러한 함수에 대해 설명합니다.

- `scds_get_rt_`*property-name*
- `scds_get_rs_`*property-name*
- `scds_get_rg_`*property-name*
- `scds_get_ext_`*property-name*

네트워크 자원 액세스 함수

이 절에 나열된 함수는 자원 및 자원 그룹에서 사용하는 네트워크 자원을 검색, 인쇄 및 해제합니다. 이 절의 `scds_get_` 함수는 RMAPI 함수를 사용하여 `Network_resources_used` 및 `Port_list`와 같은 특정 등록 정보를 쿼리하지 않고 네트워크 자원을 검색하는 편리한 방법을 제공합니다. `scds_print_`*name*`()` 함수는 `scds_get_`*name* `()` 함수가 반환하는 데이터 구조의 값을 인쇄합니다. `scds_free_`*name* `()` 함수는 `scds_get_` *name*`()` 함수에서 할당한 메모리를 비웁니다.

호스트 이름 함수

호스트 이름을 처리하는 함수는 다음과 같습니다.

- `scds_get_rs_hostnames`(3HA) – 자원에 사용되는 호스트 이름 목록을 검색합니다.
- `scds_get_rg_hostnames`(3HA) – 자원 그룹의 네트워크 자원에 사용되는 호스트 이름 목록을 검색합니다.
- `scds_print_net_list`(3HA) – `scds_get_rs_hostnames ()` 또는 `scds_get_rg_hostnames()`가 반환하는 호스트 이름 목록의 내용을 인쇄합니다.
- `scds_free_net_list`(3HA) – `scds_get_rs_hostnames()` 또는 `scds_get_rg_hostnames()`에서 할당한 메모리를 비웁니다.

포트 목록 함수

포트 목록을 처리하는 함수는 다음과 같습니다.

- `scds_get_port_list`(3HA) – 자원에 사용되는 포트-프로토콜 쌍의 목록을 검색합니다.
- `scds_print_port_list`(3HA) – `scds_get_port_list()`가 반환하는 포트-프로토콜 쌍 목록의 내용을 인쇄합니다.
- `scds_free_port_list`(3HA) – `scds_get_port_list()`에서 할당한 메모리를 비웁니다.

네트워크 주소 함수

네트워크 주소를 처리하는 함수는 다음과 같습니다.

- `scds_get_netaddr_list`(3HA) – 자원에 사용되는 네트워크 주소 목록을 검색합니다.
- `scds_print_netaddr_list`(3HA) – `scds_get_netaddr_list()`가 반환하는 네트워크 주소 목록의 내용을 인쇄합니다.
- `scds_free_netaddr_list`(3HA) – `scds_get_netaddr_list()`에서 할당한 메모리를 비웁니다.

TCP 연결을 사용한 오류 모니터

이 절의 함수는 TCP 기반 모니터를 사용 가능하게 합니다. 일반적으로 오류 모니터는 이러한 함수를 사용하여 서비스에 대한 간단한 소켓 연결을 설정하고 상태를 확인하기 위해 데이터를 읽어 서비스에 기록한 다음 서비스와의 연결을 끊습니다.

함수는 다음과 같습니다.

- `scds_fm_tcp_connect`(3HA) – IPv4 주소 지정만 사용하는 프로세스에 TCP 연결을 설정합니다.
- `scds_fm_net_connect`(3HA) – IPv4 또는 IPv6 주소 지정을 사용하는 프로세스에 TCP 연결을 설정합니다.
- `scds_fm_tcp_read`(3HA) – TCP 연결을 사용하여 모니터 중인 프로세스에서 데이터를 읽습니다.
- `scds_fm_tcp_write`(3HA) – TCP 연결을 사용하여 모니터 중인 프로세스에 데이터를 씁니다.
- `scds_simple_probe`(3HA) – 프로세스에 대한 TCP 연결을 설정하고 종료하여 프로세스를 검사합니다. 이 함수는 IPv4 주소만 처리합니다.
- `scds_simple_net_probe`(3HA) – 프로세스에 대한 TCP 연결을 설정하고 종료하여 프로세스를 검사합니다. 이 함수는 IPv4 또는 IPv6 주소를 처리합니다.
- `scds_fm_tcp_disconnect`(3HA) – 모니터 중인 프로세스에 대한 연결을 종료합니다. 이 함수는 IPv4 주소만 처리합니다.

- scds_fm_net_disconnect(3HA) – 모니터 중인 프로세스에 대한 연결을 종료합니다. 이 함수는 IPv4 또는 IPv6 주소를 처리합니다.

PMF 함수

이러한 함수는 PMF(Process Monitor Facility) 기능을 캡슐화합니다. PMF를 통해 모니터하기 위한 DSDL 모델은 pmfadm에 대해 암시적 *tag* 값을 만들어 사용합니다. 자세한 내용은 pmfadm(1M) 설명서 페이지를 참조하십시오.

또한 PMF 기능은 Restart_interval , Retry_count 및 action_script(pmfadm에 대한 - t, -n 및 -a 옵션)에 대해 암시적 값을 사용합니다. 더욱 중요한 점은 DSDL이 재시작 또는 페일오버 결정을 계산하기 위해 PMF에서 확인된 프로세스 실패 기록을 오류 모니터에서 감지된 응용 프로그램 실패 기록에 연결한다는 것입니다.

함수는 다음과 같습니다.

- scds_pmf_get_status(3HA) – 지정한 인스턴스가 PMF의 제어 하에 모니터되고 있는지 확인합니다.
- scds_pmf_restart_fm(3HA) – PMF를 사용하여 오류 모니터를 재시작합니다.
- scds_pmf_signal(3HA) – PMF의 제어 하에 실행 중인 프로세스에 지정한 신호를 보냅니다.
- scds_pmf_start(3HA) – PMF의 제어 하에 지정한 프로그램(오류 모니터 포함)을 실행합니다.
- scds_pmf_stop(3HA) – PMF의 제어 하에 실행 중인 프로세스를 종료합니다.
- scds_pmf_stop_monitoring(3HA) – PMF의 제어 하에 실행 중인 프로세스 모니터를 중지합니다.

오류 모니터 함수

이 절의 함수는 실패 기록을 유지하고 Retry_count 및 Retry_interval 등록 정보와 함께 평가하여 미리 지정된 오류 모니터 모델을 제공합니다.

함수는 다음과 같습니다.

- scds_fm_sleep(3HA) – 오류 모니터 제어 소켓에서 메시지를 대기합니다.
- scds_fm_action(3HA) – 검사가 완료된 후 작업을 수행합니다.
- scds_fm_print_probes(3HA) – 시스템 로그에 검사 상태 정보를 씁니다.

유틸리티 함수

이 절의 함수를 사용하면 메시지 및 디버깅 메시지를 시스템 로그에 기록할 수 있습니다.

- `scds_syslog`(3HA) – 시스템 로그에 메시지를 씁니다.
- `scds_syslog_debug`(3HA) – 시스템 로그에 디버깅 메시지를 씁니다.

12장

CRNP

이 장에서는 CRNP(Cluster Reconfiguration Notification Protocol)에 대해 설명합니다. CRNP를 사용하여 페일오버와 확장 가능 응용 프로그램이 "클러스터를 인식"하도록 할 수 있습니다. 특히 CRNP는 응용 프로그램이 Sun Cluster 재구성 이벤트의 후속 비동기 알림을 등록 및 수신할 수 있게 하는 기법을 제공합니다. 클러스터 내에서 실행되는 데이터 서비스와 클러스터 외부에서 실행되는 응용 프로그램은 이벤트 알림을 등록할 수 있습니다. 이벤트는 클러스터의 구성원이 변경될 경우와 자원 그룹이나 자원의 상태가 변경될 경우에 생성됩니다.

주 – `SUNW.Event` 자원 유형 구현은 Sun Cluster에 고가용성 CRNP 서비스를 제공합니다. 이 자원 유형 구현에 대한 자세한 내용은 `SUNW.Event`(5) 설명서 페이지를 참조하십시오.

이 장은 다음 내용으로 구성되어 있습니다.

CRNP 개념

CRNP는 7개 계층의 표준 OSI (Open System Interconnect) 프로토콜 스택에서 응용 프로그램, 표현 및 세션 계층을 정의합니다. 전송 계층은 TCP여야 하며 네트워크 계층은 IP여야 합니다. CRNP는 데이터 링크 및 물리 계층과 무관합니다. CRNP에서 교환되는 모든 응용 프로그램 계층 메시지는 XML 1.0에 기초합니다.

CRNP 작동 방법

CRNP는 클러스터 재구성 이벤트를 생성하고 클러스터를 통해 이벤트를 라우팅하며 원하는 클라이언트에게 보내는 기법과 데몬을 제공합니다.

`cl_apid` 데몬은 클라이언트와 상호 작용하며Sun Cluster RGM(Resource Group Manager)은 클러스터 재구성 이벤트를 생성합니다. 이 데몬은 `sysevent`d를 사용하여 각 로컬 노드에서 이벤트를 전송합니다. `cl_apid` 데몬은 TCP/IP를 통한 XML(Extensible Markup Language)을 사용하여 원하는 클라이언트와 통신합니다.

다음 다이어그램은 CRNP 구성 요소 간의 이벤트 흐름을 보여줍니다. 이 다이어그램에서 한 클라이언트는 클러스터 노드 2에서 실행되고 있고 다른 클라이언트는 클러스터의 일부가 아닌 컴퓨터에서 실행되고 있습니다.

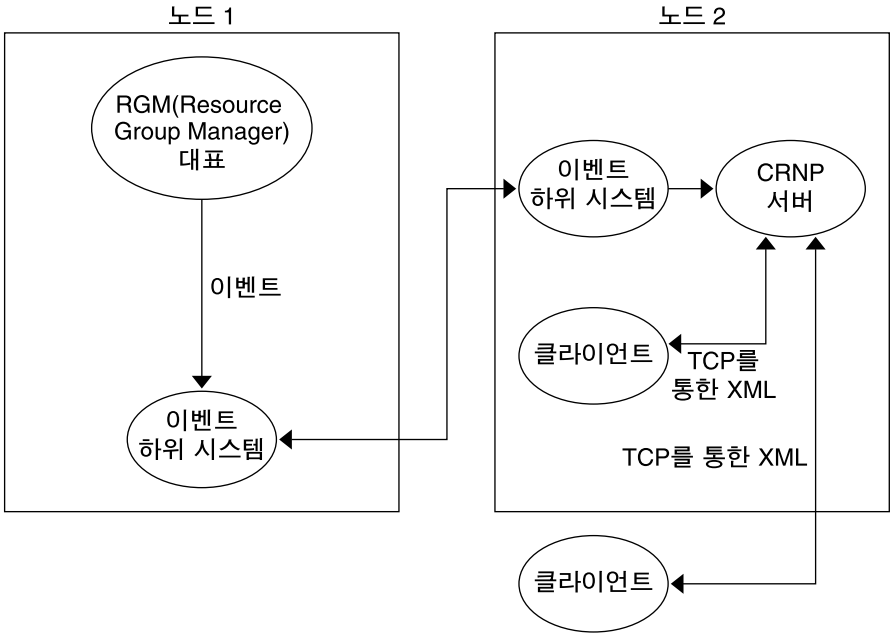


그림 12–1 CRNP 구성 요소 간의 이벤트 흐름

CRNP 의미

클라이언트는 등록 메시지(`SC_CALLBACK_RG`)를 서버에 보내 통신을 시작합니다. 이 등록 메시지는 클라이언트가 알림을 받으려는 이벤트 유형과 이벤트를 수신할 포트를 지정합니다. 등록 연결 및 지정한 포트의 소스 IP가 함께 콜백 주소를 구성합니다.

클러스터 내에서 클라이언트에 대한 해당 이벤트가 생성될 때마다 서버는 콜백 주소(IP 및 포트)의 클라이언트에 연결되어 이벤트(SC_EVENT)를 클라이언트에 전달합니다. 서버는 클러스터 자체 내에서 실행되며 높은 수준의 가용성을 가집니다. 서버는 클러스터가 재부트된 후에도 지속되는 저장소에 클라이언트 등록을 저장합니다.

클라이언트는 등록 메시지(REMOVE_CLIENT 메시지를 포함하는 SC_CALLBACK_RG)를 서버에 보내 등록을 취소합니다. 클라이언트는 SC_REPLY 메시지를 서버로부터 받은 후에 연결을 닫습니다.

다음 다이어그램은 클라이언트와 서버 간의 통신 흐름을 보여줍니다.

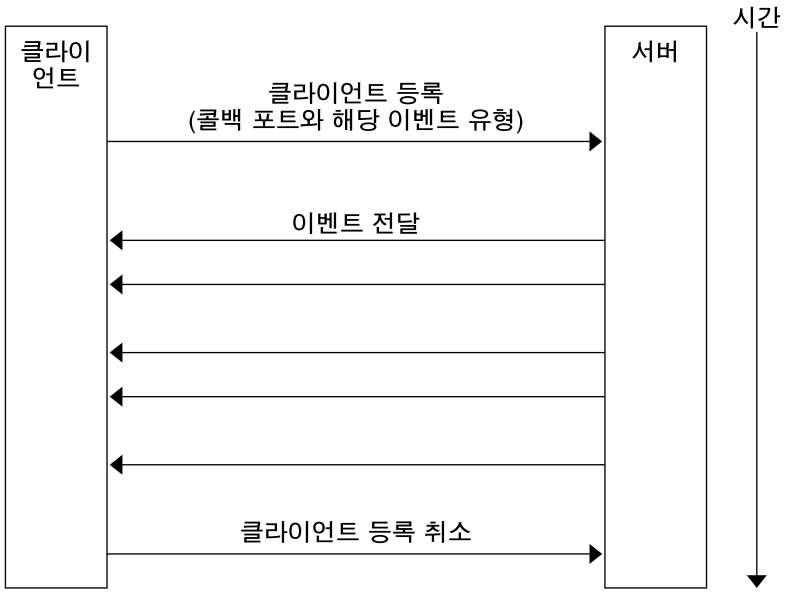


그림 12–2 클라이언트와 서버 간의 통신 흐름

CRNP 메시지 유형

CRNP는 다음 표에 설명된 세 가지 유형의 XML 기반 메시지를 사용합니다. 이러한 메시지 유형에 대해서는 나중에 자세히 설명됩니다.

CRNP 메시지 유형	설명
SC_CALLBACK_REG	이 메시지는ADD_CLIENT, REMOVE_CLIENT , ADD_EVENTS 및 REMOVE_EVENTS의 네 가지 형식을 가집니다. 이러한 형식은 각각 다음 정보를 포함합니다. ■ 프로토콜 버전 ■ ASCII 형식의 콜백 포트(이진 형식 아님) 또한 ADD_CLIENT, ADD_EVENTS 및 REMOVE_EVENTS 형식은 각각 다음 정보를 포함하는 바운드되지 않은 이벤트 유형 목록을 포함합니다. ■ 이벤트 클래스 ■ (옵션) 이벤트 하위 클래스 ■ (옵션) 이름 및 값 쌍 목록 이벤트 클래스와 이벤트 하위 클래스가 함께 고유한 "이벤트 유형"을 정의합니다. SC_CALLBACK_REG의 클래스가 생성되는 문서 유형 정의(DTD)는 SC_CALLBACK_REG입니다. 이 DTD에 대한 자세한 내용은 부록 F를 참조하십시오.
SC_REPLY	이 메시지에는 다음 정보가 포함되어 있습니다. ■ 프로토콜 버전 ■ 오류 코드 ■ 오류 메시지 SC_REPLY의 클래스가 생성되는 DTD는 SC_REPLY입니다. 이 DTD에 대한 자세한 내용은 부록 F를 참조하십시오.

SC_EVENT	이 메시지에는 다음 정보가 포함되어 있습니다.
	■ 프로토콜 버전
	■ 이벤트 클래스
	■ 이벤트 하위 클래스
	■ 공급업체
	■ 게시자
	■ 이름 및 값 쌍 목록(0개 이상의 이름 및 값 쌍 데이터 구조)
	  ■ 이름(문자열)
	  ■ 값(문자열 또는 문자열 배열)
	
	SC_EVENT의 값은 입력할 수 없습니다. SC_EVENT의 클래스가 생성되는 DTD는 SC_EVENT입니다. 이 DTD에 대한 자세한 내용은 부록 F를 참조하십시오.

클라이언트가 서버에 등록되는 방법

이 절에서는 클러스터 관리자가 서버를 설정하는 방법, 클라이언트가 식별되는 방법, 응용 프로그램 및 세션 계층을 통해 정보가 전송되는 방법 및 오류 상태에 대해 설명합니다.

관리자의 서버 설정 방법에 대한 가정

클러스터 관리자는 고가용성 IP 주소(즉, 클러스터의 특정 시스템에 연결되지 않은 IP 주소)와 포트 번호를 사용하여 서버를 구성해야 합니다. 또한 이 네트워크 주소를 예정된 클라이언트에 게시해야 합니다. CRNP는 이 서버 이름을 클라이언트가 사용할 수 있게 하는 방법을 정의하지 않습니다. 클러스터 관리자는 클라이언트가 서버의 네트워크 주소를 동적으로 찾을 수 있도록 하는 이름 지정 서비스를 사용하거나 클라이언트가 읽을 수 있도록 구성 파일에 네트워크 이름을 추가합니다. 서버는 클러스터 내에서 페일오버 자원 유형으로 실행됩니다.

서버가 클라이언트를 식별하는 방법

각 클라이언트는 콜백 주소, 즉 해당 IP 주소와 포트 번호에 의해 고유하게 식별됩니다. 포트는 SC_CALLBACK_REG 메시지에 지정되며 IP 주소는 TCP 등록 연결로부터 가져옵니다. CRNP는 동일한 콜백 주소를 가진 후속 SC_CALLBACK_REG 메시지가 동일한 클라이언트로부터 온다고 가정하며 이는 메시지를 보낸 원본 포트가 다른 경우에도 마찬가지입니다.

클라이언트와 서버 간에 SC_CALLBACK_REG 메시지가 전달되는 방법

클라이언트는 서버의 IP 주소와 포트 번호에 대한 TCP 연결을 열어 등록을 시작합니다. TCP 연결이 설정되고 쓰기가 가능해지면 클라이언트는 해당 등록 메시지를 보내야 합니다. 등록 메시지는 메시지 앞뒤에 추가 바이트를 포함하지 않아야 하며 서식이 올바로 지정된 단일 SC_CALLBACK_REG 메시지여야 합니다.

모든 바이트가 스트림에 기록되고 나면 클라이언트는 서버로부터 응답을 받기 위해 연결을 열어 두어야 합니다. 클라이언트가 메시지의 서식을 올바로 지정하지 않을 경우 서버는 클라이언트를 등록하지 않으며 오류 응답을 클라이언트에게 보냅니다. 그러나 서버가 응답을 보내기 전에 클라이언트가 소켓 연결을 닫을 경우 서버는 정상적으로 클라이언트를 등록합니다.

클라이언트는 언제든지 서버에 연결할 수 있습니다. 클라이언트는 서버에 연결할 때마다 SC_CALLBACK_REG 메시지를 보내야 합니다. 잘못되었거나, 문제가 있거나, 유효하지 않은 메시지가 전달되면 서버는 클라이언트에게 오류 응답을 보냅니다.

클라이언트는 ADD_CLIENT 메시지를 보내기 전까지 ADD_EVENTS, REMOVE_EVENTS 또는 REMOVE_CLIENT 메시지를 보낼 수 없습니다. 클라이언트는 ADD_CLIENT 메시지를 보내기 전까지 REMOVE_CLIENT 메시지를 보낼 수 없습니다.

클라이언트가 이미 등록되어 있는 상태에서 ADD_CLIENT 메시지가 전달될 경우 서버는 이 메시지를 허용할 수 있습니다. 이 경우 서버는 이전 클라이언트 등록을 두 번째 ADD_CLIENT 메시지에 지정된 새 클라이언트 등록으로 자동으로 대체합니다.

대부분의 경우 클라이언트는 시작될 때 ADD_CLIENT 메시지를 보냄으로써 서버에 한 차례 등록합니다. 또한 클라이언트는 REMOVE_CLIENT 메시지를 서버로 보내 등록을 한 번 취소합니다. 그러나 CRNP는 자신의 이벤트 유형 목록을 동적으로 수정해야 하는 클라이언트를 위한 더 많은 유연성을 제공합니다.

SC_CALLBACK_REG 메시지의 내용

각 ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS 및 REMOVE_EVENTS 메시지는 이벤트 목록을 포함합니다. 다음 표에는 CRNP에서 허용하는 이벤트 유형과 함께 필수 이름 및 값 쌍이 설명되어 있습니다.

클라이언트가 다음 작업 중 하나를 수행하면 서버는 이러한 메시지를 자동으로 무시합니다.

- 클라이언트가 이전에 등록되지 않은 이벤트 유형을 한 개 이상 지정하는 REMOVE_EVENTS 메시지를 보내는 경우
- 동일한 이벤트 유형을 두 번 등록할 경우

클래스 및 하위 클래스	이름 및 값 쌍	설명
EC_Cluster ESC_cluster_membership	필수: 없음 선택 사항: 없음	모든 클러스터 구성원 변경 이벤트(노드 상실 또는 결합)를 등록합니다.
EC_Cluster ESC_cluster_rg_state	다음과 같은 필수 항목 하나: rg_*name* 값 유형: 문자열 선택 사항: 없음	자원 그룹 *name*에 대한 모든 상태 변경 이벤트를 등록합니다.
EC_Cluster ESC_cluster_r_state	다음과 같은 필수 항목 하나: r_*name* 값 유형: 문자열 선택 사항: 없음	자원 **이름**에 대한 모든 상태 변경 이벤트를 등록합니다.

클래스 및 하위 클래스	이름 및 값 쌍	설명
EC_Cluster	필수: 없음	모든 Sun Cluster 이벤트를 등록합니다.
없음	선택 사항: 없음	

서버에서 클라이언트에 응답하는 방법

등록을 처리한 후 등록 요청을 받은 서버는 클라이언트에서 열린 TCP 연결을 통해 SC_REPLY 메시지를 보낸 다음 연결을 닫습니다. 클라이언트는 서버로부터 SC_REPLY 메시지를 받을 때까지 TCP 연결을 열어 두어야 합니다.

예를 들어, 클라이언트는 다음 작업을 수행합니다.

1. 서버에 대한 TCP 연결을 엽니다.
2. 연결이 "쓸 수 있는 상태"가 될 때까지 기다립니다.
3. ADD_CLIENT 메시지를 포함하는 SC_CALLBACK_REG 메시지를 보냅니다.
4. 서버로부터 SC_REPLY 메시지를 기다립니다.
5. 서버로부터 SC_REPLY 메시지를 수신합니다.
6. 서버가 연결을 닫았다는 표시를 수신합니다(소켓에서 0바이트를 읽음).
7. 연결을 닫습니다.

그런 다음 클라이언트는 나중에 다음 작업을 수행합니다.

1. 서버에 대한 TCP 연결을 엽니다.
2. 연결이 "쓸 수 있는 상태"가 될 때까지 기다립니다.
3. REMOVE_CLIENT 메시지를 포함하는 SC_CALLBACK_REG 메시지를 보냅니다.
4. 서버로부터 SC_REPLY 메시지를 기다립니다.
5. 서버로부터 SC_REPLY 메시지를 수신합니다.
6. 서버가 연결을 닫았다는 표시를 수신합니다(소켓에서 0바이트를 읽음).
7. 연결을 닫습니다.

서버는 클라이언트로부터 SC_CALLBACK_REG 메시지를 받을 때마다 열려 있는 동일한 연결을 통해 SC_REPLY 메시지를 보냅니다. 이 메시지는 작업의 성공 또는 실패 여부를 지정합니다. 318 페이지 "SC_REPLY XML DTD"에는 SC_REPLY 메시지의 XML 문서 유형 정의와 이 메시지가 포함할 수 있는 가능한 오류 메시지가 포함되어 있습니다.

SC_REPLY 메시지의 내용

SC_REPLY 메시지는 작업에 성공 또는 실패했는지를 지정합니다. 이 메시지는 CRNP 메시지의 버전, 상태 코드 및 상태 코드를 자세하게 설명하는 상태 메시지를 포함합니다. 다음 표에는 가능한 상태 코드 값이 설명되어 있습니다.

상태 코드	설명
OK	메시지가 성공적으로 처리되었습니다.
RETRY	일시적인 오류로 인하여 서버에서 클라이언트 등록을 거부했습니다. 클라이언트가 다른 인자를 사용하여 다시 등록을 시도해야 합니다.
LOW_RESOURCE	클러스터 자원이 부족하므로 클라이언트가 나중에 다시 시도해야 합니다. 클러스터의 클러스터 관리자가 클러스터의 자원을 늘릴 수도있습니다.
SYSTEM_ERROR	심각한 문제가 발생했습니다. 클러스터 관리자에게 클러스터에 대해 문의하십시오.
FAIL	인증에 실패했거나 다른 문제로 인해 등록에 실패했습니다.
MALFORMED	XML 요청이 잘못되었거나 구문 분석하지 못했습니다.
INVALID	XML 요청이 유효하지 않습니다(즉, XML 사양을 충족하지 않음).
VERSION_TOO_HIGH	메시지 버전이 너무 높아 메시지를 성공적으로 처리하지 못했습니다.
VERSION_TOO_LOW	메시지의 버전이 너무 낮아 메시지를 성공적으로 처리하지 못했습니다.

클라이언트의 오류 상태 처리 방법

정상적인 상태에서 SC_CALLBACK_REG 메시지를 보내는 클라이언트는 등록이 성공 또는 실패했는지를 나타내는 응답을 받습니다.

그러나 클라이언트를 등록하는 동안 서버에 오류 상태가 발생하여 서버에서 SC_REPLY 메시지를 클라이언트로 보내지 못할 수 있습니다. 이러한 경우 등록은 오류 상태가 발생하기 전에 성공했거나, 실패했거나, 아직 처리되지 않았을 수 있습니다.

서버는 클러스터에서 페일오버 또는 고가용성 서버로 작동해야 하므로 이 오류 상태로 인해 서비스가 종료되는 것은 아닙니다. 실제로 서버는 새로 등록된 클라이언트에 대한 이벤트 전송을 곧바로 시작할 수 있습니다.

이러한 문제를 해결하려면 클라이언트가 다음 작업을 수행해야 합니다.

- SC_REPLY 메시지를 대기 중인 등록 연결에 대해 응용 프로그램 수준의 시간 초과를 지정합니다. 시간이 초과되면 클라이언트는 등록을 다시 시도해야 합니다.
- 이벤트 콜백을 등록하기 전에 이벤트 전달을 위한 해당 콜백 IP 주소 및 포트 번호를 수신합니다. 클라이언트는 등록 확인 메시지와 이벤트 전달을 동시에 기다려야 합니다. 클라이언트는 확인 메시지를 받기 전에 이벤트를 받기 시작한 경우 등록 연결을 자동으로 닫아야 합니다.

서버에서 클라이언트에 이벤트를 전달하는 방법

클러스터 내에서 이벤트가 생성되면 CRNP 서버는 해당 유형의 이벤트를 요청한 모든 클라이언트에게 이벤트를 전달합니다. 이러한 전달은 SC_EVENT 메시지를 클라이언트의 콜백 주소로 보내는 것으로 이루어집니다. 각 이벤트의 전달은 새 TCP 연결에서 수행됩니다.

ADD_CLIENT 메시지나 ADD_EVENT 메시지를 포함하는 SC_CALLBACK_REG 메시지를 통해 클라이언트가 이벤트 유형을 등록한 직후에 서버는 해당 유형의 최신 이벤트를 클라이언트에게 보냅니다. 이렇게 하면 클라이언트는 후속 이벤트를 보내는 시스템의 현재 상태를 확인할 수 있습니다.

서버는 클라이언트에 대한 TCP 연결을 시작할 때 정확하게 하나의 SC_EVENT 메시지를 연결을 통해 보냅니다. 그런 다음 서버는 전이중 닫기를 실행합니다.

예를 들어, 클라이언트는 다음 작업을 수행합니다.

1. 서버가 TCP 연결을 시작하기를 기다립니다.
2. 서버로부터 들어오는 연결을 수락합니다.
3. 서버로부터 SC_EVENT 메시지를 기다립니다.
4. 서버로부터 SC_EVENT 메시지를 읽습니다.
5. 서버가 연결을 닫았다는 표시를 수신합니다(소켓에서 0바이트를 읽음).
6. 연결을 닫습니다.

등록된 모든 클라이언트는 해당 콜백 주소(IP 주소 및 포트 번호)에서 들어오는 이벤트 전달 연결을 계속 수신해야 합니다.

서버가 이벤트 전달을 위해 클라이언트에 연결하는 데 실패할 경우 지정된 간격으로 여러 번에 걸쳐 이벤트 전달을 다시 시도합니다. 모든 시도에 실패하면 서버의 클라이언트 목록에서 해당 클라이언트가 제거됩니다. 또한 클라이언트는 ADD_CLIENT 메시지를 포함하는 다른 SC_CALLBACK_REG 메시지를 보내 다시 등록해야만 추가 이벤트를 받을 수 있습니다.

이벤트 전달을 보장하는 방법

클러스터에서의 전체 이벤트 생성 순서가 각 클라이언트에 대한 이벤트 전달 순서가 됩니다. 다시 말해서 클러스터 내에서 이벤트 A가 이벤트 B보다 먼저 생성된 경우 클라이언트 X는 이벤트 B를 받기 전에 이벤트를 A를 받습니다. 그러나 **모든** 클라이언트에 대해 전체 이벤트 전달 순서가 유지되는 것은 **아닙니다**. 즉, 클라이언트 Y는 클라이언트 X가 이벤트 A를 받기 전에 이벤트 A와 B를 모두 받을 수 있습니다. 이는 속도가 느린 클라이언트가 모든 클라이언트에 대한 이벤트 전달을 방해하지 않는다는 것을 의미합니다.

클러스터가 생성한 이벤트를 누락시키는 오류가 서버에서 발생하는 경우를 제외하고 서버가 전달하는 모든 이벤트(하위 클래스에 대한 첫 번째 이벤트와 서버 오류 이후의 이벤트는 제외)는 클러스터가 생성하는 실제 이벤트에 응답하여 발생합니다. 이 경우 서버는 해당 유형의 시스템의 현재 상태를 나타내는 각 이벤트 유형에 대해 이벤트를 생성합니다. 각 이벤트는 해당 이벤트 유형을 원하는 등록된 클라이언트에게 보내집니다.

이벤트 전달에는 "최소한 한 번 이상"이라는 표현이 적용됩니다. 즉, 서버에서 동일한 이벤트를 클라이언트에 두 번 이상 보낼 수 있습니다. 이 기능은 서버가 일시적으로 다운되었다가 다시 원래 상태로 돌아왔을 때 클라이언트가 최신 정보를 받았는지 확인할 수 없는 경우에 필요합니다.

SC_EVENT 메시지의 내용

SC_EVENT 메시지는 클러스터 내에서 생성되어 SC_EVENT XML 메시지 형식에 맞게 변환된 실제 메시지를 포함합니다. 다음 표에는 이름 및 값 쌍, 게시자 및 공급업체를 비롯하여 CRNP가 제공하는 이벤트 유형이 설명되어 있습니다.

주 – state_list에 대한 배열 요소의 위치는 node_list에 대한 배열 요소의 위치와 동기화됩니다. 즉, node_list 배열에 첫 번째로 나열된 노드의 상태는 state_list 배열에서도 첫 번째로 나열됩니다.

ev_로 시작하는 추가 이름과 관련 값이 존재할 수 있지만 이러한 값은 클라이언트가 사용하도록 되어 있지 않습니다.

클래스 및 하위 클래스	게시자 및 공급업체	이름 및 값 쌍
EC_Cluster	게시자: rgm	이름: node_list
ESC_cluster_membership	공급업체: SUNW	값 유형: 문자열 배열
		이름: state_list
		state_list는 ASCII로 표시되는 숫자만 포함합니다. 각 숫자는 클러스터의 해당 노드에 대한 현재 구현 번호를 나타냅니다. 이 번호가 이전 메시지에서 받은 번호와 같을 경우 노드의 클러스터에 대한 관계(이탈, 결합 또는 재결합)가 변경되지 않은 것입니다. 구현 번호가 –1일 경우 노드는 클러스터의 구성원이 아닙니다. 구현 번호가 음수가 아닌 숫자일 경우 노드는 클러스터의 구성원입니다.
		값 유형: 문자열 배열

클래스 및 하위 클래스	게시자 및 공급업체	이름 및 값 쌍
EC_Cluster	게시자: rgm	이름: rg_name
ESC_cluster_rg_state	공급업체: SUNW	값 유형: 문자열
		이름: node_list
		값 유형: 문자열 배열
		이름: state_list
		state_list는 자원 그룹의 상태에 대한 문자열 표시를 포함합니다. 유효한 값은 scha_cmds(1HA) 명령으로 검색할 수 있는 값입니다.
		값 유형: 문자열 배열
EC_Cluster	게시자: rgm	이름: r_name
ESC_cluster_r_state	공급업체: SUNW	값 유형: 문자열
		이름: node_list
		값 유형: 문자열 배열
		이름: state_list
		state_list는 자원의 상태에 대한 문자열 표시를 포함합니다. 유효한 값은 scha_cmds(1HA) 명령으로 검색할 수 있는 값입니다.
		값 유형: 문자열 배열

CRNP의 클라이언트 및 서버 인증 방법

서버는 TCP 래퍼 형태를 사용하여 클라이언트를 인증합니다. 이벤트가 전달되는 콜백 IP 주소로도 사용되는 등록 메시지의 소스 IP 주소는 서버에서 허용하는 클라이언트 목록 에 있어야 합니다. 소스 IP 주소 및 등록 메시지가 거부된 클라이언트 목록에 있으면 안 됩니다. 소스 IP 주소 및 등록이 목록에 없을 경우 서버는 요청을 거부하고 클라이언트에게 오류 응답을 보냅니다.

서버가 SC_CALLBACK_REG ADD_CLIENT 메시지를 받으면 해당 클라이언트에 대한 후속 SC_CALLBACK_REG 메시지는 첫 번째 메시지의 소스 IP 주소와 동일한 소스 IP 주소를 포함해야 합니다. CRNP 서버는 이 요구 사항을 충족하지 않는 SC_CALLBACK_REG를 받을 경우 다음 작업 중 하나를 수행합니다.

- 요청을 무시하고 클라이언트에게 오류 응답을 보냅니다.
- SC_CALLBACK_REG 메시지의 내용에 따라 새 클라이언트에서 요청을 보냈다고 가정합니다.

이 보안 기법은 누군가가 정당한 클라이언트의 등록을 취소하려고 시도하는 서비스 거부 공격을 방지하는 데 도움이 됩니다.

또한 클라이언트도 마찬가지로 서버를 인증해야 합니다. 클라이언트는 소스 IP 주소와 포트 번호가 자신이 사용했던 등록 IP 주소 및 포트 번호와 같은 서버의 이벤트 전달만 수락하면 됩니다.

CRNP 서비스의 클라이언트가 클러스터를 보호하는 방화벽 안쪽에 있다고 가정하기 때문에 CRNP에는 추가 보안 기법이 포함되지 않습니다.

CRNP를 사용하는 Java 응용 프로그램 생성 예

다음 예에서는 CRNP를 사용하는 `CrnpClient`라는 간단한 Java 응용 프로그램을 개발하는 방법을 보여줍니다. 이 응용 프로그램은 클러스터의 CRNP 서버에 이벤트 콜백을 등록하고 이러한 이벤트 콜백을 수신하며 해당 내용을 인쇄하여 이벤트를 처리합니다. 종료하기 전에 이 응용 프로그램은 이벤트 콜백에 대한 요청을 등록 취소합니다.

다음 사항에 주의하면서 이 예를 검토합니다.

- 샘플 응용 프로그램은 JAXP(Java API for XML Processing)를 사용하여 XML을 생성하고 구문을 분석합니다. JAXP를 사용하는 방법은 이 예에서 다루지 않습니다. AXP에 대한 자세한 내용은 `http://java.sun.com/xml/jaxp/index.html`을 참조하십시오.
- 이 예에서는 전체 응용 프로그램의 일부를 제공하며 응용 프로그램의 전체 내용은 부록 G에 나와 있습니다. 특정 개념을 보다 효율적으로 보여주기 위해 이 장에서 제공하는 예는 부록 G에 나온 전체 응용 프로그램과 약간 다릅니다.
- 내용을 간단히 하기 위해 이 장의 예에서는 샘플 코드의 주석을 제외했습니다. 부록 G의 전체 응용 프로그램에는 주석이 포함되어 있습니다.
- 이 예에 나온 응용 프로그램은 단순히 응용 프로그램을 종료하는 방법으로 대부분의 오류 상태를 처리합니다. 실제 응용 프로그램은 보다 안정적인 방법으로 오류를 처리해야 합니다.

▼ 환경 설정 방법

단계 1. JAXP와 올바른 버전의 Java 컴파일러 및 가상 머신을 다운로드하여 설치합니다.

`http://java.sun.com/xml/jaxp/index.html`에서 지침을 확인할 수 있습니다.

주 – 이 예를 사용하려면 Java 1.3.1 이상이 필요합니다.

2. **소스 파일이 있는 디렉토리에서 다음을 입력합니다.**

```
% javac -classpath jaxp-root/dom.jar:jaxp-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xsltc.jar -sourcepath . source-filename.java
```

여기서 *jaxp-root*는 JAXP jar 파일이 있는 디렉토리의 절대 또는 상대 경로 이며 *source-filename*은 Java 소스 파일의 이름입니다.

컴파일러가 JAXP 클래스를 찾을 수 있도록 컴파일 명령줄에 `classpath` 를 지정해야 합니다.

3. **응용 프로그램을 실행할 때 응용 프로그램이 적절한 JAXP 클래스 파일을 로드할 수 있도록 다음과 같이 `classpath`를 지정합니다. `classpath`의 첫 번째 경로는 현재 디렉토리입니다.**

```
% java -cp .:jaxp-root/dom.jar:jaxp-rootjaxp-api. \
jar:jaxp-rootsax.jar:jaxp-rootxalan.jar:jaxp-root/xercesImpl \
.jar:jaxp-root/xsltc.jar source-filename arguments
```

이제 환경이 구성되었으므로 응용 프로그램을 개발할 수 있습니다.

▼ 응용 프로그램 개발 시작 방법

이 부분의 예에서는 명령줄 인자를 구문 분석하고 `CrnpClient` 객체를 생성하는 주 메소드를 가진 `CrnpClient`라는 기본 클래스를 만듭니다. 이 객체는 클래스에 명령줄 인자를 전달하고 사용자가 응용 프로그램을 종료하기를 기다렸다가 `CrnpClient`에서 `shutdown`을 호출한 다음 종료합니다.

`CrnpClient` 클래스의 구성자는 다음 작업을 실행해야 합니다.

- XML 처리 객체를 설정합니다.
- 이벤트 콜백을 수신하는 스레드를 만듭니다.
- CRNP 서버에 연결하고 이벤트 콜백을 등록합니다.

단계 ● **앞의 논리를 구현하는 Java 코드를 만듭니다.**

다음 예에서는 `CrnpClient` 클래스의 스켈레톤 코드를 보여줍니다. 구성자 및 종료 메소드에서 참조되는 4개의 도우미 메소드에 대한 구현은 나중에 표시됩니다. 필요한 모든 패키지를 가져오는 코드가 표시됩니다.

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
```

```
import org.w3c.dom.*;
import java.net.*;
import java.io.*;
import java.util.*;

class CrnpClient
{
        public static void main(String []args)
        {
                InetAddress regIp = null;
                int regPort = 0, localPort = 0;
                try {
                        regIp = InetAddress.getByName(args[0]);
                        regPort = (new Integer(args[1])).intValue();
                        localPort = (new Integer(args[2])).intValue();
                } catch (UnknownHostException e) {
                        System.out.println(e);
                        System.exit(1);
                }
                CrnpClient client = new CrnpClient(regIp, regPort,
                    localPort, args);
                System.out.println("Hit return to terminate demo...");
                try {
                        System.in.read();
                } catch (IOException e) {
                        System.out.println(e.toString());
                }
                client.shutdown();
                System.exit(0);
        }

        public CrnpClient(InetAddress regIpIn, int regPortIn,
            int localPortIn, String []clArgs)
        {
                try {
                        regIp = regIpIn;
                        regPort = regPortIn;
                        localPort = localPortIn;
                        regs = clArgs;
                        setupXmlProcessing();
                        createEvtRecepThr();
                        registerCallbacks();
                } catch (Exception e) {
                        System.out.println(e.toString());
                        System.exit(1);
                }
        }

        public void shutdown()
        {
                try {
                        unregister();
                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
```

```
                }
        }

        private InetAddress regIp;
        private int regPort;
        private EventReceptionThread evtThr;
        private String regs[];
        public int localPort;
        public DocumentBuilderFactory dbf;
}
```

구성원 변수에 대해서는 나중에 자세히 설명됩니다.

▼ 명령줄 인자 구문 분석 방법

단계 ● **명령줄 인자를 구문 분석하려면 부록 G의 코드를 참조하십시오.**

▼ 이벤트 수신 스레드 정의 방법

이벤트 스레드가 차단되어 이벤트 콜백을 대기하는 동안 응용 프로그램에서 다른 작업을 계속할 수 있으려면 코드에서 이벤트 수신이 별도의 스레드에서 수행되도록 지정해야 합니다.

주 – XML 설정에 대해서는 나중에 자세히 설명됩니다.

단계 **1. `ServerSocket`을 만들고 이 소켓에서 이벤트가 도착하기를 기다리는 `EventReceptionThread`라는 `Thread` 하위 클래스를 코드에 정의합니다.**

이 코드 예 부분에서는 이벤트를 읽거나 처리하지 않습니다. 이벤트 읽기 및 처리에 대해서는 나중에 자세히 설명됩니다. `EventReceptionThread`는 와일드카드 인터네트워킹 프로토콜 주소에서 `ServerSocket`을 만듭니다. `EventReceptionThread`는 `CrnpClient` 객체에 대한 참조를 유지하므로 `EventReceptionThread`에서 처리할 `CrnpClient` 객체에 이벤트를 보낼 수 있습니다.

```
class EventReceptionThread extends Thread
{
        public EventReceptionThread(CrnpClient clientIn) throws IOException
        {
                client = clientIn;
                listeningSock = new ServerSocket(client.localPort, 50,
                    InetAddress.getLocalHost());
        }

        public void run()
```

```
        {
                try {
                        DocumentBuilder db = client.dbf.newDocumentBuilder();
                        db.setErrorHandler(new DefaultHandler());

                        while(true) {
                                Socket sock = listeningSock.accept();
                                // Construct event from the sock stream and process it
                                sock.close();
                        }
                        // UNREACHABLE

                } catch (Exception e) {
                        System.out.println(e);
                        System.exit(1);
                }
        }

        /* private member variables */
        private ServerSocket listeningSock;
        private CrnpClient client;
}
```

2. `createEvtRecepThr` 객체를 생성합니다.

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
}
```

▼ 콜백 등록 및 등록 취소 방법

등록 작업은 다음과 같이 구성됩니다.

- 등록 인터네트워킹 프로토콜 및 포트에 대한 기본 TCP 소켓 열기
- XML 등록 메시지 생성
- 소켓에서 XML 등록 메시지 보내기
- 소켓으로부터 XML 응답 메시지 읽기
- 소켓 닫기

단계 1. 앞의 논리를 구현하는 Java 코드를 만듭니다.

다음 예제 코드에서는 `CrnpClient` 구성자에 의해 호출되는 `CrnpClient` 클래스의 `registerCallbacks` 메소드에 대한 구현을 보여줍니다. `createRegistrationString()` 및 `readRegistrationReply()` 호출에 대해서는 나중에 자세히 설명됩니다.

`regIp` 및 `regPort`는 구성자에 의해 설정되는 객체 구성원입니다.

```
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. **unregister 메소드를 구현합니다.**

   이 메소드는 CrnpClient의 shutdown 메소드에 의해 호출됩니다. createUnregistrationString 구현에 대해서는 나중에 자세히 설명됩니다.

```
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();
}
```

▼ XML 생성 방법

이제 응용 프로그램의 구조를 설정하고 모든 네트워킹 코드를 작성했으므로 XML을 생성 및 구문 분석하는 코드를 작성합니다. 먼저 SC_CALLBACK_REG XML 등록 메시지를 생성하는 코드를 작성합니다.

SC_CALLBACK_REG 메시지는 등록 유형(ADD_CLIENT, REMOVE_CLIENT, ADD_EVENTS 또는 REMOVE_EVENTS), 콜백 포트 및 원하는 이벤트 목록으로 구성됩니다. 각 이벤트는 클래스 및 하위 클래스로 구성되며 이름 및 값 쌍 목록이 그 뒤에 옵니다.

이 예 부분에서는 등록 유형, 콜백 포트 및 등록 이벤트 목록을 저장하는 CallbackReg 클래스를 작성합니다. 이 클래스는 또한 SC_CALLBACK_REG XML 메시지에 대해 자신을 일련화할 수 있습니다.

이 클래스에서 흥미로운 메소드는 클래스 구성원에서 SC_CALLBACK_REG XML 메시지 문자열을 만드는 convertToXml 메소드입니다. http://java.sun.com/xml/jaxp/index.html의 JAXP 설명서에는 이 메소드의 코드가 자세히 설명되어 있습니다.

다음 예제 코드에는 Event 클래스의 구현이 나와 있습니다. CallbackReg 클래스는 하나의 이벤트를 저장하고 이 이벤트를 XML Element로 변환할 수 있는 Event 클래스를 사용합니다.

단계 1. 앞의 논리를 구현하는 Java 코드를 만듭니다.

```
class CallbackReg
{
```

```
public static final int ADD_CLIENT = 0;
public static final int ADD_EVENTS = 1;
public static final int REMOVE_EVENTS = 2;
public static final int REMOVE_CLIENT = 3;

public CallbackReg()
{
        port = null;
        regType = null;
        regEvents = new Vector();
}

public void setPort(String portIn)
{
        port = portIn;
}

public void setRegType(int regTypeIn)
{
        switch (regTypeIn) {
        case ADD_CLIENT:
                regType = "ADD_CLIENT";
                break;
        case ADD_EVENTS:
                regType = "ADD_EVENTS";
                break;
        case REMOVE_CLIENT:
                regType = "REMOVE_CLIENT";
                break;
        case REMOVE_EVENTS:
                regType = "REMOVE_EVENTS";
                break;
        default:
                System.out.println("Error, invalid regType " +
                    regTypeIn);
                regType = "ADD_CLIENT";
                break;
        }
}

public void addRegEvent(Event regEvent)
{
        regEvents.add(regEvent);
}

public String convertToXml()
{
        Document document = null;
        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
                DocumentBuilder builder = factory.newDocumentBuilder();
                document = builder.newDocument();
        } catch (ParserConfigurationException pce) {
                // Parser with specified options can't be built
```

```
                pce.printStackTrace();
                System.exit(1);
        }

        // Create the root element
        Element root = (Element) document.createElement("SC_CALLBACK_REG");

        // Add the attributes
        root.setAttribute("VERSION", "1.0");
        root.setAttribute("PORT", port);
        root.setAttribute("regType", regType);

        // Add the events
        for (int i = 0; i < regEvents.size(); i++) {
                Event tempEvent = (Event)
                    (regEvents.elementAt(i));
                root.appendChild(tempEvent.createXmlElement(document));
        }
        document.appendChild(root);

        // Convert the whole thing to a string
        DOMSource domSource = new DOMSource(document);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
                Transformer transformer = tf.newTransformer();
                transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
                System.out.println(e.toString());
                return ("");
        }
        return (strWrite.toString());
    }

    private String port;
    private String regType;
    private Vector regEvents;
}
```

2. **Event 및 NVPair 클래스를 구현합니다.**

   CallbackReg 클래스는 NVPair 클래스를 직접 사용하는 Event 클래스를 사용합니다.

```
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;
        }
```

▼ 등록 및 등록 취소 메시지 작성 방법

이제 XML 메시지를 생성하는 도우미 클래스를 만들었으므로 `createRegistrationString` 메소드의 구현을 작성할 수 있습니다. 이 메소드는 210 페이지 "콜백 등록 및 등록 취소 방법"에 설명된 `registerCallbacks` 메소드에 의해 호출됩니다.

`createRegistrationString`은 `CallbackReg` 객체를 생성하고 해당 등록 유형 및 포트를 설정합니다. 그런 다음 `createRegistrationString`은 `createAllEvent`, `createMembershipEvent`, `createRgEvent` 및 `createREvent` 도우미 메소드를 사용하여 다양한 이벤트를 생성합니다. `CallbackReg` 객체가 만들어진 후에 각 이벤트는 이 객체에 추가됩니다. 마지막으로 `createRegistrationString`은 `CallbackReg` 객체에서 `convertToXml` 메소드를 호출하여 `String` 형태로 XML 메시지를 검색합니다.

`regs` 구성원 변수는 사용자가 응용 프로그램에 제공하는 명령줄 인자를 저장합니다. 다섯 번째 및 그 이후의 인자는 응용 프로그램이 등록해야 하는 이벤트를 지정합니다. 네 번째 인자는 등록 유형을 지정하지만 이 예에서는 무시됩니다. 부록 G의 전체 코드에는 이 네 번째 인자를 사용하는 방법이 나와 있습니다.

단계 1. 앞의 논리를 구현하는 Java 코드를 만듭니다.

```
private String createRegistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);

        cbReg.setRegType(CallbackReg.ADD_CLIENT);

        // add the events
        for (int i = 4; i < regs.length; i++) {
                if (regs[i].equals("M")) {
                        cbReg.addRegEvent(createMembershipEvent());
                } else if (regs[i].equals("A")) {
                        cbReg.addRegEvent(createAllEvent());
                } else if (regs[i].substring(0,2).equals("RG")) {
```

```
                        cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
                } else if (regs[i].substring(0,1).equals("R")) {
                        cbReg.addRegEvent(createREvent(regs[i].substring(2)));
                }
        }

        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}

private Event createAllEvent()
{
        Event allEvent = new Event();
        allEvent.setClass("EC_Cluster");
        return (allEvent);
}

private Event createMembershipEvent()
{
        Event membershipEvent = new Event();
        membershipEvent.setClass("EC_Cluster");
        membershipEvent.setSubclass("ESC_cluster_membership");
        return (membershipEvent);
}

private Event createRgEvent(String rgname)
{
        Event rgStateEvent = new Event();
        rgStateEvent.setClass("EC_Cluster");
        rgStateEvent.setSubclass("ESC_cluster_rg_state");

        NVPair rgNvpair = new NVPair();
        rgNvpair.setName("rg_name");
        rgNvpair.setValue(rgname);
        rgStateEvent.addNvpair(rgNvpair);

        return (rgStateEvent);
}

private Event createREvent(String rname)
{
        Event rStateEvent = new Event();
        rStateEvent.setClass("EC_Cluster");
        rStateEvent.setSubclass("ESC_cluster_r_state");

        NVPair rNvpair = new NVPair();
        rNvpair.setName("r_name");
        rNvpair.setValue(rname);
        rStateEvent.addNvpair(rNvpair);

        return (rStateEvent);
}
```

2. 등록 취소 문자열을 만듭니다.

등록 취소 문자열을 만드는 것은 이벤트를 수용할 필요가 없기 때문에 등록 문자열을 만드는 것보다 쉽습니다.

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}
```

▼ XML 구문 분석기 설정 방법

이제 응용 프로그램의 네트워킹 및 XML 생성 코드를 만들었습니다. `CrnpClient` 구성자는 `setupXmlProcessing` 메소드를 호출합니다. 이 메소드는 `DocumentBuilderFactory` 객체를 만들고 이 객체에 대한 다양한 구분 분석 등록 정보를 설정합니다. JAXP 설명서에 이 메소드가 자세히 설명되어 있습니다. `http://java.sun.com/xml/jaxp/index.html`을 참조하십시오.

단계 ● **앞의 논리를 구현하는 Java 코드를 만듭니다.**

```
private void setupXmlProcessing() throws Exception
{
        dbf = DocumentBuilderFactory.newInstance();

        // We don't need to bother validating
        dbf.setValidating(false);
        dbf.setExpandEntityReferences(false);

        // We want to ignore comments and whitespace
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);

        // Coalesce CDATA sections into TEXT nodes.
        dbf.setCoalescing(true);
}
```

▼ 등록 응답 구문 분석 방법

등록 또는 등록 취소 메시지에 응답하여 CRNP 서버가 보낸 `SC_REPLY` XML 메시지를 구분 분석하려면 `RegReply` 도우미 클래스가 필요합니다. 이 클래스는 XML 문서에서 생성할 수 있습니다. 이 클래스는 상태 코드와 상태 메시지에 대한 액세서를 제공합니다. 서버의 XML 스트림을 구문 분석하려면 새 XML 문서를 만들고 이 문서의 구문 분석 메소드를 사용해야 합니다. `http://java.sun.com/xml/jaxp/index.html`의 JAXP 설명서에 이 메소드가 자세히 설명되어 있습니다.

단계 **1. 앞의 논리를 구현하는 Java 코드를 만듭니다.**

readRegistrationReply 메소드는 새 RegReply 클래스를 사용한다는 점에 주의합니다.

```
private void readRegistrationReply(InputStream stream) throws Exception
{
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();
        db.setErrorHandler(new DefaultHandler());

        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
}
```

2. **RegReply 클래스를 구현합니다.**

   retrieveValues 메소드가 XML 문서의 DOM 트리를 통과하면서 상태 코드와 상태 메시지를 추출한다는 것에 주의합니다.
   http://java.sun.com/xml/jaxp/index.html의 JAXP 설명서에 자세한 내용이 나와 있습니다.

```
class RegReply
{
        public RegReply(Document doc)
        {
                retrieveValues(doc);
        }

        public String getStatusCode()
        {
                return (statusCode);
        }

        public String getStatusMsg()
        {
                return (statusMsg);
        }
        public void print(PrintStream out)
        {
                out.println(statusCode + ": " +
                    (statusMsg != null ? statusMsg : ""));
        }

        private void retrieveValues(Document doc)
        {
                Node n;
                NodeList nl;
                String nodeName;

                // Find the SC_REPLY element.
                nl = doc.getElementsByTagName("SC_REPLY");
                if (nl.getLength() != 1) {
                        System.out.println("Error in parsing: can't find "
```

```
                    + "SC_REPLY node.");
                return;
            }

            n = nl.item(0);

            // Retrieve the value of the statusCode attribute
            statusCode = ((Element)n).getAttribute("STATUS_CODE");

            // Find the SC_STATUS_MSG element
            nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
            if (nl.getLength() != 1) {
                System.out.println("Error in parsing: can't find "
                    + "SC_STATUS_MSG node.");
                return;
            }
            // Get the TEXT section, if there is one.
            n = nl.item(0).getFirstChild();
            if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Not an error if there isn't one, so we just silently return.
                return;
            }

            // Retrieve the value
            statusMsg = n.getNodeValue();
        }

        private String statusCode;
        private String statusMsg;
}
```

▼ 콜백 이벤트 구문 분석 방법

마지막 단계는 실제 콜백 이벤트를 구분 분석 및 처리하는 것입니다. 이 작업을 지원하기 위해 211 페이지 "XML 생성 방법"에서 만든 `Event` 클래스를 수정하여 이 클래스가 XML 문서에서 `Event`를 생성하고 XML `Element`를 만들 수 있게 합니다. 이러한 변경 작업에는 XML 문서를 사용하는 추가 구성자, `retrieveValues` 메소드, 두 개의 추가 구성원 변수(`vendor` 및 `publisher`), 모든 필드에 대한 액세서 메소드 및 마지막으로 인쇄 메소드가 필요합니다.

단계 1. 앞의 논리를 구현하는 Java 코드를 만듭니다.

이 코드는 217 페이지 "등록 응답 구문 분석 방법"에 설명된 `RegReply` 클래스의 코드와 비슷합니다.

```
public Event(Document doc)
        {
                nvpairs = new Vector();
                retrieveValues(doc);
        }
        public void print(PrintStream out)
```

```
{
        out.println("\tCLASS=" + regClass);
        out.println("\tSUBCLASS=" + regSubclass);
        out.println("\tVENDOR=" + vendor);
        out.println("\tPUBLISHER=" + publisher);
        for (int i = 0; i < nvpairs.size(); i++) {
                NVPair tempNv = (NVPair)
                    (nvpairs.elementAt(i));
                out.print("\t\t");
                tempNv.print(out);
        }
}

private void retrieveValues(Document doc)
{
        Node n;
        NodeList nl;
        String nodeName;

        // Find the SC_EVENT element.
        nl = doc.getElementsByTagName("SC_EVENT");
        if (nl.getLength() != 1) {
           System.out.println("Error in parsing: can't find "
               + "SC_EVENT node.");
           return;
        }

        n = nl.item(0);

        //
        // Retrieve the values of the CLASS, SUBCLASS,
        // VENDOR and PUBLISHER attributes.
        //
        regClass = ((Element)n).getAttribute("CLASS");
        regSubclass = ((Element)n).getAttribute("SUBCLASS");
        publisher = ((Element)n).getAttribute("PUBLISHER");
        vendor = ((Element)n).getAttribute("VENDOR");

        // Retrieve all the nv pairs
        for (Node child = n.getFirstChild(); child != null;
            child = child.getNextSibling())
        {
               nvpairs.add(new NVPair((Element)child));
        }
}

public String getRegClass()
{
        return (regClass);
}

public String getSubclass()
{
        return (regSubclass);
}
```

```
public String getVendor()
{
        return (vendor);
}

public String getPublisher()
{
        return (publisher);
}

public Vector getNvpairs()
{
        return (nvpairs);
}

private String vendor, publisher;
```

2. **XML 구문 분석을 지원하는 `NVPair` 클래스에 대한 추가 구성자와 메소드를 구현합니다.**

   단계 1에 표시된 것처럼 `Event` 클래스를 변경하려면 `NVPair` 클래스를 비슷하게 변경해야 합니다.

```
public NVPair(Element elem)
        {
                retrieveValues(elem);
        }
        public void print(PrintStream out)
        {
                out.println("NAME=" + name + " VALUE=" + value);
        }
        private void retrieveValues(Element elem)
        {
                Node n;
                NodeList nl;
                String nodeName;

                // Find the NAME element
                nl = elem.getElementsByTagName("NAME");
                if (nl.getLength() != 1) {
                   System.out.println("Error in parsing: can't find "
                       + "NAME node.");
                   return;
                }
                // Get the TEXT section
                n = nl.item(0).getFirstChild();
                if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                   System.out.println("Error in parsing: can't find "
                       + "TEXT section.");
                   return;
                }

                // Retrieve the value
                name = n.getNodeValue();
```

```
                    // Now get the value element
                    nl = elem.getElementsByTagName("VALUE");
                    if (nl.getLength() != 1) {
                       System.out.println("Error in parsing: can't find "
                           + "VALUE node.");
                       return;
                    }
                    // Get the TEXT section
                    n = nl.item(0).getFirstChild();
                    if (n == null || n.getNodeType() != Node.TEXT_NODE) {
                    System.out.println("Error in parsing: can't find "
                                + "TEXT section.");
                            return;
                    }

                    // Retrieve the value
                    value = n.getNodeValue();
                    }

            public String getName()
            {
                    return (name);
            }

            public String getValue()
            {
                    return (value);
            }
}
```

3. **이벤트 콜백을 대기하는 `EventReceptionThread`에서 `while` 루프를 구현합니다.**

   `EventReceptionThread`는 209 페이지 "이벤트 수신 스레드 정의 방법"에 설명되어 있습니다.

```
while(true) {
                    Socket sock = listeningSock.accept();
                    Document doc = db.parse(sock.getInputStream());
                    Event event = new Event(doc);
                    client.processEvent(event);
                    sock.close();
            }
```

▼ 응용 프로그램 실행 방법

단계 1. **수퍼유저가 되거나 동등한 역할을 맡습니다.**

2. **응용 프로그램을 실행합니다.**

   # **`java CrnpClient`** *crnpHost crnpPort localPort* **`...`**

   `CrnpClient` 응용 프로그램의 전체 코드는 부록 G에 나와 있습니다.

부록 A

표준 등록 정보

이 부록에서는 표준 자원 유형, 자원 및 자원 그룹 등록 정보에 대해 설명합니다. 이 부록에서는 시스템 정의 등록 정보를 변경하고 확장 등록 정보를 만드는 데 사용할 수 있는 자원 등록 정보 속성에 대해서도 설명합니다.

주 – 자원 유형, 자원 및 자원 그룹에 대한 등록 정보 이름은 대소문자를 구분하지 **않습니다**. 등록 정보 이름을 지정할 때는 대소문자를 임의로 조합해서 사용할 수 있습니다.

이 부록은 다음 내용으로 구성되어 있습니다.

자원 유형 등록 정보

다음 정보에서는 Sun Cluster 소프트웨어에서 정의한 자원 유형 등록 정보에 대해 설명합니다. 등록 정보 값의 범주는 다음과 같습니다.

- **필수적**. 이 등록 정보는 자원 유형 등록(RTR) 파일에 명시적인 값이 있어야 합니다. 그렇지 않으면 해당 등록 정보가 속한 객체를 생성할 수 없습니다. 공백이나 빈 문자열은 값으로 사용할 수 없습니다.
- **조건적**. 존재하려면 RTR 파일에서 등록 정보를 선언해야 합니다. 그렇지 않으면 RGM에서 등록 정보를 만들지 않고 해당 등록 정보를 관리 유틸리티에 사용할 수 없습니다. 공백이나 빈 문자열이 허용됩니다. 등록 정보가 RTR 파일에 선언되었지만 값이 지정되지 않았다면 RGM에서 기본값을 제공합니다.

- **조건적 또는 명시적**. 존재하려면 RTR 파일에서 명시적 값으로 등록 정보를 선언해야 합니다. 그렇지 않으면 RGM에서 등록 정보를 만들지 않고 해당 등록 정보를 관리 유틸리티에 사용할 수 없습니다. 공백이나 빈 문자열은 허용되지 않습니다.
- **선택적**. RTR 파일에서 등록 정보를 선언할 수 있습니다. 등록 정보가 RTR 파일에 선언되지 않은 경우 RGM은 등록 정보를 만든 다음 기본값을 제공합니다. RTR 파일에 등록 정보를 선언하고 값을 지정하지 않은 경우 RTR 파일에 등록 정보를 선언하지 않은 것처럼 RGM에서 동일한 기본값을 제공합니다.
- **쿼리 전용** – 관리 도구에서 직접 설정할 수 없습니다.

RTR 파일에서 선언할 수 없고 클러스터 관리자가 설정해야 하는 `Installed_nodes` 및 `RT_system`을 제외하고 관리 유틸리티에서 자원 유형 등록 정보를 업데이트할 수 없습니다.

등록 정보 이름이 먼저 표시되고 그 뒤에 설명이 표시됩니다.

주 – `API_version` 및 `Boot`와 같은 자원 유형 등록 정보 이름은 대소문자를 구분하지 **않습니다**. 등록 정보 이름을 지정하는 경우 대문자와 소문자를 임의로 조합해서 사용할 수 있습니다.

`API_version(integer)`
이 자원 유형 구현을 지원하는 데 필요한 자원 관리 API의 최소 버전입니다.

다음 정보는 각 Sun Cluster 릴리스에서 지원하는 최대 `API_version`을 요약한 것입니다.

3.1 이하 버전	2
3.1 10/03	3
3.1 4/04	4
3.1 9/04	5
3.1 8/05	6

RTR 파일에서 2보다 큰 `API_version` 값을 선언하면 이보다 낮은 최대 버전을 지원하는 Sun Cluster 버전에 자원 유형을 설치할 수 없습니다. 예를 들어, 자원 유형에 대해 `API_version=5`를 선언하는 경우 3.1 9/04 이전에 릴리스된 Sun Cluster 버전에 해당 자원 유형을 설치할 수 없습니다.

주 – 이 등록 정보를 선언하지 않거나 이 등록 정보를 기본값(2)으로 설정하면 Sun Cluster 3.0부터 모든 Sun Cluster 버전에 데이터 서비스를 설치할 수 있습니다.

범주: 선택적

기본값: 2

조정 가능: NONE

Boot(string)
선택적 콜백 메소드: 이 유형의 자원이 이미 관리되는 경우 클러스터에 가입하거나 다시 가입하는 노드에서 RGM이 실행하는 프로그램의 경로입니다. 이 메소드는 Init 메소드와 같이 이 유형의 자원을 초기화합니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Failover(boolean)
TRUE는 동시에 여러 노드에서 온라인 상태가 될 수 있는 그룹에 이 유형의 자원을 구성할 수 없음을 나타냅니다.

다음 표에서는 이 자원 유형 등록 정보를 Scalable 자원 등록 정보와 함께 사용하는 방법을 보여줍니다.

Failover 자원 유형의 값	Scalable 자원의 값	설명
TRUE	TRUE	이 조합은 비논리적이므로 지정하지 마십시오.
TRUE	FALSE	페일오버 서비스에 대해 이 조합을 지정합니다.
FALSE	TRUE	네트워크 로드 균형 조정을 위해 SharedAddress 자원을 사용하는 확장 가능한 서비스에 이 조합을 지정합니다. **Solaris OS용 Sun Cluster 개념 안내서**에서는 SharedAddress에 대해 자세히 설명합니다.
FALSE	FALSE	일반적인 조합은 아니지만 이 조합을 사용하여 네트워크 로드 균형 조정을 사용하지 않는 다중 마스터 서비스를 선택할 수 있습니다.

r_properties(5) 설명서 페이지의 Scalable에 대한 설명과 **Solaris OS용 Sun Cluster 개념 안내서**의 3 장, "시스템 관리자와 응용 프로그램 개발자를 위한 주요 개념"에 추가 정보가 포함되어 있습니다.

범주: 선택적

기본값: FALSE

조정 가능: NONE

Fini(string)
선택적 콜백 메소드: RGM 관리에서 이 유형의 자원을 제거할 경우 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

`Init(string)`
선택적 콜백 메소드: 이 유형의 자원을 RGM에서 관리할 경우 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

`Init_nodes(enum)`
RGM이 `Init`, `Fini`, `Boot` 및 `Validate` 메소드를 호출하는 노드를 나타냅니다. 값은 `RG_PRIMARIES`(자원을 마스터할 수 있는 노드만) 또는 `RT_INSTALLED_NODES`(자원 유형이 설치된 모든 노드)일 수 있습니다.

범주: 선택적

기본값: `RG_PRIMARIES`

조정 가능: NONE

`Installed_nodes(string_array)`
자원 유형이 실행될 수 있는 클러스터 노드 이름 목록입니다. RGM에서 자동으로 이 등록 정보를 만듭니다. 해당 클러스터 관리자는 값을 설정할 수 있습니다. RTR 파일에서 이 등록 정보를 선언할 수 없습니다.

범주: 클러스터 관리자가 구성할 수 있음

기본값: 모든 클러스터 노드

조정 가능: ANYTIME

`Is_logical_hostname(boolean`
`TRUE`는 이 자원 유형이 페일오버 인터넷 프로토콜(IP) 주소를 관리하는 `LogicalHostname` 자원 유형의 일부 버전임을 나타냅니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

`Is_shared_address(boolean)`
`TRUE`는 이 자원 유형이 페일오버 인터넷 프로토콜(IP) 주소를 관리하는 `SharedAddress` 자원 유형의 일부 버전임을 나타냅니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

Monitor_check(string)
선택적 콜백 메소드: 이 자원 유형의 모니터에서 요청하는 페일오버를 수행하기 전에 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Monitor_start(string)
선택적 콜백 메소드: 이 유형의 자원에 대한 오류 모니터를 시작하기 위해 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Monitor_stop(string)
Monitor_start가 설정된 경우 필요한 콜백 메소드: 이 유형의 자원에 대한 오류 모니터를 중지하기 위해 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Pkglist(string_array)
자원 유형 설치에 포함된 패키지의 선택적 목록입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Postnet_stop(string)
선택적 콜백 메소드: 이 유형의 자원이 종속되는 네트워크 주소 자원(Network_resources_used)의 Stop 메소드를 호출한 후 RGM에서 실행하는 프로그램의 경로입니다. 네트워크 인터페이스를 비활성으로 구성한 후 이 메소드에서 Stop 작업을 수행해야 합니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

Prenet_start(string)
선택적 콜백 메소드: 이 유형의 자원이 종속되는 네트워크 주소 자원(Network_resources_used)의 Start 메소드를 호출하기 전에 RGM에서 실행하는 프로그램의 경로입니다. 이 메소드는 네트워크 인터페이스를 구성하기 전에 수행해야 하는 Start 작업을 수행합니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

`Resource_list(string_array)`
자원 유형의 모든 자원 목록입니다. 클러스터 관리자는 이 등록 정보를 직접 설정하지 않습니다. 그 대신 클러스터 관리자가 이 유형의 자원을 자원 그룹에서 추가 또는 제거할 때 RGM이 이 등록 정보를 업데이트합니다.

범주: 쿼리 전용

기본값: 빈 목록

조정 가능: NONE

`Resource_type(string)`
자원 유형의 이름. 현재 등록된 자원 유형 이름을 보려면 다음을 사용합니다.

scrgadm -p

Sun Cluster 3.1 이후 릴리스에서는 다음과 같이 자원 유형 이름에 버전이 반드시 포함됩니다.

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

자원 유형 이름의 세 구성 요소는 RTR 파일에 *vendor-id*, *resource-type* 및 *rt-version*으로 지정된 등록 정보입니다. `scrgadm` 명령은 마침표(.)와 콜론(:)분리자를 삽입합니다. 자원 유형 이름의 *rt-version* 접미어는 `RT_version` 등록 정보와 같은 값입니다. *vendor-id*가 고유한지 확인하려면 자원 유형을 만드는 회사의 주식 기호를 사용합니다. Sun Cluster 3.1 이전에 만든 자원 유형 이름의 형식은 다음과 같습니다.

vendor-id.*resource-type*

범주: 필수적

기본값: 빈 문자열

조정 가능: NONE

`RT_basedir(string)`
콜백 메소드에 대한 상대 경로를 완성하는 데 사용되는 디렉토리 경로입니다. 이 경로는 자원 유형 패키지가 설치된 디렉토리로 설정해야 합니다. 전체 경로, 즉 슬래시(/)로 시작하는 경로여야 합니다.

범주: 모든 메소드 경로 이름이 절대적이지 않은 경우 필수적

기본값: 없음

조정 가능: NONE

`RT_description(string)`
자원 유형에 대한 간단한 설명입니다.

범주: 조건적

기본값: 빈 문자열

조정 가능: NONE

RT_system(boolean)
자원 유형에 대한 RT_system 등록 정보가 TRUE이면 해당 자원 유형(scrgadm -r -t *resource-type-name*)을 삭제할 수 없습니다. 이 등록 정보는 LogicalHostname과 같이 클러스터 인프라를 지원하는 데 사용되는 자원 유형이 실수로 삭제되는 것을 방지합니다. 그러나 모든 자원 유형에 RT_system 등록 정보를 적용할 수 있습니다.

RT_system 등록 정보가 TRUE로 설정된 자원 유형을 삭제하려면 먼저 이 등록 정보를 FALSE로 설정해야 합니다. 해당 자원이 클러스터 서비스를 지원하는 자원 유형을 삭제할 때는 주의하십시오.

범주: 선택적

기본값: FALSE

조정 가능: ANYTIME

RT_version(string)
Sun Cluster 3.1에서 사용되기 시작한 이 자원 유형 구현의 필수 버전 문자열입니다. RT_version은 전체 자원 유형 이름의 접미어 구성 요소입니다. Sun Cluster 3.0에서 선택적이었던 RT_version 등록 정보가 Sun Cluster 3.1 이후 릴리스에서는 필수적입니다.

범주: 조건적/명시적 또는 필수적

기본값: 없음

조정 가능: NONE

Single_instance(boolean)
TRUE인 경우에는 이 유형의 자원이 클러스터에 오직 하나만 존재할 수 있음을 나타냅니다. RGM을 사용하면 이 유형의 자원을 한 번에 하나만 클러스터 전체에 실행할 수 있습니다.

범주: 선택적

기본값: FALSE

조정 가능: NONE

Start(string)
콜백 메소드: 이 유형의 자원을 시작하기 위해 RGM에서 실행하는 프로그램의 경로입니다.

범주: RTR 파일에서 Prenet_start 메소드를 선언하지 않는 경우 필수적

기본값: 없음

조정 가능: NONE

Stop(string)
콜백 메소드: 이 유형의 자원을 중지하기 위해 RGM에서 실행하는 프로그램의 경로입니다.

범주: RTR 파일에서 Postnet_stop 메소드를 선언하지 않는 경우 필수적

기본값: 없음

조정 가능: NONE

`Update(string)`
선택적 콜백 메소드: 실행 중인 이 자원 유형의 등록 정보가 변경되었을 때 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

`Validate(string)`
선택적 콜백 메소드: 이 자원 유형의 등록 정보 값을 확인하기 위해 RGM에서 실행하는 프로그램의 경로입니다.

범주: 조건적 또는 명시적

기본값: 없음

조정 가능: NONE

`Vendor_ID(string)`
`Resource_type` 등록 정보를 참조하십시오.

범주: 조건적

기본값: 없음

조정 가능: NONE

자원 등록 정보

이 절에서는 Sun Cluster 소프트웨어에서 정의하는 자원 등록 정보에 대해 설명합니다. 등록 정보 값의 범주는 다음과 같습니다.

- **필수적**. 클러스터 관리자가 관리 유틸리티를 사용하여 자원을 생성할 때 반드시 값을 지정해야 합니다.
- **선택적**. 클러스터 관리자가 자원 그룹을 생성할 때 값을 지정하지 않으면 시스템이 기본값을 제공합니다.
- **조건적**. 등록 정보가 RTR 파일에 선언된 경우에만 RGM에서 등록 정보를 만듭니다. 그렇지 않으면 등록 정보가 존재하지 않고 클러스터 관리자가 사용할 수 없습니다. RTR 파일에 선언된 조건적 등록 정보는 RTR 파일에 기본값이 지정되었는지 여부에 따라 선택적이거나 필수적입니다. 자세한 내용은 조건적 등록 정보의 설명을 참조하십시오.
- **쿼리 전용**. 관리 도구에서 직접 설정할 수 없습니다.

253 페이지 “자원 등록 정보 속성”에 설명된 조정 가능한 속성은 다음과 같이 자원 등록 정보를 업데이트할 수 있는지 여부와 업데이트할 수 있는 시기를 나타냅니다.

FALSE 또는 NONE	절대 안 함
TRUE 또는 ANYTIME	언제든지
AT_CREATION	해당 자원이 클러스터에 추가될 때
WHEN_DISABLED	해당 자원이 비활성화될 때

등록 정보 이름이 먼저 표시되고 그 뒤에 설명이 표시됩니다.

Affinity_timeout(integer)
: 자원의 모든 서비스에 지정된 클라이언트 IP 주소의 연결을 동일한 서버 노드로 전송하는 시간(초)입니다.

  Load_balancing_policy가 Lb_sticky 또는 Lb_sticky_wild일 경우에만 이 등록 정보가 해당됩니다. 또한 Weak_affinity를 FALSE로 설정해야 합니다.

  이 등록 정보는 확장 가능 서비스에서만 사용합니다.

  범주: 선택적

  기본값: 없음

  조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Boot_timeout(integer)
: RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

  범주: 조건적 또는 선택적

  기본값: RTR 파일에서 메소드 자체가 선언된 경우 3600(1시간)

  조정 가능: ANYTIME

Cheap_probe_interval(integer)
: 자원에 대한 빠른 오류 검사 호출 사이의 시간(초)입니다. 등록 정보가 RTR 파일에서 선언된 경우에만 RGM에서 이 등록 정보를 만들고 클러스터 관리자가 사용할 수 있습니다. 기본값이 RTR 파일에 지정되지 않았다면 이 등록 정보는 선택적입니다.

  RTR 파일에 Tunable 속성이 지정되지 않은 경우 해당 등록 정보에 대한 Tunable 값은 WHEN_DISABLED입니다.

  범주: 조건적

  기본값: 없음

  조정 가능: WHEN_DISABLED

확장 등록 정보
: 해당 자원 유형의 RTR 파일에 선언된 확장명 등록 정보입니다. 이 자원 유형의 구현 방식에 따라 이 등록 정보가 정의됩니다. 253 페이지 “자원 등록 정보 속성”에는 확장 등록 정보에 대해 설정할 수 있는 개별 속성 정보가 포함되어 있습니다.

범주: 조건적

기본값: 없음

조정 가능: 특정 등록 정보에 따라 다름

`Failover_mode`(enum)
자원이 성공적으로 시작 또는 중지되지 않거나 자원 모니터에서 자원 상태가 불량임을 발견하고 재시작 또는 페일오버를 요청한 경우 RGM에서 수행할 복구 작업을 수정합니다.

`NONE`, `SOFT` 또는 `HARD`(메소드 실패)

이러한 설정은 시작 또는 중지 메소드(`Prenet_start`, `Start`, `Monitor_stop`, `Stop`, `Postnet_stop`)가 실패한 경우의 페일오버 동작에만 적용됩니다. 자원이 성공적으로 시작된 경우 `NONE`, `SOFT` 및 `HARD` 는 자원 모니터가 `scha_control` 명령이나 `scha_control()` 함수를 사용하여 나중에 시작한 자원 재시작 또는 giveover 동작에 적용되지 않습니다. `scha_control`(1HA) 및 `scha_control`(3HA) 설명서 페이지를 참조하십시오. `NONE`은 앞에 나열된 시작 또는 중지 메소드 중 하나가 실패할 경우 RGM에서 어떤 복구 작업도 수행하지 않음을 나타냅니다. `SOFT` 또는 `HARD`는 `Start` 또는 `Prenet_start` 메소드가 실패할 경우 RGM에서 자원 그룹을 다른 노드로 재배치함을 나타냅니다. `Start` 또는 `Prenet_start` 실패에서는 `SOFT`와 `HARD`가 같습니다.

중지 메소드(`Monitor_stop`, `Stop` 또는 `Postnet_stop`) 실패에서는 `SOFT`와 `NONE`이 같습니다. `Failover_mode`를 `HARD`로 설정하면 이러한 중지 메소드 중 하나가 실패할 경우 RGM이 노드를 재부트하여 자원 그룹을 오프라인 상태로 만듭니다. 그런 다음 RGM은 다른 노드에서 해당 그룹을 시작합니다.

`RESTART_ONLY` 또는 `LOG_ONLY`

시작 또는 중지 메소드 실패 시 페일오버 동작에 적용되는 `NONE`, `SOFT` 및 `HARD`와 달리 `RESTART_ONLY` 와 `LOG_ONLY`는 모든 페일오버 동작에 적용됩니다. 페일오버 동작에는 모니터에서 시작한(`scha_control`) 자원 및 자원 그룹 재시작, 자원 모니터에서 시작한 giveover(`scha_control`) 등이 포함됩니다. `RESTART_ONLY`는 모니터에서 `scha_control`을 실행하여 자원 또는 자원 그룹을 재시작할 수 있음을 나타냅니다. RGM은 `Retry_interval`내에서 `Retry_count`만큼의 재시도를 허용합니다. `Retry_count`를 초과하면 더 이상 재시작이 허용되지 않습니다. `Failover_mode`를 `LOG_ONLY`로 설정하면 자원 재시작 또는 giveover가 허용되지 않습니다. `Failover_mode`를 `LOG_ONLY`로 설정하는 것은 `Failover_mode`를 `RESTART_ONLY`로 설정하고 `Retry_count`를 0으로 설정하는 것과 같습니다.

`RESTART_ONLY` 또는 `LOG_ONLY`(메소드 실패)

`Prenet_start`, `Start`, `Monitor_stop` , `Stop` 또는 `Postnet_stop` 메소드가 실패하면 `RESTART_ONLY` 및 `LOG_ONLY`와 `NONE` 이 같습니다. 즉, 노드를 페일오버하거나 재부트하지 않습니다.

데이터 서비스에 대한 `Failover_mode` 설정의 영향

Failover_mode에 대한 각 설정이 데이터 서비스에 미치는 영향은 데이터 서비스의 모니터링 여부와 DSDL(Data Services Development Library) 기반인지 여부에 따라 달라집니다.

- Monitor_start 메소드를 구현하고 자원 모니터링을 활성화하면 데이터 서비스가 모니터링됩니다. RGM은 자원 자체를 시작한 후 Monitor_start 메소드를 실행하여 자원 모니터를 시작합니다. 자원 모니터는 자원의 상태를 검사합니다. 검사가 실패하면 자원 모니터에서 scha_control() 함수를 호출하여 재시작이나 페일오버를 요청할 수도 있습니다. DSDL 기반 자원의 경우 검사에서 데이터 서비스의 부분 실패(저하) 또는 전체 실패를 나타낼 수도 있습니다. 반복된 부분 실패는 전체 실패로 누적됩니다.
- Monitor_start 메소드를 제공하지 않거나 자원 모니터링이 비활성화된 경우 데이터 서비스의 모니터링이 해제됩니다.
- DSDL 기반 데이터 서비스에는 GDS를 통해 또는 DSDL을 직접 사용하여 Agent Builder로 개발된 서비스가 포함됩니다. 일부 데이터 서비스(예: HA Oracle)는 DSDL을 사용하지 않고 개발되었습니다.

NONE, SOFT 또는 HARD(검사 실패)

Failover_mode를 NONE, SOFT 또는 HARD로 설정하고 데이터 서비스가 모니터링된 DSDL 기반 서비스인 경우 및 검사가 완전히 실패한 경우 모니터는 scha_control () 함수를 호출하여 자원 재시작을 요청합니다. 검사가 계속해서 실패하면 자원이 Retry_interval 내에서 최대 Retry_count 횟수까지 재시작됩니다. Retry_count 횟수만큼 재시작한 후 검사가 다시 실패하면 모니터는 자원 그룹의 페일오버를 다른 노드로 요청합니다.

Failover_mode를 NONE, SOFT 또는 HARD로 설정하고 데이터 서비스가 모니터링 해제된 DSDL 기반 서비스인 경우 감지되는 유일한 실패는 자원 프로세스 트리의 상실입니다. 자원 프로세스 트리가 상실되면 자원이 재시작됩니다.

데이터 서비스가 DSDL 기반 서비스가 아닌 경우 재시작 또는 페일오버 동작은 자원 모니터의 코딩 방법에 따라 달라집니다. 예를 들어 Oracle 자원 모니터는 자원 또는 자원 그룹을 재시작하거나 자원 그룹을 페일오버하여 복구됩니다.

RESTART_ONLY(검사 실패)

Failover_mode를 RESTART_ONLY로 설정하고 데이터 서비스가 모니터링된 DSDL 기반 서비스인 경우 및 검사가 완전히 실패한 경우 자원이 Retry_interval 내에서 Retry_count만큼 재시작됩니다. 그러나 Retry_count를 초과하면 자원 모니터가 종료되고 자원 상태가 FAULTED로 설정되며 "응용 프로그램 오류가 발생했지만 재시작되지 않았습니다. 검사가 종료됩니다."라는 상태 메시지가 생성됩니다. 이때 모니터링이 활성화되어 있어도 자원은 클러스터 관리자가 복구 및 재시작할 때까지 실제로 모니터링 해제됩니다.

Failover_mode를 RESTART_ONLY로 설정하고 데이터 서비스가 모니터링 해제된 DSDL 기반 서비스인 경우 및 프로세스 트리가 상실된 경우 자원이 재시작되지 **않습니다**.

모니터링된 데이터 서비스가 DSDL 기반이 아니면 복구 동작은 자원 모니터의 코딩 방법에 따라 달라집니다. Failover_mode를 RESTART_ONLY로 설정하면 Retry_interval 내에서 Retry_count만큼 scha_control() 함수를 호출하여

자원 또는 자원 그룹을 재시작할 수 있습니다. 자원 모니터가 Retry_count를 초과하면 재시작 시도가 실패합니다. 모니터가 scha_control() 함수를 호출하여 페일오버를 요청하면 해당 요청도 실패합니다.

LOG_ONLY(검사 실패)

임의의 데이터 서비스에 대해 Failover_mode를 LOG_ONLY로 설정하면 자원 또는 자원 그룹을 재시작하거나 그룹을 페일오버하는 모든 scha_control() 요청이 수행되지 않습니다. 데이터 서비스가 DSDL 기반이면 검사가 완전히 실패할 경우 메시지가 기록되지만 자원이 재시작되지는 않습니다. 검사가 Retry_interval 내에서 Retry_count 횟수 이상 완전히 실패하면 자원 모니터가 종료되고 자원 상태가 FAULTED로 설정되며 "응용 프로그램 오류가 발생했지만 재시작되지 않았습니다. 검사가 종료됩니다."라는 상태 메시지가 생성됩니다. 이때 모니터링이 활성화되어 있어도 자원은 클러스터 관리자가 복구 및 재시작할 때까지 실제로 모니터링 해제됩니다.

Failover_mode를 LOG_ONLY로 설정하고 데이터 서비스가 모니터링 해제된 DSDL 기반 서비스인 경우 및 프로세스 트리가 상실된 경우 메시지가 기록되지만 자원이 재시작되지는 않습니다.

모니터링된 데이터 서비스가 DSDL 기반이 아니면 복구 동작은 자원 모니터의 코딩 방법에 따라 달라집니다. Failover_mode를 LOG_ONLY로 설정하면 자원 또는 자원 그룹을 재시작하거나 그룹을 페일오버하는 모든 scha_control() 요청이 실패합니다.

범주: 선택적

기본값: NONE

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Fini_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Init_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

Load_balancing_policy(string)
사용 중인 로드 균형 조정 정책을 정의하는 문자열입니다. 이 등록 정보는 확장 가능 서비스에 대해서만 사용 가능합니다. Scalable 등록 정보가 RTR 파일에 선언된 경우 RGM에서 자동으로 이 등록 정보를 만듭니다. Load_balancing_policy에 설정할 수 있는 값은 다음과 같습니다.

Lb_weighted (기본값). Load_balancing_weights 등록 정보에 설정된 가중치에 따라 여러 노드에 로드를 분산합니다.

Lb_sticky. 확장 가능 서비스의 해당 클라이언트(클라이언트 IP 주소로 식별)는 항상 같은 클러스터 노드로 전송됩니다.

Lb_sticky_wild. 와일드카드 고정 서비스의 IP 주소에 연결하는 지정된 클라이언트의 IP 주소는 IP 주소와 연결된 포트 번호와 관계없이 항상 동일한 클러스터 노드로 전송됩니다.

범주: 조건적 또는 선택적

기본값: Lb_weighted

조정 가능: AT_CREATION

Load_balancing_weights(string_array)
확장 가능한 자원에만 해당합니다. Scalable 등록 정보가 RTR 파일에 선언된 경우 RGM에서 자동으로 이 등록 정보를 만듭니다. 형식은 *weight@node, weight@node*입니다. 여기서 *weight*는 지정한 *node*로 분산된 로드의 상대적인 분량을 반영하는 정수입니다. 노드로 분산되는 로드의 분수는 이 노드에 대한 가중치를 모든 가중치의 합으로 나눈 것입니다. 예를 들어 1@1, 3@2는 노드 1이 로드의 1/4을 받고 노드 2가 로드의 3/4을 받도록 지정합니다. 빈 문자열(“”)은 기본값으로 균일하게 분산되도록 설정합니다. 명시적으로 가중치가 할당되지 않은 노드는 기본 가중치인 1을 수신합니다.

Tunable 속성이 RTR 파일에 지정되지 않은 경우 등록 정보에 대한 Tunable 값은 ANYTIME입니다. 이 등록 정보를 변경하면 새 연결에 대한 분산에만 적용됩니다.

범주: 조건적 또는 선택적

기본값: 빈 문자열(“”)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Monitor_check_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Monitor_start_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Monitor_stop_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

Monitored_switch(enum)
클러스터 관리자가 관리 유틸리티를 사용하여 모니터를 활성화하거나 비활성화할 경우 RGM에서 Enabled 또는 Disabled로 설정합니다. Disabled로 설정하면 자원 자체가 온라인 상태인 경우에도 자원에 대한 모니터링이 중지됩니다. 모니터링을 다시 활성화할 때까지 Monitor_start 메소드가 호출되지 않습니다. 해당 자원에 모니터 콜백 메소드가 없는 경우에는 이 등록 정보가 존재하지 않습니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

Network_resources_used(string_array)
자원이 사용하는 논리 호스트 이름이나 공유 주소 네트워크 자원의 목록입니다. 확장 가능 서비스의 경우 이 등록 정보는 반드시 별도 자원 그룹에 존재하는 공유 주소 자원을 참조해야 합니다. 페일오버 서비스의 경우, 이 등록 정보는 같은 자원 그룹에 존재하는 논리 호스트 이름이나 공유 주소 자원을 참조합니다. Scalable 등록 정보가 RTR 파일에 선언된 경우 RGM에서 자동으로 이 등록 정보를 만듭니다. Scalable이 RTR 파일에 선언되지 않은 경우 RTR 파일에 명시적으로 선언될 때까지 Network_resources_used를 사용할 수 없습니다.

Tunable 속성이 RTR 파일에 지정되지 않은 경우 등록 정보에 대한 Tunable 값은 AT_CREATION입니다.

주 – SUNW.Event(5) 설명서 페이지에서는 CRNP에 대해 이 등록 정보를 설정하는 방법을 설명합니다.

범주: 조건적 또는 필수적

기본값: 없음

조정 가능: AT_CREATION

각 클러스터 노드에 대한 Num_resource_restarts(integer)
직접 이 등록 정보를 설정할 수는 없습니다. RGM은 이 등록 정보를 이전 *n*초 내에 이 노드의 이 자원에 대해 수행된 scha_control, Resource_restart 또는 Resource_is_restarted 호출 수로 설정합니다. *n*은 자원의 Retry_interval 등록 정보 값입니다. giveover 시도의 성공 여부에 관계없이 이 자원이 scha_control giveover를 실행할 때마다 RGM은 자원 재시작 카운터를 영(0)으로 재설정합니다.

자원 유형에서 Retry_interval 등록 정보를 선언하지 않은 경우 해당 유형의 자원에 대해 Num_resource_restarts 등록 정보를 사용할 수 없습니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

각 클러스터 노드에 대한 Num_rg_restarts(integer)
직접 이 등록 정보를 설정할 수는 없습니다. RGM은 이 등록 정보를 이전 *n*초 내에 이 노드의 포함 자원 그룹에 대해 자원이 수행한 scha_control Restart 호출 수로 설정합니다. *n*은 자원의 Retry_interval 등록 정보 값입니다. 자원 유형이 Retry_interval 등록 정보를 선언하지 않으면 해당 유형의 자원에 대해 Num_rg_restarts 등록 정보를 사용할 수 없습니다.

범주: 설명 참조

기본값: 없음

조정 가능: NONE

On_off_switch(enum)
클러스터 관리자가 관리 유틸리티를 사용하여 모니터를 활성화하거나 비활성화할 경우 RGM에서 Enabled 또는 Disabled로 설정합니다. 비활성화하면 자원이 오프라인 상태가 되며 다시 활성화할 때까지 콜백이 실행되지 않습니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

Port_list(string_array)
서버가 수신 중인 포트 번호 목록. 각 포트 번호에 슬래시(/)가 추가되고 뒤에 해당 포트에서 사용 중인 프로토콜이 옵니다(예: Port_list=80/tcp 또는 Port_list=80/tcp6,40/udp6). 지정할 수 있는 프로토콜 값은 다음과 같습니다.

- tcp, TCP IPv4의 경우
- tcp6, TCP IPv6의 경우
- udp, UDP IPv4의 경우
- udp6, UDP IPv6의 경우

Scalable 등록 정보가 RTR 파일에 선언된 경우 RGM에서 자동으로 Port_list를 만듭니다. 그렇지 않으면 이 등록 정보는 RTR 파일에 명시적으로 선언될 때까지는 사용할 수 없습니다.

Apache에 대한 이 등록 정보 설정은 **Sun Cluster Data Service for Apache Guide for Solaris OS**에 설명되어 있습니다.

범주: 조건적 또는 필수적

기본값: 없음

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Postnet_stop_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Prenet_start_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

R_description(string)
자원에 대한 간단한 설명입니다.

범주: 선택적

기본값: 빈 문자열

조정 가능: ANYTIME

Resource_dependencies(string_array)
Resource_dependencies 자원이 강한 종속성을 가진 동일한 그룹 또는 여러 그룹의 자원 목록입니다. 목록의 자원이 온라인 상태가 아닌 경우 이 자원을 시작할 수 없습니다. 이 자원과 목록에 있는 자원 중 하나가 동시에 시작되면 RGM은 목록에 있는 자원이 시작될 때까지 이 자원의 시작을 대기합니다. 이 자원의 Resource_dependencies 목록에 있는 자원이 시작되지 않으면 이 자원은 계속 오프라인 상태로 있습니다. 목록에 있는 자원의 자원 그룹이 오프라인 상태이거나 START_FAILED 상태에 있어서 이 자원 목록의 자원이 시작되지 않을 수도 있습니다. 시작되지 않은 다른 자원 그룹의 자원에 대한 종속성 때문에 이 자원이 오프라인 상태로 유지되면 이 자원 그룹은 PENDING_ONLINE_BLOCKED 상태가 됩니다.

이 자원이 목록의 자원과 동시에 오프라인화되면 이 자원이 목록의 자원보다 먼저 중지됩니다. 그러나 이 자원이 온라인 상태를 유지하거나 중지하는 데 실패하면 목록 중 다른 자원 그룹에 속하는 자원은 중지합니다. 이 자원이 먼저 비활성화되지 않으면 목록의 자원은 비활성화될 수 없습니다.

기본적으로 자원 그룹에서 응용 프로그램 자원은 네트워크 주소 자원에 대해 강한 암시적 자원 종속성을 갖습니다. 245 페이지 "자원 그룹 등록 정보"의 Implicit_network_dependencies에는 추가 정보가 포함되어 있습니다.

자원 그룹 내에서 Prenet_start 메소드는 종속 순서에 따라 Start 메소드보다 먼저 실행됩니다. Postnet_stop 메소드는 Stop 메소드 후에 종속성 순서대로 실행됩니다. 여러 자원 그룹에서 종속 자원은 Prenet_start를 실행하기 전에 종속 대상 자원이 Prenet_start 및 Start를 완료할 때까지 대기합니다. 종속 대상 자원은 Stop을 실행하기 전에 종속 자원이 Stop 및 Postnet_stop을 완료할 때까지 대기합니다.

범주: 선택적

기본값: 빈 목록

조정 가능: ANYTIME

Resource_dependencies_restart(string_array)
Resource_dependencies_restart 자원이 재시작 종속성을 가진 동일한 그룹 또는 여러 그룹의 자원 목록입니다.

이 등록 정보는 Resource_dependencies처럼 작동하지만 한 가지 예외가 있습니다. 재시작 종속성 목록에 있는 자원이 재시작되면 이 자원도 재시작됩니다. RGM은 목록에 있는 자원이 온라인 상태로 돌아온 후 이 자원을 재시작합니다.

범주: 선택적

기본값: 빈 목록

조정 가능: ANYTIME

Resource_dependencies_weak(string_array)
Resource_dependencies_weak 자원이 약한 종속성을 가진 동일한 그룹 또는 여러 그룹의 자원 목록입니다. 약한 종속성은 메소드 호출 순서를 결정합니다. RGM은 이 자원의 Start 메소드 전에 이 목록에 있는 자원의 Start 메소드를 호출합니다. 또한 목록에 있는 자원의 Stop 메소드 전에 이 자원의 Stop 메소드를 호출합니다. 목록의 자원이 시작하지 못하거나 오프라인 상태를 유지하더라도 이 자원은 시작할 수 있습니다.

이 자원과 Resource_dependencies_weak 목록에 있는 자원이 동시에 시작되면 RGM은 목록에 있는 자원이 시작될 때까지 이 자원의 시작을 대기합니다. 목록에 있는 자원이 시작되지 않으면(예: 목록에 있는 자원의 자원 그룹이 오프라인 상태로 유지되거나 목록에 있는 자원이 START_FAILED 상태인 경우) 이 자원이 시작됩니다. 이 자원의 Resource_dependencies_weak 목록에 있는 자원이 시작되면 이 자원의 자원 그룹이 일시적으로 PENDING_ONLINE_BLOCKED 상태가 될 수 있습니다. 목록에 있는 모든 자원이 시작되거나 시작되지 않으면 이 자원이 시작되고 해당 그룹이 다시 PENDING_ONLINE 상태가 됩니다.

이 자원이 목록에 있는 자원과 동시에 오프라인 상태가 되면 이 자원이 목록에 있는 자원보다 먼저 중지됩니다. 이 자원이 온라인 상태로 유지되거나 중지하는 데 실패하더라도 목록에 있는 자원은 중지됩니다. 이 자원이 비활성화되지 않으면 목록에 있는 자원을 비활성화할 수 없습니다.

자원 그룹 내에서 `Prenet_start` 메소드는 종속 순서에 따라 `Start` 메소드보다 먼저 실행됩니다. `Postnet_stop` 메소드는 `Stop` 메소드 후에 종속성 순서대로 실행됩니다. 여러 자원 그룹에서 종속 자원은 `Prenet_start`를 실행하기 전에 종속 대상 자원이 `Prenet_start` 및 `Start`를 완료할 때까지 대기합니다. 종속 대상 자원은 `Stop`을 실행하기 전에 종속 자원이 `Stop` 및 `Postnet_stop`을 완료할 때까지 대기합니다.

범주: 선택적

기본값: 빈 목록

조정 가능: `ANYTIME`

`Resource_name(string)`
자원 인스턴스의 이름입니다. 이 이름은 클러스터 구성 내부에서 고유해야 하며 자원이 만들어진 후에는 변경할 수 없습니다.

범주: 필수적

기본값: 없음

조정 가능: `NONE`

`Resource_project_name(string)`
해당 자원과 관련된 Solaris 프로젝트 이름입니다. CPU 공유와 자원 풀 같은 Solaris 자원 관리 기능을 클러스터 데이터 서비스에 적용하려면 이 등록 정보를 사용합니다. RGM에서 자원을 온라인으로 가져올 경우 이 프로젝트 이름으로 관련된 프로세스를 시작합니다. 이 등록 정보를 지정하지 않으면 자원이 포함된 자원 그룹의 `RG_project_name` 등록 정보에서 프로젝트 이름을 가져옵니다(`rg_properties`(5) 설명서 페이지 참조). 등록 정보를 지정하지 않은 경우 RGM은 미리 정의된 프로젝트 이름 `default`를 사용합니다. 지정한 프로젝트 이름이 프로젝트 데이터베이스에 있어야 합니다(`projects`(1) 설명서 페이지 및 **System Administration Guide: Solaris Containers-Resource Management and Solaris Zones** 참조).

이 등록 정보는 Solaris 9 릴리스부터 지원됩니다.

주 – 이 등록 정보에 대한 변경 사항은 다음에 자원을 시작할 때 적용됩니다.

범주: 선택적

기본값: null

조정 가능: `ANYTIME`

각 클러스터 노드에 대한 Resource_state(enum)
각 클러스터 노드에 대해 RGM에서 결정한 자원 상태입니다. 가능한 상태는 ONLINE, OFFLINE, START_FAILED , STOP_FAILED, MONITOR_FAILED, ONLINE_NOT_MONITORED, STARTING 및 STOPPING 입니다.

이 등록 정보를 구성할 수 없습니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

Retry_count(integer)
실패한 경우 모니터가 자원을 재시작하기 위해 시도하는 횟수입니다. Retry_count를 초과하면 특정 데이터 서비스와 Failover_mode 등록 정보의 설정에 따라 모니터가 다음 작업 중 하나를 수행할 수 있습니다.

- 자원이 오류 상태에 있는 경우에도 자원 그룹이 현재 기본 노드에서 유지될수 있게 허용
- 자원 그룹의 페일오버를 다른 노드로 요청

등록 정보가 RTR 파일에 선언되어 있는 경우에만 RGM에서 등록 정보를 만들고 클러스터 관리자가 사용할 수 있습니다. 기본값이 RTR 파일에 지정되지 않았다면 이 등록 정보는 선택적입니다.

Tunable 속성이 RTR 파일에 지정되지 않은 경우 등록 정보에 대한 Tunable 값은 WHEN_DISABLED입니다.

주 – 이 등록 정보에 대해 음수 값을 지정하면 모니터가 수에 제한없이 자원을 재시작합니다.

범주: 조건적

기본값: 위의 내용 참조

조정 가능: WHEN_DISABLED

Retry_interval(integer)
장애가 발생한 자원을 재시작하기 위한 시도를 세는 시간(초)입니다. 자원 모니터는 이 등록 정보를 Retry_count와 함께 사용합니다. 등록 정보가 RTR 파일에 선언되어 있는 경우에만 RGM에서 등록 정보를 만들고 클러스터 관리자가 사용할 수 있습니다. 기본값이 RTR 파일에 지정되지 않았다면 이 등록 정보는 선택적입니다.

Tunable 속성이 RTR 파일에 지정되지 않은 경우 등록 정보에 대한 Tunable 값은 WHEN_DISABLED입니다.

범주: 조건적

기본값: 기본값 없음(위의 내용 참조)

조정 가능: WHEN_DISABLED

`Scalable`(`boolean`)
자원이 확장 가능한지, 즉 자원이 Sun Cluster 소프트웨어의 네트워킹 로드 균형 조정 기능을 사용하는지 여부를 나타냅니다.

이 등록 정보가 RTR 파일에 선언된 경우에는 RGM에서 해당 유형의 자원에 대하여 다음과 같은 확장 가능 서비스 등록 정보를 자동으로 만듭니다. `Affinity_timeout`, `Load_balancing_policy`, `Load_balancing_weights`, `Network_resources_used`, `Port_list`, `UDP_affinity` 및 `Weak_affinity`. 이러한 등록 정보는 RTR 파일에 명시적으로 선언되지 않았다면 기본값을 사용합니다. `Scalable`의 기본값은 RTR 파일에서 선언된 경우 `TRUE`입니다.

RTR 파일에서 이 등록 정보를 선언하면 `AT_CREATION` 이외의 `Tunable` 속성을 할당할 수 없습니다.

RTR 파일에 등록 정보가 선언되지 않고 자원이 확장 가능하지 않을 경우 이 등록 정보를 조정할 수 없고 RGM에서 확장 가능 서비스 등록 정보를 설정하지 않습니다. 그러나 RTR 파일에서 `Network_resources_used` 및 `Port_list` 등록 정보를 명시적으로 선언할 수 있습니다. 이러한 등록 정보는 확장 가능 서비스와 확장 불가능 서비스에서 유용할 수 있습니다.

`Failover` 자원 유형 등록 정보와 조합된 이 자원 등록 정보의 사용에 대해서는 `r_properties`(5) 설명서 페이지에서 자세히 설명합니다.

범주: 선택적

기본값: 없음

조정 가능: `AT_CREATION`

유형의 각 콜백 메소드에 대한 `Start_timeout`(`integer`)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 `3600`(1시간)

조정 가능: `ANYTIME`

각 클러스터 노드에 대한 `Status`(`enum`)
자원 모니터에서 `scha_resource_setstatus` 명령 또는 `scha_resource_setstatus()` 함수를 사용하여 설정합니다. 가능한 값은 `OK`, `degraded`, `faulted`, `unknown` 및 `offline`입니다. 자원이 온라인 또는 오프라인 상태가 되면 RGM은 자원의 모니터 또는 메소드에서 `Status` 값을 설정하지 않은 경우 `Status` 값을 자동으로 설정합니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: `NONE`

각 클러스터 노드에 대한 Status_msg(string)
자원 모니터에 의해 Status 등록 정보와 동시에 설정됩니다. 자원이 온라인 또는 오프라인으로 전환될 때 자원의 메소드가 이 등록 정보를 설정하지 않는 경우 RGM이 자동으로 이 등록 정보를 빈 문자열로 재설정합니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

유형의 각 콜백 메소드에 대한 Stop_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

Thorough_probe_interval(integer)
자원에 대한 높은 오버헤드 오류 검사 호출 사이의 시간(초)입니다. 등록 정보가 RTR 파일에 선언되어 있는 경우에만 RGM에서 등록 정보를 만들고 클러스터 관리자가 사용할 수 있습니다. 기본값이 RTR 파일에 지정되지 않았다면 이 등록 정보는 선택적입니다.

Tunable 속성이 RTR 파일에 지정되지 않은 경우 등록 정보에 대한 Tunable 값은 WHEN_DISABLED입니다.

범주: 조건적

기본값: 없음

조정 가능: WHEN_DISABLED

Type(string)
이 자원이 인스턴스인 자원 유형입니다.

범주: 필수적

기본값: 없음

조정 가능: NONE

Type_version(string)
현재 이 자원과 관련된 자원 유형의 버전을 지정합니다. RGM은 RTR 파일에서 선언할 수 없는 이 등록 정보를 자동으로 만듭니다. 이 등록 정보 값은 해당 자원 유형의 RT_version 등록 정보와 같습니다. 자원을 만들 경우 Type_version 등록 정보는 자원 유형 이름의 접미어로 표시될 수는 있지만 명시적으로 지정되지는 않습니다. 자원을 편집할 경우 Type_version 등록 정보를 새 값으로 변경할 수 있습니다.

이 등록 정보의 조정 가능 여부는 다음 소스에 의해 결정됩니다.

- 자원 유형의 현재 버전

- RTR 파일의 #$upgrade_from 지시어

범주: 설명 참조

기본값: 없음

조정 가능: 설명 참조

UDP_affinity(boolean)
이 등록 정보를 TRUE로 설정하면 지정한 클라이언트의 모든 UDP 트래픽을 현재 클라이언트의 모든 TCP 트래픽을 처리하는 서버 노드로 보냅니다.

Load_balancing_policy가 Lb_sticky 또는 Lb_sticky_wild일 경우에만 이 등록 정보가 해당됩니다. 또한 Weak_affinity를 FALSE로 설정해야 합니다.

이 등록 정보는 확장 가능 서비스에서만 사용합니다.

범주: 선택적

기본값: 없음

조정 가능: WHEN_DISABLED

유형의 각 콜백 메소드에 대한 Update_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

유형의 각 콜백 메소드에 대한 Validate_timeout(integer)
RGM이 메소드의 호출이 실패했다고 결정할 때까지의 경과 시간(초)입니다. 지정한 자원 유형에서 시간 초과 등록 정보는 RTR 파일에서 선언된 메소드에 대해서만 정의됩니다.

범주: 조건적 또는 선택적

기본값: 메소드가 RTR 파일에 선언된 경우 3600(1시간)

조정 가능: ANYTIME

Weak_affinity(boolean)
이 등록 정보를 TRUE로 설정하면 약한 형태의 클라이언트 유사성이 활성화됩니다. 약한 형태의 클라이언트 유사성을 사용하면 다음과 같은 경우를 제외하고, 지정된 클라이언트의 연결을 동일한 서버 노드로 전송할 수 있습니다.

- Server Listener가 오류 모니터 재시작, 자원의 페일오버 또는 전환, 실패 후 노드의 클러스터 다시 결합 등에 응답하여 시작되는 경우
- 클러스터 관리자가 관리 작업을 수행했기 때문에 확장 가능 자원에 대한 Load_balancing_weights가 변경되는 경우

약한 유사성은 메모리 사용 및 프로세서 사이클 측면에서 기본 형태에 대한 낮은 오버헤드 대안을 제공합니다.

Load_balancing_policy가 Lb_sticky 또는 Lb_sticky_wild일 경우에만 이 등록 정보가 해당됩니다.

이 등록 정보는 확장 가능 서비스에서만 사용합니다.

범주: 선택적

기본값: 없음

조정 가능: WHEN_DISABLED

자원 그룹 등록 정보

다음 정보에서는 Sun Cluster 소프트웨어에서 정의한 자원 그룹 등록 정보에 대해 설명합니다. 등록 정보 값의 범주는 다음과 같습니다.

- **필수적**. 클러스터 관리자가 관리 유틸리티를 사용하여 자원 그룹을 생성할 때 반드시 값을 지정해야 합니다.
- **선택적**. 클러스터 관리자가 자원 그룹을 생성할 때 값을 지정하지 않으면 시스템에서 기본값을 지정합니다.
- **쿼리 전용**. 관리 도구에서 직접 설정할 수 없습니다.

등록 정보 이름이 먼저 표시되고 그 뒤에 설명이 표시됩니다.

Auto_start_on_new_cluster(boolean)
이 등록 정보는 새 클러스터가 구성될 때 Resource Group Manager(RGM)가 자동으로 자원 그룹을 시작할 것인지를 제어합니다. 기본값은 TRUE입니다.

TRUE로 설정하면 클러스터의 모든 노드가 동시에 다시 부트될 때 RGM이 자원 그룹을 자동으로 시작하여 Desired_primaries를 만듭니다.

FALSE로 설정하면 클러스터가 다시 부트될 때 자원 그룹이 자동으로 재시작되지 않습니다. 자원 그룹은 scswitch 명령을 사용하거나 이와 동등한 GUI 명령을 사용하여 처음으로 자원 그룹을 온라인 상태로 수동 전환할 때까지 오프라인 상태로 유지됩니다. 전환 후 자원 그룹은 정상적인 페일오버 동작을 계속합니다.

범주: 선택적

기본값: TRUE

조정 가능: ANYTIME

Desired_primaries(integer)
그룹이 동시에 실행될 수 있는 선호하는 노드 수입니다.

기본값은 1입니다. RG_mode 등록 정보가 Failover인 경우 이 등록 정보의 값은 1보다 작아야 합니다. RG_mode 등록 정보가 Scalable일 경우 1보다 큰 값이 허용됩니다.

범주: 선택적

기본값: 1

조정 가능: ANYTIME

`Failback(boolean)`
클러스터 구성원 자격이 변경될 때 그룹이 온라인 상태인 노드 집합을 재계산할지 여부를 나타내는 부울 값입니다. 재계산을 하면 RGM에서 해당 그룹으로 하여금 덜 선호하는 노드에서는 오프라인이, 그리고 더 선호하는 노드에서는 온라인이 되도록 합니다.

범주: 선택적

기본값: FALSE

조정 가능: ANYTIME

`Global_resources_used(string_array)`
클러스터 파일 시스템이 이 자원 그룹의 자원에 의해 사용되는지를 가리킵니다. 클러스터 관리자가 지정할 수 있는 유효한 값은 모든 전역 자원을 가리키는 별표(*)와 어떠한 전역 자원도 가리키지 않는 빈 문자열(“”)입니다.

범주: 선택적

기본값: 모든 전역 자원

조정 가능: ANYTIME

`Implicit_network_dependencies(boolean)`
TRUE일 경우 RGM이 그룹 내 네트워크 주소 자원에서 비네트워크 주소 자원의 암시적인 강력한 종속성을 적용해야 함을 나타내는 부울 값입니다. 즉, RGM은 그룹 내의 다른 모든 자원보다 먼저 네트워크 주소 자원을 시작하고, 또한 다른 모든 자원보다 나중에 네트워크 주소 자원을 중지합니다. 네트워크 주소 자원은 논리 호스트 이름과 공유 주소 자원 유형을 포함합니다.

확장 가능 자원 그룹은 네트워크 주소 자원을 포함하지 않기 때문에 이 등록 정보가 확장 가능 등록 그룹에 영향을 미치지 않습니다.

범주: 선택적

기본값: TRUE

조정 가능: ANYTIME

`Maximum_primaries(integer)`
그룹이 동시에 온라인 상태가 될 수 있는 최대 노드 수입니다.

RG_mode 등록 정보가 Failover인 경우 이 등록 정보의 값은 1보다 작아야 합니다. RG_mode 등록 정보가 Scalable일 경우 1보다 큰 값이 허용됩니다.

범주: 선택적

기본값: 1

조정 가능: ANYTIME

Nodelist(string_array)
우선 순위에 따라 그룹이 온라인 상태가 될 수 있는 클러스터 노드의 목록입니다. 이러한 노드들은 기본 노드가 될 수 있는 노드나 자원 그룹의 마스터로 알려져 있습니다.

범주: 선택적

기본값: 임의의 순서로 정렬된 모든 클러스터 노드 목록

조정 가능: ANYTIME

Pathprefix(string)
그룹의 자원이 필수적인 관리 파일을 쓸 수 있는 클러스터 파일 시스템의 디렉토리. 몇몇 자원에는 이 등록 정보가 필요할 수 있습니다. 각 자원 그룹에 대해 Pathprefix를 고유하게 하십시오.

범주: 선택적

기본값: 빈 문자열

조정 가능: ANYTIME

Pingpong_interval(integer)
RGM에서 다음 인스턴스의 자원 그룹을 온라인 상태로 만들 위치를 결정하는 데 사용하는 음수가 아닌 정수 값(초)입니다.

- 재구성 시
- scha_control GIVEOVER 명령 또는 함수의 실행 결과로

재구성 시 자원 그룹이 특정 노드에서 지난 Pingpong_interval초 내에 두 번 이상 온라인 상태가 되지 못할 수 있습니다. 이 실패는 자원의 Start 또는 Prenet_start 메소드가 0이 아닌 상태로 종료했거나 시간 초과되었기 때문입니다. 그 결과, 해당 노드는 자원 그룹을 호스트하기에 부적절한 것으로 간주되고 RGM은 다른 마스터를 찾습니다.

지정한 노드에서 자원이 scha_control 명령 또는 scha_control GIVEOVER 명령을 실행하여 자원 그룹이 다른 노드로 페일오버된 경우 scha_control이 실행된 첫 번째 노드는 Pingpong_interval 초가 경과할 때까지 동일한 자원에 의한 다른 scha_control GIVEOVER의 대상이 될 수 없습니다.

범주: 선택적

기본값: 3600(1시간)

조정 가능: ANYTIME

Resource_list(string_array)
그룹에 포함된 자원 목록입니다. 클러스터 관리자는 이 등록 정보를 직접 설정하지 않습니다. 그 대신 클러스터 관리자가 자원 그룹에서 자원을 추가하거나 제거할 때 RGM이 이 등록 정보를 업데이트합니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

RG_affinities(string)
RGM은 다른 자원 그룹의 현재 마스터인 노드(양수 유사성) 또는 지정한 자원 그룹의 현재 마스터가 아닌 노드(음수 유사성)에서 자원 그룹을 찾습니다.

RG_affinities를 다음 문자열로 설정할 수 있습니다.

- ++ 또는 강한 양수 유사성
- + 또는 약한 양수 유사성
- - 또는 약한 음수 유사성
- -- 또는 강한 음수 유사성
- +++ 또는 페일오버 위임이 있는 강한 양수 유사성

예를 들어, RG_affinities=+RG2,--RG3은 이 자원 그룹에 RG2에 대한 약한 양수 유사성이 있고 RG3에 대한 강한 음수 유사성이 있음을 나타냅니다.

RG_affinities 사용에 대해서는 **Sun Cluster Data Services Planning and Administration Guide for Solaris OS**의 2 장, "Administering Data Service Resources"에서 설명합니다.

범주: 선택적

기본값: 빈 문자열

조정 가능: ANYTIME

RG_dependencies(string_array)
동일한 노드에서 다른 그룹을 온라인 또는 오프라인 상태로 만드는 우선 순서를 나타내는 선택적 자원 그룹 목록입니다. 모든 강한 RG_affinities(양수 및 음수)와 RG_dependencies가 결합된 그래프는 주기를 포함할 수 없습니다.

예를 들어 자원 그룹 RG2가 자원 그룹 RG1의 RG_dependencies 목록에 나열되어 있다고 가정합니다. 즉, RG1은 RG2에 대해 자원 그룹 종속성을 가지고 있습니다. 다음 목록에는 이 자원 그룹 종속성의 영향이 요약되어 있습니다.

- 노드가 클러스터에 결합될 때 해당 노드의 모든 Boot 메소드가 RG2의 자원에서 완료할 때까지는 RG1의 어떤 자원에 대해서도 Boot 메소드가 실행되지 않습니다.
- RG1과 RG2가 동일한 노드에서 동시에 PENDING_ONLINE 상태이면 RG2의 모든 자원이 시작 메소드를 완료할 때까지 RG1의 모든 자원에서 시작 메소드(Prenet_start 또는 Start)가 실행되지 않습니다.
- RG1과 RG2가 동일한 노드에서 동시에 PENDING_OFFLINE 상태이면 RG1의 모든 자원이 중지 메소드를 완료할 때까지 RG2의 모든 자원에서 중지 메소드(Stop 또는 Postnet_stop)가 실행되지 않습니다.
- RG1을 임의의 노드에서 온라인 상태로 두고 RG2를 모든 노드에서 오프라인 상태로 두는 경우 RG1이나 RG2의 기본 노드 전환 시도가 실패합니다. The scswitch(1M) 및 scsetup(1M) 설명서 페이지에 추가 정보가 포함되어 있습니다.
- RG2에서 Desired_primaries를 0으로 설정하면 RG1에서 Desired_primaries 등록 정보를 0보다 큰 값으로 설정할 수 없습니다.
- RG2에서 Auto_start_on_new_cluster가 FALSE로 설정되어 있는 경우 RG1에서 Auto_start_on_new_cluster 등록 정보를 TRUE로 설정할 수 없습니다.

범주: 선택적

기본값: 빈 목록

조정 가능: ANYTIME

RG_description(string)
자원 그룹에 대한 간단한 설명입니다.

범주: 선택적

기본값: 빈 문자열

조정 가능: ANYTIME

RG_is_frozen(boolean)
자원 그룹이 종속된 전역 장치가 스위치오버되는지 여부를 나타내는 부울 값이 등록 정보를 TRUE로 설정하면 전역 장치가 스위치오버됩니다. 이 등록 정보를 FALSE로 설정하면 전역 장치가 스위치오버되지 않습니다. 자원 그룹은 Global_resources_used 등록 정보에 표시된 전역 장치에 종속됩니다.

RG_is_frozen 등록 정보를 직접 설정하지 마십시오. 전역 장치의 상태가 변경되면 RGM에서 RG_is_frozen 등록 정보를 자동으로 업데이트합니다.

범주: 선택적

기본값: 없음

조정 가능: NONE

RG_mode(enum)
자원 그룹이 페일오버 또는 확장 가능한 그룹인지를 나타냅니다. 값이 Failover일 경우 RGM에서 그룹의 Maximum_primaries 등록 정보를 1로 설정하고 단일 노드에서 마스터할 자원 그룹을 제한합니다.

이 등록 정보 값이 Scalable일 경우 RGM은 Maximum_primaries 등록 정보가 1보다 큰 값으로 설정되도록 허용합니다. 따라서 여러 노드에서 동시에 그룹을 마스터할 수 있습니다. RGM은 Failover 등록 정보가 TRUE인 자원을 RG_mode가 Scalable인 자원 그룹에 추가하도록 허용하지 않습니다.

If Maximum_primaries가 1이면 기본값은 Failover입니다. Maximum_primaries가 1보다 큰 경우 기본값은 Scalable입니다.

범주: 선택적

기본값: Maximum_primaries의 값에 따라 다릅니다.

조정 가능: NONE

RG_name(string)
자원 그룹의 이름입니다. 이 등록 정보는 필수적이며 클러스터 내에서 고유해야 합니다.

범주: 필수적

기본값: 없음

조정 가능: NONE

RG_project_name(string)
자원 그룹과 관련된 Solaris 프로젝트 이름(projects(1) 설명서 페이지 참조)입니다. CPU 공유와 자원 풀 같은 Solaris 자원 관리 기능을 클러스터 데이터 서비스에 적용하려면 이 등록 정보를 사용합니다. RGM은 자원 그룹을 온라인 상태로 만들 때 Resource_project_name 등록 정보가 설정되지 않은 자원에 대해 이 프로젝트 이름으로 관련 프로세스를 시작합니다(r_properties(5) 설명서 페이지 참조). 지정한 프로젝트 이름이 프로젝트 데이터베이스에 있어야 합니다(projects(1) 설명서 페이지 및 **System Administration Guide: Solaris Containers-Resource Management and Solaris Zones** 참조).

이 등록 정보는 Solaris 9 릴리스부터 지원됩니다.

주 – 이 등록 정보에 대한 변경 사항은 다음에 자원을 시작할 때 적용됩니다.

범주: 선택적

기본값: 텍스트 문자열 “default”

조정 가능: ANYTIME

각 클러스터 노드에 대한 RG_state(enum)
각 클러스터 노드에 대한 그룹 상태를 설명하기 위해 RGM에서 UNMANAGED, ONLINE, OFFLINE, PENDING_ONLINE, PENDING_OFFLINE, ERROR_STOP_FAILED, ONLINE_FAULTED 또는 PENDING_ONLINE_BLOCKED로 설정합니다.

이 등록 정보를 구성할 수 없습니다. 그러나 scswitch 명령을 실행하거나 이와 동등한 scsetup 또는 SunPlex Manager 명령을 사용하여 이 등록 정보를 간접적으로 설정할 수 있습니다. 그룹이 RGM의 제어를 받지 않으면 UNMANAGED 상태가 될 수 있습니다.

각 상태에 대한 요약 설명은 다음과 같습니다.

주 – 모든 노드에서 적용되는 UNMANAGED 상태를 제외하고 상태는 개별 노드에만 적용됩니다. 예를 들어 노드 A에서는 자원 그룹이 OFFLINE이고 노드 B에서는 PENDING_ONLINE일 수 있습니다.

UNMANAGED — 새로 만든 자원 그룹의 초기 상태이거나 이전 관리 대상 자원 그룹의 상태입니다. Init 메소드가 그룹의 자원에서 아직 실행되지 않았거나 Fini 메소드가 그룹의 자원에서 실행되었습니다.

이 그룹은 RGM에서 관리되지 않습니다.

ONLINE	자원 그룹이 노드에서 시작되었습니다. 즉, 각 자원에 해당하는 시작 메소드 Prenet_start, Start 및 Monitor_start가 그룹에서 활성화된 모든 자원에 대해 성공적으로 실행되었습니다.
OFFLINE	자원 그룹이 노드에서 중지되었습니다. 즉, 각 자원에 해당하는 중지 메소드 Monitor_stop, Stop 및 Postnet_stop이 그룹에서 활성화된 모든 자원에 대해 성공적으로 실행되었습니다. 이 상태는 노드에서 자원 그룹이 처음 시작되기 전에도 적용됩니다.
PENDING_ONLINE	자원 그룹이 노드에서 시작 중입니다. 각 자원에 해당하는 시작 메소드 Prenet_start, Start, and Monitor_start가 그룹에서 활성화된 자원에 대해 실행 중입니다.
PENDING_OFFLINE	자원 그룹이 노드에서 중지 중입니다. 각 자원에 해당하는 중지 메소드 Monitor_stop, Stop 및 Postnet_stop이 그룹에서 활성화된 자원에 대해 실행 중입니다.
ERROR_STOP_FAILED	자원 그룹 내의 자원이 한 개 이상 성공적으로 중지되지 않았으며 Stop_failed 상태입니다. 그룹 내의 다른 자원은 온라인 또는 오프라인 상태로 유지될 수 있습니다. 이 자원 그룹은 ERROR_STOP_FAILED 상태가 해제될 때까지 모든 노드에서 시작할 수 없습니다. Stop_failed 자원을 수동으로 강제 종료하고 상태를 OFFLINE으로 재설정하려면 scswitch -c와 같은 관리 명령을 사용해야 합니다.
ONLINE_FAULTED	자원 그룹이 PENDING_ONLINE 상태였으며 이 노드에서 시작을 완료했습니다. 그러나 한 개 이상의 자원이 Start_failed 상태 또는 Faulted 상태로 종료되었습니다.
PENDING_ONLINE_BLOCKED	다른 자원 그룹의 자원에 대한 강한 자원 종속성이 만족되지 않은 자원이 자원 그룹에 한 개 이상 있으므로 자원 그룹이

완전히 시작되지 않았습니다. 해당 자원은 OFFLINE 상태로 유지됩니다. 자원 종속성이 만족되면 해당 자원 그룹은 자동으로 PENDING_ONLINE 상태로 돌아갑니다.

범주: 쿼리 전용

기본값: 없음

조정 가능: NONE

RG_system(boolean)
자원 그룹에 대한 RG_system 등록 정보가 TRUE이면 해당 자원 그룹 및 자원 그룹에 포함된 자원에 대해 특정 작업이 제한됩니다. 이 제한은 중요 자원 그룹 및 자원을 실수로 수정하거나 삭제하는 것을 금지하기 위한 것입니다. 이 등록 정보는 scrgadm 및 scswitch 명령에만 영향을 줍니다. scha_control(1HA) 및 scha_control(3HA) 작업은 영향을 받지 않습니다.

자원 그룹 또는 자원 그룹의 자원에서 제한된 작업을 수행하기 전에 해당 자원 그룹의 RG_system 등록 정보를 FALSE로 설정해야 합니다. 클러스터 서비스를 지원하는 자원 그룹을 수정하거나 삭제할 때 또는 해당 자원 그룹에 포함된 자원을 수정하거나 삭제할 때 주의하십시오.

작업	예
자원 그룹 삭제	**scrgadm -r -g RG1**
자원 그룹 등록 정보 편집(RG_system 제외)	**scrgadm -c -t RG1 -y nodelist=...**
자원 그룹에 자원 추가	**scrgadm -a -j R1 -g RG1**
자원 그룹에서 자원 삭제	**scrgadm -r -j R1 -g RG1**
자원 그룹에 속하는 자원의 등록 정보 편집	**scrgadm -c -j R1**
자원 그룹을 오프라인 상태로 전환	**scswitch -F -g RG1**
자원 그룹 관리	**scswitch -o -g RG1**
자원 그룹 관리 해제	**scswitch -u -g RG1**
자원 활성화	**scswitch -e -j R1**
자원에 대한 모니터링 사용 가능	**scswitch -e -M -j R1**
자원 비활성화	**scswitch -n -j R1**
자원에 대한 모니터링 사용 불가	**scswitch -n -M -j R1**

자원 그룹에 대한 RG_system 등록 정보가 TRUE 이면 자원 그룹에 대해 편집 가능한 등록 정보는 RG_system 등록 정보뿐입니다. 즉, RG_system 등록 정보 편집은 결코 제한되지 않습니다.

범주: 선택적

기본값: FALSE

조정 가능: ANYTIME

자원 등록 정보 속성

이 절에서는 시스템 정의 등록 정보를 변경하거나 확장 등록 정보를 만드는 데 사용할 수 있는 자원 등록 정보 속성에 대해 설명합니다.

주의 – boolean, enum 또는 int 유형에 대한 기본값으로 Null이나 빈 문자열(“”)을 지정할 수 없습니다.

등록 정보 이름이 먼저 표시되고 그 뒤에 설명이 표시됩니다.

Array_maxsize
  stringarray 유형의 경우 허용된 배열 요소의 최대 개수입니다.

Array_minsize
  stringarray 유형의 경우 허용된 배열 요소의 최소 개수입니다.

Default
  등록 정보에 대한 기본값을 나타냅니다.

Description
  등록 정보를 간단하게 설명하는 문자열 주석입니다. 시스템에서 정의한 등록 정보에 대해 RTR 파일에 Description 속성을 설정할 수 없습니다.

Enumlist
  enum 유형의 경우 등록 정보에 대해 허용된 문자열 값 세트입니다.

Extension
  사용된 경우, RTR 파일 항목이 자원 유형 구현에 의해 정의된 확장명 등록 정보를 선언하였음을 가리킵니다. 그렇지 않으면 이 항목은 시스템에서 정의한 등록 정보입니다.

Max
  int 유형의 경우 등록 정보에 대해 허용된 최대값입니다.

Maxlength
  string 및 stringarray 유형의 경우 허용된 최대 문자열 길이입니다.

Min
  int 유형의 경우 등록 정보에 대해 허용된 최소값입니다.

Minlength
  string 및 stringarray 유형의 경우 허용된 최소 문자열 길이입니다.

Property
  자원 등록 정보 이름입니다.

Tunable
  클러스터 관리자가 자원에서 이 등록 정보의 값을 설정할 수 있는 시기를 나타냅니다. 클러스터 관리자가 등록 정보를 설정하지 못하도록 NONE 또는 FALSE로 설정할 수 있습니다. 클러스터 관리자의 등록 정보 조정을 허용하는 값은 TRUE 또는 ANYTIME(언제든지), AT_CREATION(자원 생성 시에만) 또는 WHEN_DISABLED(자원이 비활성화된 경우)입니다. "모니터링이 비활성화된 경우" 또는 "오프라인 시"와 같은 다른 조건을 설정하려면 이 속성을 ANYTIME으로 설정하고 Validate 메소드로 자원 상태를 검증합니다.

  다음 절에 표시된 것처럼 기본값은 표준 자원 등록 정보별로 달라집니다. RTR 파일에 지정되어 있지 않은 경우 확장 등록 정보 조정에 대한 기본 설정은 TRUE(ANYTIME)입니다.

등록 정보 유형
  허용 가능한 유형은 string, boolean, integer, enum 및 stringarray입니다. 시스템에서 정의한 등록 정보에 대해 RTR 파일 항목에서 유형 속성을 설정할 수 없습니다. 유형은 조건에 맞는 등록 정보 값과 RTR 파일 항목에서 허용된 유형별 속성을 결정합니다. enum 유형은 문자열 값 세트입니다.

부록 B

샘플 데이터 서비스 코드 목록

이 부록에서는 샘플 데이터 서비스의 각 메소드에 대한 전체 코드를 제공합니다. 또한 이 부록에는 자원 유형 등록(RTR) 파일의 내용이 나열되어 있습니다.

이 부록은 다음 내용으로 구성되어 있습니다.

자원 유형 등록(RTR) 파일 목록

RTR 파일은 클러스터 관리자가 데이터 서비스를 등록할 때 데이터 서비스의 초기 구성을 정의하는 자원 및 자원 유형 등록 정보 선언을 포함합니다.

예 B–1 SUNW.Sample RTR 파일

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

예 **B–1** SUNW.Sample RTR 파일 (계속)

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

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

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

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

Start              = dns_svc_start;
Stop               = dns_svc_stop;

Validate           = dns_validate;
Update             = dns_update;

Monitor_start      = dns_monitor_start;
Monitor_stop       = dns_monitor_stop;
Monitor_check      = dns_monitor_check;

# A list of bracketed resource property declarations follows the
# resource type declarations. The property-name declaration must be
# the first attribute after the open curly bracket of each entry.
#
# The <method>_timeout properties set the value in seconds after which
# the RGM concludes invocation of the method has failed.

# The MIN value for all method timeouts is set to 60 seconds. This
# prevents administrators from setting shorter timeouts, which do not
# improve switchover/failover performance, and can lead to undesired
# RGM actions (false failovers, node reboot, or moving the resource group
# to ERROR_STOP_FAILED state, requiring operator intervention). Setting
# too-short method timeouts leads to a *decrease* in overall availability
# of the data service.
{
        PROPERTY = Start_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Stop_timeout;
        MIN=60;
        DEFAULT=300;
}
{
        PROPERTY = Validate_timeout;
        MIN=60;
        DEFAULT=300;
}
{
```

예 B–1 SUNW.Sample RTR 파일 (계속)

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

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

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

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

#
# Extension Properties
#
```

예 B–1 SUNW.Sample RTR 파일 (계속)

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

# Time out value in seconds before declaring the probe as failed.
{
   PROPERTY = Probe_timeout;
   EXTENSION;
   INT;
   DEFAULT = 30;
   TUNABLE = ANYTIME;
   DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Start 메소드 코드 목록

RGM은 데이터 서비스 자원을 포함하는 자원 그룹이 해당 노드에서 온라인 상태가 되거나 자원이 사용 가능할 때 클러스터 노드에서 Start 메소드를 실행합니다. 샘플 응용 프로그램에서 Start 메소드는 해당 노드의 in.named(DNS) 데몬을 활성화합니다.

예 B–2 dns_svc_start 메소드

```
#!/bin/ksh
#
# Start Method for HA-DNS.
#
# This method starts the data service under the control of PMF. Before starting
# the in.named process for DNS, it performs some sanity checks. The PMF tag for
# the data service is $RESOURCE_NAME.named. PMF tries to start the service a
# specified number of times (Retry_count) and if the number of attempts exceeds
# this value within a specified interval (Retry_interval) PMF reports a failure
# to start the service. Retry_count and Retry_interval are both properties of the
# resource set in the RTR file.


#pragma ident   "@(#)dns_svc_start   1.1   00/05/24 SMI"
```

예 B–2 dns_svc_start 메소드 (계속)

```
################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;

                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;

                esac
        done

}



################################################################################
# MAIN
#
################################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

PMF_TAG=$RESOURCE_NAME.named
```

예 **B–2** dns_svc_start 메소드 (**계속**)

```
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Get the value of the Confdir property of the resource in order to start
# DNS. Using the resource name and the resource group entered, find the value of
# Confdir value set by the cluster administrator when adding theresource.
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get returns the "type" as well as the "value" for the extension
# properties. Get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

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

# Change to the $CONFIG_DIR directory in case there are relative
# path names in the data files.
cd $CONFIG_DIR

# Check that the named.conf file is present in the $CONFIG_DIR directory.
if [ ! -s named.conf ]; then
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
       "${ARGV0} File $CONFIG_DIR/named.conf is missing or empty"
   exit 1
fi

# Get the value for Retry_count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the value for Retry_interval from the RTR file. Convert this value, which is in
# seconds, to minutes for passing to pmfadm. Note that this is a conversion with
# round-up, for example, 50 seconds rounds up to one minute.
((RETRY_INTRVAL = `scha_resource_get -O Retry_interval -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM? 60))

# Start the in.named daemon under the control of PMF. Let it crash and restart
# up to $RETRY_COUNT times in a period of $RETRY_INTERVAL; if it crashes
# more often than that, PMF will cease trying to restart it. If there is a
# process already registered under the tag <$PMF_TAG>, then, PMF sends out
# an alert message that the process is already running.
echo "Retry interval is "$RETRY_INTRVAL
pmfadm -c $PMF_TAG.named -n $RETRY_CNT -t $RETRY_INTRVAL \
    /usr/sbin/in.named -c named.conf

# Log a message indicating that HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} HA-DNS successfully started"
fi
```

예 B–2 dns_svc_start 메소드 *(계속)*

```
exit 0
```

Stop 메소드 코드 목록

RGM은 HA-DNS 자원을 포함하는 자원 그룹이 해당 노드에서 오프라인 상태가 되거나 자원이 비활성화될 때 클러스터 노드에서 Stop 메소드를 실행합니다. 이 메소드는 해당 노드에서 in.named(DNS) 데몬을 중지합니다.

예 B–3 dns_svc_stop 메소드

```
#!/bin/ksh
#
# Stop method for HA-DNS
#
# Stop the data service using PMF. If the service is not running the
# method exits with status 0 as returning any other value puts the resource
# in STOP_FAILED state.


#pragma ident   "@(#)dns_svc_stop   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
```

예 **B–3** dns_svc_stop 메소드 (**계속**)

```
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done

}

###############################################################################
# MAIN
#
###############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# Obtain the Stop_timeout value from the RTR file.
STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT -R $RESOURCE_NAME -G \
$RESOURCEGROUP_NAM?

# Attempt to stop the data service in an orderly manner using a SIGTERM
# signal through PMF. Wait for up to 80% of the Stop_timeout value to
# see if SIGTERM is successful in stopping the data service. If not, send SIGKILL
# to stop the data service. Use up to 15% of the Stop_timeout value to see
# if SIGKILL is successful. If not, there is a failure and the method exits with
# non-zero status. The remaining 5% of the Stop_timeout is for other uses.
((SMOOTH_TIMEOUT=$STOP_TIMEOUT * 80/100))

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

# See if in.named is running, and if so, kill it.
if pmfadm -q $PMF_TAG.named; then
   # Send a SIGTERM signal to the data service and wait for 80% of the
   # total timeout value.
   pmfadm -s $PMF_TAG.named -w $SMOOTH_TIMEOUT TERM
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.info -t [SYSLOG_TAG] \
          "${ARGV0} Failed to stop HA-DNS with SIGTERM; Retry with \
           SIGKILL"

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

예 B–3 dns_svc_stop 메소드 *(계속)*

```
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESSFUL"

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

   # Even if HA-DNS is not running, exit success to avoid putting
   # the data service in STOP_FAILED State.
   exit 0
fi

# Successfully stopped DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

gettime 유틸리티 코드 목록

gettime 유틸리티는 PROBE 프로그램에서 검사 재시작 간의 경과 시간을 추적하기 위해 사용하는 C 프로그램입니다. 이 프로그램을 컴파일하여 콜백 메소드와 동일한 디렉토리, 즉 RT_basedir 등록 정보에서 가리키는 디렉토리에 배치해야 합니다.

예 B–4 gettime 유틸리티

```
# This utility program, used by the probe method of the data service, tracks
# the elapsed time in seconds from a known reference point (epoch point). It
# must be compiled and placed in the same directory as the data service callback
# methods (RT_basedir).

#pragma ident   "@(#)gettime.c   1.1   00/05/24 SMI"

#include <stdio.h>
#include <sys/types.h>
#include <time.h>

main()
{
    printf("%d\n", time(0));
    exit(0);
}
```

PROBE 프로그램 코드 목록

PROBE 프로그램은 nslookup 명령(nslookup(1M) 설명서 페이지 참조)을 사용하여 데이터 서비스의 가용성을 검사합니다. Monitor_start 콜백 메소드는 이 프로그램을 시작하고 Monitor_stop 콜백 메소드는 이 프로그램을 중지합니다.

예 B–5 dns_probe 프로그램

```
#!/bin/ksh
#pragma ident   "@(#)dns_probe   1.1   00/04/19 SMI"
#
# Probe method for HA-DNS.
#
# This program checks the availability of the data service using nslookup, which
# queries the DNS server to look for the DNS server itself. If the server
# does not respond or if the query is replied to by some other server,
# then the probe concludes that there is some problem with the data service
# and fails the service over to another node in the cluster. Probing is done
# at a specific interval set by THOROUGH_PROBE_INTERVAL in the RTR file.

#pragma ident   "@(#)dns_probe   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
```

예 B–5 dns_probe 프로그램 (계속)

```
        done
}

#############################################################################
# restart_service ()
#
# This function tries to restart the data service by calling the Stop method
# followed by the Start method of the dataservice. If the dataservice has
# already died and no tag is registered for the dataservice under PMF,
# then this function fails the service over to another node in the cluster.
#
function restart_service
{
        # To restart the dataservice, first, verify that the
        # dataservice itself is still registered under PMF.
        pmfadm -q $PMF_TAG
        if [[ $? -eq 0 ]]; then
                # Since the TAG for the dataservice is still registered under
                # PMF, first stop the dataservice and start it back up again.
                # Obtain the Stop method name and the STOP_TIMEOUT value for
                # this resource.
                STOP_TIMEOUT=`scha_resource_get -O STOP_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                STOP_METHOD=`scha_resource_get -O STOP \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $STOP_TIMEOUT $RT_BASEDIR/$STOP_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

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

                # Obtain the Start method name and the START_TIMEOUT value for
                # this resource.
                START_TIMEOUT=`scha_resource_get -O START_TIMEOUT \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                START_METHOD=`scha_resource_get -O START \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?
                hatimerun -t $START_TIMEOUT $RT_BASEDIR/$START_METHOD \
                        -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
                        -T $RESOURCETYPE_NAME

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

        else
                # The absence of the TAG for the dataservice
```

예 **B–5** dns_probe 프로그램 *(계속)*

```
                # implies that the dataservice has already
                # exceeded the maximum retries allowed under PMF.
                # Therefore, do not attempt to restart the
                # dataservice again, but try to failover
                # to another node in the cluster.
                scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
                        -R $RESOURCE_NAME
        fi

        return 0
}

#############################################################################
# decide_restart_or_failover ()
#
# This function decides the action to be taken upon the failure of a probe:
# restart the data service locally or fail over to another node in the cluster.
#
function decide_restart_or_failover
{

   # Check if this is the first restart attempt.
   if [ $retries -eq 0 ]; then
         # This is the first failure. Note the time of
         # this first attempt.
         start_time=`$RT_BASEDIR/gettim?
         retries=`expr $retries + 1`
         # Because this is the first failure, attempt to restart
         # the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart data service."
            exit 1
         fi
   else
      # This is not the first failure
      current_time=`$RT_BASEDIR/gettim?
      time_diff=`expr $current_time - $start_tim?
      if [ $time_diff -ge $RETRY_INTERVAL ]; then
         # This failure happened after the time window
         # elapsed, so reset the retries counter,
         # slide the window, and do a retry.
         retries=1
         start_time=$current_time
         # Because the previous failure occurred more than
         # Retry_interval ago, attempt to restart the data service.
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
```

예 B–5 dns_probe 프로그램 (계속)

```
      elif [ $retries -ge $RETRY_COUNT ]; then
         # Still within the time window,
         # and the retry counter expired, so fail over.
         retries=0
         scha_control -O GIVEOVER -G $RESOURCEGROUP_NAME \
             -R $RESOURCE_NAME
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failover attempt failed."
            exit 1
         fi
      else
         # Still within the time window,
         # and the retry counter has not expired,
         # so do another retry.
         retries=`expr $retries + 1`
         restart_service
         if [ $? -ne 0 ]; then
            logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                "${ARGV0} Failed to restart HA-DNS."
            exit 1
         fi
      fi
fi
}

##############################################################################
# MAIN
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# The interval at which probing is to be done is set in the system defined
# property THOROUGH_PROBE_INTERVAL. Obtain the value of this property with
# scha_resource_get
PROBE_INTERVAL=scha_resource_get -O THOROUGH_PROBE_INTERVAL \
-R $RESOURCE_NAME -G $RESOURCEGROUP_NAM?

# Obtain the timeout value allowed for the probe, which is set in the
# PROBE_TIMEOUT extension property in the RTR file. The default timeout for
# nslookup is 1.5 minutes.
probe_timeout_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Probe_timeout`
PROBE_TIMEOUT=`echo $probe_timeout_info | awk '{print $2}'`
```

예 **B–5** dns_probe 프로그램 *(계속)*

```
# Identify the server on which DNS is serving by obtaining the value
# of the NETWORK_RESOURCES_USED property of the resource.
DNS_HOST=`scha_resource_get -O NETWORK_RESOURCES_USED -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the retry count value from the system defined property Retry_count
RETRY_COUNT =`scha_resource_get -O RETRY_COUNT -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Get the retry interval value from the system defined property
Retry_interval
RETRY_INTERVAL=scha_resource_get -O RETRY_INTERVAL -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the full path for the gettime utility from the
# RT_basedir property of the resource type.
RT_BASEDIR=scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# The probe runs in an infinite loop, trying nslookup commands.
# Set up a temporary file for the nslookup replies.
DNSPROBEFILE=/tmp/.$RESOURCE_NAME.probe
probefail=0
retries=0

while :
do
   # The interval at which the probe needs to run is specified in the
   # property THOROUGH_PROBE_INTERVAL. Therefore, set the probe to sleep for a
   # duration of <THOROUGH_PROBE_INTERVAL>
   sleep $PROBE_INTERVAL

   # Run the probe, which queries the IP address on
   # which DNS is serving.
   hatimerun -t $PROBE_TIMEOUT /usr/sbin/nslookup $DNS_HOST $DNS_HOST \
           > $DNSPROBEFILE 2>&1

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

   # Make sure that the reply to nslookup command comes from the HA-DNS
   # server and not from another name server listed in the
   # /etc/resolv.conf file.
   if [ $probefail -eq 0 ]; then
      # Get the name of the server that replied to the nslookup query.
                 SERVER=` awk ' $1=="Server:" {print $2 }' \
                 $DNSPROBEFILE | awk -F. ' { print $1 } ' `
               if [ -z "$SERVER" ];
                   then
                       probefail=1
```

예 B–5 dns_probe 프로그램 (계속)

```
                else
                        if [ $SERVER != $DNS_HOST ]; then
                                probefail=1
                        fi
                fi
        fi

   # If the probefail variable is not set to 0, either the nslookup command
   # timed out or the reply to the query was came from another server
   # (specified in the /etc/resolv.conf file). In either case, the DNS server is
   # not responding and the method calls decide_restart_or_failover,
   # which evaluates whether to restart the data service or to fail it over
   # to another node.

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

Monitor_start 메소드 코드 목록

이 메소드는 데이터 서비스의 PROBE 프로그램을 시작합니다.

예 B–6 dns_monitor_start 메소드

```
#!/bin/ksh
#
# Monitor start Method for HA-DNS.
#
# This method starts the monitor (probe) for the data service under the
# control of PMF. The monitor is a process that probes the data service
# at periodic intervals and if there is a problem restarts it on the same node
# or fails it over to another node in the cluster. The PMF tag for the
# monitor is $RESOURCE_NAME.monitor.

#pragma ident   "@(#)dns_monitor_start   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt
```

예 **B–6** dns_monitor_start 메소드 *(계속)*

```
        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
          logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                     "ERROR: Option $OPTARG unknown"
                      exit 1
                      ;;
                esac
        done

}

##############################################################################
# MAIN
#
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the data service.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Start the probe for the data service under PMF. Use the infinite retries
# option to start the probe. Pass the resource name, group, and type to the
# probe method.
pmfadm -c $PMF_TAG.monitor -n -1 -t -1 \
    $RT_BASEDIR/dns_probe -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
```

예 B–6 dns_monitor_start 메소드 *(계속)*

```
    -T $RESOURCETYPE_NAME

# Log a message indicating that the monitor for HA-DNS has been started.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
           "${ARGV0} Monitor for HA-DNS successfully started"
fi
exit 0
```

Monitor_stop 메소드 코드 목록

이 메소드는 데이터 서비스의 PROBE 프로그램을 중지합니다.

예 B–7 dns_monitor_stop 메소드

```
#!/bin/ksh
# Monitor stop method for HA-DNS
# Stops the monitor that is running using PMF.

#pragma ident   "@(#)dns_monitor_stop   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
```

예 B–7 dns_monitor_stop 메소드 (계속)

```
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done
}

#############################################################################
# MAIN
#
#############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# See if the monitor is running, and if so, kill it.
if pmfadm -q $PMF_TAG.monitor; then
   pmfadm -s $PMF_TAG.monitor KILL
   if [ $? -ne 0 ]; then
      logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
          "${ARGV0} Could not stop monitor for resource " \
          $RESOURCE_NAME
           exit 1
   else
      # Could successfully stop the monitor. Log a message.
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
          "${ARGV0} Monitor for resource " $RESOURCE_NAME \
          " successfully stopped"
   fi
fi
exit 0
```

Monitor_check 메소드 코드 목록

이 메소드는 Confdir 등록 정보에서 가리키는 디렉토리가 있는지 확인합니다. RGM은 PROBE 메소드가 데이터 서비스를 새 노드로 페일오버할 때마다, 그리고 잠재적 마스터인 노드를 검사하기 위해 Monitor_check을 호출합니다.

예 B–8 dns_monitor_check 메소드

```
#!/bin/ksh#
# Monitor check  Method for DNS.
```

예 B–8 dns_monitor_check 메소드 (계속)

```
#
# The RGM calls this method whenever the fault monitor fails the data service
# over to a new node. Monitor_check calls the Validate method to verify
# that the configuration directory and files are available on the new node.

#pragma ident   "@(#)dns_monitor_check 1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
function parse_args # [args ...]
{
   typeset opt

   while getopts 'R:G:T:' opt
   do
      case "$opt" in

      R)
      # Name of the DNS resource.
      RESOURCE_NAME=$OPTARG
      ;;

      G)
      # Name of the resource group in which the resource is
      # configured.
      RESOURCEGROUP_NAME=$OPTARG
      ;;

      T)
      # Name of the resource type.
      RESOURCETYPE_NAME=$OPTARG
      ;;

      *)
      logger -p ${SYSLOG_FACILITY}.err \
      -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
      "ERROR: Option $OPTARG unknown"
      exit 1
      ;;
      esac
   done

}

################################################################################
# MAIN
################################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`
```

예 B–8 dns_monitor_check 메소드 (계속)

```
# Parse the arguments that have been passed to this method.
parse_args "$@"

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

# Obtain the full path for the Validate method from
# the RT_basedir property of the resource type.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the name of the Validate method for this resource.
VALIDATE_METHOD=`scha_resource_get -O VALIDATE -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# Obtain the value of the Confdir property in order to start the
# data service. Use the resource name and the resource group entered to
# obtain the Confdir value set at the time of adding the resource.
config_info=`scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`

# scha_resource_get returns the type as well as the value for extension
# properties. Use awk to get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

# Call the validate method so that the dataservice can be failed over
# successfully to the new node.
$RT_BASEDIR/$VALIDATE_METHOD -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME \
-T $RESOURCETYPE_NAME -x Confdir=$CONFIG_DIR

# Log a message indicating that monitor check was successful.
if [ $? -eq 0 ]; then
   logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS successful."
   exit 0
else
   logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
      "${ARGV0} Monitor check for DNS not successful."
   exit 1
fi
```

Validate 메소드 코드 목록

이 메소드는 Confdir 등록 정보에서 가리키는 디렉토리가 있는지 확인합니다. RGM은 데이터 서비스가 만들어질 때와 데이터 서비스 등록 정보가 클러스터 관리자에 의해 업데이트될 때 이 메소드를 호출합니다. Monitor_check 메소드는 오류 모니터가 데이터 서비스를 새 노드로 페일오버할 때마다 이 메소드를 호출합니다.

예 B–9 dns_validate 메소드

```
#!/bin/ksh
# Validate method for HA-DNS.
# This method validates the Confdir property of the resource. The Validate
# method gets called in two scenarios. When the resource is being created and
# when a resource property is getting updated. When the resource is being
# created, this method gets called with the -c flag and all the system-defined
# and extension properties are passed as command-line arguments. When a resource
# property is being updated, the Validate method gets called with the -u flag,
# and only the property/value pair of the property being updated is passed as a
# command-line argument.
#
# ex: When the resource is being created command args will be
#
# dns_validate -c -R <..> -G <...> -T <..> -r <sysdef-prop=value>...
#       -x <extension-prop=value>.... -g <resourcegroup-prop=value>....
#
# when the resource property is being updated
#
# dns_validate -u -R <..> -G <...> -T <..> -r <sys-prop_being_updated=value>
#   OR
# dns_validate -u -R <..> -G <...> -T <..> -x <extn-prop_being_updated=value>

#pragma ident   "@(#)dns_validate   1.1   00/05/24 SMI"

################################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
   typeset opt

   while getopts 'cur:x:g:R:T:G:' opt
   do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                r)
                        #The method is not accessing any system defined
                        #properties, so this is a no-op.
                        ;;
                g)
                        # The method is not accessing any resource group
```

예 **B–9** dns_validate 메소드 (**계속**)

```
                        # properties, so this is a no-op.
                        ;;
                c)
                        # Indicates the Validate method is being called while
                        # creating the resource, so this flag is a no-op.
                        ;;
                u)
                        # Indicates the updating of a property when the
                        # resource already exists. If the update is to the
                        # Confdir property then Confdir should appear in the
                        # command-line arguments. If it does not, the method must
                        # look for it specifically using scha_resource_get.
                        UPDATE_PROPERTY=1
                        ;;
                x)
                        # Extension property list. Separate the property and
                        # value pairs using "=" as the separator.
                        PROPERTY=`echo $OPTARG | awk -F= '{print $1}'`
                        VAL=echo $OPTARG | awk -F= '{print $2}'`

                        # If the Confdir extension property is found on the
                        # command line, note its value.
                        if [ $PROPERTY == "Confdir" ];
                        then
                        CONFDIR=$VAL
                        CONFDIR_FOUND=1
                        fi
                        ;;
                *)
                        logger -p ${SYSLOG_FACILITY}.err \
                        -t [$SYSLOG_TAG] \
                        "ERROR: Option $OPTARG unknown"
                        exit 1
                        ;;
                esac
   done
}

##############################################################################
# MAIN
#
##############################################################################

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

# Obtain the syslog facility to use to log messages.
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Set the Value of CONFDIR to null. Later, this method retrieves the value
# of the Confdir property from the command line or using scha_resource_get.
CONFDIR=""
UPDATE_PROPERTY=0
CONFDIR_FOUND=0
```

예 B–9 dns_validate 메소드 *(계속)*

```
# Parse the arguments that have been passed to this method.
parse_args "$@"

# If the validate method is being called due to the updating of properties
# try to retrieve the value of the Confdir extension property from the command
# line. Otherwise, obtain the value of Confdir using scha_resource_get.
if ( (( $UPDATE_PROPERTY == 1 )) &&  (( CONFDIR_FOUND == 0 )) ); then
   config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
       -G $RESOURCEGROUP_NAME Confdir`
   CONFDIR=`echo $config_info | awk '{print $2}'`
fi

# Verify that the Confdir property has a value. If not there is a failure
# and exit with status 1.
if [[ -z $CONFDIR ]]; then
   logger -p ${SYSLOG_FACILITY}.err \
       "${ARGV0} Validate method for resource "$RESOURCE_NAME " failed"
   exit 1
fi

# Now validate the actual Confdir property value.

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

# Check that the named.conf file is present in the Confdir directory.
if [ ! -s $CONFDIR/named.conf ]; then
        logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
            "${ARGV0} File $CONFDIR/named.conf is missing or empty"
        exit 1
fi

# Log a message indicating that the Validate method was successful.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
   "${ARGV0} Validate method for resource "$RESOURCE_NAME \
   " completed successfully"

exit 0
```

Update 메소드 코드 목록

RGM은 실행 중인 자원에 해당 등록 정보가 변경되었다는 것을 알리기 위해 Update 메소드를 호출합니다.

예 B–10 dns_update 메소드

```
#!/bin/ksh
# Update method for HA-DNS.
# The actual updates to properties are done by the RGM. Updates affect only
# the fault monitor so this method must restart the fault monitor.

#pragma ident   "@(#)dns_update   1.1   00/05/24 SMI"

###############################################################################
# Parse program arguments.
#
function parse_args # [args ...]
{
        typeset opt

        while getopts 'R:G:T:' opt
        do
                case "$opt" in
                R)
                        # Name of the DNS resource.
                        RESOURCE_NAME=$OPTARG
                        ;;
                G)
                        # Name of the resource group in which the resource is
                        # configured.
                        RESOURCEGROUP_NAME=$OPTARG
                        ;;
                T)
                        # Name of the resource type.
                        RESOURCETYPE_NAME=$OPTARG
                        ;;
                *)
                    logger -p ${SYSLOG_FACILITY}.err \
                    -t [$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME] \
                    "ERROR: Option $OPTARG unknown"
                     exit 1
                     ;;
                esac
        done
}
###############################################################################
# MAIN
###############################################################################
export PATH=/bin:/usr/bin:/usr/cluster/bin:/usr/sbin:/usr/proc/bin:$PATH

# Obtain the syslog facility to use to log messages.
```

예 B–10 dns_update 메소드 *(계속)*

```
SYSLOG_FACILITY=`scha_cluster_get -O SYSLOG_FACILITY`

# Parse the arguments that have been passed to this method
parse_args "$@"

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

# Find where the probe method resides by obtaining the value of the
# RT_basedir property of the resource.
RT_BASEDIR=`scha_resource_get -O RT_basedir -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAM?

# When the Update method is called, the RGM updates the value of the property
# being updated. This method must check if the fault monitor (probe)
# is running, and if so, kill it and then restart it.
if pmfadm -q $PMF_TAG.monitor; then

# Kill the monitor that is running already
        pmfadm -s $PMF_TAG.monitor TERM
        if [ $? -ne 0 ]; then
                logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                    "${ARGV0} Could not stop the monitor"
                exit 1
        else
                # Could successfully stop DNS. Log a message.
                logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully stopped"
        fi
   # Restart the monitor.
   pmfadm -c $PMF_TAG.monitor -n -1 -t -1 $RT_BASEDIR/dns_probe \
      -R $RESOURCE_NAME -G $RESOURCEGROUP_NAME -T $RESOURCETYPE_NAME
   if [ $? -ne 0 ]; then
          logger -p ${SYSLOG_FACILITY}.err -t [$SYSLOG_TAG] \
                  "${ARGV0} Could not restart monitor for HA-DNS "
      exit 1
   else
      logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
                    "Monitor for HA-DNS successfully restarted"
   fi
fi
exit 0
```

부록 C

DSDL 샘플 자원 유형 코드 목록

이 부록에서는 SUNW.xfnts 자원 유형의 각 메소드에 대한 전체 코드를 제공합니다. 이 부록에는 콜백 메소드가 호출하는 서브루틴에 대한 코드가 들어 있는 xfnts.c 목록이 포함되어 있습니다. 8 장에서 샘플 자원 유형 SUNW.xfnts에 대해 자세히 설명합니다.

이 부록은 다음 내용으로 구성되어 있습니다.

xfnts.c 파일 목록

이 파일은 SUNW.xfnts 메소드에서 호출하는 서브루틴을 구현합니다.

예 C–1 xfnts.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
 * to probe the health of the data service. The probe just returns either
```

예 C–1 xfnts.c (계속)

```
 * success or failure. Action is taken based on this returned value in the
 * method found in the file xfnts_probe.c
 *
 */

#pragma ident "@(#)xfnts.c 1.47 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <scha.h>
#include <rgm/libdsdev.h>
#include <errno.h>
#include "xfnts.h"

/*
 * The initial timeout allowed for the HAXFS data service to
 * be fully up and running. We will wait for 3 % (SVC_WAIT_PCT)
 * of the start_timeout time before probing the service.
 */
#define   SVC_WAIT_PCT      3

/*
 * We need to use 95% of probe_timeout to connect to the port and the
 * remaining time is used to disconnect from port in the svc_probe function.
 */
#define   SVC_CONNECT_TIMEOUT_PCT      95

/*
 * SVC_WAIT_TIME is used only during starting in svc_wait().
 * In svc_wait() we need to be sure that the service is up
 * before returning, thus we need to call svc_probe() to
 * monitor the service. SVC_WAIT_TIME is the time between
 * such probes.
 */
#define   SVC_WAIT_TIME      5

/*
 * This value will be used as disconnect timeout, if there is no
 * time left from the probe_timeout.
 */
#define   SVC_DISCONNECT_TIMEOUT_SECONDS      2


/*
 * svc_validate():
 *
```

예 C–1 xfnts.c (계속)

```
 * Do HA-XFS specific validation of the resource configuration.
 *
 * svc_validate will check for the following
 * 1. Confdir_list extension property
 * 2. fontserver.cfg file
 * 3. xfs binary
 * 4. port_list property
 * 5. network resources
 * 6. other extension properties
 *
 * If any of the above validation fails then, Return > 0 otherwise return 0 for
 * success
 */

int
svc_validate(scds_handle_t scds_handle)
{
   char   xfnts_conf[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_net_resource_list_t *snrlp;
   int rc;
   struct stat statbuf;
   scds_port_list_t   *portlist;
   scha_err_t   err;

   /*
    * Get the configuration directory for the XFS dataservice from the
    * confdir_list extension property.
    */
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

   /*
    * Construct the path to the configuration file from the extension
    * property confdir_list. Since HA-XFS has only one configuration
    * we will need to use the first entry of the confdir_list property.
    */
   (void) sprintf(xfnts_conf, "%s/fontserver.cfg", confdirs->str_array[0]);

   /*
    * Check to see if the HA-XFS configuration file is in the right place.
    * Try to access the HA-XFS configuration file and make sure the
    * permissions are set properly
    */
   if (stat(xfnts_conf, &statbuf) != 0) {
      /*
       * suppress lint error because errno.h prototype
```

예 C–1 xfnts.c (계속)

```
        * is missing void arg
        */
       scds_syslog(LOG_ERR,
           "Failed to access file <%s> : <%s>",
           xfnts_conf, strerror(errno));   /*lint !e746 */
       return (1);
   }

   /*
    * Make sure that xfs binary exists and that the permissions
    * are correct. The XFS binary are assumed to be on the local
    * File system and not on the Global File System
    */
   if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
       scds_syslog(LOG_ERR,
           "Cannot access XFS binary : <%s> ", strerror(errno));
       return (1);
   }

   /* HA-XFS will have only port */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
       scds_syslog(LOG_ERR,
           "Could not access property Port_list: %s.",
          scds_error_string(err));
       return (1); /* Validation Failure */
   }

#ifdef TEST
   if (portlist->num_ports != 1) {
       scds_syslog(LOG_ERR,
           "Property Port_list must have only one value.");
       scds_free_port_list(portlist);
       return (1); /* Validation Failure */
   }
#endif

   /*
    * Return an error if there is an error when trying to get the
    * available network address resources for this resource
    */
   if ((err = scds_get_rs_hostnames(scds_handle, &snrlp))
       != SCHA_ERR_NOERR) {
       scds_syslog(LOG_ERR,
           "No network address resource in resource group: %s.",
          scds_error_string(err));
       return (1); /* Validation Failure */
   }

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

예 C–1 xfnts.c (계속)

```
      rc = 1;
      goto finished;
   }

   /* Check to make sure other important extension props are set */
   if (scds_get_ext_monitor_retry_count(scds_handle) <= 0)
   {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_count is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }
   if (scds_get_ext_monitor_retry_interval(scds_handle) <= 0) {
      scds_syslog(LOG_ERR,
          "Property Monitor_retry_interval is not set.");
      rc = 1; /* Validation Failure */
      goto finished;
   }

   /* All validation checks were successful */
   scds_syslog(LOG_INFO, "Successful validation.");
   rc = 0;

finished:
   scds_free_net_list(snrlp);
   scds_free_port_list(portlist);

   return (rc); /* return result of validation */
}

/*
 * svc_start():
 *
 * Start up the X font server
 * Return 0 on success, > 0 on failures.
 *
 * The XFS service will be started by running the command
 * /usr/openwin/bin/xfs -config <fontserver.cfg file> -port <port to listen>
 * XFS will be started under PMF. XFS will be started as a single instance
 * service. The PMF tag for the data service will be of the form
 * <resourcegroupname,resourcename,instance_number.svc>. In case of XFS, since
 * there will be only one instance the instance_number in the tag will be 0.
 */

int
svc_start(scds_handle_t scds_handle)
{
   char    xfnts_conf[SCDS_ARRAY_SIZE];
   char   cmd[SCDS_ARRAY_SIZE];
   scha_str_array_t *confdirs;
   scds_port_list_t    *portlist;
   scha_err_t   err;
```

예 C–1 xfnts.c (계속)

```
   /* get the configuration directory from the confdir_list property */
   confdirs = scds_get_ext_confdir_list(scds_handle);

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

   /* obtain the port to be used by XFS from the Port_list property */
   err = scds_get_port_list(scds_handle, &portlist);
   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Could not access property Port_list.");
      return (1);
   }

   /*
    * Construct the command to start HA-XFS.
    * NOTE: XFS daemon prints the following message while stopping the XFS
    * "/usr/openwin/bin/xfs notice: terminating"
    * In order to suppress the daemon message,
    * the output is redirected to /dev/null.
    */
   (void) sprintf(cmd,
       "/usr/openwin/bin/xfs -config %s -port %d 2>/dev/null",
       xfnts_conf, portlist->ports[0].port);

   /*
    * Start HA-XFS under PMF. Note that HA-XFS is started as a single
    * instance service. The last argument to the scds_pmf_start function
    * denotes the level of children to be monitored. A value of -1 for
    * this parameter means that all the children along with the original
    * process are to be monitored.
    */
   scds_syslog(LOG_INFO, "Issuing a start request.");
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_SVC,
      SCDS_PMF_SINGLE_INSTANCE, cmd, -1);

   if (err == SCHA_ERR_NOERR) {
      scds_syslog(LOG_INFO,
          "Start command completed successfully.");
   } else {
      scds_syslog(LOG_ERR,
          "Failed to start HA-XFS ");
   }

   scds_free_port_list(portlist);
   return (err); /* return Success/failure status */
}


/*
 * svc_stop():
 *
 * Stop the XFS server
 * Return 0 on success, > 0 on failures.
```

예 C–1 xfnts.c (계속)

```
 *
 * svc_stop will stop the server by calling the toolkit function:
 * scds_pmf_stop.
 */
int
svc_stop(scds_handle_t scds_handle)
{
   scha_err_t   err;

   /*
    * The timeout value for the stop method to succeed is set in the
    * Stop_Timeout (system defined) property
    */
   scds_syslog(LOG_ERR, "Issuing a stop request.");
   err = scds_pmf_stop(scds_handle,
       SCDS_PMF_TYPE_SVC, SCDS_PMF_SINGLE_INSTANCE, SIGTERM,
       scds_get_rs_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop HA-XFS.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Successfully stopped HA-XFS.");
   return (SCHA_ERR_NOERR); /* Successfully stopped */
}

/*
 * svc_wait():
 *
 * wait for the data service to start up fully and make sure it is running
 * healthy
 */

int
svc_wait(scds_handle_t scds_handle)
{
   int rc, svc_start_timeout, probe_timeout;
   scds_netaddr_list_t   *netaddr;

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

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

예 C–1 xfnts.c (계속)

```
      return (1);
   }

   /*
    * Get the Start method timeout, port number on which to probe,
    * the Probe timeout value
    */
   svc_start_timeout = scds_get_rs_start_timeout(scds_handle);
   probe_timeout = scds_get_ext_probe_timeout(scds_handle);

   /*
    * sleep for SVC_WAIT_PCT percentage of start_timeout time
    * before actually probing the dataservice. This is to allow
    * the dataservice to be fully up in order to reply to the
    * probe. NOTE: the value for SVC_WAIT_PCT could be different
    * for different data services.
    * Instead of calling sleep(),
    * call scds_svc_wait() so that if service fails too
    * many times, we give up and return early.
    */
   if (scds_svc_wait(scds_handle, (svc_start_timeout * SVC_WAIT_PCT)/100)
      != SCHA_ERR_NOERR) {

      scds_syslog(LOG_ERR, "Service failed to start.");
      return (1);
   }

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

      /*
       * Dataservice is still trying to come up. Sleep for a while
       * before probing again. Instead of calling sleep(),
       * call scds_svc_wait() so that if service fails too
       * many times, we give up and return early.
       */
      if (scds_svc_wait(scds_handle, SVC_WAIT_TIME)
         != SCHA_ERR_NOERR) {
         scds_syslog(LOG_ERR, "Service failed to start.");
         return (1);
      }
```

예 C–1 xfnts.c (계속)

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

}

/*
 * This function starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as <RG-name,RS-name,instance_number.mon>. The restart option
 * of PMF is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

int
mon_start(scds_handle_t scds_handle)
{
   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling MONITOR_START method for resource <%s>.",
      scds_get_resource_name(scds_handle));

   /*
    * The probe xfnts_probe is assumed to be available in the same
    * subdirectory where the other callback methods for the RT are
    * installed. The last parameter to scds_pmf_start denotes the
    * child monitor level. Since we are starting the probe under PMF
    * we need to monitor the probe process only and hence we are using
    * a value of 0.
    */
   err = scds_pmf_start(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, "xfnts_probe", 0);

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to start fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Started the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully started Monitor */
}


/*
 * This function stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on <RG-name_RS-name,instance_number.mon>.
 */

int
```

예 C–1 xfnts.c (계속)

```
mon_stop(scds_handle_t scds_handle)
{

   scha_err_t   err;

   scds_syslog_debug(DBG_LEVEL_HIGH,
      "Calling scds_pmf_stop method");

   err = scds_pmf_stop(scds_handle, SCDS_PMF_TYPE_MON,
       SCDS_PMF_SINGLE_INSTANCE, SIGKILL,
       scds_get_rs_monitor_stop_timeout(scds_handle));

   if (err != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to stop fault monitor.");
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Stopped the fault monitor.");

   return (SCHA_ERR_NOERR); /* Successfully stopped monitor */
}

/*
 * svc_probe(): Do data service specific probing. Return a float value
 * between 0 (success) and 100(complete failure).
 *
 * The probe does a simple socket connection to the XFS server on the specified
 * port which is configured as the resource extension property (Port_list) and
 * pings the dataservice. If the probe fails to connect to the port, we return
 * a value of 100 indicating that there is a total failure. If the connection
 * goes through and the disconnect to the port fails, then a value of 50 is
 * returned indicating a partial failure.
 */
int
svc_probe(scds_handle_t scds_handle, char *hostname, int port, int
timeout)
{
   int  rc;
   hrtime_t   t1, t2;
   int    sock;
   char   testcmd[2048];
   int    time_used, time_remaining;
   time_t      connect_timeout;


   /*
    * probe the dataservice by doing a socket connection to the port
    * specified in the port_list property to the host that is
    * serving the XFS dataservice. If the XFS service which is configured
    * to listen on the specified port, replies to the connection, then
    * the probe is successful. Else we will wait for a time period set
```

예 C–1 xfnts.c (계속)

```
 * in probe_timeout property before concluding that the probe failed.
 */

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

/*
 * the probe makes a connection to the specified hostname and port.
 * The connection is timed for 95% of the actual probe_timeout.
 */
rc = scds_fm_tcp_connect(scds_handle, &sock, hostname, port,
    connect_timeout);
if (rc) {
   scds_syslog(LOG_ERR,
       "Failed to connect to port <%d> of resource <%s>.",
       port, scds_get_resource_name(scds_handle));
   /* this is a complete failure */
   return (SCDS_PROBE_COMPLETE_FAILURE);
}

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

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

time_used = (int)(t2 - t1);

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

time_remaining = timeout - (int)time_used;

/*
 * If all the time is used up, use a small hardcoded timeout
 * to still try to disconnect. This will avoid the fd leak.
 */
if (time_remaining <= 0) {
   scds_syslog_debug(DBG_LEVEL_LOW,
       "svc_probe used entire timeout of "
       "%d seconds during connect operation and exceeded the "
```

예 C–1 xfnts.c (계속)

```
            "timeout by %d seconds. Attempting disconnect with timeout"
            " %d ",
            connect_timeout,
            abs(time_used),
            SVC_DISCONNECT_TIMEOUT_SECONDS);

        time_remaining = SVC_DISCONNECT_TIMEOUT_SECONDS;
    }

    /*
     * Return partial failure in case of disconnection failure.
     * Reason: The connect call is successful, which means
     * the application is alive. A disconnection failure
     * could happen due to a hung application or heavy load.
     * If it is the later case, don't declare the application
     * as dead by returning complete failure. Instead, declare
     * it as partial failure. If this situation persists, the
     * disconnect call will fail again and the application will be
     * restarted.
     */
    rc = scds_fm_tcp_disconnect(scds_handle, sock, time_remaining);
    if (rc != SCHA_ERR_NOERR) {
        scds_syslog(LOG_ERR,
            "Failed to disconnect to port %d of resource %s.",
            port, scds_get_resource_name(scds_handle));
        /* this is a partial failure */
        return (SCDS_PROBE_COMPLETE_FAILURE/2);
    }

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

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

    /*
     * The connection and disconnection to port is successful,
     * Run the fsinfo command to perform a full check of
     * server health.
     * Redirect stdout, otherwise the output from fsinfo
     * ends up on the console.
     */
    (void) sprintf(testcmd,
        "/usr/openwin/bin/fsinfo -server %s:%d > /dev/null",
```

예 C–1 xfnts.c (계속)

```
        hostname, port);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Checking the server status with %s.", testcmd);
   if (scds_timerun(scds_handle, testcmd, time_remaining,
      SIGKILL, &rc) != SCHA_ERR_NOERR || rc != 0) {

      scds_syslog(LOG_ERR,
         "Failed to check server status with command <%s>",
         testcmd);
      return (SCDS_PROBE_COMPLETE_FAILURE/2);
   }
   return (0);
}
```

xfnts_monitor_check 메소드 코드 목록

이 메소드는 기본 자원 유형 구성이 유효한지 확인합니다.

예 C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_check.c - Monitor Check method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * just make a simple validate check on the service
 */

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

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
```

예 C–2 xfnts_monitor_check.c (계속)

```
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc =  svc_validate(scds_handle);
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "monitor_check method "
       "was called and returned <%d>.", rc);

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

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

xfnts_monitor_start 메소드 코드 목록

이 메소드는 xfnts_probe 메소드를 시작합니다.

예 C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_start.c - Monitor Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as RG-name,RS-name.mon. The restart option of PMF
 * is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

int
```

예 C–3 xfnts_monitor_start.c *(계속)*

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

   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = mon_start(scds_handle);

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

   /* Return the result of monitor_start method */
   return (rc);
}
```

xfnts_monitor_stop 메소드 코드 목록

이 메소드는 xfnts_probe 메소드를 중지합니다.

예 C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_stop.c - Monitor Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on RG-name_RS-name.mon.
 */
```

예 C–4 xfnts_monitor_stop.c (계속)

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

   scds_handle_t   scds_handle;
   int    rc;

   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = mon_stop(scds_handle);

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

   /* Return the result of monitor stop method */
   return (rc);
}
```

xfnts_probe 메소드 코드 목록

xfnts_probe 메소드는 응용 프로그램의 가용성을 확인하고 데이터 서비스를 페일오버할 것인지 아니면 재시작할 것인지 결정합니다. xfnts_monitor_start 콜백 메소드는 이 프로그램을 시작하고 xfnts_monitor_stop 콜백 메소드는 이 프로그램을 중지합니다.

예 C–5 xfnts_probe.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_probe.c - Probe for HA-XFS
 */

#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
```

예 C–5 xfnts_probe.c (계속)

```
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"


/*
 * main():
 * Just an infinite loop which sleep()s for sometime, waiting for
 * the PMF action script to interrupt the sleep(). When interrupted
 * It calls the start method for HA-XFS to restart it.
 *
 */

int
main(int argc, char *argv[])
{
   int         timeout;
   int         port, ip, probe_result;
   scds_handle_t      scds_handle;

   hrtime_t      ht1, ht2;
   unsigned long      dt;

   scds_netaddr_list_t *netaddr;
   char   *hostname;

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Get the ip addresses available for this resource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

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

   /*
    * Set the timeout from the X props. This means that each probe
    * iteration will get a full timeout on each network resource
    * without chopping up the timeout between all of the network
    * resources configured for this resource.
    */
```

예 C–5 xfnts_probe.c (계속)

```
   timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {

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

      /*
       * Now probe all ipaddress we use. Loop over
       * 1. All net resources we use.
       * 2. All ipaddresses in a given resource.
       * For each of the ipaddress that is probed,
       * compute the failure history.
       */
      probe_result = 0;
      /*
       * Iterate through the all resources to get each
       * IP address to use for calling svc_probe()
       */
      for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS supports only one port and
          * hence obtain the port value from the
          * first entry in the array of ports.
          */
         ht1 = gethrtime(); /* Latch probe start time */
         scds_syslog(LOG_INFO, "Probing the service on "
             "port: %d.", port);

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

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

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

         /*
```

예 C–5 xfnts_probe.c (계속)

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

xfnts_start 메소드 코드 목록

RGM은 데이터 서비스 자원이 포함된 자원 그룹이 클러스터 노드에서 온라인 상태이거나 자원이 사용 가능할 때 해당 노드에서 Start 메소드를 실행합니다. xfnts_start 메소드는 해당 노드에서 xfs 데몬을 활성화합니다.

예 C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_start.c - Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * The start method for HA-XFS. Does some sanity checks on
 * the resource settings then starts the HA-XFS under PMF with
 * an action script.
 */

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

   /*
    * Process all the arguments that have been passed to us from RGM
    * and do some initialization for syslog
    */

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

예 C–6 xfnts_start.c (계속)

```
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Validate the configuration and if there is an error return back */
   rc = svc_validate(scds_handle);
   if (rc != 0) {
      scds_syslog(LOG_ERR,
          "Failed to validate configuration.");
      return (rc);
   }

   /* Start the data service, if it fails return with an error */
   rc = svc_start(scds_handle);
   if (rc != 0) {
      goto finished;
   }

   /* Wait for the service to start up fully */
   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Calling svc_wait to verify that service has started.");

   rc = svc_wait(scds_handle);

   scds_syslog_debug(DBG_LEVEL_HIGH,
       "Returned from svc_wait");

   if (rc == 0) {
      scds_syslog(LOG_INFO, "Successfully started the service.");
   } else {
      scds_syslog(LOG_ERR, "Failed to start the service.");
   }


finished:
   /* Free up the Environment resources that were allocated */
   scds_close(&scds_handle);

   return (rc);
}
```

xfnts_stop 메소드 코드 목록

RGM은 HA-XFS 자원이 포함된 자원 그룹이 클러스터 노드에서 오프라인 상태이거나 자원이 비활성화될 때 해당 노드에서 Stop 메소드를 실행합니다. 이 메소드는 해당 노드에서 xfs 데몬을 중지합니다.

예 C–7 xfnts_stop.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_stop.c - Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_stop.c 1.10 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Stops the HA-XFS process using PMF
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int      rc;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   rc = svc_stop(scds_handle);

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

   /* Return the result of svc_stop method */
   return (rc);
}
```

xfnts_update 메소드 코드 목록

RGM은 실행 중인 자원에 해당 등록 정보가 변경되었다는 것을 알리기 위해 Update 메소드를 호출합니다. RGM은 관리 작업이 자원이나 자원 그룹의 등록 정보를 성공적으로 설정한 후 Update를 실행합니다.

예 C–8 xfnts_update.c

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
```

예 C–8 xfnts_update.c (계속)

```
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_update.c - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>

/*
 * Some of the resource properties might have been updated. All such
 * updatable properties are related to the fault monitor. Hence, just
 * restarting the monitor should be enough.
 */

int
main(int argc, char *argv[])
{
   scds_handle_t   scds_handle;
   scha_err_t   result;

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

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

   scds_syslog(LOG_INFO, "Restarting the fault monitor.");
   result = scds_pmf_restart_fm(scds_handle, 0);
   if (result != SCHA_ERR_NOERR) {
      scds_syslog(LOG_ERR,
          "Failed to restart fault monitor.");
      /* Free up all the memory allocated by scds_initialize */
      scds_close(&scds_handle);
      return (1);
   }

   scds_syslog(LOG_INFO,
       "Completed successfully.");

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

예 C–8 xfnts_update.c (계속)

```
   return (0);
}
```

xfnts_validate 메소드 코드 목록

이 메소드는 Confdir_list 등록 정보에서 가리키는 디렉토리가 있는지 확인합니다. RGM은 데이터 서비스가 만들어질 때와 데이터 서비스 등록 정보가 클러스터 관리자에 의해 업데이트될 때 이 메소드를 호출합니다. Monitor_check 메소드는 오류 모니터가 데이터 서비스를 새 노드로 페일오버할 때마다 이 메소드를 호출합니다.

예 C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2005 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_validate.c - validate method for HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Check to make sure that the properties have been set properly.
 */

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

   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = svc_validate(scds_handle);

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

   /* Return the result of validate method */
   return (rc);
```

예 **C–9** xfnts_validate.c *(계속)*

```
}
```

부록 D

유효한 RGM 이름 및 값

이 부록에서는 RGM(Resource Group Manager) 이름 및 값으로 유효한 문자에 대한 요구 사항을 나열합니다.

이 부록은 다음 내용으로 구성되어 있습니다.

RGM 유효 이름

RGM 이름은 다음과 같은 범주에 속합니다.

- 자원 그룹 이름
- 자원 유형 이름
- 자원 이름
- 등록 정보 이름
- 열거 리터럴 이름

자원 유형 이름을 제외한 이름 규칙

자원 유형 이름을 제외한 모든 이름은 다음 규칙을 준수해야 합니다.

- ASCII여야 합니다.
- 문자로 시작해야 합니다.
- 대소문자, 숫자, 대시(-) 및 밑줄(_)을 포함할 수 있습니다.
- 255자를 넘지 않아야 합니다.

자원 유형 이름 형식

자원 유형의 전체 이름 형식은 자원 유형에 따라 다음과 같습니다.

- 자원 유형의 자원 유형 등록(RTR) 파일에 `#$upgrade` 지시어가 포함되어 있는 경우 형식은 다음과 같습니다.

  vendor-id.*base-rt-name*:*rt-version*

- 자원 유형의 RTR 파일에 `#$upgrade` 지시어가 포함되어 있지 **않은** 경우 형식은 다음과 같습니다.

  vendor-id.*base-rt-name*

마침표는 *vendor-id*와 *base-rt-name*을 구분합니다. 콜론은 *base-rt-name*과 *rt-version*을 구분합니다.

이 형식에서 변수 요소는 다음과 같습니다.

vendor-id 공급업체 ID 접두어를 지정합니다. 공급업체 ID 접두어는 RTR 파일의 `Vendor_id` 자원 유형 등록 정보 값입니다. 자원 유형을 개발 중이면 회사의 주식 티커 기호와 같이 공급업체를 고유하게 식별하는 공급업체 ID 접두어를 선택합니다. 예를 들어 Sun Microsystems, Inc.에서 개발한 자원 유형의 공급업체 ID 접두어는 `SUNW`입니다.

base-rt-name 기본 자원 유형 이름을 지정합니다. 기본 자원 유형 이름은 RTR 파일의 `Resource_type` 자원 유형 등록 정보 값입니다.

rt-version 버전 접미어를 지정합니다. 버전 접미어는 RTR 파일의 `RT_version` 자원 유형 등록 정보 값입니다. RTR 파일에 `#$upgrade` 지시어가 포함되어 있는 경우 전체 자원 유형 이름의 **유일한** 부분은 접미어입니다. `#$upgrade` 지시어는 Sun Cluster 제품의 릴리스 3.1에서 도입되었습니다.

주 – 기본 자원 유형 이름 버전을 하나만 등록하는 경우 `scrgadm` 명령에 전체 이름을 사용할 필요가 없습니다. 공급업체 ID 접두어, 버전 번호 접미어 또는 둘 다를 생략할 수 있습니다.

자세한 내용은 223 페이지 “자원 유형 등록 정보”를 참조하십시오.

예 D–1 `#$upgrade` 지시어가 있는 자원 유형의 전체 이름

이 예에서는 RTR 파일에서 등록 정보가 다음과 같이 설정된 자원 유형의 전체 이름을 보여줍니다.

- `Vendor_id=SUNW`
- `Resource_type=sample`
- `RT_version=2.0`

이 RTR 파일에 정의되는 자원 유형의 전체 이름은 다음과 같습니다.

예 D–1 #$upgrade 지시어가 있는 자원 유형의 전체 이름 (**계속**)

```
SUNW.sample:2.0
```

예 D–2 #$upgrade 지시어가 없는 자원 유형의 전체 이름

이 예에서는 RTR 파일에서 등록 정보가 다음과 같이 설정된 자원 유형의 전체 이름을 보여줍니다.

- `Vendor_id=SUNW`
- `Resource_type=nfs`

이 RTR 파일에 정의되는 자원 유형의 전체 이름은 다음과 같습니다.

```
SUNW.nfs
```

RGM 값

RGM 값은 등록 정보 값과 설명 값의 두 범주로 구분됩니다. 두 범주는 다음과 같은 동일한 규칙을 공유합니다.

- 값은 ASCII여야 합니다.
- 값의 최대 길이는 4MB-1, 즉 4,194,303바이트입니다.
- 값에 다음 문자를 사용할 수 없습니다.
  - null
  - 개행
  - 쉼표(,)
  - 세미콜론(;)

부록 E

비클러스터 인식 응용 프로그램 요구 사항

일반 비클러스터 인식 응용 프로그램은 고가용성(HA)의 후보가 되기 위해 특정 요구 사항을 충족해야 합니다. 29 페이지 “응용 프로그램 적합성 분석” 절에 이 요구 사항이 나열되어 있습니다. 이 부록에서는 해당 목록의 특정 항목에 대한 추가 정보를 제공합니다.

자원을 자원 그룹에 구성하여 응용 프로그램의 가용성을 높입니다. 응용 프로그램의 데이터를 가용성이 높은 클러스터 파일 시스템에 배치하여 하나의 서버가 실패한 경우 남은 서 버에서 데이터를 액세스할 수 있게 합니다. 클러스터 파일 시스템에 대한 자세한 내용은 **Solaris OS용 Sun Cluster 개념 안내서**를 참조하십시오.

네트워크의 클라이언트에서 네트워크를 액세스하기 위해 동일한 자원 그룹에 데이터 서비스 자원으로 포함된 논리 호스트 이름 자원에 논리 네트워크 IP 주소가 구성됩니다. 데이터 서비스 자원 및 네트워크 주소 자원이 함께 페일오버하면 데이터 서비스의 네트워크 클라이언트가 새로운 호스트에서 데이터 서비스 자원을 액세스하게 됩니다.

이 부록은 다음 내용으로 구성되어 있습니다.

멀티호스트된 데이터

가용성이 높은 클러스터 파일 시스템의 장치는 멀티호스트되기 때문에 물리적 호스트가 충돌할 경우 충돌에서 살아남은 호스트 중 하나가 이 장치를 액세스할 수 있습니다. 가용성이 높은 응용 프로그램을 만들려면 해당 데이터의 가용성이 높아야 하므로 응용 프로그램 데이터가 여러 클러스터 노드에서 액세스할 수 있는 파일 시스템에 있어야 합니다. Sun Cluster에서 지원하는 가용성이 높은 파일 시스템의 예에는 전역 HA 파일 시스템, FFS(Failover File System) 및 Oracle Real Application Clusters을 사용하는 환경의 경우 QFS 공유 파일 시스템이 포함됩니다.

클러스터 파일 시스템은 독립 항목으로 만들어진 장치 그룹에 마운트됩니다. 개발자는 일부 장치 그룹을 마운트된 클러스터 파일 시스템으로 사용하고 다른 일부는 HA Oracle 소프트웨어 같은 데이터 서비스용 원시 장치로 사용하도록 선택할 수 있습니다.

응용 프로그램에는 명령줄 스위치나 데이터 파일의 위치를 가리키는 구성 파일이 있을 수 있습니다. 응용 프로그램에서 하드 연결된 경로 이름을 사용하는 경우 응용 프로그램 코드를 변경하지 않고 클러스터 파일 시스템의 파일을 가리키는 심볼릭 링크로 경로 이름을 변경할 수 있습니다. 심볼릭 링크 사용에 대한 자세한 내용은 310 페이지 “멀티 호스트된 데이터 배치에 심볼릭 링크 사용”을 참조하십시오.

최악의 경우 응용 프로그램의 소스 코드를 수정하여 실제 데이터 위치를 가리키는 기법을 제공해야 합니다. 추가 명령줄 인자를 생성하여 이 기법을 구현할 수 있습니다.

Sun Cluster 소프트웨어는 볼륨 관리자에 구성된 UNIX UFS 파일 시스템 및 HA 원시 장치의 사용을 지원합니다. Sun Cluster 소프트웨어 설치 및 구성 시 클러스터 관리자는 UFS 파일 시스템에 사용할 디스크 자원과 원시 장치에 사용할 디스크 자원을 지정해야 합니다. 일반적으로 데이터베이스 서버와 멀티미디어 서버에서만 원시 장치를 사용합니다.

멀티 호스트된 데이터 배치에 심볼릭 링크 사용

때로는 하드 연결된 경로 이름을 대체할 기법 없이 응용 프로그램 데이터 파일의 경로 이름이 하드 연결되는 경우가 있습니다. 응용 프로그램 코드의 수정을 방지하기 위해 때로 심볼릭 링크를 사용할 수 있습니다.

예를 들어, 응용 프로그램에서 하드 연결된 경로 이름`/etc/mydatafile`을 사용하여 데이터 파일의 이름을 지정하는 것으로 가정합시다. 개발자는 논리 호스트 파일 시스템 중 하나에 있는 파일을 가리키는 값을 가지는 심볼릭 링크로 파일 경로를 변경할 수 있습니다. 예를 들어, 경로를 `/global/phys-schost-2/mydatafile`에 대한 심볼릭 링크로 만들 수 있습니다.

응용 프로그램이나 관리 절차 중 하나에서 해당 내용뿐만 아니라 데이터 파일 이름을 수정할 경우 이 심볼릭 링크 사용에 문제가 생길 수 있습니다. 예를 들어, 응용 프로그램에서 먼저 새로운 임시 파일 /etc/mydatafile.new를 만들어 업데이트를 수행한다고 가정합니다. 그런 다음 rename() 시스템 호출이나 mv 명령을 사용하여 임시 파일이 실제 파일 이름을 갖도록 이름을 변경합니다. 데이터 서비스는 임시 파일을 만든 다음 이름을 실제 파일 이름으로 변경하여 데이터 파일 내용이 항상 올바로 구성되도록 보장합니다.

유감스럽게도 rename() 작업을 수행하면 심볼릭 링크가 완전히 삭제됩니다. 이제 /etc/mydatafile 이름은 정규 파일이며 클러스터의 클러스터 파일 시스템이 아니라 /etc 디렉토리와 동일한 파일 시스템에 있습니다. /etc 파일 시스템은 각 호스트에서 개인 파일 시스템이기 때문에 페일오버 또는 스위치오버 후에는 해당 데이터를 사용할 수 없습니다.

이 상황에서 발생할 수 있는 문제는 기존 응용 프로그램이 심볼릭 링크를 인식하지 못하여 심볼릭 링크를 처리하도록 작성되지 않았다는 것입니다. 심볼릭 링크를 사용하여 데이터 액세스를 논리 호스트의 파일 시스템으로 재지정하려면 응용 프로그램 구현에서 심볼릭 링크를 제거하지 않는 방식으로 작동해야 합니다. 따라서 심볼릭 링크는 클러스터의 파일 시스템에 데이터를 배치하는 문제에 대한 완벽한 해결책이 아닙니다.

호스트 이름

데이터 서비스에서 데이터 서비스가 실행 중인 서버의 호스트 이름을 알 필요가 있는지 여부를 결정해야 합니다. 알아야 하는 경우 데이터 서비스에서 물리적 호스트 이름 대신 논리 호스트 이름을 사용하도록 수정해야 할 수도 있습니다. 이때 논리 호스트 이름은 응용 프로그램 자원과 동일한 자원 그룹에 있는 논리 호스트 이름 자원으로 구성된 호스트 이름입니다.

때로 데이터 서비스의 클라이언트-서버 프로토콜에서 서버는 클라이언트에 대한 메시지 내용의 일부로 고유한 호스트 이름을 클라이언트에 반환합니다. 그런 프로토콜의 경우 클라이언트는 서버에 연결할 때 사용할 호스트 이름으로 이 반환된 호스트 이름을 사용할 수 있습니다. 페일오버 또는 스위치오버 후 반환된 호스트 이름을 사용할 수 있게 하려면 호스트 이름은 물리적 호스트 이름이 아니라 자원 그룹의 논리 호스트 이름이어야 합니다. 이 경우 논리 호스트 이름을 클라이언트에 반환하려면 데이터 서비스 코드를 수정해야 합니다.

다중 홈 호스트

다중 홈 호스트는 둘 이상의 공용 네트워크에 있는 호스트를 설명합니다. 그런 호스트에는 여러 호스트 이름과 IP 주소가 있습니다. 각 네트워크에는 하나의 호스트 이름–IP 주소 쌍이 있습니다. Sun Cluster는 단 하나의 네트워크(다중 홈이 아닌 경우)를 포함하여 여러 네트워크에 호스트가 나타나도록 설계되어 있습니다. 물리적 호스트 이름에 여러 호스트 이름–IP 주소 쌍이 있는 것처럼 각 자원 그룹에는 각 공용 네트워크에 하나씩 여러 개의 호스트 이름–IP 주소 쌍이 있을 수 있습니다. Sun Cluster에서 자원 그룹을 하나의 물리적 호스트에서 다른 물리적 호스트로 이동하면 해당 자원 그룹의 전체 호스트 이름–IP 주소 쌍 세트가 이동됩니다.

자원 그룹에 대한 호스트 이름–IP 주소 쌍 세트는 자원 그룹에 포함된 논리 호스트 이름 자원으로 구성됩니다. 자원 그룹을 만들고 구성할 때 클러스터 관리자는 이러한 네트워크 주소 자원을 지정합니다. Sun Cluster Data Service API에는 이 호스트 이름–IP 주소 쌍을 쿼리하는 기능이 포함되어 있습니다.

Solaris 운영 체제용으로 작성된 대부분의 사용 가능 데이터 서비스 데몬은 이미 다중 홈 호스트를 올바로 처리합니다. 많은 데이터 서비스에서는 Solaris 와일드카드 주소 `INADDR_ANY`에 바인드하여 모든 네트워크 통신을 수행합니다. 이렇게 바인드하면 자동으로 데이터 서비스에서 모든 네트워크 인터페이스의 모든 IP 주소를 처리합니다. `INADDR_ANY`는 시스템에 현재 구성된 모든 IP 주소에 효율적으로 바인드합니다. 일반적으로 Sun Cluster 논리 네트워크 주소를 처리하기 위해 `INADDR_ANY`를 사용하는 데이터 서비스 데몬을 변경할 필요는 없습니다.

INADDR_ANY에 바인드 대 특정 IP 주소에 바인드

다중 홈이 아닌 호스트를 사용할 경우에도 Sun Cluster 논리 네트워크 주소 개념을 사용하면 시스템에 둘 이상의 IP 주소가 있을 수 있습니다. 시스템에는 자체 물리적 호스트에 대한 하나의 IP 주소와 현재 시스템에서 마스터하는 각 네트워크 주소(논리 호스트 이름) 자원에 대한 추가 IP 주소가 있습니다. 시스템이 네트워크 주소 자원의 마스터가 될 경우 시스템에서는 동적으로 추가 IP 주소를 가져옵니다. 시스템이 네트워크 주소 자원의 마스터를 포기한 경우 동적으로 IP 주소를 양도합니다.

일부 데이터 서비스는 `INADDR_ANY`에 바인드될 경우 Sun Cluster 환경에서 올바로 작동하지 않습니다. 이 데이터 서비스에서는 자원 그룹이 마스터되거나 마스터 해제될 때 자신이 바인드되는 IP 주소 세트를 동적으로 변경해야 합니다. 재바인드를 완료하기 위한 한 가지 방법은 이 데이터 서비스에 대한 시작 및 중지 메소드를 사용하여 데이터 서비스 데몬을 다시 시작하는 것입니다.

Network_resources_used 자원 등록 정보에서는 최종 사용자가 응용 프로그램 자원을 바인드할 특정 네트워크 주소 자원 세트를 구성할 수 있도록 허용합니다. 이 기능이 필요한 자원 유형의 경우 해당 자원 유형의 RTR 파일에서 Network_resources_used 등록 정보를 선언해야 합니다.

RGM에서 자원 그룹을 온라인이나 오프라인으로 전환할 경우 RGM은 호출 데이터 서비스 자원 메소드를 호출하는 시기와 관련하여 네트워크 주소를 연결, 연결 해제, 활성으로 구성 및 비활성으로 구성하는 특정 순서를 따릅니다. 44 페이지 "사용할 Start 및 Stop 메소드 결정"을 참조하십시오.

데이터 서비스의 Stop 메소드가 반환될 때까지는 자원 그룹의 네트워크 주소를 사용하여 데이터 서비스를 중지했어야 합니다. 마찬가지로 Start 메소드가 반환될 때까지는 네트워크 주소를 사용하려면 데이터 서비스를 시작했어야 합니다.

데이터 서비스가 개별 IP 주소가 아니라 INADDR_ANY에 바인드될 경우 데이터 서비스 자원 메소드를 호출하는 순서와 네트워크 주소 메소드를 호출하는 순서는 관련이 없습니다.

데이터 서비스의 Stop 및 Start 메소드가 데이터 서비스의 데몬을 강제 종료하고 다시 시작하여 해당 작업을 완료하면 적절한 시간에 네트워크 주소를 사용하여 데이터 서비스가 중지 및 시작됩니다.

클라이언트 재시도

네트워크 클라이언트에게 페일오버 또는 스위치오버는 빠른 재부트가 수반되는 논리 호스트의 충돌로 나타납니다. 이상적으로 클라이언트 응용 프로그램과 클라이언트-서버 프로토콜은 어느 정도의 재시도를 수행하도록 구성되어 있습니다. 응용 프로그램과 프로토콜에서 이미 단일 서버 충돌 및 재부트를 처리한 경우 자원 그룹의 페일오버 또는 스위치 오버도 처리할 수 있습니다. 일부 응용 프로그램에서는 무한대로 재시도하도록 선택할 수 있습니다. 좀더 복잡한 응용 프로그램에서는 사용자에게 시간이 오래 걸리는 재시도가 진행 중임을 알리고 사용자가 계속할지 여부를 선택하도록 할 수 있습니다.

부록 F

CRNP용 문서 유형 정의

이 부록에는 다음과 같은 CRNP(Cluster Reconfiguration Notification Protocol)용 문서 유형 정의(DTD)가 포함되어 있습니다.

SC_CALLBACK_REG XML DTD

주 – SC_CALLBACK_REG 및 SC_EVENT에서 사용하는 NVPAIR 데이터 구조는 한 번만 정의됩니다.

```
<!— SC_CALLBACK_REG XML format specification
        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:

A client of the Cluster Reconfiguration Notification Protocol should use this xml format
to register initially with the service, to subsequently register for more events, to
subsequently remove registration of some events, or to remove itself from the service
entirely.

A client is uniquely identified by its callback IP and port.  The port is defined in the
SC_CALLBACK_REG element, and the IP is taken as the source IP of the registration
connection.  The final attribute of the root SC_CALLBACK_REG element is either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS, depending on which form of the
message the client is using.
```

```
The SC_CALLBACK_REG contains 0 or more SC_EVENT_REG sub-elements.

One SC_EVENT_REG is the specification for one event type.  A client may specify only the
CLASS (an attribute of the SC_EVENT_REG element), or may specify a SUBCLASS (an optional
attribute) for further granularity.  Also, the SC_EVENT_REG has as subelements 0 or more
NVPAIRs, which can be used to further specify the event.

Thus, the client can specify events to whatever granularity it wants.  Note that a client
cannot both register for and unregister for events in the same message.  However a client
can subscribe to the service and sign up for events in the same message.

Note on versioning: the VERSION attribute of each root element is marked "fixed", which
means that all message adhering to these DTDs must have the version value specified.  If a
new version of the protocol is created, the revised DTDs will have a new value for this
fixed" VERSION attribute, such that all message adhering to the new version must have the
new version number.
—>
<!— SC_CALLBACK_REG definition

The root element of the XML document is a registration message.  A registration message
consists of the callback port and the protocol version as attributes, and either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS attribute, specifying the
registration type.  The ADD_CLIENT, ADD_EVENTS, and REMOVE_EVENTS types should have one or
more SC_EVENT_REG subelements.  The REMOVE_CLIENT should not specify an SC_EVENT_REG
subelement.
        ATTRIBUTES:
               VERSION        The CRNP protocol version of the message.
               PORT           The callback port.
               REG_TYPE       The type of registration.  One of:
                       ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENTS:
SUBELEMENTS: SC_EVENT_REG (0 or more)
—>
<!ELEMENT SC_CALLBACK_REG (SC_EVENT_REG*)>
<!ATTLIST SC_CALLBACK_REG
       VERSION        NMTOKEN                                                  #FIXED
       PORT           NMTOKEN                                                  #REQUIRED
       REG_TYPE       (ADD_CLIENT|ADD_EVENTS|REMOVE_CLIENT|REMOVE_EVENTS)      #REQUIRED

>
<!— SC_EVENT_REG definition

The SC_EVENT_REG defines an event for which the client is either registering or
unregistering interest in receiving event notifications.  The registration can be for any
level of granularity, from only event class down to specific name/value pairs that must be
present.  Thus, the only required attribute is the CLASS.  The SUBCLASS attribute, and the
NVPAIRS sub-elements are optional, for higher granularity.

Registrations that specify name/value pairs are registering interest in notification of
messages from the class/subclass specified with ALL name/value pairs present.
Unregistrations that specify name/value pairs are unregistering interest in notifications
that have EXACTLY those name/value pairs in granularity previously specified.
Unregistrations that do not specify name/value pairs unregister interest in ALL event
notifications of the specified class/subclass.
```

```
        ATTRIBUTES:
               CLASS:        The event class for which this element is registering
                             or unregistering interest.
               SUBCLASS:     The subclass of the event (optional).

        CONTENTS:
               SUBELEMENTS: 0 or more NVPAIRs.
—>
<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA                #REQUIRED
       SUBCLASS       CDATA                #IMPLIED
>
```

NVPAIR XML DTD

```
<!— NVPAIR XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:
              An nvpair element is meant to be used in an SC_EVENT or SC_CALLBACK_REG
              element.
—>
<!— NVPAIR definition

       The NVPAIR is a name/value pair to represent arbitrary name/value combinations.
       It is intended to be a direct, generic, translation of the Solaris nvpair_t
       structure used by the sysevent framework.  However, there is no type information
       associated with the name or the value (they are both arbitrary text) in this xml
       element.

       The NVPAIR consists simply of one NAME element and one or more VALUE elements.
       One VALUE element represents a scalar value, while multiple represent an array
       VALUE.

       ATTRIBUTES:

       CONTENTS:
              SUBELEMENTS: NAME(1), VALUE(1 or more)
—>
<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— NAME definition

        The NAME is simply an arbitrary length string.

        ATTRIBUTES:
```

```
        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT NAME (#PCDATA)>

<!— VALUE definition
        The VALUE is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>
<!ELEMENT VALUE (#PCDATA)>
```

SC_REPLY XML DTD

```
<!— SC_REPLY XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.
—>
<!— SC_REPLY definition

        The root element of the XML document represents a reply to a message.  The reply
        contains a status code and a status message.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                STATUS_CODE:    The return code for the message.  One of the
                                following: OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
                                MALFORMED, INVALID_XML, VERSION_TOO_HIGH, or
                                VERSION_TOO_LOW.

                CONTENTS:
                        SUBELEMENTS: SC_STATUS_MSG(1)
—>
<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
        VERSION         NMTOKEN                                         #FIXED   "1.0"
        STATUS_CODE     OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
                        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>
<!— SC_STATUS_MSG definition
        The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
        code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.
```

```
        ATTRIBUTES:

        CONTENTS:
                Arbitrary string.
—>
<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

SC_EVENT XML DTD

주 – SC_CALLBACK_REG 및 SC_EVENT에서 사용하는 NVPAIR 데이터 구조는 한 번만 정의됩니다.

```
<!— SC_EVENT XML format specification

        Copyright 2001-2005 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        The root element of the XML document is intended to be a direct, generic,
        translation of the Solaris syseventd message format.  It has attributes to
        represent the class, subclass, vendor, and publisher, and contains any number of
        NVPAIR elements.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                CLASS:          The sysevent class of the event
                SUBCLASS:       The subclass of the event
                VENDOR:         The vendor associated with the event
                PUBLISHER:      The publisher of the event
        CONTENTS:
                SUBELEMENTS: NVPAIR (0 or more)
—>
<!ELEMENT SC_EVENT (NVPAIR*)>
<!ATTLIST SC_EVENT
        VERSION         NMTOKEN         #FIXED "1.0"
        CLASS           CDATA           #REQUIRED
        SUBCLASS        CDATA           #REQUIRED
        VENDOR          CDATA           #REQUIRED
        PUBLISHER       CDATA           #REQUIRED
>
```

부록 G

CrnpClient.java 응용 프로그램

이 부록에는 12 장에 자세히 설명된 CrnpClient.java 응용 프로그램의 전체 내용이 나와 있습니다.

CrnpClient.java의 내용

```
/*
 * CrnpClient.java
 * ================
 *
 * Note regarding XML parsing:
 *
 * This program uses the Sun Java Architecture for XML Processing (JAXP) API.
 * See http://java.sun.com/xml/jaxp/index.html for API documentation and
 * availability information.
 *
 * This program was written for Java 1.3.1 or higher.
 *
 * Program overview:
 *
 * The main thread of the program creates a CrnpClient object, waits for the
 * user to terminate the demo, then calls shutdown on the CrnpClient object
 * and exits the program.
 *
 * The CrnpClient constructor creates an EventReceptionThread object,
 * opens a connection to the CRNP server (using the host and port specified
 * on the command line), constructs a registration message (based on the
 * command-line specifications), sends the registartion message, and reads
 * and parses the reply.
 *
 * The EventReceptionThread creates a listening socket bound to
 * the hostname of the machine on which this program runs, and the port
 * specified on the command line. It waits for an incoming event callback,
```

```
 * at which point it constructs an XML Document from the incoming socket
 * stream, which is then passed back to the CrnpClient object to process.
 *
 * The shutdown method in the CrnpClient just sends an unregistration
 * (REMOVE_CLIENT) SC_CALLBACK_REG message to the crnp server.
 *
 * Note regarding error handling: for the sake of brevity, this program just
 * exits on most errors.  Obviously, a real application would attempt to handle
 * some errors in various ways, such as retrying when appropriate.
 */

// JAXP packages
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.w3c.dom.*;

// standard packages
import java.net.*;
import java.io.*;
import java.util.*;

/*
 * class CrnpClient
 * ----------------
 * See file header comments above.
 */
class CrnpClient
{
    /*
     * main
     * ----
     * The entry point of the execution, main simply verifies the
     * number of command-line arguments, and constructs an instance
     * of a CrnpClient to do all the work.
     */
    public static void main(String []args)
    {
        InetAddress regIp = null;
        int regPort = 0, localPort = 0;

        /* Verify the number of command-line arguments */
        if (args.length < 4) {
            System.out.println(
                "Usage: java CrnpClient crnpHost crnpPort "
                + "localPort (-ac | -ae | -re) "
                + "[(M | A | RG=name | R=name) [...]]");
            System.exit(1);
        }


        /*
```

```
     * We expect the command line to contain the ip/port of the
     * crnp server, the local port on which we should listen, and
     * arguments specifying the type of registration.
     */
    try {
        regIp = InetAddress.getByName(args[0]);
        regPort = (new Integer(args[1])).intValue();
        localPort = (new Integer(args[2])).intValue();
    } catch (UnknownHostException e) {
        System.out.println(e);
        System.exit(1);
    }

    // Create the CrnpClient
    CrnpClient client = new CrnpClient(regIp, regPort, localPort,
        args);

    // Now wait until the user wants to end the program
    System.out.println("Hit return to terminate demo...");

    // read will block until the user enters something
    try {
        System.in.read();
    } catch (IOException e) {
        System.out.println(e.toString());
    }

    // shutdown the client
    client.shutdown();
    System.exit(0);
}

/*
 * ======================
 * public methods
 * ======================
 */

/*
 * CrnpClient constructor
 * ----------------------
 * Parses the command line arguments so we know how to contact
 * the crnp server, creates the event reception thread, and starts it
 * running, creates the XML DocumentBuilderFactory obect, and, finally,
 * registers for callbacks with the crnp server.
 */
public CrnpClient(InetAddress regIpIn, int regPortIn, int localPortIn,
    String []clArgs)
{
    try {

        regIp = regIpIn;
        regPort = regPortIn;
        localPort = localPortIn;
        regs = clArgs;
```

```
        /*
         * Setup the document builder factory for
         * xml processing.
         */
        setupXmlProcessing();

        /*
         * Create the EventReceptionThread, which creates a
         * ServerSocket and binds it to a local ip and port.
         */
        createEvtRecepThr();

        /*
         * Register with the crnp server.
         */
        registerCallbacks();

    } catch (Exception e) {
        System.out.println(e.toString());
        System.exit(1);
    }
}

/*
 * processEvent
 * ---------------
 * Callback into the CrnpClient, used by the EventReceptionThread
 * when it receives event callbacks.
 */
public void processEvent(Event event)
{
    /*
     * For demonstration purposes, simply print the event
     * to System.out.  A real application would obviously make
     * use of the event in some way.
     */
    event.print(System.out);
}

/*
 * shutdown
 * -------------
 * Unregister from the CRNP server.
 */
public void shutdown()
{
    try {
        /* send an unregistration message to the server */
        unregister();
    } catch (Exception e) {
        System.out.println(e);
        System.exit(1);
    }
}
```

```
/*
 * ======================
 * private helper methods
 * ======================
 */

/*
 * setupXmlProcessing
 * --------------------
 * Create the document builder factory for
 * parsing the xml replies and events.
 */
private void setupXmlProcessing() throws Exception
{
    dbf = DocumentBuilderFactory.newInstance();

    // We don't need to bother validating
    dbf.setValidating(false);
    dbf.setExpandEntityReferences(false);

    // We want to ignore comments and whitespace
    dbf.setIgnoringComments(true);
    dbf.setIgnoringElementContentWhitespace(true);

    // Coalesce CDATA sections into TEXT nodes.
    dbf.setCoalescing(true);
}

/*
 * createEvtRecepThr
 * -------------------
 * Creates a new EventReceptionThread object, saves the ip
 * and port to which its listening socket is bound, and
 * starts the thread running.
 */
private void createEvtRecepThr() throws Exception
{
    /* create the thread object */
    evtThr = new EventReceptionThread(this);

    /*
     * Now start the thread running to begin listening
     * for event delivery callbacks.
     */
    evtThr.start();
}


/*
 * registerCallbacks
 * ------------------
 * Creates a socket connection to the crnp server and sends
 * an event registration message.
```

```
 */
private void registerCallbacks() throws Exception
{
    System.out.println("About to register");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the registration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createRegistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}

/*
 * unregister
 * ----------
 * As in registerCallbacks, we create a socket connection to
 * the crnp server, send the unregistration message, wait for
 * the reply from the server, then close the socket.
 */
private void unregister() throws Exception
{
    System.out.println("About to unregister");

    /*
     * Create a socket connected to the registration ip/port
     * of the crnp server and send the unregistration information.
     */
    Socket sock = new Socket(regIp, regPort);
    String xmlStr = createUnregistrationString();
    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.print(xmlStr);

    /*
     * Read the reply
     */
    readRegistrationReply(sock.getInputStream());

    /*
     * Close the socket connection.
     */
    sock.close();
}
```

```
/*
 * createRegistrationString
 * ------------------
 * Constructs a CallbackReg object based on the command line arguments
 * to this program, then retrieves the XML string from the CallbackReg
 * object.
 */
private String createRegistrationString() throws Exception
{
    /*
     * create the actual CallbackReg class and set the port.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);

    // set the registration type
    if (regs[3].equals("-ac")) {
        cbReg.setRegType(CallbackReg.ADD_CLIENT);
    } else if (regs[3].equals("-ae")) {
        cbReg.setRegType(CallbackReg.ADD_EVENTS);
    } else if (regs[3].equals("-re")) {
        cbReg.setRegType(CallbackReg.REMOVE_EVENTS);
    } else {
        System.out.println("Invalid reg type: " + regs[3]);
        System.exit(1);
    }

    // add the events
    for (int i = 4; i < regs.length; i++) {
        if (regs[i].equals("M")) {
            cbReg.addRegEvent(createMembershipEvent());
        } else if (regs[i].equals("A")) {
            cbReg.addRegEvent(createAllEvent());
        } else if (regs[i].substring(0,2).equals("RG")) {
            cbReg.addRegEvent(createRgEvent(regs[i].substring(3)));
        } else if (regs[i].substring(0,1).equals("R")) {
            cbReg.addRegEvent(createREvent(regs[i].substring(2)));
        }
    }

    String xmlStr = cbReg.convertToXml();
    System.out.println(xmlStr);
    return (xmlStr);
}

/*
 * createAllEvent
 * ----------------
 * Creates an XML registartion event with class EC_Cluster, and no
 * subclass.
 */
private Event createAllEvent()
{
    Event allEvent = new Event();
```

```
    allEvent.setClass("EC_Cluster");
    return (allEvent);
}

/*
 * createMembershipEvent
 * ---------------------
 * Creates an XML registration event with class EC_Cluster, subclass
 * ESC_cluster_memberhip.
 */
private Event createMembershipEvent()
{
    Event membershipEvent = new Event();
    membershipEvent.setClass("EC_Cluster");
    membershipEvent.setSubclass("ESC_cluster_membership");
    return (membershipEvent);
}

/*
 * createRgEvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
 * subclass ESC_cluster_rg_state, and one "rg_name" nvpair (based
 * on input parameter).
 */
private Event createRgEvent(String rgname)
{
    /*
     * Create a Resource Group state change event for the
     * rgname Resource Group.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    /*
     * Construct the event object and set the class and subclass.
     */
    Event rgStateEvent = new Event();
    rgStateEvent.setClass("EC_Cluster");
    rgStateEvent.setSubclass("ESC_cluster_rg_state");

    /*
     * Create the nvpair object and add it to the Event.
     */
    NVPair rgNvpair = new NVPair();
    rgNvpair.setName("rg_name");
    rgNvpair.setValue(rgname);
    rgStateEvent.addNvpair(rgNvpair);

    return (rgStateEvent);
}

/*
 * createREvent
 * ----------------
 * Creates an XML registration event with class EC_Cluster,
```

```
 * subclass ESC_cluster_r_state, and one "r_name" nvpair (based
 * on input parameter).
 */
private Event createREvent(String rname)
{
    /*
     * Create a Resource state change event for the
     * rgname Resource.  Note that we supply
     * a name/value pair (nvpair) for this event type, to
     * specify in which Resource Group we are interested.
     */
    Event rStateEvent = new Event();
    rStateEvent.setClass("EC_Cluster");
    rStateEvent.setSubclass("ESC_cluster_r_state");

    NVPair rNvpair = new NVPair();
    rNvpair.setName("r_name");
    rNvpair.setValue(rname);
    rStateEvent.addNvpair(rNvpair);

    return (rStateEvent);
}

/*
 * createUnregistrationString
 * ------------------
 * Constructs a REMOVE_CLIENT CallbackReg object, then retrieves
 * the XML string from the CallbackReg object.
 */
private String createUnregistrationString() throws Exception
{
    /*
     * Crate the CallbackReg object.
     */
    CallbackReg cbReg = new CallbackReg();
    cbReg.setPort("" + localPort);
    cbReg.setRegType(CallbackReg.REMOVE_CLIENT);

    /*
     * we marshall the registration to the OutputStream
     */
    String xmlStr = cbReg.convertToXml();

    // Print the string for debugging purposes
    System.out.println(xmlStr);
    return (xmlStr);
}


/*
 * readRegistrationReply
 * ------------------------
 * Parse the xml into a Document, construct a RegReply object
 * from the document, and print the RegReply object.  Note that
 * a real application would take action based on the status_code
```

```
     * of the RegReply object.
     */
    private void readRegistrationReply(InputStream stream)
        throws Exception
    {
        // Create the document builder
        DocumentBuilder db = dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        //parse the input file
        Document doc = db.parse(stream);

        RegReply reply = new RegReply(doc);
        reply.print(System.out);
    }

    /* private member variables */
    private InetAddress regIp;
    private int regPort;
    private EventReceptionThread evtThr;
    private String regs[];

    /* public member variables */
    public int localPort;
    public DocumentBuilderFactory dbf;
}

/*
 * class EventReceptionThread
 * --------------------------
 * See file header comments above.
 */
class EventReceptionThread extends Thread
{
    /*
     * EventReceptionThread constructor
     * --------------------------------
     * Creates a new ServerSocket, bound to the local hostname and
     * a wildcard port.
     */
    public EventReceptionThread(CrnpClient clientIn) throws IOException
    {
        /*
         * keep a reference to the client so we can call it back
         * when we get an event.
         */
        client = clientIn;

        /*
         * Specify the IP to which we should bind.  It's
```

```
     * simply the local host ip.  If there is more
     * than one public interface configured on this
     * machine, we'll go with whichever one
     * InetAddress.getLocalHost comes up with.
     *
     */
    listeningSock = new ServerSocket(client.localPort, 50,
        InetAddress.getLocalHost());
        System.out.println(listeningSock);
}

/*
 * run
 * ---
 * Called by the Thread.Start method.
 *
 * Loops forever, waiting for incoming connections on the ServerSocket.
 *
 * As each incoming connection is accepted, an Event object
 * is created from the xml stream, which is then passed back to
 * the CrnpClient object for processing.
 */
public void run()
{
    /*
     * Loop forever.
     */
    try {
        //
        // Create the document builder using the document
        // builder factory in the CrnpClient.
        //
        DocumentBuilder db = client.dbf.newDocumentBuilder();

        //
        // Set an ErrorHandler before parsing
        // Use the default handler.
        //
        db.setErrorHandler(new DefaultHandler());

        while(true) {
            /* wait for a callback from the server */
            Socket sock = listeningSock.accept();

            // parse the input file
            Document doc = db.parse(sock.getInputStream());

            Event event = new Event(doc);
            client.processEvent(event);

            /* close the socket */
            sock.close();
        }
        // UNREACHABLE
```

```
        } catch (Exception e) {
            System.out.println(e);
            System.exit(1);
        }
    }

    /* private member variables */
    private ServerSocket listeningSock;
    private CrnpClient client;
}

/*
 * class NVPair
 * ------------
 * This class stores a name/value pair (both Strings).  It knows how to
 * construct an NVPAIR XML message from its members, and how to parse
 * an NVPAIR XML Element into its members.
 *
 * Note that the formal specification of an NVPAIR allows for multiple values.
 * We make the simplifying assumption of only one value.
 */
class NVPair
{
    /*
     * Two constructors: the first creates an empty NVPair, the second
     * creates an NVPair from an NVPAIR XML Element.
     */
    public NVPair()
    {
        name = value = null;
    }

    public NVPair(Element elem)
    {
        retrieveValues(elem);
    }

    /*
     * Public setters.
     */
    public void setName(String nameIn)
    {
        name = nameIn;
    }

    public void setValue(String valueIn)
    {
        value = valueIn;
    }

    /*
     * Prints the name and value on a single line.
     */
    public void print(PrintStream out)
    {
```

```
    out.println("NAME=" + name + " VALUE=" + value);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an NVPAIR XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.
 */
public Element createXmlElement(Document doc)
{
    // Create the element.
    Element nvpair = (Element)
        doc.createElement("NVPAIR");
    //
    // Add the name.  Note that the actual name is
    // a separate CDATA section.
    //
    Element eName = doc.createElement("NAME");
    Node nameData = doc.createCDATASection(name);
    eName.appendChild(nameData);
    nvpair.appendChild(eName);
    //
    // Add the value.  Note that the actual value is
    // a separate CDATA section.
    //
    Element eValue = doc.createElement("VALUE");
    Node valueData = doc.createCDATASection(value);
    eValue.appendChild(valueData);
    nvpair.appendChild(eValue);

    return (nvpair);
}

/*
 * retrieveValues
 * ----------------
 * Parse the XML Element to retrieve the name and value.
 */
private void retrieveValues(Element elem)
{
    Node n;
    NodeList nl;

    //
    // Find the NAME element
    //
    nl = elem.getElementsByTagName("NAME");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "NAME node.");
        return;
    }
```

```
        //
        // Get the TEXT section
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            System.out.println("Error in parsing: can't find "
                + "TEXT section.");
            return;
        }

        // Retrieve the value
        name = n.getNodeValue();

        //
        // Now get the value element
        //
        nl = elem.getElementsByTagName("VALUE");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "VALUE node.");
            return;
        }

        //
        // Get the TEXT section
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            System.out.println("Error in parsing: can't find "
                + "TEXT section.");
            return;
        }

        // Retrieve the value
        value = n.getNodeValue();
    }


    /*
     * Public accessors
     */
    public String getName()
    {
        return (name);
    }

    public String getValue()
    {
        return (value);
    }

    // Private member vars
    private String name, value;
}
```

```
/*
 * class Event
 * -----------
 * This class stores an event, which consists of a class, subclass, vendor,
 * publisher, and list of name/value pairs.  It knows how to
 * construct an SC_EVENT_REG XML Element from its members, and how to parse
 * an SC_EVENT XML Element into its members.  Note that there is an assymetry
 * here: we parse SC_EVENT elements, but construct SC_EVENT_REG elements.
 * That is because SC_EVENT_REG elements are used in registration messages
 * (which we must construct), while SC_EVENT elements are used in event
 * deliveries (which we must parse).  The only difference is that SC_EVENT_REG
 * elements don't have a vendor or publisher.
 */
class Event
{

    /*
     * Two constructors: the first creates an empty Event; the second
     * creates an Event from an SC_EVENT XML Document.
     */
    public Event()
    {
        regClass = regSubclass = null;
        nvpairs = new Vector();
    }

    public Event(Document doc)
    {

        nvpairs = new Vector();

        //
        // Convert the document to a string to print for debugging
        // purposes.
        //
        DOMSource domSource = new DOMSource(doc);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return;
        }
        System.out.println(strWrite.toString());

        // Do the actual parsing.
        retrieveValues(doc);
    }

    /*
     * Public setters.
```

```
 */
public void setClass(String classIn)
{
    regClass = classIn;
}

public void setSubclass(String subclassIn)
{
    regSubclass = subclassIn;
}

public void addNvpair(NVPair nvpair)
{
    nvpairs.add(nvpair);
}

/*
 * createXmlElement
 * ------------------
 * Constructs an SC_EVENT_REG XML Element from the member variables.
 * Takes the Document as a parameter so that it can create the
 * Element.  Relies on the NVPair createXmlElement ability.
 */
public Element createXmlElement(Document doc)
{
    Element event = (Element)
        doc.createElement("SC_EVENT_REG");
    event.setAttribute("CLASS", regClass);
    if (regSubclass != null) {
        event.setAttribute("SUBCLASS", regSubclass);
    }
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        event.appendChild(tempNv.createXmlElement(doc));
    }
    return (event);
}

/*
 * Prints the member vars on multiple lines.
 */
public void print(PrintStream out)
{
    out.println("\tCLASS=" + regClass);
    out.println("\tSUBCLASS=" + regSubclass);
    out.println("\tVENDOR=" + vendor);
    out.println("\tPUBLISHER=" + publisher);
    for (int i = 0; i < nvpairs.size(); i++) {
        NVPair tempNv = (NVPair)
            (nvpairs.elementAt(i));
        out.print("\t\t");
        tempNv.print(out);
    }
}
```

```
 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the class, subclass, vendor,
 * publisher, and nvpairs.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_EVENT element.
    //
    nl = doc.getElementsByTagName("SC_EVENT");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "SC_EVENT node.");
        return;
    }

    n = nl.item(0);

    //
    // Retrieve the values of the CLASS, SUBCLASS,
    // VENDOR and PUBLISHER attributes.
    //
    regClass = ((Element)n).getAttribute("CLASS");
    regSubclass = ((Element)n).getAttribute("SUBCLASS");
    publisher = ((Element)n).getAttribute("PUBLISHER");
    vendor = ((Element)n).getAttribute("VENDOR");

    //
    // Retrieve all the nv pairs
    //
    for (Node child = n.getFirstChild(); child != null;
         child = child.getNextSibling())
    {
        nvpairs.add(new NVPair((Element)child));
    }
}

/*
 * Public accessor methods.
 */
public String getRegClass()
{
    return (regClass);
}

public String getSubclass()
{
    return (regSubclass);
}
```

```
    public String getVendor()
    {
        return (vendor);
    }

    public String getPublisher()
    {
        return (publisher);
    }

    public Vector getNvpairs()
    {
        return (nvpairs);
    }

    // Private member vars.
    private String regClass, regSubclass;
    private Vector nvpairs;
    private String vendor, publisher;
}


/*
 * class CallbackReg
 * -----------
 * This class stores a port and regType (both Strings), and a list of Events.
 * It knows how to construct an SC_CALLBACK_REG XML message from its members.
 *
 * Note that this class does not need to be able to parse SC_CALLBACK_REG
 * messages, because only the CRNP server must parse SC_CALLBACK_REG
 * messages.
 */
class CallbackReg
{
    // Useful defines for the setRegType method
    public static final int ADD_CLIENT = 0;
    public static final int ADD_EVENTS = 1;
    public static final int REMOVE_EVENTS = 2;
    public static final int REMOVE_CLIENT = 3;

    public CallbackReg()
    {
        port = null;
        regType = null;
        regEvents = new Vector();
    }

    /*
     * Public setters.
     */
    public void setPort(String portIn)
    {
        port = portIn;
    }
```

```
public void setRegType(int regTypeIn)
{
    switch (regTypeIn) {
    case ADD_CLIENT:
        regType = "ADD_CLIENT";
        break;
    case ADD_EVENTS:
        regType = "ADD_EVENTS";
        break;
    case REMOVE_CLIENT:
        regType = "REMOVE_CLIENT";
        break;
    case REMOVE_EVENTS:
        regType = "REMOVE_EVENTS";
        break;
    default:
        System.out.println("Error, invalid regType " +
            regTypeIn);
        regType = "ADD_CLIENT";
        break;
    }
}

public void addRegEvent(Event regEvent)
{
    regEvents.add(regEvent);
}


/*
 * convertToXml
 * ------------------
 * Constructs an SC_CALLBACK_REG XML Document from the member
 * variables.  Relies on the Event createXmlElement ability.
 */
public String convertToXml()
{
    Document document = null;
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.newDocument();
    } catch (ParserConfigurationException pce) {
        // Parser with specified options can't be built
        pce.printStackTrace();
        System.exit(1);
    }
    Element root = (Element) document.createElement("SC_CALLBACK_REG");
    root.setAttribute("VERSION", "1.0");
    root.setAttribute("PORT", port);
    root.setAttribute("REG_TYPE", regType);
    for (int i = 0; i < regEvents.size(); i++) {
        Event tempEvent = (Event)
```

```
                (regEvents.elementAt(i));
            root.appendChild(tempEvent.createXmlElement(document));
        }
        document.appendChild(root);

        //
        // Now convert the document to a string.
        //
        DOMSource domSource = new DOMSource(document);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return ("");
        }
        return (strWrite.toString());
    }

    // private member vars
    private String port;
    private String regType;
    private Vector regEvents;
}

/*
 * class RegReply
 * ------------
 * This class stores a status_code and status_msg (both Strings).
 * It knows how to parse an SC_REPLY XML Element into its members.
 */
class RegReply
{

    /*
     * The only constructor takes an XML Document and parses it.
     */
    public RegReply(Document doc)
    {
        //
        // Now convert the document to a string.
        //
        DOMSource domSource = new DOMSource(doc);
        StringWriter strWrite = new StringWriter();
        StreamResult streamResult = new StreamResult(strWrite);
        TransformerFactory tf = TransformerFactory.newInstance();
        try {
            Transformer transformer = tf.newTransformer();
            transformer.transform(domSource, streamResult);
        } catch (TransformerException e) {
            System.out.println(e.toString());
            return;
```

```
    }
    System.out.println(strWrite.toString());

    retrieveValues(doc);
}

/*
 * Public accessors
 */
public String getStatusCode()
{
    return (statusCode);
}

public String getStatusMsg()
{
    return (statusMsg);
}

/*
 * Prints the info on a single line.
 */
public void print(PrintStream out)
{
    out.println(statusCode + ": " +
        (statusMsg != null ? statusMsg : ""));
}

 /*
 * retrieveValues
 * ----------------
 * Parse the XML Document to retrieve the statusCode and statusMsg.
 */
private void retrieveValues(Document doc)
{
    Node n;
    NodeList nl;

    //
    // Find the SC_REPLY element.
    //
    nl = doc.getElementsByTagName("SC_REPLY");
    if (nl.getLength() != 1) {
        System.out.println("Error in parsing: can't find "
            + "SC_REPLY node.");
        return;
    }

    n = nl.item(0);

    // Retrieve the value of the STATUS_CODE attribute
    statusCode = ((Element)n).getAttribute("STATUS_CODE");

    //
    // Find the SC_STATUS_MSG element
```

```
        //
        nl = ((Element)n).getElementsByTagName("SC_STATUS_MSG");
        if (nl.getLength() != 1) {
            System.out.println("Error in parsing: can't find "
                + "SC_STATUS_MSG node.");
            return;
        }

        //
        // Get the TEXT section, if there is one.
        //
        n = nl.item(0).getFirstChild();
        if (n == null || n.getNodeType() != Node.TEXT_NODE) {
            // Not an error if there isn't one, so we
            // just silently return.
            return;
        }

        // Retrieve the value
        statusMsg = n.getNodeValue();
    }

    // private member vars
    private String statusCode;
    private String statusMsg;
}
```

색인

번호와 기호

A

B

C

D

N

O

P

R

S

T

U

정

조

종

지

코

콜

클