Sun Cluster 資料服務開發者指南 (適用於 Solaris 作業系統)

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

文件號碼: 819–0188–10
2004 年 9 月，修訂版 A

041215@10536

目錄

前言　11

1　資源管理簡介　17
Sun Cluster 應用程式環境　17
RGM 模型　18
　資源類型　19
　資源　19
　資源群組　19
資源群組管理員　20
回呼方法　20
程式設計介面　21
　RMAPI　21
　資料服務開發程式庫 (DSDL)　22
　SunPlex Agent Builder　22
資源群組管理員管理介面　22
　SunPlex Manager　22
　管理指令　23

2　開發資料服務　25
分析應用程式的適當性　25
確定要使用的介面　27
設定撰寫資料服務的開發環境　28
　▼ 設定開發環境　28
　將資料服務傳送至叢集　29
設定資源特性和資源類型特性　29
　宣告資源類型特性　30

宣告資源特性 32
宣告延伸特性 35
實施回呼方法 37
存取資源和資源群組特性資訊 37
方法的等冪性 37
一般資料服務 38
控制應用程式 38
啓動與停止資源 38
Init、Fini 以及 Boot 方法 40
監視資源 40
將訊息記錄加入資源 41
提供程序管理 41
爲資源提供管理支援 42
實施故障轉移資源 43
實施可延伸資源 43
可延伸服務的驗證檢查 45
寫入與測試資料服務 46
使用持續連接 46
測試 HA 資料服務 46
協調資源之間的相依性 47

3 升級資源類型 49
簡介 49
資源類型註冊檔案 50
資源類型名稱 50
指令 51
變更 RTR 檔案中的 RT_Version 51
舊版 Sun Cluster 中的資源類型名稱 51
資源 Type_version 屬性 52
將資源遷移至其他版本 53
升級與降級資源類型 53
▼ 如何升級資源類型 53
▼ 如何將資源的資源類型降級爲舊版本 54
預設屬性值 55
資源類型開發者說明文件 56
資源類型名稱與資源類型監視器實作 56
應用程式升級 57
資源類型升級範例 57

資源類型套件的安裝要求 59
變更 RTR 檔案前需要瞭解的資訊 60
變更監視器程式碼 60
變更方法程式碼 60

4 資源管理 API 參考 63
RMAPI 存取方法 64
RMAPI Shell 指令 64
C 函式 65
RMAPI 回呼方法 68
方法引數 68
退出碼 69
控制與初始化回呼方法 69
管理支援方法 70
網路相關的回呼方法 70
監視器控制回呼方法 71

5 資料服務範例 73
資料服務範例概觀 73
定義資源類型註冊檔案 74
RTR 檔案概觀 74
RTR 檔案範例中的資源類型特性 74
RTR 檔案範例中的資源特性 76
爲所有方法提供共用功能性 79
識別指令解譯程式與匯出路徑 79
宣告變數 PMF_TAG 與 SYSLOG_TAG 79
剖析函式引數 80
產生錯誤訊息 82
獲取特性資訊 82
控制資料服務 83
Start 方法 83
Stop 方法 85
定義故障監視器 88
探測程式 88
Monitor_start 方法 93
Monitor_stop 方法 93
Monitor_check 方法 95

處理特性更新 96
Validate 方法 96
Update 方法 100

6 資料服務開發程式庫 (DSDL) 103
DSDL 概觀 103
管理配置屬性 103
啓動與停止資料服務 104
實施故障監視器 105
存取網路位址資訊 105
對資源類型實作進行除錯 106
啓用高度可用的本機檔案系統 106

7 設計資源類型 107
RTR 檔案 107
Validate 方法 108
Start 方法 109
Stop 方法 110
Monitor_start 方法 111
Monitor_stop 方法 111
Monitor_check 方法 112
Update 方法 112
Init、Fini 和 Boot 方法 113
設計故障監視器常駐程式 113

8 DSDL 資源類型實施範例 117
X 字型伺服器 117
X 字型伺服器配置檔案 118
TCP 通訊埠編號 118
命名慣例 118
SUNW.xfnts RTR 檔案 119
scds_initialize() 函式 119
xfnts_start 方法 120
啓動前驗證服務 120
啓動服務 120
從 svc_start() 傳回 122
xfnts_stop 方法 124

xfnts_monitor_start 方法 125
xfnts_monitor_stop 方法 126
xfnts_monitor_check 方法 127
SUNW.xfnts 故障監視器 128
xfonts_probe 主迴圈 128
svc_probe() 函式 130
決定故障監視器的動作 132
xfnts_validate 方法 133
xfnts_update 方法 135

9 SunPlex Agent Builder 137
Agent Builder 簡介 137
使用 Agent Builder 之前 138
建立具有多個獨立程序樹的資源類型 138
使用 Agent Builder 139
分析應用程式 139
安裝與配置 Agent Builder 139
Agent Builder 螢幕 140
啓動 Agent Builder 140
導覽 Agent Builder 141
使用 [建立] 畫面 144
使用 [配置] 畫面 146
使用基於 Agent Builder Korn Shell 的變數 $hostnames 148
特性變數 148
重複使用完成的工作 150
▼ 如何使用 Agent Builder 的指令行版本 151
目錄結構 152
Agent Builder 輸出 152
來源檔與二進位檔 153
公用程式程序檔與線上援助頁 154
支援檔案 155
套件目錄 155
rtconfig 檔案 155
Agent Builder 的 Cluster Agent 模組 156
▼ 如何安裝與設置 Cluster Agent 模組 156
▼ 如何啓動 Cluster Agent 模組 157
使用 Cluster Agent 模組 158
Cluster Agent 模組與Agent Builder 之間的差異 159

10 一般資料服務 161
GDS 簡介 161
預先編譯的資源類型 161
使用 GDS 的優勢和劣勢 162
建立使用 GDS 的服務之方法 162
GDS 如何記錄事件 163
必需的 GDS 特性 163
可選擇的 GDS 特性 164
使用 SunPlex Agent Builder 建立使用 GDS 的服務 166
建立與配置程序檔 166
SunPlex Agent Builder 的輸出 170
使用標準 Sun Cluster 管理指令建立使用 GDS 的服務 170
▼ 如何使用 Sun Cluster 管理指令建立使用 GDS 的高度可用服務 171
▼ 如何使用 Sun Cluster 管理指令建立使用 GDS 的可延伸服務 171
SunPlex Agent Builder 的指令行介面 172
▼ 如何透過 Agent Builder 的指令行版本建立使用 GDS 的服務 172

11 資料服務開發程式庫參考 175
DSDL 函式 175
通用函式 175
特性函式 176
網路資源存取函式 176
使用 TCP 連接的故障監視 177
PMF 函式 178
故障監視器函式 178
公用程式函式 178

12 CRNP 179
CRNP 概觀 179
CRNP 協定概觀 180
CRNP 使用的訊息類型 182
用戶端如何註冊到伺服器 183
有關管理員如何設定伺服器的假定 183
伺服器識別用戶端的方式 183
在用戶端與伺服器之間傳送 SC_CALLBACK_REG 訊息的方式 183
伺服器回覆用戶端的方式 185
SC_REPLY 訊息的內容 185

用戶端處理錯誤狀況的方式 186
伺服器如何將事件發送給用戶端 186
如何保證事件的發送 187
SC_EVENT 訊息的內容 187
CRNP 如何授權用戶端與伺服器 189
建立使用 CRNP 的 Java 應用程式 190
▼ 設定環境 190
▼ 開始 191
▼ 剖析指令行引數 192
▼ 定義事件接收執行緒 193
▼ 註冊與取消註冊回呼 194
▼ 產生 XML 194
▼ 建立註冊訊息與取消註冊訊息 198
▼ 設定 XML 剖析器 200
▼ 剖析註冊回覆 201
▼ 剖析回呼事件 202
▼ 執行應用程式 205

A 標準特性 207
資源類型特性 207
資源特性 213
資源群組特性 222
資源特性性質 228

B 範例資料服務程式碼清單 231
資源類型註冊檔案清單 231
Start 方法 234
Stop 方法 237
gettime 公用程式 239
PROBE 程式 240
Monitor_start 方法 245
Monitor_stop 方法 247
Monitor_check 方法 248
Validate 方法 250
Update 方法 253

C 資料服務開發程式庫資源類型程式碼範例清單 257
xfnts.c 257
xfnts_monitor_check 方法 269
xfnts_monitor_start 方法 270
xfnts_monitor_stop 方法 271
xfnts_probe 方法 272
xfnts_start 方法 275
xfnts_stop 方法 276
xfnts_update 方法 277
xfnts_validate 方法程式碼清單 279

D 合法的 RGM 名稱和值 281
RGM 的合法名稱 281
適用於除資源類型名稱以外的名稱之規則 281
資源類型名稱的格式 282
RGM 的值 283

E 不支援叢集的應用程式的要求 285
多重主機資料 285
使用多重主機資料放置的符號連結 286
主機名稱 286
多重主目錄主機 287
連結至 INADDR_ANY 與連結至特定 IP 位址 287
用戶端重試 288

F CRNP 的文件類型定義 289
SC_CALLBACK_REG XML DTD 289
NVPAIR XML DTD 291
SC_REPLY XML DTD 292
SC_EVENT XML DTD 293

G CrnpClient.java 應用程式 295
CrnpClient.java 的內容 295

索引 317

前言

「*Sun Cluster* **資料服務開發者指南 (適用於** *Solaris* **作業系統)**」包含有關使用資源管理 API 在 SPARC® 與 x86 型系統上開發 Sun™ Cluster 資料服務的資訊。

注意 – 在本文件中，「x86」指 Intel 32 位元系列的微處理器晶片和由 AMD 製造的相容微處理器晶片。

注意 – Sun Cluster 軟體在兩個平台 (SPARC 與 x86上) 上執行。本文件中的資訊適用於這兩個平台，除非在特定章節、小節、備註、項目符號、圖形、表格或範例中另行指定。

本書適用對象

本文件適合經驗豐富且對 Sun 軟硬體非常熟悉的開發者。本書中的資訊假定使用者瞭解 Solaris™ 作業系統。

本書的組織架構

「*Sun Cluster* **資料服務開發者指南 (適用於** *Solaris* **作業系統)**」包含下列章節與附錄：

- 第 1 章提供開發資料服務所需概念的簡介。
- 第 2 章提供有關開發資料服務的詳細資訊。

- 第 3 章論述升級資源類型與遷移資源所需瞭解的問題
- 第 4 章提供了組成資源管理 API (RMAPI) 的存取函式與回呼方法的參考。
- 第 5 章提供了 `in.named()` 應用程式的 Sun Cluster 資料服務範例。
- 第 6 章概述了組成資料服務開發程式庫 (DSDL) 的應用程式設計介面。
- 第 7 章說明 DSDL 在設計和實現資源類型方面的一般用法。
- 第 8 章描述一個使用 DSDL 實現的資源類型範例。
- 第 9 章描述 SunPlex ™ Agent Builder。
- 第 10 章描述如何建立常規資料服務。
- 第 11 章描述 DSDL API 函式。
- 第 12 章提供有關叢集重新配置通知協定 (CRNP) 的資訊。CRNP 使故障轉移應用程式與可延伸應用程式能夠「支援叢集」。
- 附錄 A描述了標準資源類型、資源群組與資源特性。
- 附錄 B 提供範例資料服務中每個方法的完整代碼。
- 附錄 C 列出 `SUNW.xfnts()` 資源類型中每個方法的完整代碼。
- 附錄 D 列出對資源群組管理員 (RGM) 名稱與值的合法字元的要求。
- 附錄 E 列出對一般不支援叢集應用程式要達到高可用性的要求。
- 附錄 F 列出 CRNP 的文件類型定義。
- 附錄 G 顯示在第 12 章中論述之完整的 `CrnpClient.java` 應用程式。

相關說明文件

有關 Sun Cluster 相關主題的資訊可從下表中列出的說明文件獲得。所有 Sun Cluster 說明文件均可從 `http://docs.sun.com` 取得。

主題	文件資料
簡介	「*Sun Cluster* **簡介** *(***適用於** *Solaris* **作業系統***)*」
概念	「*Sun Cluster* **概念指南** *(***適用於** *Solaris* **作業系統***)*」
硬體安裝與管理	「*Sun Cluster 3.x Hardware Administration Manual for Solaris OS*」 個別硬體管理指南
軟體安裝	「*Sun Cluster* **軟體安裝指南** *(***適用於** *Solaris* **作業系統***)*」

主題	文件資料
資料服務安裝與管理	「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」 個別資料服務指南
資料服務開發	「*Sun Cluster* **資料服務開發者指南** *(***適用於** *Solaris* **作業系統***)*」
系統管理	「*Sun Cluster* **系統管理指南** *(***適用於** *Solaris* **作業系統***)*」
錯誤訊息	「*Sun Cluster Error Messages Guide for Solaris OS*」
指令和功能參考	「*Sun Cluster Reference Manual for Solaris OS*」

如需 Sun Cluster 文件的完整清單，請參閱 `http://docs.sun.com` 上關於您的 Sun Cluster 軟體版本之版本說明。

Sun Cluster 文件的完整清單可於 `http://docs.sun.com` 處的您 Sun Cluster 發行版本的版本說明中取得。

取得說明

如果在安裝或使用 Sun Cluster 上有問題，請與您的服務供應商聯絡並提供下列資訊。

- 您的姓名和電子郵件地址 (如果有的話)
- 您的公司名稱、地址和電話號碼
- 您系統的機型和序號
- 作業系統的版次編號 (例如，Solaris 10)
- Sun Cluster 的版次編號 (例如，Sun Cluster 3.1)

請使用以下指令爲服務供應商收集有關您系統的資訊。

指令	功能
`prtconf -v`	顯示系統記憶體的大小及報告周邊裝置的相關資訊
`psrinfo -v`	顯示處理器的相關資訊
`showrev -p`	報告安裝了哪些修補程式
SPARC: `prtdiag -v`	顯示系統診斷資訊
`/usr/cluster/bin/scinstall - pv`	顯示 Sun Cluster 版次與套件版本資訊

同時提供 `/var/adm/messages` 檔案的內容。

線上存取 Sun 說明文件

docs.sun.com℠ 網站可讓您存取 Sun 線上技術文件。您可以瀏覽 docs.sun.com 的歸檔檔案或搜尋特定書名或主題。其 URL 是 `http://docs.sun.com`。

訂購 Sun 說明文件

Sun Microsystems 提供列印的選取產品說明文件。如需文件清單及文件的訂購方式，請參閱 http://docs.sun.com/?l=zh_TW 網站上的「購買書面文件」。

印刷排版慣例

下表說明本書在印刷上所作的變更。

表 P–1 印刷排版慣例

字體或符號	涵義	範例
`AaBbCc123`	指令、檔案和目錄的名稱；電腦螢幕的輸出	編輯您的 `.login` 檔案。 使用 `ls -a` 列出所有檔案。 `machine_name% you have mail`。
`AaBbCc123`	您輸入的內容，與電腦螢幕上的輸出相對照	`machine_name%` **`su`** `Password:`
AaBbCc123	指令行預留位置：用實際名稱或值取代	若要刪除某個檔案，請輸入 **`rm`** ***檔案名稱***。
AaBbCc123	書名、新字詞、專有名詞或要強調的字	請閱讀「**使用者指南**」中的第 6 章。 這些稱爲**類別**選項。 您必須是 *root*，才能執行此動作。

指令範例中的 Shell 提示符號

以下表格列出使用於 C shell、Bourne shell 和 Korn shell 的預設系統提示符號以及超級使用者提示符號。

表 P–2 Shell 提示符號

Shell	提示符號
C shell 提示符號	`machine_name%`
C shell 超級使用者提示符號	`machine_name#`
Bourne shell 和 Korn shell 提示符號	`$`
Bourne shell 和 Korn shell 超級使用者提示符號	`#`

第 1 章

資源管理簡介

本書爲 Oracle®、Sun Java™ System Web Server (以前爲 Sun™ ONE Web Server)、DNS 等軟體應用程式提供建立資源類型的指導。因此，本書主要針對資源類型的開發人員。

本章概述了開發資料服務所需要瞭解的概念，包含下列資訊。

- 第 17 頁的 「Sun Cluster 應用程式環境」
- 第 18 頁的 「RGM 模型」
- 第 20 頁的 「資源群組管理員」
- 第 20 頁的 「回呼方法」
- 第 21 頁的 「程式設計介面」
- 第 22 頁的 「資源群組管理員管理介面」

注意 – 本書中的術語**資源類型**與**資料服務**可以交換使用。雖然術語**代理程式**在本書中很少用到，但是與**資源類型**與**資料服務**是對等的。

Sun Cluster 應用程式環境

Sun Cluster 系統可使應用程式作爲高度可用資源與可延伸資源被執行與管理。稱爲「資源群組管理員」或者 RGM 的叢集工具提供了高可用性與可延伸性的機制。形成此工具程式設計介面的元素包括以下幾種。

- 您所寫入的回呼方法集，使 RGM 可以控制叢集上的應用程式
- 資源管理 API (RMAPI)，可用來寫入回呼方法的低階 API 指令與函式集。這些 API 是在 `libscha.so`程式庫中實現的。
- 在叢集上監視與重新啓動程序的程序管理工具
- 資料服務開發程式庫 (DSDL)，爲程式庫函式集，用於封裝低階 API 與較高階程序管理功能和新增一些額外功能性，以簡化回呼方法的寫入。這些函式是在 `libdsdev.so` 程式庫中實現的。

下圖顯示這些元素之間的相互關係。

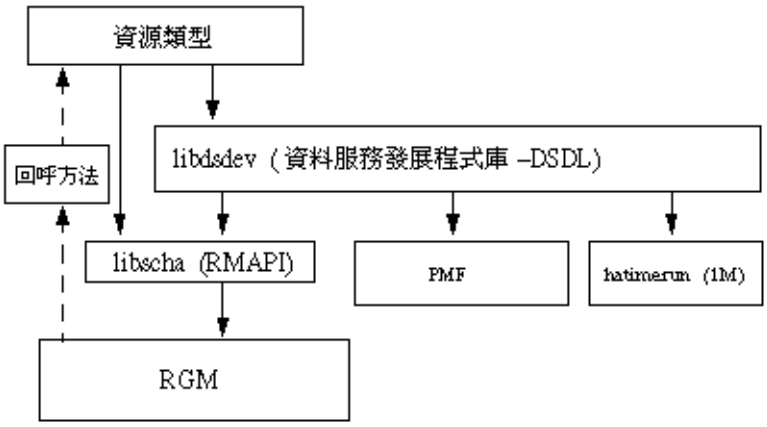


圖 1–1 程式設計架構

Sun Cluster 套裝軟體中包含 SunPlex™ Agent Builder，一種使資料服務的建立自動進行的工具 (請參閱第 9 章)。Agent Builder 在 C shell (使用 DSDL 函式寫入回呼方法) 或者 Korn shell (`ksh`) (使用低層級 API 指令寫入回呼方法) 產生資料服務程式碼。

RGM 在每個叢集節點上作爲常駐程式執行，並依照預先配置的策略自動啓動與停止所選節點上的資源。RGM 在節點發生故障或重新啓動時，透過停止受影響節點上的資源並在另一個節點上啓動資源，使該資源高度可用。RGM 還自動啓動與停止資源特定的監視器，該監視器可以偵測資源故障，並將發生故障的資源重新配置到另一個節點，或者可以監視資源效能的其他方面。

RGM 支援故障轉移資源 (一次最多可在一個節點上連線) 與可延伸資源 (同時可在多個節點上連線)。

RGM 模型

本節介紹一些基本術語，並詳細說明 RGM 及其關聯的介面。

RGM 處理以下三種主要的相互關聯物件：資源類型、資源以及資源群組。引入這些物件的一種方法是使用範例，如下所述。

開發者實施資源類型 `ha-oracle`，使得現有 Oracle DBMS 應用程式高度可用。一般使用者爲市場營銷、工程與金融定義獨立的資料庫，每個資料庫皆爲一種類型爲 `ha-oracle` 的資源。叢集管理員將這些資源放入獨立的資源群組中，以便這些資源可

以在不同的節點上執行並獨立進行故障轉移。開發者建立第二種資源類型 ha-calendar，以實施需要 Oracle 資料庫的高度可用日曆伺服器。叢集管理員將金融日曆的資源放入與金融資料庫資源相同的資源群組，以便這兩種資源在同一節點上執行並共同進行故障轉移。

資源類型

資源類型由在叢集上執行的軟體應用程式、作爲回呼方法由 RGM 用來管理應用程式 (作爲叢集資源) 的控制程式以及構成叢集部分靜態配置的特性集組成。RGM 使用資源類型特性來管理特定類型的資源。

注意 – 除軟體應用程式之外，資源類型可以表示其他系統資源 (例如，網路位址)。

資源類型開發者在資源類型註冊 (RTR) 檔案中指定資源類型的特性，並設定其值。RTR 檔案依循明確定義的格式，如第 29 頁的 「設定資源特性和資源類型特性」與 `rt_reg`(4) 線上說明手冊中所述。亦請參閱第 74 頁的 「定義資源類型註冊檔案」，以取得資源類型註冊檔案範例的說明。

第 207 頁的 「資源類型特性」提供資源類型特性的清單。

叢集管理員在叢集上安裝與註冊資源類型實作與基礎應用程式。註冊程序在叢集配置中輸入資源類型註冊檔案的資訊。「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」描述註冊資料服務的程序。

資源

資源繼承其資源類型的特性和值。此外，開發者可以在資源類型註冊檔案中宣告資源特性。第 213 頁的 「資源特性」包含資源特性的清單。

叢集管理員可以依據在資源類型註冊 (RTR) 檔案中指定特定特性的方法來變更這些特性的值。例如，特性定義可以指定可允許值的範圍，還可指定該特性何時可調 (例如，建立時、任何時候或者永不)。在這些規格內，叢集管理員可以使用管理指令對特性進行變更。

叢集管理員可以建立許多相同類型的資源，每種資源有其自己的名稱與特性值集，以便基礎應用程式的多個實例可以在叢集上執行。在叢集內，每個創設均需要唯一的名稱。

資源群組

每種資源均必須在資源群組中進行配置。RGM 使群組中的所有資源在同一節點上一同上線與離線。當 RGM 使資源群組上線或離線時，會對群組中的個別資源呼叫回呼方法。

資源群組目前在其上處於線上狀態的節點稱爲該資源群組的**主節點**或**主要節點**。資源群組由其每個主節點**主控**。每個資源群組均具有相關 `Nodelist` 特性，該特性由叢集管理員來設定，可以識別資源群組的所有**潛在主節點**或主控者。

資源群組還具有一個特性集。這些特性包括配置特性 (可以由叢集管理員來設定) 與動態特性 (由 RGM 設定，反映資源群組的使用中狀態)。

RGM 定義兩種類型的資源群組，故障轉移資源群組與可延伸資源群組。故障轉移資源群組在任何時候僅可以在一個節點上連線，而可延伸資源群組可以同時在多個節點上連線。RGM 提供支援建立每種類型資源群組的特性集。請參閱第 29 頁的 「將資料服務傳送至叢集」與第 37 頁的 「實施回呼方法」，以取得有關這些特性的詳細資訊。

第 222 頁的 「資源群組特性」包含資源群組特性的清單。

資源群組管理員

資源群組管理員 (RGM) 是作爲在叢集的每個成員節點上執行的常駐程式 `rgmd` 來實現的。所有 `rgmd` 程序互相通訊，並作爲單一叢集範圍的工具一起動作。

RGM 支援以下功能：

- 無論何時節點啓動或者當機，RGM 均會嘗試透過自動使所有受管理的資源群組在適當的主控者上上線來維護這些資源群組的可用性。
- 如果特定資源失敗，則其監視器程式可以要求在同一主控者上重新啓動資源群組，或者將該資源群組切換至新的主控者。
- 叢集管理員可以發出管理指令，以要求下列動作之一：
  - 變更資源群組的主控者
  - 啓用或停用資源群組內的特定資源
  - 建立、刪除或者修改資源、資源群組或資源類型

每當 RGM 啓動配置變更，均會在叢集的所有成員節點之間協調其動作。這種活動稱爲重新配置。爲了在個別資源上使狀態變更生效，RGM 會針對該資源呼叫資源類型特有的回呼方法。

回呼方法

Sun Cluster 框架使用一種回呼機制，提供資料服務與 RGM 之間的通訊。該框架定義一組回呼方法，包括這些方法的引數與傳回值，以及 RGM 呼叫每種方法所處的環境。

透過編碼一組個別回呼方法，並將每種方法作爲可由 RGM 呼叫的控制程式來實施，便可建立資料服務。也就是說，資料服務不是由單一可執行檔組成，而是由多個可執行程序檔 (`ksh`) 或者二進位檔 (C) 組成，其中的每個檔案均可由 RGM 直接呼叫。

使用 RGM 透過資源類型註冊 (RTR) 檔案來註冊回呼方法。在 RTR 檔案中，您要識別針對資料服務已實施的每種方法之程式。當系統管理員在叢集上註冊資料服務時，RGM 會讀取 RTR 檔案，該檔案提供在其他資訊中對回呼程式的識別。

資源類型必需的回呼方法僅僅包括啓動方法 (`Start` 或 `Prenet_start`) 與停止方法 (`Stop` 或 `Postnet_stop`)。

可以將呼叫方法群組爲下列種類：

- 控制與初始化方法
  - `Start` 與 `Stop` 啓動與停止上線或離線的群組中之資源。
  - `Init`、`Fini`、`Boot` 在資源上執行初始化程式碼與終止程式碼。
- 管理支援方法
  - `Validate` 驗證由管理動作設定的特性。
  - `Update` 更新線上資源的特性設定。
- 網路相關的方法
  - `Prenet_start` 與 `Postnet_stop` 在配置同一資源群組中的網路位址之前或取消配置這些網路位址之後，進行特定的啓動動作或者關閉動作。
- 監視器控制方法
  - `Monitor_start` 與 `Monitor_stop` 爲資源啓動或停止監視器。
  - `Monitor_check` 在將資源群組移到節點之前評估該節點的可靠性。

請參閱第 4 章與 `rt_callbacks`(1HA) 線上援助頁，以取得有關回呼方法的詳細資訊。另請參閱第 5 章與第 8 章，以取得資料服務範例中的回呼方法。

程式設計介面

若要寫入資料服務程式碼，資源管理架構會提供低層級 API 或基本 API、建立在基本 API 之上的較高層級程式庫，還會提供從您所提供的基本輸入自動產生資料服務的工具 SunPlex Agent Builder。

RMAPI

RMAPI (資源管理 API) 提供一組低層級常式，這些常式可讓資料服務在系統中存取有關資源、資源類型以及資源群組的資訊，要求本機重新啓動或者故障轉移以及設定資源狀況。您要透過 `libscha.so` 程式庫存取這些函式。RMAPI 以 shell 指令形式與 C 函式形式提供這些回呼方法。請參閱 `scha_calls`(3HA) 與第 4 章，以取得有關 RMAPI 常式的更多資訊。亦請參閱第 5 章，以取得在資料服務回呼方法範例中如何使用這些常式的範例。

資料服務開發程式庫 (DSDL)

建立在 RMAPI 頂層的是 DSDL，它在保留 RGM 的以下回呼方法模型的同時提供了更高階的整合框架。DSDL 爲資料服務開發提供了各種工具，包括：

- `libscha.so` — 低層級資源管理 API
- PMF — 程序管理工具，提供監視程序及其子代與在這些程序失敗時重新啓動它們的方法 (請參閱 `pmfadm`(1M) 與 `rpc.pmfd`(1M))。
- `hatimerun` — 在逾時情況下執行程式的工具 (請參閱 `hatimerun`(1M)。

對於大多數應用程式來說，DSDL 均會提供建立資料服務所需的大部分或者全部功能性。然而，請注意，DSDL 不會取代低層級 API，而是將其封裝與延伸。實際上，許多 DSDL 函式呼叫 `libscha.so` 函式。同樣，您可以在使用 DSDL 編碼大部分資料服務時直接呼叫 `libscha.so` 函式。`libdsdev.so` 程式庫包含 DSDL 函式。

請參閱第 6 章與 `scha_calls`(3HA) 線上援助頁，以取得有關 DSDL 的詳細資訊。

SunPlex Agent Builder

Agent Builder 是一種使資料服務的建立自動進行的工具。輸入有關目標應用程式與要建立之資料服務的基本資訊。Agent Builder 將產生完整的資料服務，包含源代碼與可執行代碼 (C 或 Korn shell)、自訂 RTR 檔案以及 Solaris™ 套裝軟體。

對於大多數應用程式來說，您可以使用 Agent Builder 來產生完整的資料服務，僅需您進行次要的手動變更即可。具有較複雜要求的應用程式 (例如對其他特性新增驗證檢查) 可能需要 Agent Builder 所無法完成的工作。然而，即使出現這些情況，您仍可以使用 Agent Builder 產生大部分程式碼，而其餘部分透過手動編碼來完成。至少，還可以使用 Agent Builder 爲您產生 Solaris 套件。

資源群組管理員管理介面

Sun Cluster 提供用於管理叢集的圖形使用者介面與指令集。

SunPlex Manager

SunPlex Manager 是一種基於 Web 的工具，使您能夠執行下列作業。

- 安裝叢集
- 管理叢集
- 建立和配置資源與資源群組
- 使用 Sun Cluster 軟體配置資料服務

請參閱「*Sun Cluster* **軟體安裝指南 (適用於** *Solaris* **作業系統)**」，以取得安裝 SunPlex Manager 與使用 SunPlex Manager 來安裝叢集軟體之方法的說明。SunPlex Mnager 提供大多數唯一管理作業的線上說明。

管理指令

管理 RGM 物件的 Sun Cluster 指令爲 `scrgadm`(1M)、`scswitch`(1M) 和 `scstat`(1M)- `g`。

`scrgadm` 指令允許檢視、建立、配置、刪除 RGM 使用的資源類型、資源群組與資源物件。指令是叢集管理介面的一部分，在同一程式設計環境中不能用來作爲應用程式介面，本章的餘下部分有相關說明。然而，`scrgadm` 是建構 API 作業所在的叢集配置之工具。瞭解管理介面可設定瞭解應用程式介面的環境。請參閱 `scrgadm`(1M) 線上說明手冊，以取得有關該指令可以執行之管理作業的詳細資訊。

`scswitch` 指令在指定的節點上將資源群組切換爲線上狀態或離線狀態，以及啓用或停用資源或者其監視器。請參閱 `scswitch`(1M) 線上說明手冊，以取得有關該指令可以執行之管理作業的詳細資訊。

`scstat -g` 指令顯示目前所有資源群組與資源的動態狀態。

第 2 章

開發資料服務

本章提供有關開發資料服務的詳細資訊。

本章包含以下主題：

- 第 25 頁的 「分析應用程式的適當性」
- 第 27 頁的 「確定要使用的介面」
- 第 28 頁的 「設定撰寫資料服務的開發環境」
- 第 29 頁的 「設定資源特性和資源類型特性」
- 第 37 頁的 「實施回呼方法」
- 第 38 頁的 「一般資料服務」
- 第 38 頁的 「控制應用程式」
- 第 40 頁的 「監視資源」
- 第 41 頁的 「將訊息記錄加入資源」
- 第 41 頁的 「提供程序管理」
- 第 42 頁的 「爲資源提供管理支援」
- 第 43 頁的 「實施故障轉移資源」
- 第 43 頁的 「實施可延伸資源」
- 第 46 頁的 「寫入與測試資料服務」

分析應用程式的適當性

建立資料服務的第一步爲確定目標應用程式滿足高度可用或可延伸的要求。如果該應用程式不能滿足所有要求，您可以修改該應用程式的來源代碼以使其滿足所有要求。

以下清單概括了使應用程式高度可用或可延伸所應滿足的要求。如果您需要更多詳細資訊，或您需要修改應用程式來源代碼，請參閱附錄 B。

注意 – 可延伸服務必須滿足所有下列條件以及一些其他準則，才能高度可用。

- 支援網路 (主從式模型) 的應用程式和不支援網路 (無用戶端) 的應用程式均可在 Sun Cluster 環境中變得高度可用或可延伸。但是，Sun Cluster 在分時環境中無法提供增強的可用性，因爲在這種環境中，執行應用程式的伺服器是透過 `telnet` 或 `rlogin` 存取的。
- 該應用程式必須爲當機容限。也就是說，它必須能在非預期的節點故障後，重新啓動時回復磁碟資料 (如有必要)。而且，當機後的回復時間必須是有限的。當機容限是應用程式高度可用所應必備的條件，因爲回復磁碟並重新啓動該應用程式的能力關係到資料完整性問題。不需要資料服務即可回復連接
- 應用程式不得依賴其執行所在節點的實體主機名稱。請參閱第 286 頁的 「主機名稱」，以取得其他資訊。
- 應用程式必須在配置多個 IP 位址的環境中正常作業；例如具有多位址主機的環境 (在這種環境中節點位於多個公用網路上)，以及具有多個節點的環境，在這些節點上多個邏輯介面被配置爲在一個硬體介面上。
- 若要高度可用，應用程式資料必須駐留在叢集檔案系統中 — 請參閱第 285 頁的 「多重主機資料」。

  如果應用程式使用該資料位置的固定連線路徑名稱，您可以將該路徑變更爲指向叢集檔案系統中某個位置的符號連結，而無需變更應用程式來源代碼。請參閱第 286 頁的 「使用多重主機資料放置的符號連結」，以取得其他資訊。
- 應用程式二進位檔和程式庫可駐留在本機的每個節點上或叢集檔案系統上。駐留在叢集檔案系統上的優勢是單一安裝即足夠。其劣勢是：由於應用程式在 RGM 控制下執行時二進位檔正處於使用中狀態，因此滾動更新就成了一個問題。
- 用戶端應該在首次嘗試逾時後能自動重試查詢。如果應用程式和協定已處理了單一伺服器當機和重新啓動的情況，則它們也將處理故障轉移或切換保護移轉資源所在資源群組的情況。請參閱第 288 頁的 「用戶端重試」，以取得其他資訊。
- 應用程式不得在叢集檔案系統中具有UNIX® 網域套接字或已命名的管道。

此外，可延伸服務必須滿足下列要求。

- 應用程式必須能執行多個實例，且多個實例均運行在叢集檔案系統的同一應用程式資料上。
- 應用程式必須爲多個節點的同時存取提供資料一致性。
- 應用程式必須透過全域可視機制 (如叢集檔案系統) 實施足夠的鎖護。

對於可延伸服務，應用程式特性還決定了負載平衡策略。例如，負載平衡策略 `LB_WEIGHTED` (允許任意實例對用戶端請求做出回應) 不適用於爲用戶端連接使用伺服器上記憶體內快取的應用程式。在此情況下，您應該指定負載平衡策略，將指定用戶端的流量限定爲應用程式的一個實例。負載平衡策略 `LB_STICKY` 和 `LB_STICKY_WILD` 多次將某個用戶端的所有請求發送至同一應用程式實例 — 在此它們可使用記憶體內快取。請注意，如果多個用戶端請求來自不同用戶端，則 RGM 在該服務的實例間分配這些請求。請參閱第 43 頁的 「實施故障轉移資源」，以取得有關設定可延伸資料服務之負載平衡策略的詳細資訊。

確定要使用的介面

Sun Cluster 開發者支援套件 (SUNWscdev) 提供了兩組介面用於編碼資料服務方法：

- 資源管理 API (RMAPI)，一組低層級常式 (位於 libscha.so 程式庫中)
- 資料服務開發程式庫 (DSDL)，一組較高層級函式 (位於 libdsdev.so 程式庫中)，它們封裝了 RMAPI 的功能性，並提供了一些其他功能性

Sun Cluster 開發者支援套件還包括 SunPlex Agent Builder，一個自動化資料服務建立的工具。

建議採用以下方法開發資料服務：

1. 決定是在 C shell 還是在 Korn shell 中編碼。如果您決定使用 Korn shell，則無法使用 DSDL，因爲 DSDL 僅提供 C 介面。
2. 執行 Agent Builder，指定要求的輸入，產生資料服務。該資料服務包括來源代碼和可執行程式碼、一個 RTR 檔案和一個套件。
3. 如果產生的資料服務需要自訂，您可以將 DSDL 程式碼加入至產生的來源檔案中。Agent Builder 透過註釋指示來源檔案中可自行加入程式碼的特定位置。
4. 如果程式碼需要進一步的自訂才能支援目標應用程式，您可以將 RMAPI 函式加入現有來源代碼中。

實際上，您可採用很多方法來建立資料服務。例如，您可以使用透過 DSDL 或 RMAPI 函式從頭撰寫的程式，完全取代產生的方法之一或產生的監視程式，而不是自行將程式碼加入至 Agent Builder 所產生程式碼的特定位置。但是，不管您採用何種模式，幾乎在所有情況下，以 Agent Builder 開始都是有道理的，其原因如下：

- Agent Builder 產生的程式碼雖然本質上一般，但已在許多資料服務中經過測試。
- Agent Builder 產生一個 RTR 檔案、一個 make 程式檔案、一個資源套件以及資料服務的其他支援檔案。即使您不使用任何資料服務程式碼，使用其他這些檔案也能節省您相當的工作量。
- 您可以修改產生的程式碼。

注意 – 與 RMAPI (提供一組 C 函式和一組在程序檔中使用的指令) 不同，DSDL 僅提供一個 C 函式介面。因此，如果您在 Agent Builder 中指定 Korn shell (ksh) 輸出，則產生的來源代碼將向 RMAPI 發出呼叫，這是因爲沒有 DSDL ksh 指令。

設定撰寫資料服務的開發環境

在開始資料服務開發之前，您必須已安裝了 Sun Cluster 開發套件 (SUNWscdev)，才能存取 Sun Cluster 標頭和程式庫檔案。儘管所有叢集節點上均已安裝了此套件，但一般是在獨立的、非叢集開發機器上而非叢集節點上開發。在此一般情況下，您必須使用 pkgadd 在開發機器上安裝 SUNWscdev 套件。

當編譯和連結程式碼時，您必須設定特別選項以識別標頭和程式庫檔案。

注意 – 在 Solaris 作業系統與 Sun Cluster 產品中，您不能混淆相容模式編譯的 C++ 程式碼與標準模式編譯的 C++ 程式碼。因此，如果您打算建立基於 C++ 的資料服務以用於 Sun Cluster，則必須如下所示編譯該資料服務：

- 對於 Sun Cluster 3.0 及更早的版本，請使用相容模式。
- 從 Sun Cluster 3.1 開始，使用標準模式。

當完成開發 (在非叢集節點上) 後，您可以將完成的資料服務傳送至某個叢集來執行和測試。

注意 – 請確保您使用的是 Solaris 5.8 或更高版本 Solaris 作業系統的 Developer 或 Entire Distribution 軟體群組。

使用本節中的程序來：

- 安裝 Sun Cluster 開發套件 (SUNWscdev) 和設定適當的編譯程式選項和連結程式選項。
- 將資料服務傳送至叢集

▼ 設定開發環境

此程序說明如何安裝 SUNWscdev 套件和如何爲資料服務開發設定編譯程式選項和連結程式選項。

1. **成為超級使用者，或成為對等角色並將目錄變更為您要使用的 CD-ROM 目錄。**

   ```
   # cd CD-ROM_directory
   ```

2. **在目前目錄中安裝 SUNWscdev 套件。**

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

3. **在 `Makefile` 中，指定識別用於資料服務程式碼的包含檔案和程式庫檔案的編譯程式和連結程式選項。**

   指定 `-I` 選項來識別 Sun Cluster 標頭檔案；指定 `-L` 選項來指定開發系統上的編譯期間程式庫搜尋路徑；指定 `-R` 選項來指定叢集上執行期間連結程式的程式庫搜尋路徑。

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

-I /usr/cluster/include

-L /usr/cluster/lib

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

將資料服務傳送至叢集

當您在開發機器上完成資料服務的開發後，必須將其傳送至叢集進行測試。爲了減少發生錯誤的機會，完成此傳送的最佳方法是將資料服務程式碼和 RTR 檔案封裝在一起，然後在叢集的所有節點上安裝該套件。

注意 – 不管您使用 `pkgadd` 還是使用某種其他方法來安裝資料服務，均必須將該資料服務放置在所有叢集節點上。Agent Builder 自動將 RTR 檔案和資料服務程式碼封裝在一起。

設定資源特性和資源類型特性

Sun Cluster 提供一組資源類型特性和資源特性，您可以使用這些特性定義資料服務的靜態配置。資源類型特性指定資源的類型、資源的版本、API 的版本等，同時還指定每個回呼方法的路徑。第 207 頁的 「資源類型特性」列出了全部資源類型特性。

資源特性 (如 `Failover_mode`、`Thorough_probe_interval`) 和方法逾時還定義了資源的靜態配置。動態資源特性 (如 `Resource_state` 和 `Status`) 反映受管理資源的作用中狀態。第 213 頁的 「資源特性」描述了資源特性。

您可在資源類型註冊 (RTR) 檔案中宣告資源類型和資源特性，該檔案是資料服務必不可少的元件。RTR 檔案在叢集管理員將資料服務註冊到 Sun Cluster 時定義資料服務的初始配置。

因爲 Agent Builder 宣告的特性集對任意資料服務都是有用且必需的，所以建議您使用 Agent Builder 產生資料服務的 RTR 檔案。例如，必須在 RTR 檔案中宣告某些特性 (如 `Resource_type`)，否則資料服務的註冊將會失敗。儘管其他特性非必需，但除非系統

管理員在 RTR 檔案中對其進行宣告，否則將無法使用；然而有些特性無論您是否對其進行宣告都可用，這是因爲 RGM 定義它們並提供預設值。爲了避免此層複雜性，您可簡單地使用 Agent Builder 來保證產生正確的 RTR 檔案。如果需要，您可以稍後編輯 RTR 檔案來變更特定值。

本小節的其餘部分將爲您完整介紹一個 Agent Builder 建立的 RTR 檔案範例。

宣告資源類型特性

叢集管理員無法配置您在 RTR 檔案中宣告的資源類型特性。它們成爲該資源類型永久配置的一部分。

注意 – 一種資源類型特性 `Installed_nodes` 可由系統管理員配置。實際上，它僅可由系統管理員配置，您無法在 RTR 檔案中宣告它。

資源類型宣告的語法爲：

property_name = value;

注意 – RGM 處理特性名稱時不區分大小寫。在 Sun 提供的 RTR 檔案中，除了方法名稱，特性的慣例是名稱的第一個字母大寫，而其餘的字母小寫。方法名稱以及特性性質包含的都是大寫字母。

以下爲資料服務範例 (`smpl`) 的 RTR 檔案中的資源類型宣告：

```
# 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;
```

提示 – 您必須宣告 `Resource_type` 特性爲 RTR 檔案中的第一個項目。否則，資源類型的註冊將會失敗。

第一組資源類型宣告提供資源類型的基本資訊，如下所示：

`Resource_type` 和 `Vendor_id`

提供資源類型的名稱。您可以單獨使用 `Resource_type` 特性來指定 (`smpl`)；或使用 `Vendor_id` 作爲前綴，並用「.」將其與資源類型分隔 (`SUNW.smpl`)，如範例中所示。如果您使用 `Vendor_id`，請將其作爲公司定義資源類型的證券符號。資源類型名稱在叢集中必須是唯一的。

注意 – 依慣例，資源類型名稱 (*Resource_typeVendor_id*) 將作爲套件名稱。由於套件名稱被限制爲最多使用九個字元，因此儘管 RGM 不執行此限制，將這兩個特性的字元總數限制爲九個或更少也是個好主意。另一方面，Agent Builder 明確地從資源類型名稱產生套件名稱，因此它確實執行九個字元的限制。

`RT_version`

識別資料服務範例的版本。

`API_version`

識別 API 的版本。例如，`API_version = 2` 表示資料服務在 Sun Cluster 版本 3.0 下執行。

`Failover = TRUE`

指示資料服務無法在可以同時於多個節點上連線的資源群組中執行，即指定故障轉移資料服務。請參閱第 29 頁的 「將資料服務傳送至叢集」，以取得詳細資訊。

`Start`、`Stop`、`Validate` 等

提供由 RGM 呼叫的各個回呼方法程式的路徑。這些路徑爲 `RT_basedir` 所指定目錄的相對路徑。

其他資源類型宣告提供配置資訊，如下所示：

`Init_nodes = RG_PRIMARIES`	指定 RGM 僅在可主控資料服務的節點上呼叫 `Init`、`Boot`、`Fini`、`Validate` 方法。`RG_PRIMARIES` 指定的節點是安裝有資料服務的所有節點的子集。將該值設定爲 `RT_INSTALLED_NODES`，以指定 RGM 在安裝資料服務的所有節點呼叫這些方法。
`RT_basedir`	指向 `/opt/SUNWsample/bin` 作爲完整相對路徑 (如回呼方法路徑) 的目錄路徑。
`Start`、`Stop`、`Validate` 等	提供由 RGM 呼叫的各個回呼方法程式的路徑。這些路徑爲 `RT_basedir` 所指定目錄的相對路徑。

宣告資源特性

就像宣告資源類型特性一樣，您在 RTR 檔案中宣告資源特性。依慣例，在 RTR 檔案中資源特性宣告緊跟在資源類型宣告之後。資源宣告的語法爲一組由大括弧包含的性質值對：

```
{
    Attribute = Value;
    Attribute = Value;
                .
                .
                .
    Attribute = Value;
}
```

對於 Sun Cluster 提供的資源特性 (所謂的**系統定義的**特性)，您可在 RTR 檔案中變更特定性質。例如，Sun Cluster 爲每個回呼方法提供方法逾時特性，並指定預設值。在 RTR 檔案中，您可以指定不同預設值。

您還可以使用 Sun Cluster 提供的特性性質集，在 RTR 檔案中定義新的資源特性，即所謂的**延伸**特性。第 228 頁的 「資源特性性質」列出了用於變更和定義資源特性的特性。在 RTR 檔案中，延伸特性宣告緊跟在系統定義特性宣告之後。

第一組系統定義的資源特性指定回呼方法的逾時值：

```
...

# 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;
        }
```

特性 (PROPERTY = *value*) 的名稱必須是每個資源特性宣告的第一個性質。在 RTR 檔案中，您可以在特性性質定義的限制內配置資源特性。例如，範例中每個方法逾時的預設值爲 300 秒。管理員可以變更此值；但允許的最小值 (由 MIN 性質指定) 爲 60 秒。第 228 頁的 「資源特性性質」包含資源特性特性的清單。

下一組資源特性定義在資料服務中具有特定用途的特性。

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

# The number of retries to be done within a certain period before concluding
```

```
# that the application cannot be successfully started on this node.
{
        PROPERTY = Retry_Count;
        MAX=10;
        DEFAULT=2;
        TUNABLE = ANYTIME;
}

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

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

這些資源特性宣告新增了 TUNABLE 性質，該性質會限制系統管理員可以變更其值的機會。AT_CREATION 表示管理員僅能在建立資源時指定值，而以後則無法變更該值。

對於這些特性中的大多數，您可以接受 Agent Builder 產生的預設值，除非需要對其進行變更。以下是有關這些特性的資訊 (如需其他資訊，請參閱第 213 頁的 「資源特性」或 r_properties(5) 線上說明手冊)：

`Failover_mode`
指示 RGM 在 `Start` 或 `Stop` 方法失敗的情況下，是應重新配置資源群組還是中斷該節點。

`Thorough_probe_interval`、`Retry_count`、`Retry_interval`
在故障監視器中使用。`Tunable` 等同於 `ANYTIME`，因此，如果故障監視器沒有以最佳方式運作，系統管理員可對其進行調整。

`Network_resources_used`
資料服務使用的邏輯主機名稱或共用位址資源清單。Agent Builder 宣告此特性，以便系統管理員可在配置資料服務時指定一系列的資源 (如果有的話)。

`Scalable`
設定爲 `FALSE`，以指示此資源不使用叢集網路 (共用位址) 工具。此設定與設定爲 `TRUE` (表示故障轉移服務) 的資源類型 `Failover` 特性一致。請參閱第 29 頁的 「將資料服務傳送至叢集」和第 37 頁的 「實施回呼方法」，以取得有關如何使用此特性的其他資訊。

`Load_balancing_policy`、`Load_balancing_weights`
雖然會自動宣告這些特性，但在故障轉移資源類型中沒有用。

`Port_list`
識別伺服器偵聽通訊埠的清單。Agent Builder 宣告此特性，以便系統管理員可在配置資料服務時指定一系列的通訊埠。

宣告延伸特性

範例 RTR 檔案的結尾部分爲延伸特性，如以下清單所示

```
# 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 Builder 會建立一些對大多數資料服務都有用的延伸特性，如下所示。

`Confdir_list`
指定應用程式配置目錄的路徑，此路徑資訊對許多應用程式均有用。系統管理員可在配置資料服務時提供此目錄的位置。

`Monitor_retry_count`、`Monitor_retry_interval` 、`Probe_timeout`
控制故障監視器自身而非伺服器常駐程式的重新啓動。

`Child_mon_level`
設定 PMF 執行的監視層次。請參閱 `pmfadm`(1M)，以取得詳細資訊。

您可以在使用者加入的程式碼註釋所分割的區域內建立附加延伸特性。

實施回呼方法

本節提供一些有關實施回呼方法的一般資訊。

存取資源和資源群組特性資訊

通常，回呼方法需要存取資源特性。RMAPI 提供 shell 指令和 C 函式，可在回呼方法中使用以存取系統定義的資源特性和延伸資源特性。請參閱 `scha_resource_get`(1HA) 和 `scha_resource_get`(3HA) 線上說明手冊。

DSDL 提供一組存取系統定義特性的 C 函式 (每個特性一個)，以及一個存取延伸特性的函式。請參閱 `scds_property_functions`(3HA) 和 `scds_get_ext_property`(3HA) 線上說明手冊。

由於沒有可用於設定資源特性的 API 函式 (除用於設定 `Status` 和 `Status_msg` 的函式外)，因此您無法使用特性機制來儲存資料服務的動態狀態資訊。相反，您應該在全域檔案中儲存動態的狀態資訊。

注意 – 叢集管理員可以使用 `scrgadm` 指令、透過可用圖形管理指令或透過可用圖形管理介面設定特定的資源特性。但是，請勿自任何回呼方法呼叫 `scrgadm`，這是因爲在叢集重新配置期間，即 RGM 呼叫該方法時，`scrgadm` 會失敗。

方法的等冪性

RGM 一般不會在同一資源上以相同的引數連續多次呼叫一個方法。但是，如果 `Start` 方法失敗，即使資源從未啓動，RGM 也會在該資源上呼叫 `Stop` 方法。同樣，資源常駐程式可能會自動終止，而 RGM 仍然會在其上呼叫其 `Stop` 方法。這些情形也同樣適用於 `Monitor_start` 和 `Monitor_stop` 方法。

鑒於這些原因，您必須在 `Stop` 和 `Monitor_stop` 方法中建置等冪性。在同一資源上用相同參數重複呼叫 `Stop` 或 `Monitor_stop` 可獲得與單一呼叫相同的結果。

等冪性的一種含意爲，即使資源或監視器已停止，並且不執行任何工作，`Stop` 和 `Monitor_stop` 也必須傳回 0 (成功)。

注意 – `Init`、`Fini`、`Boot` 以及 `Update` 方法也必須具有等冪性。`Start` 方法不需要具有等冪性。

一般資料服務

一般資料服務 (GDS) 是一種透過將簡單應用程式插入 Sun Cluster 的資源群組管理員架構，使這些應用程式高度可用或具可延伸性的機制。此機制不需要代理程式的編碼，該編碼爲使應用程式高度可用或可延伸的典型方法。

GDS 模型依賴預先編譯的資源類型 SUNW.gds 來與 RGM 框架相互作用。

請參閱第 10 章，以取得其他資訊。

控制應用程式

每當節點處於加入或離開叢集的過程中，回呼方法即會啓用 RGM 來控制基礎資源 (應用程式)。

啓動與停止資源

資源類型實現至少需要 `Start` 方法和 `Stop` 方法。RGM 在適當節點、適當時機呼叫資源類型的方法程式，以使資源群組離線與上線運作。例如，叢集節點當機之後，RGM 會將該節點主控的所有資源群組移至新節點。您必須實施 `Start` 方法，爲 RGM 提供重新啓動存活主機節點上每個資源的方法。

`Start` 方法必須在資源已啓動且在本機節點上可用之後才傳回。請確保爲初始化時間長的資源類型在其 `Start` 方法上設定足夠長的逾時 (在資源類型註冊檔案中設定 `Start_timeout` 特性的預設值和最小值)。

您必須爲 RGM 使資源群組離線的情形實施 `Stop` 方法。例如，假定某資源群組在 Node1 上離線又在 Node2 上恢復線上狀態。在使該資源群組離線時，RGM 在該群組的資源上呼叫 `Stop` 方法來停止 Node1 上的所有作業。在 Node1 上對所有資源執行完 `Stop` 方法後，RGM 將使該資源群組在 Node2 上恢復線上狀態。

`Stop` 方法必須在資源已完全停止其在本機節點上的所有活動且完全關閉後才返回。`Stop` 方法最安全的實作會終止本機節點上與資源相關的所有程序。應該爲需要長時間才能關閉的資源類型在其 `Stop` 方法上設定足夠長的逾時。在資源類型註冊檔案中設定 `Stop_timeout` 特性。

`Stop` 方法的失敗或逾時會導致資源群組進入一個需要操作者介入的錯誤狀態。爲了避免此種狀態，`Stop` 和 `Monitor_stop` 方法實作應該嘗試從所有可能的錯誤情況回復。理想情況是，這些方法以 0 (成功) 錯誤狀態退出，即已成功停止本機節點上的資源及其監視器的所有活動。

決定要使用的 Start 和 Stop 方法

本節提供有關何時使用 Start 和 Stop 方法與何時使用 Prenet_start 和 Postnet_stop 方法的一些提示。您必須全面瞭解用戶端和資料服務的主從式網路協定，才能決定要使用的正確方法。

使用網路位址資源的服務可能要求以特定順序 (相對於邏輯主機名稱位址配置) 完成啓動或停止步驟。可選用的回呼方法 Prenet_start 和 Postnet_stop 允許資源類型實現在將同一資源群組中的網路位址配置或取消配置前後，執行特殊的啓動和關閉動作。

在呼叫資料服務的 Prenet_start 方法之前，RGM 將呼叫探索網路位址 (但不配置網路位址) 的方法。在呼叫資料服務的 Postnet_stop 方法之後，RGM 將呼叫不探索網路位址的方法。當 RGM 使資源群組上線運作時，其序列如下所示。

1. 探索網路位址。
2. 呼叫資料服務的 Prenet_start 方法 (如果有的話)。
3. 配置網路位址。
4. 呼叫資料服務的 Start 方法 (如果有的話)。

當 RGM 使資源群組離線時會發生反向：

1. 呼叫資料服務的 Stop 方法 (如果有的話)。
2. 取消配置網路位址。
3. 呼叫資料服務的 Postnet_stop 方法 (如果有的話)。
4. 不探索網路位址。

當決定是否使用 Start、Stop、Prenet_start 或 Postnet_stop 方法時，請先考量伺服器端。當使包含資料服務應用程式資源和網路位址資源的資源群組上線運作時，RGM 會在呼叫資料服務資源 Start 方法前，呼叫這些方法來配置網路位址。因此，如果資料服務需要在啓動時配置網路位址，請使用 Start 方法啓動資料服務。

同樣，當使包含資料服務應用程式資源和網路位址資源的資源群組離線時，RGM 會在呼叫資料服務資源 Stop 方法後，呼叫這些方法來取消配置網路位址。因此，如果資料服務需要在停止時配置網路位址，請使用 Stop 方法停止資料服務。

例如，爲了啓動或停止資料服務，您可能必須呼叫資料服務的管理公用程式或程式庫。有時，資料服務具有使用主從式網路介面執行管理的管理公用程式或程式庫。即管理公用程式呼叫伺服器常駐程式，從而網路位址可能需要出現，以便使用管理公用程式或程式庫。在此情形下，請使用 Start 方法與 Stop 方法。

如果資料服務需要在其啓動與停止時取消配置網路位址，請使用 Prenet_start 方法或 Postnet_stop 方法啓動與停止該資料服務。考量用戶端軟體是否將依據網路位址或資料服務在叢集重新配置 (透過具有 SCHA_GIVEOVER 引數的 scha_control() 或透過 scswitch 的進行切換保護移轉) 之後是否首先上線而做出不同回應。例如，用戶端實作可能進行最低限度的重試，並在確定資料服務通訊埠不可用之後不久放棄。

如果資料服務在其啓動時不需要配置網路位址，請在配置網路介面之前啓動該資料服務。這樣可確保資料服務能夠在完成網路位址配置之後立即回應用戶端的要求，而且這些用戶端不太可能停止重試。在此情形下，請使用 Prenet_start 方法而不是 Start 方法來啓動資料服務。

如果您使用 Postnet_stop 方法，則資料服務資源在取消配置網路位址時仍會出現。僅在取消配置網路位址後才呼叫 Postnet_stop 方法。結果，只要網路位址做出回應，資料服務的 TCP 或 UDP 服務通訊埠，或者其 RPC 程式編號總是會出現，供網路上的用戶端使用。

注意 – 如果您在叢集上安裝 RPC 服務，則服務不能使用以下程式編號：100141、100142 與 100248。這些編號是分別爲 Sun Cluster 常駐程式 rgmd_receptionist、fed 與 pmfd 而保留的。如果您安裝的 RPC 服務使用這些程式編號的其中一個，則必須將該 RPC 服務變更爲使用其他程式編號。

決定使用 Start 與 Stop 方法還是使用 Prenet_start 與 Postnet_stop 方法，或者兩者都使用，均必須考慮伺服器與用戶端的需求與行爲。

Init、Fini 以及 Boot 方法

三種可選用的方法 (Init、Fini 與 Boot) 均可使 RGM 在資源上執行初始化程式碼與終止程式碼。當資源成爲受管理狀態 (即資源所在的資源群組從未受管理狀態切換爲受管理狀態，或者在已受管理的資源群組中建立資源) 時，RGM 會呼叫 Init 方法以執行資源的一次性初始化。

當資源成爲未受管理狀態 (即資源所在的資源群組切換爲未受管理狀態，或者從受管理的資源群組中刪除資源) 時，RGM 會呼叫 Fini 方法以清除該資源。清除作業必須等冪，即如果清除作業已經完成，則 Fini 會退出 0 (成功)。

RGM 在新近連結叢集 (即，已經啓動或者重新啓動) 的節點上呼叫 Boot 方法。

Boot 方法通常執行與 Init 相同的初始化。此初始化作業必須等冪，即如果已經在本機節點上初始化資源，則 Boot 與 Init 會退出 0 (成功)。

監視資源

通常，您實現監視器以在資源上執行定期的故障探測，以偵測探測的資源是否正常運作。如果故障探測失敗，則監視器會嘗試透過呼叫 scha_control() RMAPI 函式或者 scds_fm_action() DSDL 函式，在本機重新啓動或要求對受影響的資源群組執行故障轉移。

您還可以監視資源的效能，以及微調或報告效能。寫入資源類型特定的故障監視器完全是可選擇的。即使您選擇不寫入此故障監視器，資源類型也會受益於 Sun Cluster 自身對叢集所執行的基本監視。Sun Cluster 偵測主機硬體的故障、主機作業系統的嚴重故障，以及能夠在主機公用網路上通訊的主機之故障。

儘管 RGM 不直接呼叫資源監視器，它卻爲自動啓動資源的監視器做了準備。使資源離線時，RGM 會先呼叫 `Monitor_stop` 方法以停止本機節點上的資源監視器，然後才停止資源自身。在使資源上線時，RGM 會在啓動了資源自身後呼叫 `Monitor_start` 方法。

`scha_control()` RMAPI 函式與 `scds_fm_action()` DSDL 函式 (呼叫 `scha_control()`) 允許資源監視器要求對資源群組執行故障轉移至其他節點。作爲一種完整的檢查，`scha_control()` 會呼叫 `Monitor_check` (如果已定義)，以便判斷要求的節點是否可靠到足以主控包含資源的資源群組。如果 `Monitor_check` 傳回報告，表明節點不可靠或者該方法已逾時，則 RGM 會尋找符合故障轉移要求的其他節點。如果 `Monitor_check` 在所有的節點上均失敗，則會取消故障轉移。

資源監視器可以設定 `Status` 特性與 `Status_msg` 特性，以反映監視器對資源狀態的觀點。請使用 RMAPI `scha_resource_setstatus()` 函式、`scha_resource_setstatus` 指令或者 DSDL `scds_fm_action()` 函式來設定這些特性。

注意 – 儘管 `Status` 與 `Status_msg` 對資源監視器特別有用，但任何程式均可以設定這些特性。

勤參閱第 88 頁的 「定義故障監視器」，以取得透過 RMAPI 所實施的故障監視器之範例。請參閱第 128 頁的 「SUNW.xfnts 故障監視器」，以取得透過 DSDL 所實施的故障監視器之範例。請參閱「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」，以取得有關在 Sun 提供的資料服務中所建立的故障監視器的資訊。

將訊息記錄加入資源

如果您要將狀況訊息記錄在與其他叢集訊息所在的相同日誌檔中，請使用簡易函式 `scha_cluster_getlogfacility()`，以擷取記錄叢集訊息所使用的工具編號。

將此工具編號與一般的 Solaris `syslog()` 函式配合使用，以將訊息寫入叢集日誌。您還可以透過一般的 `scha_cluster_get()` 介面，存取叢集日誌工具資訊。

提供程序管理

RMAPI 與 DSDL 提供程序管理工具，以實施資源監視器與資源控制回呼。RMAPI 定義下列工具 (請參閱線上援助頁，以取得有關其中每個指令與程式的詳細資訊)：

程序監視設備：`pmfadm` 與 `rpc.pmfd`	程序監視設備 (PMF) 提供監視程序及其子代以及當程序失效時重新啓動程序的方法。該工具由用來啓動和控制受監視程序的 `pmfadm` 指令和 `rpc.pmfd` 常駐程式組成。
`halockrun`	保持檔案鎖定時用於執行子程式的程式。此指令便於在 shell 程序檔中使用。
`hatimerun`	在逾時控制下執行子程式的程式。此指令便於在 shell 程序檔中使用。

DSDL 提供 `scds_hatimerun` 函式，以實施 `hatimerun` 功能性。

DSDL 提供實施 PMF 功能性的函式集 (`scds_pmf_*`)。請參閱第 178 頁的 「PMF 函式」，以取得 DSDL PMF 功能性的概觀以及個別函式的清單。

爲資源提供管理支援

資源上的管理動作包括設定與變更資源特性。API 定義 `Validate` 回呼方法與 `Update` 回呼方法，以便可以鉤住這些管理動作。

在建立資源時以及在管理動作更新資源或資源所在群組的特性時，RGM 會呼叫可選用的 `Validate` 方法。RGM 會將資源及其所在資源群組的特性值傳送給 `Validate` 方法。RGM 在資源類型的 `Init_nodes` 特性 (請參閱第 207 頁的 「資源類型特性」或者 `rt_properties`(5) 線上說明手冊，以取得關於 `Init_nodes` 的資訊) 所指示的叢集節點集上呼叫 `Validate`。RGM 在套用建立或更新之前呼叫 `Validate`，則任何節點上的方法故障退出碼都會導致建立或更新失敗。

僅當透過管理動作變更資源或群組特性時，RGM 才會呼叫 `Validate`，而在 RGM 設定特性或監視器設定資源特性 `Status` 與 `Status_msg` 時，RGM 不會呼叫該方法。

RGM 呼叫可選用的 `Update` 方法，以通知執行中的資源其特性已變更。在管理動作成功地設定了資源或其所在群組的特性之後，RGM 將呼叫 `Update`。RGM 在資源上線運作的節點上呼叫該方法。該方法可以使用 API 存取函式，以讀取可能影響使用中資源的特性值，並相應地調整正在執行的資源。

實施故障轉移資源

故障轉移資源群組包含網路位址 (例如，內建的資源群組邏輯主機名稱與共用位址) 與故障轉移資源 (例如，故障轉移資料服務的資料服務應用程式資源)。當資料服務故障轉移或切換保護移轉時，網路位址資源及其相依的資料服務資源在叢集節點之間移動。RGM 提供多種支援故障轉移資源實作的特性。

將布林資源類型特性 `Failover` 設定爲 `TRUE`，以限制在一次可在多個節點上連線的資源群組中對資源進行配置。此特性預設爲 `FALSE`，因此您必須在 RTR 檔案中針對故障轉移資源將此特性宣告爲 `TRUE`。

`Scalable` 資源特性決定資源是否使用叢集共用位址工具。針對故障轉移資源，將 `Scalable` 設定爲 `FALSE`，因爲故障轉移資源不使用共用位址。

`RG_mode` 資源群組特性允許叢集管理員將資源群組識別爲故障轉移資源群組或可延伸資源群組。如果 `RG_mode` 爲 `FAILOVER`，則 RGM 會將群組的 `Maximum_primaries` 特性設定爲 1，並限制資源群組由單一節點主控。RGM 不允許將 `Failover` 特性爲 `TRUE` 的資源建立在 `RG_mode` 爲 `SCALABLE` 的資源群組中。

`Implicit_network_dependencies` 資源群組特性指定 RGM 應該在群組內執行非網路位址資源對所有網路位址資源 (邏輯主機名稱與共用位址) 的隱含牢固相依性。這表明在完成群組中的網路位址配置之前，將不會呼叫群組中非網路位址 (資料服務) 資源上的 `Start` 方法。`Implicit_network_dependencies` 特性預設爲 `TRUE`。

實施可延伸資源

可延伸資源可同時在多個節點上處於線上狀態。可延伸資源包含資料服務，如用於 Sun Java System Web Server 的 Sun Cluster HA (以前爲用於 Sun ONE Web Server 的 Sun Cluster HA) 和用於 Apache 的 Sun Cluster HA。

RGM 提供多種支援可延伸資源實作的特性。

將布林資源類型特性 `Failover` 設定爲 `FALSE`，以允許在一次可在多個節點上連線的資源群組中對資源進行配置。

`Scalable` 資源特性決定資源是否使用叢集共用位址工具。將此特性設定爲 `TRUE`，因爲可延伸服務使用共用位址資源，使得可延伸服務的多個實例作爲用戶端的單一服務出現。

RG_mode 特性可讓叢集管理員識別資源群組爲故障轉移資源群組或可延伸資源群組。如果 RG_mode 爲 SCALABLE，則 RGM 允許 Maximum_primaries 的值大於 1，這表明群組可以同時由多個節點主控。RGM 允許 Failover 特性爲 FALSE 的資源群組創設在 RG_mode 爲 SCALABLE 的資源群組中。

叢集管理員建立一個可延伸資源群組 (用來納入可延伸服務資源) 與一個獨立的故障轉移資源群組 (用來納入可延伸資源所依賴的共用位址資源)。

叢集管理員使用 RG_dependencies 資源群組特性，指定在某節點上使資源群組上線或者離線所依據的順序。此排序對於可延伸服務很重要，這是因爲可延伸資源與其所依賴的共用位址資源位於不同的資源群組中。可延伸資料服務要求其網路位址 (共用位址) 資源在啓動該資料服務之前已配置完成。因此，管理員必須將包含可延伸服務的資源群組之 RG_dependencies 特性設定爲納入包含共用位址資源的資源群組。

當您在 RTR 檔案中針對資源宣告 Scalable 特性時，RGM 會自動爲資源建立下列一組可延伸特性：

Network_resources_used
: 識別由此資源所使用的共用位址資源。此特性預設爲空字串，因此叢集管理員必須在建立資源時提供可延伸服務所使用的共用位址之實際清單。scsetup 指令與 SunPlex Manager 提供自動設定可延伸服務之必備資源與群組的功能。

Load_balancing_policy
: 指定資源的負載平衡策略。您可以在 RTR 檔案中明確設定該策略 (或者允許預設值 LB_WEIGHTED)。無論在哪種情況下，叢集管理員均可以在建立資源時變更該值 (除非您在 RTR 檔案中將 Load_balancing_policy 的 Tunable 設定爲 NONE 或者 FALSE)。合法值包括：

  LB_WEIGHTED
  : 負載是依照 Load_balancing_weights 特性中所設定的權重而在各節點中進行分配的。

  LB_STICKY
  : 總是將可延伸服務的給定用戶端 (由用戶端 IP 位址識別) 發送到叢集的同一個節點。

  LB_STICKY_WILD
  : 總是將連接到萬用字元黏性服務的 IP 位址之給定用戶端 (由用戶端 IP 位址識別) 發送到同一個叢集節點，而不管該用戶端將要到哪個通訊埠編號。

  對於使用 Load_balancing_policy、LB_STICKY 或者 LB_STICKY_WILD 的可延伸服務，當服務處於線上狀態時變更 Load_balancing_weights 可能導致重設現有用戶端的相似性。在這種情況下，其他節點可能會處理後續用戶端的要求，即使該用戶端先前是由叢集中的另一個節點處理的。

	同樣，在叢集上啓動服務的新實例可能重設現有用戶端的相似性。
`Load_balancing_weights`	指定要發送至每個節點的負載。格式爲 *weight@node, weight@node*，其中 *weight* 爲一個整數，反映分配給指定 *node* 負載的相對部分。分配到節點的負載分數值是使用中實例的權重總量除以此節點的權重。例如，`1@1,3@2` 表明節點 1 接收到 1/4 的負載，節點 2 接收到 3/4 的負載。
`Port_list`	識別伺服器要在其上進行偵聽的通訊埠。此特性預設爲空字串。您可以在 RTR 檔案中提供通訊埠清單。否則，叢集管理員在建立資源時必須提供通訊埠的實際清單。

您可以建立可由管理員配置爲可延伸或故障轉移的資料服務。若要如此，請在資料服務的 RTR 檔案中將 `Failover` 資源類型特性與 `Scalable` 資源特性宣告爲 `FALSE`。指定 `Scalable` 特性在建立時可調。

`Failover` 特性值 (`FALSE`) 允許將資源配置到可延伸資源群組。管理員可以透過在建立資源時將 `Scalable` 的值變更爲 `TRUE` 來啓用共用位址，從而建立可延伸服務。

另一方面，即使 `Failover` 設定爲 `FALSE`，管理員也可以將資源配置到故障轉移資源群組，以實施故障轉移服務。管理員不會變更 `Scalable` 的值 `FALSE`。爲了支援此偶然性，您應該使用 `Validate` 方法對 `Scalable` 特性進行檢查。如果 `Scalable` 爲 `FALSE`，請驗證是否已將該資源配置到故障轉移資源群組。

「*Sun Cluster* **概念指南 (適用於** *Solaris* **作業系統)**」包含有關可延伸資源的其他資訊。

可延伸服務的驗證檢查

每當使用設定爲 `TRUE` 的可延伸特性建立或更新資源時，RGM 均會驗證各種資源特性。如果未正確配置特性，則 RGM 會拒絕所嘗試的更新或建立。RGM 執行下列檢查：

- `Network_resources_used` 特性必須爲非空，且包含現有共用位址資源的名稱。包含可延伸資源的資源群組之 `Nodelist` 中的每個節點必須出現在每個具名共用位址資源的 `NetIfList` 特性或 `AuxNodeList` 特性中。
- 包含可延伸資源的資源群組之 RG_dependencies 特性必須包括在可延伸資源的 `Network_resources_used` 特性中所列出的所有共用位址資源之資源群組。
- Port_list 特性必須爲非空，且包含通訊埠協定對 (該協定爲傳輸控制協定或使用者資料封包協定) 的清單。例如，

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

寫入與測試資料服務

本節提供有關寫入與測試資料服務的一些資訊。

使用持續連接

在伺服器端，針對非作用中 (或者網路分割) 的用戶端使用 TCP 持續連接可防止伺服器浪費系統資源。如果尚未清除這些資源 (在工作了足夠長時間的伺服器中)，則最後當用戶端當機與重新啓動時浪費的資源會無限增加。

如果主從式通訊使用 TCP 串流，則用戶端與伺服器都應該啓用 TCP 持續連接機制。甚至在非 HA 的單一伺服器情形下，此規定也適用。

其他導向連接協定也可以具有持續連接機制。

在用戶端，使用 TCP 持續連接可使用戶端在網路位址資源從一個實體主機故障轉移或者切換保護移轉至另一個實體主機時得到通知。網路位址資源的該傳送會中斷 TCP 連接。然而，除非用戶端已啓用持續連接，否則，如果當時連接突然中斷，該用戶端未必會獲悉連接中斷。

例如，假定用戶端等待伺服器對長時間要求的回應，用戶端的要求訊息已經到達伺服器，並且在 TCP 層已對該訊息做出回應。在此情形下，用戶端的 TCP 模組不需要保持重新傳輸要求狀態，用戶端應用程式會被封鎖，並等待對要求的回應。

在可能的情況下，除使用 TCP 保持連接機制之外，用戶端應用程式還必須在其應用程式層級自己執行週期性保持連接，因爲 TCP 保持連接機制並非在所有可能的界限情況下均完美無缺。使用應用程式層級保持連接一般需要主從式協定支援虛擬作業，或者至少需要有效率的唯讀作業 (例如，狀況作業)。

測試 HA 資料服務

本節提供有關如何在 HA 環境中測試資料服務實現的建議。這些測試情況是一些建議，並不十分詳盡。您需要測試平台 Sun Cluster 配置的存取權限，以免測試工作影響生產機器。

測試在實體主機之間移動資源群組的各種情況下，HA 資料服務行爲是否正常。這些情況包括系統當機與使用 `scswitch` 指令。測試用戶端機器在這些事件之後是否繼續獲取服務。

測試方法的等冪性。例如，暫時將每種方法取代爲呼叫原來方法兩次或多次之簡短 shell 程序檔。

協調資源之間的相依性

有時，一種主從式資料服務在執行用戶端的要求時會對另一種主從式資料服務提出要求。通俗地說，如果資料服務 A 依賴資料服務 B，則 A 提供其服務時，B 必須提供其服務。Sun Cluster 透過允許在資源群組內配置資源相依性來提供此要求。此相依性會影響 Sun Cluster 啓動與停止資料服務所依據的順序。請參閱 `scrgadm`(1M) 線上說明手冊，以取得詳細資訊。

如果您資源類型的資源依賴另一種類型的資源，則您需要指導使用者適當地配置資源與資源群組，或者提供正確配置它們的程序檔或工具。如果相依資源必須在與其所相依資源所在的節點上執行，則必須將這兩種資源配置在同一資源群組中。

決定是使用明確的資源相依性，還是忽略它們並對 HA 資料服務特有程式碼中的其他資料服務之可用性進行輪詢。爲了使相依與被相依的兩種資源可以在不同節點上執行，請將它們配置到獨立的資源群組。在這種情況下，就需要進行輪詢，因爲不可能跨群組配置資源相依性。

某些資料服務自身未直接儲存任何資料，但是卻依賴另一個後端資料服務來儲存其所有資料。這樣的資料服務將所有讀取要求與更新要求轉換爲對後端資料服務的呼叫。例如，考慮一種假定的主從式約會日曆服務，該服務將其所有資料均保留在 SQL 資料庫 (例如，Oracle) 中。該約會日曆服務有其自己的主從式網路協定。例如，它可能已使用 RPC 規格語言(例如，ONC RPC) 定義了其協定。

在 Sun Cluster 環境中，您可以使用 HA-ORACLE 使後端 Oracle 資料庫高度可用。然後，您可以寫入用於啓動與停止約會日曆常駐程式的簡單方法。一般使用者會向 Sun Cluster 註冊約會日曆資源類型。

如果約會日曆應用程式必須在 Oracle 資料庫所在的同一節點上執行，則一般使用者會在 HA-ORACLE 資源所在的同一資源群組中配置約會日曆資源，並使約會日曆資源依賴 HA-ORACLE 資源。此相依性是使用 `scrgadm` 中的 `Resource_dependencies` 特性標記指定的。

如果 HA-ORACLE 資源能夠在約會日期資源所在節點之外的其他節點上執行，則一般使用者會將它們配置到兩個獨立的資源群組。一般使用者可以配置日曆資源群組對 Oracle 資源群組的資源群組相依性。然而，僅當在同一節點上同時啓動或者停止兩種資源時，資源群組相依性才會有效。因此，在啓動日曆資料服務常駐程式之後，該常駐程式可以在等待 Oracle 資料庫變得可用時進行輪詢。在此情況下，日曆資源類型的 `Start` 方法通常僅會傳回成功，這是因爲如果 `Start` 方法無限期地封鎖，則它會使資源群組成爲工作中狀態，這樣會防止對群組進行任何進一步的狀態變更 (例如，編輯、故障轉移或者切換保護移轉)。然而，如果日曆資源的 `Start` 方法逾時或者以非零狀態退出，則可能導致資源群組在兩個或多個節點之間交替，而 Oracle 資料庫保持無法使用狀態。

第 3 章

升級資源類型

本章討論您升級資源類型與遷移資源而需要瞭解的問題。

- 第 49 頁的 「簡介」
- 第 50 頁的 「資源類型註冊檔案」
- 第 52 頁的 「資源 Type_version 屬性」
- 第 53 頁的 「將資源遷移至其他版本」
- 第 53 頁的 「升級與降級資源類型」
- 第 55 頁的 「預設屬性值」
- 第 56 頁的 「資源類型開發者說明文件」
- 第 56 頁的 「資源類型名稱與資源類型監視器實作」
- 第 57 頁的 「應用程式升級」
- 第 57 頁的 「資源類型升級範例」
- 第 59 頁的 「資源類型套件的安裝要求」

簡介

系統管理員需要有能力安裝與註冊現有資源類型的新版本，以允許註冊多版本給定資源類型，以及將現有資源類型遷移至新版本的資源類型，而不必刪除或重新建立該資源。資源開發者需要瞭解提供資源類型升級與資源遷移的要求。

考量升級而開發的資源類型稱之爲**支援升級**。

新版本資源類型與舊版資源類型相比有以下幾方面不同：

- 資源類型屬性的性質可變更
- 宣告的資源屬性集 (包含標準屬性與延伸屬性) 可變更
- 資源屬性性質，如 `default`、`min`、`max`、`arraymin`、`arraymax` 或可調性可能會變更
- 宣告的方法集可不同
- 方法或監視器的實作可變更。

資源類型開發者決定何時可以將現有的資源從以下可調性選項中遷移至新版本。選項以受限制性從最少到最多的順序列出：

- 任何時候 (ANYTIME)
- 取消監視資源時 (WHEN_UNMONITORED)
- 資源離線時 (WHEN_OFFLIN)
- 停用資源時 (WHEN_DISABLED)
- 未管理資源群組時 (WHEN_UNMANAGED)
- 建立時 (AT_CREATION)

請參閱第 52 頁的 「資源 Type_version 屬性」，以取得每個選項的說明。

注意 – 在本章中，討論如何進行升級時均會使用 scrgadm 指令。管理員不受限於僅使用 scrgadm 指令，也可以使用 GUI 或 scsetup 指令來進行升級。

資源類型註冊檔案

資源類型名稱

資源類型名稱的三個程式元件爲在 RTR 檔案中指定爲 *Vendor_id*、*Resource_type* 和 *RT_version* 的屬性。scrgadm 指令將插入句點分割元與逗號分割元，以建立資源類型的名稱：

```
vendor_id.resource_type:rt_version
```

Vendor_id 前綴用來區別兩個由不同供應商提供的相同名稱的註冊檔案。*RT_version* 區別具有相同基本資源類型 的多個註冊版本 (升級)。若要確保 *Vendor_id* 是唯一的，建議您在建立資源類型時使用公司的證券代號。

如果 *RT_version* 字串包含空白、定位字元、斜線 (/)、反斜線 (\)、星號 (*)、問號 (?)、逗號 (,)、分號 (;)、左方括號 ([) 或右方括號 (]) 字元，則註冊資源類型將失敗。

RT_Version 屬性 (在 Sun Cluster 3.0 中可選用) 在 Sun Cluster 3.1 中是一個強制字串。

完全合格的名稱指以下指令傳回的名稱：

```
scha_resource_get -O Type -R resource_name -G resource_group_name
```

在 Sun Cluster 3.1 之前註冊的資源類型名稱將繼續使用以下語法：

```
vendor_id.resource_type
```

指令

支援升級之資源類型的 RTR 檔案必須包含 `#$upgrade` 指令，其後不跟隨以下格式的指令，或跟隨多個：

```
#$upgrade_from version tunability
```

`upgrade_from` 指令包含字串 `#$upgrade_from`，該字串後跟隨 `RT_Version`，然後是資源的可調性限制。如果對其正在執行升級的資源類型不屬於某個版本，則會將 `RT_Version` 指定爲一個空字串，如以下最後一個範例所示：

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

當系統管理員嘗試變更資源 `Type_version` 時，RGM 將實施這些針對資源的限制。如果目前版本的資源類型未顯示在清單中，則 RGM 將強加 `WHEN_UNMANAGED` 的可調性。

這些指令必須顯示在 RTR 檔案的資源類型屬性宣告區段與 RTR 檔案的資源宣告區段之間。請參閱 `rt_reg`(4)。

變更 RTR 檔案中的 `RT_Version`

每當變更 RTR 檔案中的內容時，請變更 RTR 檔案中的 `RT_Version` 字串。此屬性的值必須明確指出哪一個是新版本的資源類型以及哪一個是舊版本的資源類型。如果未對 RTR 檔案進行任何變更，則無需變更 `RT_Version` 字串。

舊版 Sun Cluster 中的資源類型名稱

Sun Cluster 3.0 中的資源類型名稱不包含版本後綴：

```
vendor_id.resource_name
```

即使在您將叢集軟體升級爲 Sun Cluster 3.1 或更高版本之後，原來在 Sun Cluster 3.0 下註冊的資源類型仍將繼續使用此格式的名稱。同樣，如果 RTR 檔案在執行於 Sun Cluster 3.1 或更高版本軟體的叢集上進行了註冊，則其 RTR 檔案遺失了 `#$upgrade` 指令的資源類型將被提供一個 Sun Cluster 3.0 格式的名稱，且沒有版本後綴。

您可以使用 Sun Cluster 3.0 中的 `#$upgrade` 或 `#$upgrade_from` 指令來註冊 RTR 檔案，但不支援將現有的資源遷移至 Sun Cluster 3.0 中的新資源類型。

資源 Type_version 屬性

標準資源 屬性 Type_version 將儲存資源類型的 RT_Version 屬性。該屬性不會顯示在 RTR 檔案中。系統管理員將透過使用以下指令來編輯此屬性值：

```
scrgadm -c -j resource -y Type_version=new_version
```

該屬性的可調性源自：

- 目前版本的資源類型
- RTR 檔案中的 #$upgrade_from 指令

使用 #$upgrade_from 指令中的以下可調性值：

ANYTIME
: 如果對何時可以升級資源沒有限制。資源可以完全處於線上。

WHEN_UNMONITORED
: 如果新資源類型版本的 Update 方法、Stop 方法、Monitor_check 方法以及 Postnet_stop 方法可以與舊資源類型版本的啓動方法 (Prenet_stop 與 Start) 相容，並且新資源類型版本的 Fini 方法可以與舊版本的 Init 方法相容。該方案僅需在升級之前停止該資源監視器程式

WHEN_OFFLINE
: 如果新資源類型版本的 Update 方法、Stop 方法、Monitor_check 方法或 Postnet_stop 方法與舊資源類型版本的啓動方法 (Prenet_stop 與 Start) 不相容，但是與舊版本的 Init 方法相容，則在將類型升級套用至資源時，資源必須離線。

WHEN_DISABLED
: 類似於 WHEN_OFFLINE。但是，此可調性值將強加停用資源的更強條件。

WHEN_UNMANAGED
: 如果新資源類型版本的 Fini 方法與舊版本的 Init 方法不相容。該可調性值需要現有資源群組切換到不受管理狀態，然後您才可以升級資源。

AT_CREATION
: 如果無法將資源升級至新的資源類型版本。僅可以建立新版本的新資源。

  AT_CREATION 的可調性意味著資源類型開發者可以禁止將現有資源遷移至新類型。在這種情況下，系統管理員必須先刪除資源然後重新建立資源。這樣相當於宣告僅可以在建立時設定資源的版本。

將資源遷移至其他版本

系統管理員編輯資源的 Type_version 屬性時，現有的資源將具有新資源類型版本。這將遵循用來編輯其他資源屬性的相同慣例執行，除了某些資訊將源自或取自新資源類型版本而非目前版本之外：

- 所有屬性的資源屬性性質 (如 min、max、arraymin、arraymax、預設以及可調性) 都取自新資源類型版本
- 可套用至 Type_version 屬性的可調性取自 RTR 檔案中的 #$upgrade_from 指令以及現有資源的資源類型之 RT_version 屬性。此可調性與 property_attributes(5) 中描述的可調性不同。
- 將會套用新資源類型版本的 Validate 方法。這將確保屬性性質對於新資源類型有效。如果現有的資源屬性性質不符合新資源類型版本的驗證條件，系統管理員必須透過 scrgadm 指令行提供此類屬性的有效值。如果新資源類型版本開始使用未在舊版本中宣告且不具有預設值的屬性，可能會出現這種情況。如果現有資源的某個屬性被指定了對於新資源類型版本無效的值，也可能出現這種情況。
- 在舊版本資源類型中宣告的資源屬性可以在新版本資源類型中不宣告。當將資源遷移至新版本時，屬性將從資源中刪除。

注意 – Validate 方法可以查詢資源的目前 Type_version (使用 scha_resource_get) 以及新 Type_version (其透過 Validate 指令行傳送)。因此，Validate 可以排除不受支援版本的升級。

升級與降級資源類型

「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」中的「Upgrading a Resource Type」一節包含有關升級或遷移資源類型的其他資訊。

▼ 如何升級資源類型

1. **讀取新資源類型的升級說明文件，以找出資源類型變更與資源可調性限制。**
2. **在所有叢集節點上安裝資源類型升級套件。**

   安裝新資源類型套件所建議的練習處於滾動升級模式：當在非叢集模式中啓動節點時，pkgadd 將會出現。

   以下是在叢集模式中的節點上可安裝新資源類型套件的情形：

- 如果資源類型套件安裝保持方法程式碼不變更，而僅更新監視器，則有必要在安裝期間停止對所有該類型資源的監視。
- 如果資源類型套件安裝保持方法程式碼與監視器程式碼都不變更，則沒有必要在安裝期間停止對資源的監視，因爲安裝僅將新 RTR 檔案置於磁碟上。

3. **使用 `scrgadm` (或對等) 指令註冊新資源類型版本，參考 RTR 檔案的升級。**

   RGM 將建立一個新資源類型，其名稱的格式爲

   ```
   vendor_id.resource_type:version
   ```

4. **如果資源類型升級僅安裝在節點的子集上，則您必須將新資源類型的 `Installed_nodes` 屬性設定為在其上實際安裝新資源類型的節點。**

   當資源具有新類型 (透過新建或更新) 時，RGM 將需要資源群組 `nodelist` 爲資源類型的 `Installed_nodes` 清單之子集。

   ```
   scrgadm -c -t resource_type -h installed_node_list
   ```

5. **對於預先升級類型 (將要遷移至升級的類型) 的每一個資源，請呼叫 `scswitch` 以將資源或其資源群組的狀態變更為由升級說明文件指示的適當狀態。**

6. **對於預先升級類型 (將要遷移至升級的類型) 的每一個資源，請編輯資源並將其 `Type_version` 屬性變更為新版本。**

   ```
   scrgadm -c -j resource -y Type_version=new_version
   ```

   如有必要，您可以透過相同指令將相同資源的其他屬性編輯爲適當的值。

7. **透過反轉在步驟 5中呼叫的指令，回復到資源或資源群組的上一個狀態。**

▼ 如何將資源的資源類型降級爲舊版本

您可以將資源的資源類型降級爲舊版本。您可以將資源降級爲舊版本資源類型時的條件比您將資源升級爲新版本資源類型時的條件更嚴格。您必須首先取消對資源群組的管理。此外，您僅可以將資源降級爲資源類型的允許升級版本。您可以透過使用 `scrgadm -p` 指令來識別允許升級的版本。在輸出中，允許升級的版本包含後綴：*version*。

1. **將包含要降級的資源之資源群組切換為離線狀態。**

   ```
   scswitch -F -g resource_group
   ```

2. **停用要降級的資源與資源群組中的所有資源。**

   ```
   scswitch -n -j resource_to_downgrade
   scswitch -n -j resource1
   scswitch -n -j resource2
   scswitch -n -j resource3
   ...
   ```

注意 – 依相依順序停用檔案，以最大相依 (應用程式資源) 開始並以最小相依 (網路位址資源) 結束。

3. **取消管理資源群組。**

   scswitch -u -g *resource_group*

4. **是否仍要在叢集中註冊要降級至的資源類型之舊版本？**

   - 若要，請跳至下一個步驟。
   - 若否，請重新註冊需要的舊版本。

     scrgadm -a -t *resource_type_name*

5. **透過指定您需要的舊版本 `Type_version` 來降級資源。**

   scrgadm -c -j *resource_to_downgrade* **-y Type_version=***old_version*

   如有必要，您可以透過相同指令將相同資源的其他屬性編輯爲適當的值。

6. **使包含降級資源的資源群組處於受管理狀態，啓用所有資源並切換群組上線運作。**

   scswitch -Z -g *resource_group*

預設屬性值

RGM 儲存所有資源，從而任何系統管理員未明確設定的預設屬性均不會儲存於 CCR (叢集配置儲存庫) 的資源項目中。當從 CCR 讀取資源時，RGM 將從資源類型獲取遺失資源屬性的預設值 (或如果資源類型中未定義，請使用系統定義的預設值)。正是這個儲存屬性的方法可允許升級的資源類型定義新屬性或現有屬性的新預設值。

當編輯資源屬性時，RGM 會將透過編輯指令指定的屬性儲存於 CCR 中。

如果升級版本的資源類型宣告預設特性的新預設值，則新預設值將由現有資源繼承，即使宣告了該特性僅在 `AT_CREATION` 或 `WHEN_DISABLED` 條件下可調亦如此。如果新預設的應用程式將引起 `Stop`、`Postnet_stop` 或 `Fini` 方法失敗，則資源類型實作者必須在升級資源時相應地限制資源的狀態。透過限制 `Type_version` 屬性的可調性來完成此作業。

新資源類型版本 `Validate` 方法可以檢查以確定現有的屬性性質是適當的。如果不適當，系統管理員可以透過編輯 `Type_version` 屬性以將資源升級至新資源類型版本的相同指令，來將現有資源的屬性編輯爲適當的值。

注意 – 當建立於 Sun Cluster 3.0 中的資源遷移至以後版本時，它們不會繼承資源類型的新預設屬性性質，因爲其預設屬性儲存於 CCR 中。

資源類型開發者說明文件

資源類型開發者必須提供具有新資源的說明文件，該新資源提供以下資訊：

- 說明所有屬性的加入、變更或刪除
- 說明如何使屬性與新需求一致
- 敘述對資源的可調性限制
- 呼出所有新預設屬性性質
- 通知系統管理員可以透過用來編輯 `Type_version` 屬性以將資源升級至新資源類型版本的相同指令，來將現有資源的屬性編輯爲適當的值。

資源類型名稱與資源類型監視器實作

您可以在 Sun Cluster 3.0 中註冊支援升級的資源類型，但其名稱記錄在 CCR 中，而不包含版本後綴。若要在 Sun Cluster 3.0 和 Sun Cluster 3.1 (以及更高版本) 中均可正常執行，該資源類型的監視器必須能夠處理二者的命名慣例：

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

監視器程式碼可決定要使用的正確名稱，方法爲透過執行對等的：

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

然後將輸出值與 `vers` 進行比較。對於 `vers` 的特定值，這些指令中僅一個會成功，因爲在兩個不同的名稱下不可能註冊兩次相同版本的資源類型。

應用程式升級

儘管某些問題相似，但是升級應用程式代碼與升級代理程式代碼仍有區別。應用程式升級可能會伴隨資源類型升級也可能不會。

資源類型升級範例

這些範例說明多種不同資源類型安裝方案與升級方案。已經基於對資源類型實作進行的變更之類型選擇了可調性資訊與封裝資訊。可調性將資源的遷移套用至新資源類型。

所有範例都假定：

- 資源類型將在 Solaris 套裝軟體中提供。請參閱 `pkgadd`(1M) 和 `pkgrm`(1M)。
- 由於僅有一個舊版本資源類型，因此在新 RTR 檔案中僅有一個 `#$upgrade_from` 指令
- 如果 RGM 有可能在從磁碟移除方法時呼叫它們，則安裝程序將不會移除或覆寫這些方法
- 除非另有註明，否則新方法與舊方法相容
- 在進行安裝或遷移之前，使用正確的 `scswitch`(1M) 指令或對等指令將資源與資源群組移至需要的狀態。以下範例展示如何將資源群組移至不受管理狀態：

  ```
  scswitch -M -n -j resource
  scswitch -n -j resource
  scswitch -F -g resource_group
  scswitch -u -g resource_group
  ```

- 您可以透過使用以下指令來註冊資源類型：

  ```
  scrgadm -a -t resource_type -f path_to_RTR_file
  ```

- 您可以透過使用以下指令來遷移資源：

  ```
  scrgadm -c -j resource -y Type_version=version \
             -y property=value \
             -x property=value ...
  ```

- 遷移之後，使用適當的 `scswitch`(1M) 指令或對等指令將資源與資源群組回復到其上一個狀態：

  ```
  scswitch -M -e -j resource
  scswitch -e -j resource
  scswitch -o -g resource_group
  scswitch -Z -g resource_group
  ```

資源類型開發者可能需要指定比這些範例中所使用可調性值具有更多限制性的可調性值。可調性值取決於對資源類型實作所做的確切變更。此外，資源類型開發者可能會選擇使用不同的封裝方案來取代這些範例中使用的 Solaris 封裝。

表 3–1 升級資源類型的範例

變更類型	可調性	封裝	程序
僅在 RTR 檔案中進行屬性變更。	ANYTIME	僅提供新 RTR 檔案。	在所有節點上執行新 RTR 檔案的 pkgadd。 註冊新資源類型。 遷移資源。
方法被更新。	ANYTIME	將更新的方法置於一個與舊方法不同的路徑中。	在所有節點上執行已更新方法的 pkgadd。 註冊新資源類型。 遷移資源。
新的監視器程式。	WHEN_UNMONITORED	僅覆寫舊版監視器。	停用監視。 在所有的節點上執行新監視器程式的 pkgadd。 註冊新資源類型。 遷移資源。 啓用監視。
方法被更新。新的 Update/Stop 方法與舊的 Start 方法不相容。	WHEN_OFFLINE	將更新的方法置於一個與舊方法不同的路徑中。	在所有節點上執行已更新方法的 pkgadd。 註冊新資源類型。 使資源離線。 遷移資源。 使資源上線。
方法將更新，而新屬性將新增至 RTR 檔案。新方法需要新屬性。(目的在於萬一資源所在的資源群組在某個節點上從離線狀態移至線上狀態，可允許資源群組保持線上狀態，但阻止資源上線運作。)	WHEN_DISABLED	覆寫舊版方法。	停用資源。 對於每一個節點： ■ 將節點取出叢集 ■ 執行 正在更新的方法之 pkgrm/pkgadd ■ 將節點回復到叢集 註冊新資源類型。 遷移資源。 啓用資源。

表 3–1 升級資源類型的範例　*(續)*

變更類型	可調性	封裝	程序
方法將更新，而新屬性將新增至 RTR 檔案。新方法不需要新屬性。	ANYTIME	覆寫舊版方法。	對於每一個節點： ■ 將節點取出叢集 ■ 執行正在更新的方法之 pkgrm/pkgadd ■ 將節點回復到叢集 在該程序期間，RGM 將呼叫新方法，即使尚未執行遷移 (將會配置新屬性)。新方法在不具有新屬性的情況下能夠工作很重要。 註冊新資源類型。 遷移資源。
方法被更新。新的 Fini 方法與舊的 Init 方法不相容。	WHEN_UNMANAGED	將更新的方法置於一個與舊方法不同的路徑中。	使資源所在的資源群組不受管理。 在所有節點上執行已更新方法的 pkgadd。 註冊資源類型。 遷移資源。 使資源所在的資源群組受管理。
方法被更新。未對 RTR 檔案進行任何變更。	不適用。未對 RTR 檔案進行任何變更	覆寫舊版方法。	對於每一個節點： ■ 將節點取出叢集 ■ 執行已更新的方法之 pkgadd ■ 將節點回復到叢集。 由於未對 RTR 檔案進行任何變更，因此不需要註冊或遷移資源。

資源類型套件的安裝要求

有兩個與新資源類型套件有關的要求：

- 註冊新的資源類型時，其 RTR 檔案在磁碟上必須是可存取的
- 當建立新類型的資源時，所有宣告的方法路徑名稱與新類型的監視器程式均必須位於磁碟上並且是可執行的。只要資源處於使用中狀態，則舊方法與監視器程式都必須保留在適當位置。

若要決定最適當的封裝，資源類型實作者必須考量以下問題：

- RTR 檔案是否變更？
- 屬性的預設值或可調性是否變更？
- 屬性的 `min` 或 `max` 值是否變更？
- 升級是否會新增或刪除屬性？
- 方法程式碼是否變更？
- 監視器程式碼是否變更？
- 新方法或監視器程式碼與舊版本是否相容？

變更 RTR 檔案前需要瞭解的資訊

一些資源類型升級不會涉及新方法或監視器程式碼。例如，資源類型升級可能僅變更資源屬性的預設值或可調性。由於方法程式未變更，因此安裝升級的唯一要求是具有可讀 RTR 檔案的有效路徑名稱。

如果不需要重新註冊舊資源類型，則新 RTR 檔案可以覆寫舊版本。否則，新的 RTR 檔案可置於新的路徑名稱中。

如果升級變更屬性的預設值或可調性，新版本的 `Validate` 方法可以在遷移時確認現有的屬性性質對於新資源類型有效。如果升級變更屬性的 `min`、`max` 或 `type` 性質，則 `scrgadm` 指令在遷移時自動驗證這些限制。

升級說明文件必須呼出所有的新預設屬性性質。說明文件必須通知系統管理員可以透過用來編輯 `Type_version` 屬性以將資源升級至新資源類型版本的相同指令，來將現有資源的屬性編輯爲適當的值。

如果升級新增或刪除屬性，則很可能也必須變更某些回呼方法或監視器程式碼。

變更監視器程式碼

如果在更新的資源類型中僅變更監視器程式碼，則套件安裝可以覆寫監視器二進位檔。在安裝新的套件前，說明文件必須通知系統管理員暫停監視。

變更方法程式碼

如果在更新的資源類型中僅變更方法程式碼，請務必確定新方法程式碼是否與舊版本相容。這樣可決定新方法程式碼是否必須儲存於新的路徑名稱中，或是否可以覆寫舊方法。

如果新的 `Stop`、`Postnet_stop` 以及 `Fini` 方法 (如果已宣告) 可以套用至由舊版 `Start`、`Prenet_stop` 或 `Init` 方法初始化或啓動的資源，則可能會以新方法覆寫舊方法。

如果新方法程式碼與舊版本不相容，則必須停止或取消配置使用舊版本方法的資源，然後才能將資源遷移至升級的資源類型。如果新方法覆寫舊方法，則在進行資源類型升級之前，新方法將要求關閉 (或可能不管理) 所有此類型的資源。如果新方法與舊方法個別儲存 (且可同時存取新方法與舊方法)，則甚至在沒有向下相容性的情況下，都可能安裝新資源類型並逐個升級資源。

即使新方法是向下相容的，也可能要求每次升級一個資源以使用新方法，而其他資源繼續使用舊方法。仍有必要將新方法儲存於個別目錄中，而非覆寫舊方法。

將每個資源類型版本的方法儲存於個別目錄中的優勢是，如果新的版本出現問題，可以很容易將資源切換回舊的資源類型版本。

一個封裝方法是包含套件中仍支援的所有舊版本。這樣可允許新的套件版本取代舊的版本，而不覆寫或刪除舊方法路徑。決定支援多少個舊版本取決於資源類型開發者。

注意 – 建議不要覆寫方法或對目前位於叢集中節點上的方法執行 `pkgrm/pkgadd`。如果 RGM 要在某個方法於磁碟上不可存取時呼叫該方法，則可能出現非預期的結果。移除或取代正在執行方法的二進位碼也可能引起非預期的結果。

第 4 章

資源管理 API 參考

本章提供有關組成資源管理 API (RMAPI) 的存取函式與回呼方法之參考，還列出並簡要說明了每個函式與方法。然而，這些函式與方法的決定性參考是 RMAPI 線上援助頁。

本章包含以下資訊：

- 第 64 頁的 「RMAPI 存取方法」 – 以 shell 程序檔指令和 C 函式的格式
  - `scha_resource_get`(1HA)、`scha_resource_close`(3HA)、`scha_resource_get`(3HA)、`scha_resource_open`(3HA)
  - `scha_resource_setstatus`(1HA)、`scha_resource_setstatus`(3HA)
  - `scha_resourcetype_get`(1HA)、`scha_resourcetype_close`(3HA)、`scha_resourcetype_get`(3HA)、`scha_resourcetype_open`(3HA)
  - `scha_resourcegroup_get`(1HA)、`scha_resourcegroup_get`(3HA)、`scha_resourcegroup_close`(3HA)、`scha_resourcegroup_open`(3HA)
  - `scha_control`(1HA)、`scha_control`(3HA)
  - `scha_cluster_get`(1HA)、`scha_cluster_close`(3HA)、`scha_cluster_get`(3HA)、`scha_cluster_open`(3HA)
  - `scha_cluster_getlogfacility`(3HA)
  - `scha_cluster_getnodename`(3HA)
  - `scha_strerror`(3HA)
- 第 68 頁的 「RMAPI 回呼方法」 – 如 `rt_callbacks`(1HA) 線上說明手冊中所述。
  - `Start`
  - `Stop`
  - `Init`
  - `Fini`
  - `Boot`
  - `Prenet_start`
  - `Postnet_stop`
  - `Monitor_start`
  - `Monitor_stop`

- `Monitor_check`
- `Update`
- `ValidateS`

RMAPI 存取方法

API 提供存取資源、資源類型、資源群組特性以及其他叢集資訊的函式。這些函式以 shell 指令形式與 C 函式形式提供，可讓資源類型提供者將控制程式作爲 shell 程序檔或 C 程式實施。

RMAPI Shell 指令

Shell 指令將被用於資源類型 (表示叢集的 RGM 控制的服務的資源類型) 的回呼方法之 shell 程序檔實現中。您可以使用這些指令來：

- 存取有關資源、資源類型、資源群組以及叢集的資訊
- 搭配使用監視器來設定資源的 `Status` 與 `Status_msg` 特性
- 要求重新啓動或重新配置資源群組

注意 – 儘管本節提供 shell 指令的簡要說明，但 1HA 區段中的個別線上援助頁仍將提供 shell 指令的決定性參考。除非另外註明，否則每個指令將具有一個同名的線上援助頁。

RMAPI 資源指令

您可以使用這些指令存取有關資源的資訊或設定資源的 `Status` 特性與 `Status_msg` 特性。

`scha_resource_get`
在 RGM 的控制下存取有關資源或資源類型的資訊。它與 `scha_resource_get()` 函式提供的資訊相同。

`scha_resource_setstatus`
在 RGM 的控制下設定資源的 `Status` 特性與 `Status_msg` 特性。資源的監視器將使用此指令來指示其感知到的資源狀態。它與 `scha_resource_setstatus()` C 函式提供的功能性相同。

注意 – 儘管 `scha_resource_setstatus()` 對資源監視器特別有用，但任何程式都可以呼叫它。

資源類型指令

此指令存取藉由 RGM 註冊之資源類型的資訊。

`scha_resourcetype_get`
該指令與 `scha_resourcetype_get()` C 函式提供的功能性相同。

資源群組指令

您可以使用這些指令存取有關這些指令的資訊或重新啓動資源群組。

`scha_resourcegroup_get`
在 RGM 的控制下存取有關資源群組的資訊。該指令與 `scha_resourcetype_get()` C 函式提供的功能性相同。

`scha_control`
要求在 RGM 控制下重新啓動資源群組或將其重新配置到其他節點。該指令與 `scha_control()` C 函式提供的功能性相同。

叢集指令

該指令存取有關叢集的資訊，如節點名稱、節點 ID 與節點狀態、叢集名稱、資源群組等。

`scha_cluster_get`
該指令與 `scha_cluster_get()` C 函式提供的資訊相同。

C 函式

C 函式將用於資源類型的回呼方法之 C 程式實作，表示叢集的 RGM 控制的服務。您可以使用這些函式來執行以下作業：

- 存取有關資源、資源類型、資源群組以及叢集的資訊
- 搭配使用監視器來設定資源的 `Status` 與 `Status_msg` 特性
- 要求重新啓動或重新配置資源群組
- 將錯誤碼轉換爲適當的錯誤訊息

注意 – 儘管本章提供 C 函式的簡要說明，但個別 (`3HA`) 線上援助頁 仍將提供 C 函式的決定性參考。除非另外註明，否則每個函式將具有一個同名的線上援助頁。請參閱 `scha_calls`(3HA) 線上援助頁，以取得有關 C 函式的輸出引數與回覆碼之資訊。

資源函式

這些函式存取有關 RGM 管理的資源之資訊，或指示監視器感知的資源狀態。

`scha_resource_open()`、`scha_resource_get()` 和 `scha_resource_close()`
這些函式共同存取有關 RGM 管理的資源之資訊。`scha_resource_open()` 函式初始化對資源的存取，並爲 `scha_resource_get()` 傳回控點，該函式將存取資源資訊。`scha_resource_close()` 函式將使控點無效，並釋放爲 `scha_resource_get()` 傳回值配置的記憶體。

透過叢集重新配置或管理動作，資源可以在 `scha_resource_open()` 傳回資源的控點之後變更，在這種情況下 `scha_resource_get()` 透過控點獲取的資訊可能不準確。如果在資源上進行叢集重新配置或管理動作，RGM 將 `scha_err_seqid` 錯誤碼傳回至 `scha_resource_get()`，以指示有關資源的資訊可能已變更。這是一種不嚴重的錯誤訊息；函式可以成功傳回。您可以選擇忽略該訊息並接受傳回的資訊，或者可以關閉目前控點並開啓新的控點，以存取有關該資源的資訊。

單一線上援助頁可說明這三個函式。您可以透過 `scha_resource_open`(3HA)、`scha_resource_get`(3HA) 或 `scha_resource_close`(3HA) 中任一個別函式來存取此線上說明手冊。

`scha_resource_setstatus()`
在 RGM 的控制下設定資源的 `Status` 特性與 `Status_msg` 特性。資源的監視器使用此函式來指示資源的狀態。

注意 – 儘管 `scha_resource_setstatus()` 對資源監視器特別有用，但任何程式都可以呼叫它。

資源類型函式

這些函式可以共同存取有關藉由 RGM 註冊之資源類型的資訊。

`scha_resourcetype_open()`、`scha_resourcetype_get()`、`scha_resourcetype_close()`
`scha_resourcetype_open()` 函式初始化對資源的存取，並爲 `scha_resourcetype_get ()` 傳回控點，該函式將存取資源類型資訊。`scha_resourcetype_close()` 函式將使控點無效，並釋放爲 `scha_resourcetype_get()` 傳回值配置的記憶體。

透過叢集重新配置或管理動作，資源類型可以在 `scha_resourcetype_open()` 傳回資源類型的控點之後變更，在這種情況下 `scha_resourcetype_get()` 透過控點獲取的資訊可能不準確。如果在資源類型上進行叢集重新配置或管理動作，RGM 將 `scha_err_seqid` 錯誤碼傳回至 `scha_resourcetype_get()`，以指示有關資源類型的資訊可能已變更。這是一種不嚴重的錯誤訊息；函式可以成功傳回。您可以選擇忽略訊息並接受傳回的資訊，或者可以關閉目前的控點並開啓新控點，以存取有關資源類型的資訊。

單一線上援助頁可說明這三個函式。您可以透過 `scha_resourcetype_open`(3HA)、`scha_resourcetype_get`(3HA) 或 `scha_resourcetype_close`(3HA) 中的任一個別函式來存取此線上說明手冊。

資源群組函式

您可以存取有關這些函式的資訊或使用這些函式重新啓動資源群組。

`scha_resourcegroup_open`(3HA)、`scha_resourcegroup_get`(3HA) 和 `scha_resourcegroup_close`(3HA)
這些指令共同存取有關 RGM 管理的資源群組之資訊。`scha_resourcegroup_open()` 函式初始化對資源群組的存取，並爲 `scha_resourcegroup_get ()` 傳回控點，該函式將存取資源群組資訊。`scha_resourcegroup_close()` 函式將使控點無效，並釋放爲 `scha_resourcegroup_get()` 傳回值配置的記憶體。

透過叢集重新配置或管理動作，資源群組可以在 `scha_resourcegroup_open()` 傳回資源群組的控點之後變更，在這種情況下 `scha_resourcegroup_get()` 透過控點獲取的資訊可能不準確。如果在資源群組上發生叢集重新配置或管理動作，RGM 將傳回 `scha_err_seqid` 錯誤碼至 `scha_resourcegroup_get()`，以指示有關資源群組的資訊可能已變更。這是一種不嚴重的錯誤訊息；函式可以成功傳回。您可以選擇忽略訊息並接受傳回的資訊，或者可以關閉目前的控點並開啓新控點，以存取有關資源群組的資訊。

`scha_control`(3HA)
要求在 RGM 控制下重新啓動資源群組或將其重新配置到其他節點。

叢集函式

這些函式將存取或傳回有關叢集的資訊。

`scha_cluster_open`(3HA)、 `scha_cluster_get`(3HA)、 `scha_cluster_close`(3HA)
這些函式將共同存取有關叢集的資訊，如節點名稱、節點 ID 與節點狀態、叢集名稱、資源群組，等等。

透過重新配置或管理動作，叢集可以在 `scha_cluster_open()` 傳回叢集的控點之後變更，在這種情況下 `scha_cluster_get()` 透過控點獲取的資訊可能不準確。如果在叢集上發生重新配置或管理動作，RGM 將傳回 `scha_err_seqid` 錯誤碼至 `scha_cluster_get()`，以指示有關叢集的資訊可能已變更。這是一種不嚴重的錯誤訊息；函式可以成功傳回。您可以選擇忽略訊息並接受傳回的資訊，或者可以關閉目前的控點並開啓新控點，以存取有關叢集的資訊。

`scha_cluster_getlogfacility`(3HA)
傳回目前作爲叢集日誌使用的系統日誌工具之編號。使用傳回值與 Solaris `syslog ()` 函式，將事件與狀況訊息記錄到叢集日誌中。

`scha_cluster_getnodename`(3HA)
傳回在其上呼叫函式之叢集節點的名稱。

公用程式函式

此函式將錯誤碼轉換爲錯誤訊息。

`scha_strerror`(3HA)
將錯誤碼 (由 `scha_` 函式之一傳回) 轉換爲相應的錯誤訊息。使用該函式與 `logger` 將訊息記錄到系統日誌 (`syslog`) 中。

RMAPI 回呼方法

回呼方法是由 API 提供用於實現資源類型的關鍵元素。如果叢集成員資格中發生變更 (如節點開機或當機)，則回呼方法啓用 RGM 來控制叢集中的資源。

注意 – 因爲用戶端程式控制叢集系統上的 HA 服務，所以回呼方法將由具有根許可權的 RGM 來執行。安裝與管理這些具有限制性檔案所有權與許可權的方法。具體來說，給予它們一個具有特權的所有者，如 `bin` 或 `root`，且使它們不可寫入。

本節說明回呼方法引數以及退出碼與退出清單，並說明以下種類的回呼方法：

- 控制與初始化方法
- 管理支援方法
- 網路相關的方法
- 監視器控制方法

注意 – 儘管本節提供回呼方法的簡要說明，其中包含在其上呼叫方法的點以及資源上預期發生的效果，但 `rt_callbacks`(1HA) 線上援助頁仍將是回呼方法的決定性參考。

方法引數

RGM 將呼叫回呼方法，如下所示：

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

方法是註冊爲 `Start`、`Stop` 或其他回呼的程式之路徑名稱。資源類型的回呼方法將在其註冊檔案中宣告。

所有回呼方法引數都將作爲標定值傳送，其中 `-R` 指示資源實例的名稱、`-T` 指示資源的類型，而 `-G` 指示將資源配置於其中的群組。配合使用引數與存取函式來擷取有關資源的資訊。

使用其他引數 (資源與在其上呼叫資源的資源群組之特性值) 呼叫 `Validate` 方法。

請參閱 `scha_calls`(3HA)，以取得詳細資訊。

退出碼

所有回呼方法均定義了相同的退出碼，以指定呼叫該方法對資源狀態產生的效果。scha_calls(3HA) 線上援助頁將說明所有這些退出碼。這些退出碼爲：

- 0 – 方法成功
- 所有非 0 值 – 方法失敗

RGM 也將處理回呼方法執行的非正常故障，如逾時與核心傾印。

方法實作必須使用每個節點上的 syslog 輸出故障資訊。不能保證寫入 stdout 或 stderr 的輸出發送至使用者 (儘管輸出目前顯示在本機節點的主控台上)。

控制與初始化回呼方法

主要控制與初始化回呼方法將啓動與停止資源。其他方法執行資源上的初始化程式碼與終止程式碼。

Start
: 當包含資源的資源群組在某個叢集節點上線運作時，將在該節點上呼叫此必需方法。這個方法將啓動該節點上的資源。

  Start 方法啓動的資源已啓動並在本機節點上可用時，該方法才應結束。因此，在 Start 方法結束之前，應輪詢資源以確定資源已啓動。此外，您應爲此方法設定足夠長的逾時值。例如，某些資源 (如資料庫常駐程式) 的啓動時間很長，因此需要該方法具有較長的逾時值。

  RGM 回應 Start 方法故障的方式取決於 Failover_mode 特性的設定。

  資源類型註冊檔案中的 START_TIMEOUT 特性設定資源的 Start 方法之逾時值。

Stop
: 當包含資源的資源群組在某個叢集節點上離線時，將在該節點上呼叫此必需方法。如果資源處於使用中狀態，則該方法將取消啓動資源。

  Stop 方法應該在其控制的資源已完全停止了在本機節點上的所有活動並已關閉了所有檔案描述元後才結束。否則，由於 RGM 假定資源已停止，而實際上資源仍在作用中，可能導致資料毀壞。避免資料毀壞的最安全方法是終止與資源相關的本機節點上的所有程序。

  在 Start 方法結束之前，應輪詢資源以決定資源已停止。此外，您應爲此方法設定足夠長的逾時值。例如，某些資源 (如資料庫常駐程式) 的停止時間很長，因此需要該方法具有較長的逾時值。

  RGM 回應 Stop 方法失敗的方式取決於 Failover_mode 特性的設定 (請參閱第 213 頁的 「資源特性」)。

  資源類型註冊檔案中的 STOP_TIMEOUT 特性設定資源的 Stop 方法之逾時值。

`Init`
當資源處於受管理狀態時 (即資源所在的資源群組從未受管理狀態切換至受管理狀態時；或在已經受管理的資源群組中建立資源時)，將呼叫此選用方法，以執行資源的一次性初始化。在 `Init_nodes` 資源特性決定的節點上呼叫該方法。

`Fini`
當資源成爲不受管理狀態時，當資源所在的資源群組切換到未受管理狀態時，或當從受管理資源群組中刪除資源時，將呼叫此選擇性方法，以執行清除資源作業。在 `Init_nodes` 資源特性決定的節點上呼叫該方法。

`Boot`
將呼叫這個與 Init 相似的方法以初始化節點上的資源，這些節點在包含資源的資源群組已置於 RGM 的管理之下後加入叢集。在 `Init_nodes` 資源特性決定的節點上呼叫該方法。當節點由於啓動或重新啓動而連結或重新連結叢集時，將呼叫 `Boot` 方法。

注意 – `Init`、`Fini` 或 `Boot` 方法的失敗將導致 `syslog()` 函式產生一條錯誤訊息，但不會另外影響資源的 RGM 管理。

管理支援方法

資源上的管理動作包含設定與變更資源特性。`Validate` 與 `Update` 回呼方法可使資源類型實作掛上這些管理動作。

`Validate`
當建立資源和管理動作更新資源或其所在資源群組的特性時，將呼叫此選擇性方法。在資源類型的 Init_nodes 特性指示的叢集節點集上呼叫該方法。在套用建立或更新之前呼叫 `Validate`，而任意節點上的方法之故障退出碼都會導致取消建立或更新。

僅當透過管理動作變更資源或資源群組特性時，才會呼叫 `Validate`，而在 RGM 設定特性或監視器設定資源特性 `Status` 與 `Status_msg` 時，不呼叫該方法。

`Update`
呼叫這個可選用的方法，以通知正在執行的資源特性已經變更。在管理動作成功地設定了資源或資源所在群組的特性之後，將呼叫 `Update`。將在資源線上運作的節點上呼叫該方法。該方法使用 API 存取函式，以讀取可能影響使用中資源的特性值，並相應地調整正在執行的資源。

`Update` 方法的失敗將導致 `syslog()` 函式產生一條錯誤訊息，但不會另外影響資源的 RGM 管理。

網路相關的回呼方法

使用網路位址資源的服務可能要求依照與網路位址配置相關的特定次序執行啓動或停止步驟。以下可選用的回呼方法 `Prenet_start` 與 `Postnet_stop` 可使資源類型實作在配置或取消配置相關網路位址前後執行特殊的啓動動作與關閉動作。

`Prenet_start`
在配置相同資源群組中的網路位址之前，將呼叫這個可選用的方法來執行特殊的啓動動作。

`Postnet_stop`
在配置完相同資源群組中的網路位址之後，將呼叫這個可選用的方法來執行特殊的關閉動作。

監視器控制回呼方法

資源類型實作可選擇性地包含一個用於監視資源性能、報告資源狀況或在發生資源故障時採取動作的程式。`Monitor_start`、`Monitor_stop` 以及 `Monitor_check` 方法支援資源類型實作中資源監視器的實作。

`Monitor_start`
在啓動資源之後，將呼叫這個可選用的方法來啓動資源的監視器。

`Monitor_stop`
在停止資源之前，將呼叫這個可選用的方法來停止資源的監視器。

`Monitor_check`
在對節點重新配置資源群組之前，將呼叫這個可選用的方法來存取該節點的可靠性。`Monitor_check` 方法必須被實施，以便該方法不與目前正在執行的其他方法衝突。

第 5 章

資料服務範例

本章說明了用於 `in.named` 應用程式的 Sun Cluster 資料服務範例 HA-DNS。`in.named` 常駐程式是網域名稱服務 (DNS) 的 Solaris 實作。資料服務範例展示如何透過使用資源管理 API 來使應用程式具有高可用性。

資源管理 API 支援 shell 程序檔介面與 C 程式介面。本章中的範例應用程式是使用 shell 程序檔介面撰寫的。

本章包含以下資訊：

- 第 73 頁的 「資料服務範例概觀」
- 第 74 頁的 「定義資源類型註冊檔案」
- 第 79 頁的 「爲所有方法提供共用功能性」
- 第 83 頁的 「控制資料服務」
- 第 88 頁的 「定義故障監視器」
- 第 96 頁的 「處理特性更新」

資料服務範例概觀

資料服務範例會針對叢集事件 (如管理動作、應用程式故障或節點故障)，在叢集的節點之間啓動、停止、重新啓動與切換 DNS 應用程式。

應用程式重新啓動由程序監視設備 (PMF) 管理。如果應用程式失效數超過了故障時間視窗中的故障數，故障監視器會將包含應用程式資源的資源群組故障轉移至其他節點。

資料服務範例以 `PROBE` 方法的形式提供故障監視功能。該方法使用 `nslookup` 指令確保應用程式狀況正常。如果探測器探測到掛起的 DNS 服務，它會嘗試透過在本機重新啓動 DNS 應用程式來校正此情形。如果這樣沒有改善情形並且探測器多次探測到服務的問題，則探測器會嘗試將服務故障轉移至該叢集中的其他節點。

具體來說，資料服務範例包含：

- 資源類型註冊檔案，用於定義資料服務的靜態特性。
- `Start` 回呼方法，當包含 HA-DNS 資料服務的資源群組上線時，RGM 將呼叫該回呼方法，以啓動 `in.named` 常駐程式。
- `Stop` 回呼方法，當包含 HA-DNS 資料服務的資源群組離線時，RGM 將呼叫該回呼方法，以停止 `in.named` 常駐程式。
- 故障監視器，透過確認 DNS 伺服器正在執行來檢查服務可用性。故障監視器透過使用者定義的 `PROBE` 方法來實施、透過 `Monitor_start` 與 `Monitor_stop` 回呼方法來啓動與停止。
- `Validate` 回呼方法，由 RGM 呼叫以驗證服務的配置目錄是可存取的。
- `Update` 回呼方法，當系統管理員變更資源特性的值時，RGM 將呼叫該回呼方法，以重新啓動故障監視器。

定義資源類型註冊檔案

此範例中的資源類型註冊 (RTR) 檔案定義 DNS 資源類型的靜態配置。此類型資源將繼承 RTR 檔案中所定義的特性。

當叢集管理員註冊 HA-DNS 資料服務時，RGM 將讀取 RTR 檔案中的資訊。

RTR 檔案概觀

RTR 檔案採用明確定義的格式。在此檔案中，首先定義資源類型特性，接著定義系統定義的資源特性，最後定義延伸特性。請參閱 `rt_reg`(4) 線上說明手冊與第 29 頁的 「設定資源特性和資源類型特性」，以取得詳細資訊。

本節說明 RTR 檔案範例中特定的特性，還提供檔案不同部分的清單。如需 RTR 檔案範例內容的完整清單，請參閱第 231 頁的 「資源類型註冊檔案清單」。

RTR 檔案範例中的資源類型特性

如以下清單所示，RTR 檔案範例以註釋開始，其後跟隨用於定義 HA-DNS 配置的資源類型特性。

```
#
# Copyright (c) 1998-2004 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#
```

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

RESOURCE_TYPE = "sample";
VENDOR_ID = SUNW;
RT_DESCRIPTION = "Domain Name Service on Sun Cluster";

RT_VERSION ="1.0";
API_VERSION = 2;
FAILOVER = TRUE;

RT_BASEDIR=/opt/SUNWsample/bin;
PKGLIST = SUNWsample;

START         =   dns_svc_start;
STOP          =   dns_svc_stop;

VALIDATE      =   dns_validate;
UPDATE        =   dns_update;

MONITOR_START =   dns_monitor_start;
MONITOR_STOP  =   dns_monitor_stop;
MONITOR_CHECK =   dns_monitor_check;
```

提示 – 您必須宣告 `Resource_type` 特性爲 RTR 檔案中的第一個項目。否則，資源類型的註冊將會失敗。

注意 – RGM 處理特性名稱時不區分大小寫。在 Sun 提供的 RTR 檔案中，除了方法名稱，特性的慣例是名稱的第一個字母大寫，而其餘的字母小寫。方法名稱以及特性性質包含的都是大寫字母。

以下是有關這些特性的某些資訊：

- 您可以單獨使用 `Resource_type` 特性來指定資源類型名稱 (`sample`) ；或使用 `Vendor_id` 作爲前綴，並用「.」將其與資源類型分隔開 (`SUNW.sample`)。

  如果您使用 `Vendor_id`，請將其作爲公司定義資源類型的證券符號。資源類型名稱在叢集中必須是唯一的。
- `RT_version` 特性識別由供應商指定的資料服務範例版本。
- `API_version` 特性識別 Sun Cluster 版本。例如，`API_version = 2` 指示資料服務在 Sun Cluster 版本 3.0 下執行。
- `Failover = TRUE` 指示資料服務無法在可以一次在多個節點上處於線上狀態的資源群組中執行。
- `RT_basedir` 指向 `/opt/SUNWsample/bin` 作爲完整相對路徑 (如回呼方法路徑) 的目錄路徑。
- `Start`、`Stop` 以及 `Validate` 等提供 RGM 呼叫的各個回呼方法程式的路徑。這些路徑爲 `RT_basedir` 所指定目錄的相對路徑。

- `Pkglist` 識別 `SUNWsample` 作爲包含資料服務範例安裝的套件。

未在此 RTR 檔案中指定的資源類型特性 (如 `Single_instance`、`Init_nodes` 與 `Installed_nodes`) 將取得它們的預設值。第 207 頁的 「資源類型特性」包含資源類型特性的完整清單，其中包括它們的預設值。

叢集管理員無法變更 RTR 檔案中爲資源類型特性指定的值。

RTR 檔案範例中的資源特性

依照慣例，由您宣告 RTR 檔案中跟隨資源類型特性的資源特性。資源特性包含由 Sun Cluster 提供的系統定義的特性以及您定義的延伸特性。對於兩者中任一種類型，您都可以指定一些由 Sun Cluster 提供的特性性質，如最小值、最大值和預設值。

RTR 檔案中系統定義的特性

以下清單將顯示 RTR 檔案範例中系統定義的特性。

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

儘管 Sun Cluster 提供系統定義的特性，但您可以使用資源特性性質來設定不同的預設值。請參閱第 228 頁的 「資源特性性質」，以取得可套用至資源特性的性質之完整清單。

請注意以下有關 RTR 檔案範例中系統定義資源特性的資訊。

- Sun Cluster 爲所有逾時提供最小值 (1 秒) 與預設值 (3600 秒)。RTR 檔案範例會將最小值變更爲 60 秒，而將預設值變更爲 300 秒。叢集管理員可以接受此預設值，或將逾時值變更爲其他值 (60 或更高的值)。Sun Cluster 不具有最大的允許值。
- `Thorough_Probe_Interval`、`Retry_count` 以及 `Retry_interval` 特性將 `TUNABLE` 性質設定爲 `ANYTIME`。此設定意味著叢集管理員可以變更這些特性的值，即使資料服務正在執行。透過資料服務範例實施的故障監視器將使用這些特性。當透過管理動作變更這些或其他資源特性時，資料服務範例將實施 `Update` 方法來停止與重新啓動故障監視器。請參閱第 100 頁的 「`Update` 方法」。
- 資源特性分爲
  - **必需** — 叢集管理員必須在建立資源時指定一個值；
  - **可選用** — 如果管理員未指定值，則系統會提供一個預設值。
  - **條件式** — 只有在 RTR 檔案中宣告了特性，RGM 才會建立該特性。

  資料服務範例的故障監視器將使用 `Thorough_probe_interval`、`Retry_count`、`Retry_interval` 和 `Network_resources_used` 條件式特性，因此開發人員需要在 RTR 檔案中宣告它們。請參閱 `r_properties`(5) 線上說明手冊或第 213 頁的 「資源特性」，以取得有關如何將特性分類的資訊。

RTR 檔案中的延伸特性

RTR 檔案範例的末尾爲延伸特性，如以下清單中所示

```
# 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)";
}
```

RTR 檔案範例定義 `Confdir` 與 `Probe_timeout` 兩個延伸特性。`Confdir` 指定 DNS 配置目錄的路徑。此目錄包含 `in.named` 檔案，DNS 需要此檔案才能成功執行。資料服務範例的 `Start` 方法與 `Validate` 方法都將使用此特性，以驗證在啓動 DNS 之前配置目錄與 `in.named` 檔案是否均可存取。

配置資料服務時，Validate 方法將驗證新目錄是否可存取。

資料服務範例的 PROBE 方法不是 Sun Cluster 回呼方法，而是使用者定義的方法。因此，Sun Cluster 不爲其提供 Probe_timeout 特性。開發者已在 RTR 檔案中定義了延伸特性，以允許叢集管理員配置 Probe_timeout 值。

爲所有方法提供共用功能性

本節描述在資料服務範例的所有回呼方法中使用的以下功能性：

- 第 79 頁的 「識別指令解譯程式與匯出路徑」。
- 第 79 頁的 「宣告變數 PMF_TAG 與 SYSLOG_TAG」。
- 第 80 頁的 「剖析函式引數」。
- 第 82 頁的 「產生錯誤訊息」。
- 第 82 頁的 「獲取特性資訊」。

識別指令解譯程式與匯出路徑

Shell 程序檔的第一行必須識別指令解譯程式。資料服務範例中每個方法程序檔都識別指令解譯程式，如下所示：

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

應用程式範例中的所有方法程序檔均匯出 Sun Cluster 二進位檔和程式庫的路徑，而不依賴使用者的 PATH 設定。

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

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

宣告變數 PMF_TAG 與 SYSLOG_TAG

所有方法程序檔 (Validate 除外) 均使用 pmfadm 來啓動 (或停止) 資料服務或監視器，並傳送資源的名稱。每個程序檔都定義一個可傳送至 pmfadm 的變數 PMF_TAG，以識別資料服務或監視器。

同樣，每個方法程序檔也都透過系統登錄使用 logger 指令來記錄訊息。每個程序檔都定義一個可使用 -t 選項傳送至 logger 的變數 SYSLOG_TAG，以識別將爲其記錄訊息的資源之資料類型、資源群組以及資源名稱。

所有方法都將以相同的方式定義 SYSLOG_TAG，如以下範例所示。dns_probe、dns_svc_start、dns_svc_stop 和 dns_monitor_check 方法按如下所示定義 PMF_TAG (pmfadm 和 logger 的使用源自 dns_svc_stop 方法)：

```
##############################################################################
# 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 以及 dns_update 方法都將定義 PMF_TAG，如下所示 (pmfadm 的使用來自 dns_monitor_stop 方法)：

```
##############################################################################
# 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
```

剖析函式引數

RGM 呼叫所有的回呼方法 (除了 Validate)，如下所示。

method_name -R *resource_name* -T *resource_type_name* -G *resource_group_name*

方法名稱是實施回呼方法的程式之路徑名稱。資料服務爲 RTR 檔案中的每個方法指定路徑名稱。這些路徑名稱相對於 RT_basedir 特性指定的目錄，也位於 RTR 檔案中。例如，在資料服務範例的 RTR 檔案中，將如下所示指定基本目錄與方法名稱。

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

所有回呼方法引數都將作爲標定值傳送，其中 -R 指示資源實例的名稱、-T 指示資源的類型，而 -G 指示將資源配置於其中的群組。請參閱 rt_callbacks(1HA) 線上援助頁，以取得有關回呼方法的詳細資訊。

注意 – 使用其他引數 (資源與在其上呼叫資源的資源群組之特性值) 呼叫 `Validate` 方法。請參閱第 96 頁的 「處理特性更新」，以取得詳細資訊。

每個回呼方法都需要一個函式，以剖析傳送給它的引數。由於傳送給所有回呼相同的引數，因此資料服務提供一個單一剖析函式，該函式用於應用程式內的所有回呼中。

以下顯示了用於應用程式範例中回呼方法的 `parse_args()` 函式。

```
##############################################################################
# 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
}
```

注意 – 儘管應用程式範例中的 `PROBE` 方法是使用者定義的方法 (而不是 Sun Cluster 回呼方法)，但是也可透過與回呼方法使用相同的引數來呼叫。因此，該方法包含一個與其他回呼方法所使用剖析函式完全相同的剖析函式。

該剖析函式在 `MAIN` 中稱爲：

```
parse_args "$@"
```

產生錯誤訊息

建議回呼方法使用 syslog 工具將錯誤訊息輸出至一般使用者。資料服務範例中的所有回呼方法都使用 scha_cluster_get() 函式，以擷取用於叢集登錄的 syslog 工具之編號，如下所示：

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

該值將儲存於 shell 變數 SYSLOG_FACILITY 中，並可以作爲 logger 指令的功能使用，以記錄叢集登錄中的訊息。例如，資料服務範例中的 Start 方法將擷取 syslog 工具，並記錄資料服務已啓動的訊息，如下所示：

```
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
```

請參閱 scha_cluster_get(1HA) 線上說明手冊，以取得詳細資訊。

獲取特性資訊

大多數回呼方法需要獲取有關資料服務的資源與資源類型特性的資訊。API 將爲此目的提供 scha_resource_get() 函式。

兩種資源特性 (系統定義的特性與延伸特性) 可用。儘管已預先定義了系統定義的特性，但是您仍要在 RTR 檔案中定義延伸特性。

當您使用 scha_resource_get() 來獲取系統定義的特性值時，您可以使用參數 -O 指定特性的名稱。指令將僅傳回特性的**值**。例如，在資料服務範例中，Monitor_start 方法需要尋找探測程式，以便該方法可以啓動此程式。測試程式位於資料服務的基本目錄中，RT_basedir 特性指向該基本目錄，因此 Monitor_start 方法將擷取 RT_basedir 的值，並將其置於 RT_basedir 變數中，如下所示。

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

對於延伸特性，您必須使用參數 -O 指定特性爲延伸特性，並提供特性名稱作爲最後一個參數。對於延伸特性，指令將傳回特性的**類型**與**值**。例如，在資料服務範例中，探測程式將擷取 probe_timeout 延伸特性的類型與值，然後使用 awk 以僅將值置於 PROBE_TIMEOUT shell 變數中，如下所示。

```
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}'`
```

控制資料服務

資料服務必須提供 Start 方法或 Prenet_start 方法來啓動叢集上的應用程式常駐程式，並提供 Stop 方法或 Postnet_stop 方法來停止叢集上的應用程式常駐程式。資料服務範例將實施 Start 方法與 Stop 方法。請參閱第 39 頁的 「決定要使用的 Start 和 Stop 方法」，以取得有關您何時要使用 Prenet_start 方法與 Postnet_stop 方法來取代以上兩個方法的資訊。

Start 方法

當使包含資料服務資源的資源群組在叢集節點上線時或該資源群組已處於線上狀態並啓用資源時，RGM 將呼叫該叢集節點上的 Start 方法。在應用程式範例中，Start 方法將在該節點上啓動 in.named (DNS) 常駐程式。

本節說明應用程式範例的 Start 方法之主要部分，但未說明如第 79 頁的 「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 Start 方法的完整清單，請參閱第 234 頁的 「Start 方法」。

Start 概觀

在嘗試啓動 DNS 之前，資料服務範例中的 Start 方法將驗證配置目錄與配置檔案 (named.conf) 是否可存取並且可用。named.conf 中的資訊對於 DNS 的成功作業非常重要。

此回呼方法使用 PMF (pmfadm) 來啓動 DNS 常駐程式 (in.named)。如果 DNS 當機或無法啓動，則 PMF 會嘗試在指定的間隔時間啓動 DNS 指定的次數。重試的次數與間隔時間透過資料服務的 RTR 檔案中的特性來指定。

驗證配置

爲了作業，DNS 需要配置目錄內 named.conf 檔案的資訊。因此，Start 方法將執行一些完整檢查，以驗證在嘗試啓動 DNS 之前目錄與檔案是否均可存取。

Confdir 延伸特性提供配置目錄的路徑。特性本身定義在 RTR 檔案中。然而，叢集管理員會在配置資料服務時指定實際位置。

在資料服務範例中，Start 方法使用 scha_resource_get() 函式擷取配置目錄的位置。

注意 – 由於 Confdir 是一個延伸特性，因此 scha_resource_get() 將傳回類型與值。awk 指令僅擷取值，並將其置於 shell 變數 CONFIG_DIR 中。

```
# 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}'`
```

然後 Start 方法使用 CONFIG_DIR 的值來驗證目錄是否可存取。如果目錄不可存取，則 Start 將記錄一條錯誤訊息，並以錯誤狀態退出。請參閱第 85 頁的 「Start 退出狀態」。

```
# 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
```

在啓動應用程式常駐程式之前，該方法將執行最後的檢查，以驗證 named.conf 檔案是否存在。如果檔案不存在，Start 將記錄一條錯誤訊息，並以錯誤狀態退出。

```
# 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
```

啓動應用程式

該方法將使用程序管理員工具 (pmfadm) 來啓動應用程式。pmfadm 指令可讓您設定在指定時間框架期間重新啓動應用程式的次數。RTR 檔案包含兩個特性，其中 Retry_count 指定嘗試重新啓動應用程式的次數，而 Retry_interval 指定重新啓動的時間間隔。

Start 方法使用 scha_resource_get() 函式來擷取 Retry_count 與 Retry_interval 的值，並將它們的值儲存於 shell 變數中。然後使用 -n 與 -t 選項將這些值傳送至 pmfadm。

```
# 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 退出狀態

Start 方法應該在基礎應用程式實際執行並可用時 (特別是如果其他資料服務與該方法相依)，才以成功狀態退出。確認成功的一種方式是探測應用程式，以確認該應用程式在退出 Start 方法之前正在執行。對於複雜的應用程式 (如資料庫)，請確定將 RTR 檔案中 Start_timeout 特性的值設定的足夠高，以允許應用程式初始化與執行當機恢復的時間。

注意 – 由於資料服務範例中的應用程式資源 DNS 會快速啓動，因此資料服務範例在以成功狀態退出之前不會輪詢以確認其正在執行。

如果該方法無法啓動 DNS 並以失敗狀態退出，RGM 將檢查 Failover_mode 特性 (其決定如何反應)。由於資料服務範例不會明確地設定 Failover_mode 特性，因此該特性具有一個預設值 NONE (除非叢集管理員已置換了預設值並指定了其他值)。在這種情況下，RGM 僅會設定資料服務的狀態，而不採取其他任何動作。在相同的節點上重新啓動或故障轉移至其他節點，都需要使用者介入。

Stop 方法

當包含 HA-DNS 資源的資源群組在叢集節點上處於離線狀態時，或資源群組處於線上狀態而資源被停用時，將在該叢集節點上呼叫 Stop 方法。此方法將在該節點上停止 in.named (DNS) 常駐程式。

本節說明應用程式範例的 Stop 方法之主要部分，但未說明如第 79 頁的 「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 Stop 方法的完整清單，請參閱第 237 頁的 「Stop 方法」。

Stop 概觀

當嘗試停止資料服務時，有兩個需要考量的主要問題。第一個需要考量的問題是提供一個順序關機。透過 pmfadm 發送 SIGTERM 訊號，是用來完成順序關機的最佳方法。

第二個需要考量的問題是確保資料服務確實已停止，以避免將資料服務置於 Stop_failed 狀態。完成此作業的最佳方法是透過 pmfadm 傳送 SIGKILL 訊號。

資料服務範例中的 Stop 方法將考量這兩個問題。該方法首先發送 SIGTERM 訊號。如果此訊號無法停止資料服務，則該方法將發送 SIGKILL 訊號。

在嘗試停止 DNS 之前，這個 Stop 方法會確認程序確實正在執行。如果程序正在運行，Stop 將使用 PMF (pmfadm) 來停止它。

這個 Stop 方法被保證是等冪的。儘管 RGM 不應該在未首先透過呼叫資料服務的 Start 方法來啓動資料服務的情況下呼叫 Stop 方法兩次，但他仍可以呼叫資源上的 Stop 方法，即使資源從未啓動或自願失敗。因此，即使 DNS 不在執行，這個 Stop 方法仍會以成功狀態退出。

停止應用程式

Stop 方法提供一個停止資料服務的雙重方法：透過 pmfadm 使用 SIGTERM 訊號的有序或平穩方法，和使用 SIGKILL 訊號的突然或強制方法。Stop 方法將獲取 Stop_timeout 值 (Stop 方法必須在此時間範圍內返回)。Stop 然後將此時間的 80% 分配給順利停止，將此時間的 15% 分配給突然停止 (保留 5%)，如以下範例中所示。

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

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

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

Stop 方法使用 pmfadm -q 來 確認 DNS 常駐程式是否正在執行。如果此常駐程式正在執行，Stop 將會首先使用 pmfadm -s 來發送 TERM 訊號，以終止 DNS 程序。如果該訊號在 80% 的逾時值之後仍無法終止程序，則 Stop 將發送 SIGKILL 訊號。如果該訊號在 15% 的逾時值時間範圍內亦無法終止程序，則該方法將會記錄錯誤訊息，並以錯誤狀態退出。

如果 pmfadm 終止了程序，則方法將記錄一條程序已停止的訊息並以成功狀態退出。

如果 DNS 程序未在執行，則方法會記錄一條程序未在執行的訊息並無論如何會以成功狀態退出。以下程式碼範例顯示了 Stop 如何使用 pmfadm 來停止 DNS 程序。

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

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

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

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

   exit 0

fi

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

Stop 退出狀態

Stop 方法應該在基礎應用程式實際已停止時 (特別是如果其他資料服務與該方法相依)，才以成功狀態退出。無法執行此作業將導致資料毀壞。

對於複雜的應用程式 (如資料庫)，請確定將 RTR 檔案中 Stop_timeout 特性的值設定的足夠高，以允許應用程式在停止時進行清除所用的時間。

如果此方法無法停止 DNS 並以失敗狀態退出，則 RGM 將檢查 Failover_mode 特性 (其決定如何反應)。由於資料服務範例不會明確地設定 Failover_mode 特性，因此該特性具有一個預設值 NONE (除非叢集管理員已置換了預設值並指定了其他值)。在這種情況下，RGM 除了會設定資料服務的狀態爲 Stop_failed 外，不會採取任何動作。要求使用者介入以強制停止應用程式，並清除 Stop_failed 狀態。

定義故障監視器

應用程式範例實現基本故障監視器，以監視 DNS 資源 (in.named) 的可靠性。故障監視器包含：

- dns_probe，一個使用 nslookup 來驗證由資料服務範例控制的 DNS 資源是否正在執行的使用者定義的程式。如果 DNS 未在執行，此方法將嘗試在本機重新啓動 DNS，或依據嘗試重新啓動的次數，要求 RGM 將資料服務重新置於其他節點。
- dns_monitor_start，一個啓動 dns_probe 的回呼方法。如果啓用了監視，RGM 將在資料服務範例上線之後自動呼叫 dns_monitor_start。
- dns_monitor_stop，一個停止 dns_probe 的回呼方法。RGM 將在資料服務範例離線之前自動呼叫 dns_monitor_stop。
- dns_monitor_check，一個回呼方法，當 PROBE 程式將資料服務故障轉移至新節點時，該方法將呼叫 Validate 方法來驗證配置目錄是否可用。

探測程式

dns_probe 程式將實施一個連續的執行程序，該程序用來驗證由資料服務範例控制的 DNS 資源是否正在執行。dns_probe 由 dns_monitor_start 方法來啓動，RGM 會在資料服務範例上線之後自動呼叫該方法。資料服務由 dns_monitor_stop 方法來停止，RGM 會在資料服務範例離線之前呼叫該方法。

本節說明應用程式範例的 PROBE 方法之主要部分，但未說明如第 79 頁的 「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 PROBE 方法的完整清單，請參閱第 240 頁的 「PROBE 程式」。

探測概觀

探測在無窮迴圈中執行。它使用 nslookup 來驗證正確的 DNS 資源是否正在執行。如果 DNS 正在執行，探測將在規定間隔 (由 Thorough_probe_interval 系統定義的特性設定) 內休息，然後再進行檢查。如果 DNS 未在執行，此程式將嘗試在本機重新啓動 DNS，或依據嘗試重新啓動的次數，要求 RGM 將資料服務重新置於其他節點。

獲取特性值

此程式需要以下特性的值：

- `Thorough_probe_interval` – 設定探測休息的時間
- `Probe_timeout` – 對執行探測作業的 `nslookup` 指令，實施探測的逾時值
- `Network_resources_used` – 獲取 DNS 正在其上執行的 IP 位址
- `Retry_count` 與 `Retry_interval` – 決定嘗試重新啓動的次數與透過其計算次數的時間間隔
- `RT_basedir` – 獲取包含 `PROBE` 程式和 `gettime` 公用程式的目錄

`scha_resource_get()` 函式將獲取這些特性的值，並將這些值儲存於 shell 變數中，如下所示。

```
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‘
```

注意 – 對於系統定義的特性 (如 `Thorough_probe_interval`)，`scha_resource_get()` 將僅傳回值。對於延伸特性 (如 `Probe_timeout`)，`scha_resource_get()` 將傳回類型與值。使用 `awk` 指令來僅獲取值。

檢查服務的可靠性

探測本身是 `nslookup` 指令的一個無窮 `while` 迴圈。在 `while` 迴圈之前，將會設定一個暫存檔以存放 `nslookup` 回覆。*probefail* 和 *retries* 變數均初始化爲 0。

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

while 迴圈本身：

- 設定探測的休息間隔
- 使用 hatimerun 啓動 nslookup ，傳送 Probe_timeout 值與識別目標主機
- 基於 nslookup 回覆碼的成功或失敗，設定 *probefail* 變數
- 如果 *probefail* 設定爲 1 (失敗)，將會確認對 nslookup 的回覆來自資料服務範例而非其他 DNS 伺服器

以下是 while 迴圈碼。

```
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
```

評估重新啓動與故障轉移

probefail 變數是除了 0 (成功) 以外的某個值，意味著 nslookup 指令逾時或回覆來自範例服務的 DNS 之外的伺服器。在其中任何一種情況下，DNS 伺服器都不會發揮預期的功能，而故障監視器會呼叫 decide_restart_or_failover() 函式，以決定是在本機重新啓動資料服務還是要求 RGM 將資料服務重新置於其他節點。如果 *probefail* 變數爲 0，則會產生一條探測成功的訊息。

```
   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()` 函式使用時間視窗 (`Retry_interval`) 與失敗計數 (`Retry_count`) 來決定是在本機重新啓動 DNS 還是要求 RGM 將資料服務重新置於其他節點。該函式實施以下條件式程式碼 (請參閱第 240 頁的 「PROBE 程式」中 `decide_restart_or_failover()` 的程式碼清單)。

- 如果這是第一次失敗，請重新啓動資料服務。記錄錯誤訊息並按 `retries` 變數中的計數器。
- 如果這不是第一次失敗，但已超出視窗，則請重新啓動資料服務。記錄錯誤訊息、重設計數器並滑動視窗。
- 如果時間仍在視窗內，但已超出重試計數器，則故障轉移至其他節點。如果故障轉移未成功，請記錄錯誤訊息並以狀態 1 (失敗) 結束探測程式。
- 如果時間仍在視窗內，且尚未超出重試計數器，則請重新啓動資料服務。記錄錯誤訊息並按 `retries` 變數中的計數器。

如果重新啓動的次數達到了時間間隔內的限制次數，函式將要求 RGM 將資料服務重新置於其他節點。如果重新啓動的次數在限制範圍之內，或已超出間隔從而計數器重新開始計數，函式將嘗試重新啓動相同節點上的 DNS。請注意以下有關此函式的資訊：

- `gettime` 公用程式用於追蹤重新啓動之間的時間。這是位於 (`RT_basedir`) 目錄中的 C 程式。
- 系統定義的資源特性 `Retry_count` 與 `Retry_interval` 將決定嘗試重新啓動的次數以及透過其計數的間隔。儘管叢集管理員可以變更設定，但這些 RTR 檔案中的特性將預設爲在 5 分鐘 (300 秒) 的間隔內嘗試重新啓動 2 次。
- 將呼叫 `restart_service()` 函式，以嘗試重新啓動相同節點上的資料服務。請參閱下一節 (第 91 頁的 「重新啓動資料服務」)，以取得有關此函式的資訊。
- 透過 `GIVEOVER` 選項，`scha_control()` API 函式使包含資料服務範例的資源群組離線，並在其他節點上重新上線。

重新啓動資料服務

`restart_service()` 函式將由 `decide_restart_or_failover()` 呼叫，以嘗試重新啓動同一節點上的資料服務。此函式將執行以下作業。

- 此函式決定是否仍在 PMF 下註冊資料服務。如果仍註冊服務，函式將：
  - 獲取 `Stop` 方法名稱與資料服務的 `Stop_timeout` 值。
  - 使用 `hatimerun` 啓動資料服務的 `Stop` 方法，傳送 `Stop_timeout` 值。
  - (如果成功地停止了資料服務) 獲取 `Start` 方法名稱與資料服務的 `Start_timeout` 值。
  - 使用 `hatimerun` 啓動資料服務的 `Start` 方法，並傳送 `Start_timeout` 值。
- 如果已不再在 PMF 下註冊資料服務，意味著資料服務已超出在 PMF 下允許重試的最大次數，因此將使用 `GIVEOVER` 選項來呼叫 `scha_control()` 函式，以將資料服務故障轉移至其他節點。

```
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
}
```

探測退出狀態

如果嘗試本機重新啓動失敗並且嘗試將故障轉移至其他節點亦失敗，則資料服務範例的 PROBE 程式將以失敗狀態結束。該程式將會記錄訊息「故障轉移嘗試失敗」。

Monitor_start 方法

在資料服務範例上線之後，RGM 將呼叫 Monitor_start 方法來啓動 dns_probe 方法。

本節說明應用程式範例的 Monitor_start 方法之主要部分，但未說明如第 79 頁的「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 Monitor_start 方法的完整清單，請參閱第 245 頁的 「Monitor_start 方法」。

Monitor_start 概觀

此方法使用 PMF (pmfadm) 來啓動測試。

啓動探測

Monitor_start 方法將獲取 RT_basedir 特性的值，以建構 PROBE 程式的完整路徑名稱。該方法將使用無限重試選項 pmfadm (-n -1, -t -1) 來啓動探測，此選項意味著如果探測無法啓動，PMF 將在無限時間內嘗試無限次數來啓動探測。

```
# 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 方法

當資料服務範例離線時，RGM 將呼叫 Monitor_stop 方法來停止 dns_probe 的執行。

本節說明應用程式範例的 Monitor_stop 方法之主要部分，但未說明如第 79 頁的「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 Monitor_stop 方法的完整清單，請參閱第 247 頁的 「Monitor_stop 方法」。

Monitor_stop 概觀

該方法使用 PMF (pmfadm) 查看測試是否正在運行，如果是，請停止測試。

停止監視器

Monitor_stop 方法使用 pmfadm -q 查看探測是否正在執行，如果是，則使用 pmfadm -s 來停止探測。如果探測已停止，則該方法無論如何會以成功狀態結束，這樣會保證該方法的等?性。

```
# 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
```

注意 – 確定搭配 pmfadm 使用 KILL 訊號與而不是遮罩訊號 (如 TERM) 來停止探測。否則，Monitor_stop 方法可能無限期地掛起並最終逾時。出現此問題的原因是，當有必要重新啓動資料服務或故障轉移資料服務時，PROBE 方法呼叫 scha_control() 。當 scha_control() 呼叫 Monitor_stop 作爲使資料服務離線程序的一部分時，如果 Monitor_stop 使用遮罩訊號，它會掛起等待 scha_control() 完成，scha_control() 會掛起等待 Monitor_stop 完成。

Monitor_stop 退出狀態

如果 Monitor_stop 方法無法停止 PROBE 方法，則前者會記錄一條錯誤訊息。RGM 會在主要節點上將資料服務範例置於 MONITOR_FAILED 狀態，這樣可能會使該節點發生混亂。

Monitor_stop 應該在停止探測之後結束。

Monitor_check 方法

每當 PROBE 方法嘗試將包含資料服務的資源群組故障轉移至新節點時，RGM 均將呼叫 Monitor_check 方法。

本節說明應用程式範例的 Monitor_check 方法之主要部分，但未說明如第 79 頁的「爲所有方法提供共用功能性」中所描述的所有回呼方法的共用功能性，如 parse_args() 函式與獲取 syslog 工具。

如需 Monitor_check 方法的完整清單，請參閱第 248 頁的 「Monitor_check 方法」。

必須實現 Monitor_check 方法，以免與同時執行的其他方法發生衝突。

Monitor_check 方法呼叫 Validate 方法，以確認 DNS 配置目錄在新節點上可用。Confdir 延伸特性指向 DNS 配置目錄。因此，Monitor_check 將獲取 Validate 方法的路徑與名稱以及 Confdir 的值。它將傳送此值至 Validate，如以下清單所示。

```
# 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
```

請參閱第 96 頁的 「Validate 方法」，以查看應用程式範例如何確認節點托管資料服務的適合性。

處理特性更新

資料服務範例實現 Validate 方法與 Update 方法以處理叢集管理員進行的特性更新。

Validate 方法

當建立資源和管理動作更新資源或資源所在群組的特性時，RGM 將呼叫 Validate 方法。在套用建立或更新之前，RGM 將呼叫 Validate，並且此方法在任何節點上的故障結束碼均將導致取消建立或取消更新。

僅當透過管理動作變更資源或群組特性時，RGM 才會呼叫 Validate，而在 RGM 設定特性或監視器設定資源特性 Status 與 Status_msg 時，RGM 不會呼叫該方法。

注意 – 每當 PROBE 方法嘗試將資料服務之故障轉移至新節點時，Monitor_check 方法都將明確地呼叫 Validate 方法。

Validate 概觀

RGM 藉由傳送至其他方法的引數以及其他引數 (包括正在更新的特性與值)，來呼叫 Validate。因此，資料服務範例中的此方法必須實施一個不同的函式 parse_args ()，以處理其他引數。

資料服務範例中的 Validate 方法確認單一特性 (Confdir 延伸特性)。此特性指向 DNS 配置目錄，該目錄對於 DNS 的成功作業非常重要。

注意 – 由於配置目錄在執行 DNS 時無法變更，因此 Confdir 特性在 RTR 檔案中被宣告爲 TUNABLE = AT_CREATION。因此，從不會由於更新而呼叫 Validate 方法以確認 Confdir 特性，而是僅當建立資料服務資源時才呼叫該方法。

如果 Confdir 是 RGM 傳送至 Validate 的特性之一，parse_args() 函式將擷取並儲存其值。然後 Validate 將確認由 Confdir 的新值指向的目錄是可存取的，並確認該目錄中存在 named.conf 檔案且包含一些資料。

如果 parse_args() 函式無法從 RGM 傳送的指令行引數擷取 Confdir 的值，Validate 仍將嘗試驗證 Confdir 特性。Validate 使用 scha_resource_get () 來從靜態配置獲取 Confdir 的值。然後，它將執行相同的檢查，以確認配置目錄是可存取的，並包含非空的 named.conf 檔案。

如果 Validate 以失敗狀態退出，則所有特性 (而非僅僅 Confdir) 的更新或建立都將失敗。

Validate 方法剖析函式

RGM 將對 Validate 方法傳送一組與其他回呼方法不同的參數，因此 Validate 需要與其他方法不同的剖析引數之函式。請參閱 rt_callbacks(1HA) 線上說明手冊，以取得有關傳送至 Validate 與其他回呼方法之參數的詳細資訊。以下說明了 Validate parse_args() 函式。

```
#############################################################################
# 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
}
```

與其他方法的 `parse_args()` 函式一樣，此函式將提供三個旗號：提供 `R` 抓取資源名稱，提供 `G` 抓取資源群組名稱，以及提供 `T` 抓取 RGM 傳送的資源類型。

將忽略旗號 `r` (指示系統定義的特性)、旗號 `g` (指示資源群組特性) 以及旗號 `c` (指示驗證發生在建立資源期間)，因爲將在更新資源時呼叫此方法來驗證延伸特性。

`u` 旗號將 `UPDATE_PROPERTY` shell 變數的值設定爲 `1` (`TRUE`)。`x` 旗號抓取正在更新的特性之名稱與值。如果 `Confdir` 是正更新的特性之一，其值將置於 *CONFDIR* shell 變數中，且變數 *CONFDIR_FOUND* 設定爲 `1` (`TRUE`)。

驗證 `Confdir`

在 `Validate` 的 `MAIN` 函式中，`Validate` 首先將 *CONFDIR* 變數設定爲空字串，並將 *UPDATE_PROPERTY* 和 *CONFDIR_FOUND* 設定爲 0。

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

然後 `Validate` 將呼叫 `parse_args()`，以剖析 RGM 傳送的引數。

```
parse_args "$@"
```

然後 `Validate` 將檢查是否由於特性的更新而呼叫 `Validate`，以及 `Confdir` 延伸特性是否位於指令行上。然後 `Validate` 將驗證 `Confdir` 特性是否具有一個值，如果不具有，則以失敗狀態與一條錯誤訊息退出。

```
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
```

注意 – 具體地說，前面的程式碼將檢查是否由於更新 ($UPDATE_PROPERTY == 1) 而呼叫 Validate，是否**未**在指令行 (CONFDIR_FOUND == 0) 上找到特性，在此情況下，它會使用 scha_resource_get() 擷取 Confdir 的現有值。如果在指令行 (CONFDIR_FOUND == 1) 上找到 Confdir，則 *CONFDIR* 的值將來自 parse_args() 函式，而非來自 scha_resource_get ()。

然後 Validate 方法將使用 *CONFIG_DIR* 的值來驗證目錄是否可存取。如果目錄是不可存取的，Validate 將記錄一條錯誤訊息，並以錯誤狀態退出。

```
# 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
```

在驗證 Confdir 特性的更新之前，Validate 將執行最終檢查來驗證 named.conf 檔案是否存在。如果不存在，該方法將記錄一條錯誤訊息並以錯誤狀態退出。

```
# 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
```

如果通過了最終檢查，Validate 將記錄指示成功的訊息，並以成功狀態退出。

```
# 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 退出狀態

如果 Validate 以成功狀態 (0) 退出，則會使用新值建立 Confdir。如果 Validate 以失敗狀態 (1) 退出，則不會建立 Confdir 與任何其他特性，並會將一條指示原因的訊息發送至叢集管理員。

Update 方法

RGM 將呼叫 Update 方法，以通知正在執行的資源其特性已變更。在管理動作成功地設定了資源或其群組的特性之後，RGM 將呼叫 Update。將在資源線上運作的節點上呼叫該方法。

Update 概觀

Update 方法不會更新由 RGM 完成的特性。相反，它會通知正在執行的程序已發生更新。資料服務範例中唯一受特性更新影響的程序是故障監視器，因此 Update 方法停止與重新啓動的正是此程序。

Update 方法必須驗證故障監視器是否正在執行，然後使用 pmfadm 來終止它。該方法將獲取實施故障監視器的探測程式之位置，然後再次使用 pmfadm 來重新啓動此程式。

使用 Update 停止監視器

Update 方法將使用 pmfadm -q 驗證監視器是否正在執行，如果正在執行，則使用 pmfadm -s TERM 來終止它。如果成功地終止了監視器，系統會將一條大意爲此的訊息發送至使用者管理員。如果無法停止監視器，Update 將以失敗狀態退出，並發送一條錯誤訊息至使用者管理員。

```
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
```

重新啓動監視器

若要重新啓動監視器，Update 方法必須找到實現探測程式的程序檔。測試程式位於資料服務的基本目錄中，RT_basedir 特性指向該基本目錄。Update 擷取 RT_basedir 的值，並將其儲存於變數 RT_BASEDIR 中，如下所示。

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

然後，Update 將搭配 pmfadm 使用 RT_BASEDIR 的值，來重新啓動 dns_probe 程式。如果成功，Update 將以成功狀態退出，並將一條大意爲此的訊息發送至使用者管理員。如果 pmfadm 無法啓動探測程式，則 Update 將以失敗狀態退出並記錄一條錯誤訊息。

Update 退出狀態

Update 方法失敗將導致資源被置於「更新失敗」狀態。該狀態對資源的 RGM 管理沒有影響，但透過 syslog 工具指示對管理工具的更新動作失敗。

第 6 章

資料服務開發程式庫 (DSDL)

本章概述了構成資料服務開發程式庫 (DSDL) 的應用程式設計介面。DSDL 於 `libdsdev.so` 程式庫中實施，並包含於 Sun Cluster 套件中。

本章包含以下主題：

- 第 103 頁的 「DSDL 概觀」
- 第 103 頁的 「管理配置屬性」
- 第 104 頁的 「啓動與停止資料服務」
- 第 105 頁的 「實施故障監視器」
- 第 105 頁的 「存取網路位址資訊」
- 第 106 頁的 「對資源類型實作進行除錯」

DSDL 概觀

DSDL API 位於 RMAPI 的頂層。就其本身而論，它不會取代 RMAPI，而是將封裝並延伸 RMAPI 功能性。DSDL 透過對特定 Sun Cluster 整合問題提供預先決定的解決方案，簡化資料服務開發。因此，您可以將大部分開發時間用於應用程式固有的高可用性與高可延伸性問題，避免花費大量的時間來將應用程式啓動、關閉以及監視程序與 Sun Cluster 整合。

管理配置屬性

所有回呼方法均需要存取配置屬性。DSDL 透過以下方式支援對屬性的存取：

- 初始化環境
- 提供用來擷取特性值的簡易函式集

scds_initialize 函式 (必須在每個回呼方法開始時呼叫該函式) 執行以下作業：

- 檢查並處理 RGM 傳送至回呼方法的指令行引數 (argc 與 argv[])，讓您無需寫入指令行剖析函式。
- 設定供其他 DSDL 函式使用的內部資料結構。例如，從 RGM 擷取特性值的簡易函式會將值儲存於這些結構中。同樣，指令行的值 (優先於從 RGM 擷取的值) 也將儲存於這些資料結構中。

注意 – 對於 Validate 方法，scds_initialize 剖析在指令上傳送的特性值，讓您無需爲 Validate 寫入剖析函式。

---

scds_initialize 函式也會初始化記錄環境並驗證故障監視器探測設定。

DSDL 提供用來擷取資源、資源類型、資源類型群組特性以及常用延伸特性的函式集。這些函式透過使用以下慣例標準化對特性的存取。

- 每個函式僅使用一個控點引數 (由 scds_initialize 傳回)。
- 每個函式都對應一個特定的特性。函式的傳回值類型符合函式擷取的特性值類型。
- 函式不會傳回錯誤，因爲這些值已經由 scds_initialize 預先運算。除非在指令行傳送新的值，否則函式會從 RGM 擷取值。

啓動與停止資料服務

Start 方法預期會執行在叢集節點上啓動資料服務所需要執行的動作。通常這些動作包含：擷取資源特性、尋找應用程式特定的可執行檔案與配置檔案以及使用適當的指令行引數啓動應用程式。

scds_initialize 函式擷取資源配置。Start 方法可以使用特性簡易函式擷取特定特性 (如 Confdir_list) 的值，這些特性可識別用於啓動應用程式的配置目錄與配置檔案。

Start 方法可以呼叫 scds_pmf_start，以在程序監視設備 (PMF) 控制下啓動應用程式。PMF 可讓您指定要套用至程序的監視層次，並且提供在發生故障時重新啓動程序的功能。請參閱第 120 頁的 「xfnts_start 方法」，以取得透過 DSDL 實施的 Start 方法之範例。

即使在應用程式未執行時於某節點上呼叫 Stop 方法，此方法也必須等冪，以便其以成功狀態退出。如果 Stop 方法失敗，則正被停止的資源會被設定爲 STOP_FAILED 狀態，這樣將導致叢集強制重新啓動。

若要避免將資源置於 STOP_FAILED 狀態，Stop 方法必須盡力停止資源。scds_pmf_stop 函式會嘗試分階段停止資源。它將首先嘗試使用 SIGTERM 訊號來停止資源，如果失敗，則使用 SIGKILL 訊號。請參閱 scds_pmf_stop(3HA)，以取得詳細資訊。

實施故障監視器

DSDL 透過提供預先決定的模型來瞭解實施故障監視器的大部分複雜性。當資源在某節點上啓動時，`Monitor_start` 方法將在 PMF 的控制下啓動故障監視器。只要資源正在該節點上執行，故障監視器就會在迴圈中執行。DSDL 故障監視器的高階邏輯如下所示。

- `scds_fm_sleep` 函式使用 `Thorough_probe_interval` 特性決定探測之間的間隔時間。在此間隔期間由 PMF 決定的任何應用程式程式故障都將導致重新啓動資源。
- 探測本身將傳回一個值，指示從 `0` (無故障) 到 `100` (完全故障) 的故障程度。
- 探測傳回值被傳送給 `scds_action` 函式，該函式維護在 `Retry_interval` 特性的間隔時間內的累計故障歷史。
- `scds_action` 函式決定在發生故障情況下應執行的作業，如下所示。
  - 如果累計的故障低於 `100`，則不執行任何作業。
  - 如果累計的故障達到 `100` (完全故障)，則重新啓動資料服務。如果超過 `Retry_interval`，則重設歷史。
  - 如果在 `Retry_interval` 指定的時間內，重新啓動的次數超過 `Retry_count` 特性值，則對資料服務進行故障轉移。

存取網路位址資訊

DSDL 提供了方便的函式，以爲資源與資源群組傳回網路位址資訊。例如，`scds_get_netaddr_list` 擷取資源使用的網路位址資源，啓用故障監視器探測應用程式。

DSDL 還提供用於基於 TCP 的監視的函式集。通常這些函式會建立與服務的簡單套接字連接，從服務讀取資料以及將資料寫入服務，然後斷開與服務的連接。探測的結果可發送至 DSDL `scds_fm_action` 函式，以決定要採取的動作。

請參閱第 133 頁的 「`xfnts_validate` 方法」，以取得基於 TCP 的故障監視的範例。

對資源類型實作進行除錯

DSDL 具有可協助您對資料服務除錯的內建功能。

DSDL 公用程式 `scds_syslog_debug()` 提供一個基本框架，用於將除錯敘述加入至資源類型實作。每個叢集節點的每個資源類型實作都可以動態設定除錯**層次** (1-9 之間的數字)。所有資源類型回呼方法均讀取名爲 `/var/cluster/rgm/rt/`*rtname*`/loglevel` 的檔案 (僅包含 1 至 9 之間的一個整數)。DSDL 常式 `scds_initialize()` 將讀取此檔案並將除錯層次內部設定爲指定的層次。預設除錯層次 0 指定資料服務不記錄任何除錯訊息。

`scds_syslog_debug()` 函式使用 `scha_cluster_getlogfacility()` 函式以 `LOG_DEBUG` 優先順序傳回的工具。您可以在 `/etc/syslog.conf` 中配置這些除錯訊息。

您可以將某些除錯訊息轉換爲資訊訊息，用於使用 `scds_syslog` 公用程式的資源類型 (也許以 `LOG_INFO` 優先順序) 之正規作業。如果您查看第 8 章中的 DSDL 應用程式範例，將會注意到它充分利用 `scds_syslog_debug` 函式與 `scds_syslog` 函式。

啓用高度可用的本機檔案系統

您可以使用 `HAStoragePlus` 資源類型來使本機檔案系統在 Sun Cluster 環境內高度可用。本機檔案系統分割區必須位於全域磁碟群組上。必須啓用相互關係切換，並且 Sun Cluster 環境必須配置故障轉移。該設定可讓使用者從任何直接連接至多重主機磁碟的主機上，存取這些多重主機磁碟上的任何檔案系統。對於某些 I/O 密集的資料服務，強烈建議使用高度可用的本機檔案系統。「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」中的「Enabling Highly Available Local File Systems」包含有關配置 `HAStoragePlus` 資源類型的資訊。

第 7 章

設計資源類型

本章說明 DSDL 在設計和實施資源類型方面的一般用法，還重點介紹了設計資源類型以驗證資源配置以及啓動、停止和監視資源。本章最後說明如何使用 DSDL 實施資源類型回呼方法。

請參閱 `rt_callbacks`(1HA) 線上援助頁，以取得其他資訊。

您需要資源的特性設定存取權限來完成這些作業。DSDL 公用程式 `scds_initialize()` 向您提供了一個統一的存取資源特性的方法。此函式被設計爲在每個回呼方法開始時呼叫。此公用程式函式從叢集框架擷取資源的所有特性，並使資源可用於 `scds_get`*name*`()` 函式家族。

本章包含以下主題：

- 第 107 頁的 「RTR 檔案」
- 第 108 頁的 「`Validate` 方法」
- 第 109 頁的 「`Start` 方法」
- 第 110 頁的 「`Stop` 方法」
- 第 111 頁的 「`Monitor_start` 方法」
- 第 111 頁的 「`Monitor_stop` 方法」
- 第 112 頁的 「`Monitor_check` 方法」
- 第 112 頁的 「`Update` 方法」
- 第 113 頁的 「`Init`、`Fini` 和 `Boot` 方法」
- 第 113 頁的 「設計故障監視器常駐程式」

RTR 檔案

資源類型註冊 (RTR) 檔案是資源類型的重要元件。此檔案將關於資源類型的詳細資訊指定給 Sun Cluster。這些詳細資訊包括實作所需的特性、特性的資料類型、特性的預設值、用於資源類型實作的回呼方法之檔案系統路徑，以及系統定義特性的各種設定值。

隨附於 DSDL 的範例 RTR 檔案會滿足大多數資源類型實作的需要。您需要做的就是編輯某些基本元素，例如資源類型名稱和資源類型回呼方法的路徑名稱。如果需要新特性以實施資源類型，您可以宣告它爲資源類型實作的資源類型註冊 (RTR) 檔案的延伸特性，然後使用 DSDL `scds_get_ext_property()` 公用程式存取新特性。

Validate 方法

在下列兩個條件中的其中一個條件下，將由 RGM 呼叫資源類型實施的 `Validate` 方法。

- 資源類型的新資源正在建立中
- 資源或資源群組的特性正在更新中

可以由傳送至資源的 `Validate` 方法的指令行選項爲 `-c` (建立) 還是 `-u` (更新) 來辨別這兩種情形。

可在節點集的每個節點上呼叫 `Validate` 方法，節點集由資源類型特性 `INIT_NODES` 的值來定義。如果 `INIT_NODES` 設定爲 `RG_PRIMARIES`，則會在能夠存放包含資源之資源群組的每個節點上呼叫 `Validate`。如果將 `INIT_NODES` 設定爲 `RT_INSTALLED_NODES` ，則會在安裝資源類型軟體的每個節點上呼叫 `Validate`，通常爲叢集中的所有節點。`INIT_NODES` 的預設值爲 `RG_PRIMARIES` (請參閱 `rt_reg`(4))。呼叫 `Validate` 方法時，RGM 尚未建立資源 (就建立回呼而言)，或尚未套用要更新特性的更新值 (就更新回呼而言)。資源類型實施之 `Validate` 回呼方法的目的，是檢查所建議的資源設定 (由所建議的資源特性設定指定) 對資源類型來說是可接受的。

注意 – 如果您使用的是由 `HAStoragePlus` 管理的本機檔案系統，請使用 `scds_hasp_check` 來檢查 `HAStoragePlus` 資源的狀態。此資訊是使用爲資源定義的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系統特性，從該資源所依賴之所有 `SUNW.HAStoragePlus`(5) 資源的狀態 (線上或其他狀態) 獲取請參閱 `scds_hasp_check`(3HA)，以取得從 `scds_hasp_check` 呼叫傳回之狀態程式碼的完整清單。

DSDL 函式 `scds_initialize()` 以下列模式處理這些情形：

- 就資源建立而言，它剖析指令行上傳送的建議資源特性。這樣，資源特性的建議值即可被資源類型開發者使用，正如資源已在系統中建立。
- 就資源或資源群組更新而言，從指令行讀入所建議的由管理員更新的特性值，並使用資源管理 API 從 Sun Cluster 讀入剩餘特性 (其值未更新)。使用 DSDL 的資源類型開發者無需關心所有這些事務性工作。開發者可以完成資源的驗證，如同資源的所有特性對於他們來說都是可用的。

假定實施資源特性驗證的函式稱爲 `svc_validate()`，此函式使用 `scds_get_`*name*`()` 函式家族在驗證中查看其關注的特性。假定在此函式中用 0 回覆碼表示一個可接受的資源設定，資源類型的 `Validate` 方法便可由下列程式碼分段來表示：

```
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);
}
```

驗證函式也應該記錄資源驗證失敗的原因。如果省略該詳細資訊 (請參閱下一章，以取得關於驗證函式的更實際的處理資訊)，則 `svc_validate()` 函式可以實施爲的一個簡單範例：

```
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 */
}
```

這樣，資源類型開發者必須只關心 `svc_validate()` 函式的實作。典型的資源類型實作範例可以是確保名爲 app.conf 的應用程式配置檔案存在於 `Confdir_list` 特性之下。這可以由源自 `Confdir_list` 特性的適當路徑名稱上的 `stat()` 系統呼叫便利地實施。

Start 方法

資源類型實作的 `Start` 回呼方法由 RGM 在選擇的叢集節點上呼叫以啓動資源。資源群組名稱、資源名稱以及資源類型名稱將在指令行上進行傳送。`Start` 方法預期將執行在叢集節點上啓動資料服務資源所需的動作。通常包括擷取資源特性、尋找應用程式特定的可執行檔和/或配置檔、以及用適當的指令行引數啓動應用程式。

使用 DSDL，資源配置已經由 `scds_initialize()` 公用程式擷取。應用程式的啓動動作可以包含在函式 `svc_start()` 中。可以呼叫另一個函式 `svc_wait()` 來驗證應用程式確實啓動。用於 `Start` 方法的簡化程式碼變爲：

```
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));
}
```

此啓動方法實現將呼叫 `svc_validate()`，以驗證資源配置。如果失敗，則資源配置和應用程式配置不符合，或目前在叢集節點上出現關於系統的問題。例如，資源所需的全域檔案系統目前可能在此叢集節點上不可用。在此情況下，即使嘗試在此叢集節點上啓動資源也是無效的。最好讓 RGM 嘗試在其他節點上啓動資源。但是，請注意以上假定 `svc_validate()` 應足夠地保守 (以便於它只檢查應用程式絕對需要的叢集節點上的資源)，否則資源可能無法在所有的叢集節點上啓動，因而處於 `START_FAILED` 狀態。請參閱 `scswitch`(1M) 與「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」，以取得此狀態的說明。

`svc_start()` 函式必須傳回 0 才表示成功啓動節點上的資源。如果啓動函式遇到問題，則必定傳回非零值。此函式失敗時，RGM 將嘗試在其他叢集節點上啓動資源。

若要盡可能利用 DSDL，`svc_start()` 函式可以使用 `scds_pmf_start()` 公用程式在程序管理設備 (PMF) 下啓動應用程式。此公用程式還利用 PMF 的故障回呼動作功能 (請參閱 `pmfadm`(1M) 中的 `-a` 動作旗號)，以實施程序故障偵測。

Stop 方法

資源類型實現的 `Stop` 回呼方法由 RGM 在叢集節點上呼叫來停止應用程式。`Stop` 方法的回呼語義要求：

- `Stop` 方法必須是**等冪的**，因爲即使 `Start` 方法未在節點上順利完成，RGM 也可以呼叫 `Stop` 方法。這樣，`Stop` 方法必定會成功 (退出零)，即使應用程式目前未在叢集節點上執行，並且無任何工作可做。
- 如果資源類型的 `Stop` 方法在叢集節點上失敗 (退出非零)，正被停止的資源將以 `STOP_FAILED` 狀態結束。依靠資源上的 `Failover_mode` 設定，這可能導致 RGM 強制重新啓動叢集節點。因此，請務必設計 `Stop` 方法以便於應用程式無法終止時，它會盡力確實停止應用程式，甚至突然地強制結束應用程式 (例如，使用 `SIGKILL`)。它還應該確保能夠及時完成此作業，因爲框架將 `Stop_timeout` 的過

期視爲停止失敗，並使資源處於 STOP_FAILED 狀態。

當 DSDL 公用程式 scds_pmf_stop() 第一次嘗試軟式 (透過 SIGTERM) 停止應用程式 (假定它透過 scds_pmf_start() 在 PMF 下啓動)，然後發送 SIGKILL 至程序時，它應該滿足大多數應用程式的需要。請參閱第 178 頁的 「PMF 函式」，以取得關於此公用程式的詳細資訊。

遵循到目前爲止使用的程式碼模型，假定要停止應用程式的應用程式特定函式名爲 svc_stop() (svc_stop() 實作是否使用 scds_pmf_stop() 在此處不考慮，而取決於是否透過 Start 方法在 PMG 下啓動應用程式)，Stop 方法可以實施爲

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

在 Stop 方法實作過程中不使用 svc_validate() 方法，因爲即使系統目前出現問題，Stop 方法也會嘗試停止此節點上的應用程式。

Monitor_start 方法

RGM 呼叫 Monitor_start 方法以啓動資源的故障監視器。故障監視器監視由資源管理的應用程式的運作狀況。資源類型實作通常將故障監視器作爲在後台執行的獨立常駐程式來實施。Monitor_start 回呼方法用於以適當的引數啓動此常駐程式。

由於監視器常駐程式自身易於出現故障 (例如，它可能停止，使應用程式處於未監視狀態)，因此您應該使用 PMF 啓動監視器常駐程式。DSDL 公用程式 scds_pmf_start() 具有用於啓動故障監視器的內建支援。此公用程式使用監視器常駐程式的相對路徑名稱 (相對於用於資源類型回呼方法實作的位置之 RT_basedir)。它使用由 DSDL 管理的 Monitor_retry_interval 和 Monitor_retry_count 延伸特性，以防止常駐程式無限制地重新啓動。雖然 RGM 決不會直接呼叫監視器常駐程式，但是它將爲所有回呼方法定義的相同指令行語法 (即 -R *resource* -G *resource_group* -T *resource_type*) 強加於監視器常駐程式。它允許監視器常駐程式實作自己利用 scds_initialize() 公用程式設定自己的環境。主要的工作在於設計監視器常駐程式自身。

Monitor_stop 方法

RGM 呼叫 Monitor_stop 方法，以停止透過 Monitor_start 方法啓動的故障監視器常駐程式。對此回呼方法失敗的處理方式與對Stop 方法的失敗之處理方式完全相同；因此，Monitor_stop 方法必須與 Stop 方法一樣等冪和可靠。

如果使用 scds_pmf_start() 公用程式來啓動故障監視器公用程式，請使用 scds_pmf_stop() 公用程式來停止它。

Monitor_check 方法

在指定資源的節點上呼叫資源的 Monitor_check 回呼方法，以確定叢集節點是否能夠主控資源 (即資源所管理的應用程式是否能在節點上成功執行？）。通常，這種情形包括確定應用程式需要的所有系統資源確實在叢集節點上可用。與在第 108 頁的「Validate 方法」中說明的一樣，預定開發者實施的函式 svc_validate() 至少能確定這一點。

依靠資源類型實作所管理的特定應用程式，可以寫入 Monitor_check 方法以執行某些額外作業。必須實施 Monitor_check 方法，以便它不會與同時在執行的其他方法發生衝突。對於使用 DSDL 的開發者，建議 Monitor_check 方法利用 svc_validate() 函式，此函式是爲了實施應用程式特定的資源特性驗證而寫入的。

Update 方法

RGM 呼叫資源類型實現的 Update 方法，以套用系統管理員對作用中資源配置所做的所有變更。僅在資源目前在線上的節點 (如果存在) 上呼叫 Update 方法。

由於 RGM 在執行資源類型的 Validate 方法後才執行 Update 方法，因此可以確保對資源配置所做的變更對於資源類型實作來說是可接受的。在資源或資源群組特性變更之前呼叫 Validate 方法，Validate 方法可以禁止所建議的變更。在套用變更之後呼叫 Update 方法，以使作用中 (線上) 的資源可以注意到新的設定。

作爲資源類型開發人員，您需要謹慎地決定要動態更新的特性，並在 RTR 檔案中用 TUNABLE = ANYTIME 設定標示這些特性。通常，您可以指定要動態更新故障監視器常駐程式使用的資源類型實作之任意特性，前提是 Update 方法實作至少重新啓動監視器常駐程式。

可能的候選特性如下：

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

這些特性會影響故障監視器常駐程式檢查服務運作狀態的方式、該常駐程式執行檢查的頻率、該常駐程式用來記錄錯誤的歷史間隔，以及 PMF 設定的重新啓動臨界值。爲了實施這些特性的更新，在 DSDL 中提供了公用程式 scds_pmf_restart()。

如果您需要能夠動態更新資源特性，但是修改特性可能影響執行中的應用程式，則您需要執行適當的動作，以便對該特性的更新可以正確套用至應用程式的任何執行中的實例。目前沒有透過 DSDL 簡化此作業的方法。無法在指令行上將修改的特性傳送至 `Update` (與 `Validate` 相同)。

Init、Fini 和 Boot 方法

這些方法是由資源管理 API 規格定義的**一次性動作**方法。DSDL 所包含的範例實施未對這些方法的使用進行說明。但是，如果資源類型開發者需要這些方法，則 DSDL 中的所有功能對這些方法也是可用的。通常，`Init` 和 `Boot` 方法對於資源類型實作實施**一次性動作**是完全相同的。`Fini` 方法通常執行**還原** `Init` 或 `Boot` 方法的動作之動作。

設計故障監視器常駐程式

使用 DSDL 的資源類型實作通常具有擔負下列職責的故障監視器常駐程式。

- 定期監視被管理的應用程式的運作狀況。監視器常駐程式的這個特定方面嚴格相依於應用程式，並隨資源類型的不同而有很大不同。DSDL 具有某些內建公用程式函式，以執行基於 TCP 的簡單服務運作狀況檢查。執行基於 ASCII 的協定的應用程式 (例如 HTTP、NNTP、IMAP 和 POP3) 可以使用這些公用程式來實施。
- 由使用資源特性 `Retry_interval` 和 `Retry_count` 的應用程式記錄遇到的問題。在應用程式完全失敗時，決定 PMF 動作程序檔是否應該重新啓動服務，或應用程式失敗是否已經累積得如此迅速，以至於可能需要考慮故障轉移。DSDL 公用程式 `scds_fm_action()` 和 `scds_fm_sleep()` 預定會幫助您實施此機制。
- 採取適當的動作 (通常重新啓動應用程式或嘗試對所包含的資源群組進行故障轉移)。DSDL 公用程式 `scds_fm_action()` 實施此類演算法。爲達到此目的，它計算在過去的 Retry_interval 秒內，探測失敗的目前累積量。
- 更新資源狀態，以便 `scstat` 指令與叢集管理 GUI 可以使用應用程式運作狀態。

設計了 DSDL 公用程式，從而故障監視器常駐程式的主迴圈便可以由下列虛擬程式碼來表示。

對於使用 DSDL 來實施的故障監視器：

- 由於透過 PMF 的程序失敗通知非同步，因此透過 `scds_fm_sleep()` 偵測應用程式程序失敗非常迅速。與故障監視器不時喚醒以檢查服務運作狀況並尋找失敗應用程式的情況進行對比。故障偵測時間明顯減少，因此服務可用性增加。
- 如果 RGM 拒絕透過 `scha_control(3HA)` API 轉移服務故障的嘗試，則 `scds_fm_action()` **重設** (遺忘) 其目前的失敗歷史。原因是失敗歷史已經在 `Retry_count` 之上，如果監視器常駐程式在下一次重覆中喚醒，並無法成功完成常

駐程式的運作狀況檢查，它將再次嘗試呼叫 scha_control() 呼叫。由於在上次重覆中導致拒絕的情形仍然有效，因此呼叫可能還會被拒絕。重設歷史將確保故障監視器至少嘗試在下一個重覆中在本機校正此情形 (例如，透過應用程式重新啓動)。

- 一旦重新啓動失敗，scds_fm_action() 將**不**重設應用程式失敗歷史，因爲如果此情形未自行校正，則通常可能很快嘗試 scha_control()。
- 根據失敗歷史，公用程式 scds_fm_action() 會將資源狀況更新至 SCHA_RSSTATUS_OK、SCHA_RSSTATUS_DEGRADED 或 SCHA_RSSTATUS_FAULTED。從而此狀況對於叢集系統管理來說是可用的。

在大多數情況下，應用程式特定的運作狀況檢查動作可以在單獨的獨立式公用程式 (例如，svc_probe()) 中實施，並與此一般主迴圈整合。

```
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 */
```

第 8 章

DSDL 資源類型實施範例

本章說明了藉由 DSDL 實施的資源類型範例 `SUNW.xfnts`。此資料服務寫入 C 。基礎應用程式爲 X 字型伺服器 (基於 TCP/IP 的服務)。

本章包含以下資訊：

- 第 117 頁的 「X 字型伺服器」
- 第 119 頁的 「`SUNW.xfnts` RTR 檔案」
- 第 119 頁的 「`scds_initialize()` 函式」
- 第 120 頁的 「`xfnts_start` 方法」
- 第 124 頁的 「`xfnts_stop` 方法」
- 第 125 頁的 「`xfnts_monitor_start` 方法」
- 第 126 頁的 「`xfnts_monitor_stop` 方法」
- 第 127 頁的 「`xfnts_monitor_check` 方法」
- 第 128 頁的 「`SUNW.xfnts` 故障監視器」
- 第 133 頁的 「`xfnts_validate` 方法」

X 字型伺服器

X 字型伺服器是一種基於 TCP/IP 的簡單服務，可爲用戶端提供字型檔案。用戶端連接至伺服器要求字型集，伺服器便從磁碟中讀取字型檔案，然後將其提供給用戶端。X 字型伺服器常駐程式包含伺服器二進位檔 `/usr/openwin/bin/xfs`。常駐程式一般從 `inetd` 啓動，但對於目前的範例，假定了 `/etc/inetd.conf` 檔案中的相應項目已被停用 (例如，透過 `fsadmin -d` 指令)，因此常駐程式僅受 Sun Cluster 控制。

X 字型伺服器配置檔案

依預設，X 字型伺服器從檔案 `/usr/openwin/lib/X11/fontserver.cfg` 中讀取它的配置資訊。此檔案中的目錄項目包含可用於該常駐程式提供服務的字型目錄清單。叢集管理員可以在全域檔案系統上找到字型目錄 (以透過維護該系統上字型資料庫的單一複本，來最佳化 X 字型伺服器在 Sun Cluster 上的使用)。如果如此，管理員必須編輯 `fontserver.cfg`，才能反映字型目錄的新路徑。

爲便於配置，管理員也可以將配置檔案本身放置在全域檔案系統上。`xfs` 常駐程式提供指令行引數來置換此檔案的預設內建位置。`SUNW.xfnts` 資源類型使用以下指令來啓動 Sun Cluster 所控制的常駐程式。

```
/usr/openwin/bin/xfs -config <location_of_cfg_file>/fontserver.cfg \
-port <portnumber>
```

在 `SUNW.xfnts` 資源類型實作中，您可以使用 `Confdir_list` 特性來管理 `fontserver.cfg` 配置檔案的位置。

TCP 通訊埠編號

`xfs` 伺服器常駐程式偵聽的 TCP 通訊埠編號一般爲「fs」通訊埠 (在 `/etc/services` 檔案中通常被定義爲 `7100`)。不過，`xfs` 指令行上的 `-port` 選項可讓系統管理員置換預設設定值。您可以使用 `SUNW.xfnts` 資源類型中的 `Port_list` 特性來設定預設值和支援在 `xfs` 指令行上使用 `-port` 選項。在 RTR 檔案中，此特性的預設值定義爲 `7100/tcp`。在 `SUNW.xfnts` `Start` 方法中，您會將 `Port_list` 傳送至 `xfs` 指令行上的 `-port` 選項。因此，不需要此資源類型的使用者指定埠號碼 — 埠預設爲 `7100/tcp` — 但如果使用者願意，在配置資源類型時，他們完全可以透過爲 `Port_list` 特性指定其他值來指定其他埠。

命名慣例

記住下列慣例，便可以識別各種範例程式碼片段。

- RMAPI 函式以 `scha_` 開始。
- DSDL 函式以 `scds_` 開始。
- 回呼方法以 `xfnts_` 開始。
- 使用者寫入的函式以 `svc_` 開始。

SUNW.xfnts RTR 檔案

本節說明了 SUNW.xfnts RTR 檔案中的幾個主要特性。此節未說明該檔案中每個特性的用途。如需此類說明，請參閱第 29 頁的 「設定資源特性和資源類型特性」。

Confdir_list 延伸特性識別配置目錄 (或目錄清單)，如下所示。

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

Confdir_list 特性不指定預設值。叢集管理員必須在建立資源時指定目錄。由於可調性受限於 AT_CREATION，因此以後無法變更該值。

Port_list 特性識別伺服器常駐程式偵聽的通訊埠，如下所示。

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

由於該特性宣告了預設值，因此，叢集管理員可以在建立資源時指定新值或接受此預設值。由於可調性受限於 AT_CREATION，因此以後無法變更該值。

scds_initialize() 函式

DSDL 要求每種回呼方法在其開始時均呼叫 scds_initialize(3HA) 函式。此函式執行下列作業：

- 檢查並處理框架傳送至資料服務方法的指令行引數 (argc 與 argv)。此方法無須對這些指令行引數進行任何額外的處理。
- 設定內部資料結構，以供 DSDL 中的其他函式使用。
- 初始化記錄環境。
- 驗證故障監視器的探測設定。

使用 scds_close() 函式，可回收 scds_initialize() 分配的資源。

xfnts_start 方法

當包含資料服務資源的資源群組在叢集節點上處於線上運作狀態或當啓用該資源時，RGM 將在該節點上呼叫 Start 方法。在 SUNW.xfnts 範例資源類型中，xfnts_start 方法可在該節點上啓動 xfs 常駐程式。

xfnts_start 方法呼叫 scds_pmf_start()，以在 PMF 下啓動此常駐程式。PMF 提供了自動故障通知與重新啓動功能，以及與故障監視器的整合。

注意 – xfnts_start 的第一次呼叫是呼叫 scds_initialize()，它執行某些必需的**事務性工作**功能 (第 119 頁的 「scds_initialize() 函式」與 scds_initialize(3HA) 線上援助頁包含更多詳細資訊)。

啓動前驗證服務

在嘗試啓動 X 字型伺服器之前，xfnts_start 方法將呼叫 svc_validate()，以確認正確的配置已就緒來支援 xfs 常駐程式 (請參閱第 133 頁的 「xfnts_validate 方法」以取得詳細資訊)，如下所示。

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

啓動服務

xfnts_start 方法呼叫在 xfnts.c 中定義的 svc_start() 方法來啓動 xfs 常駐程式。本節說明了 svc_start ()。

用來啓動 xfs 常駐程式的指令如下所述。

```
xfs -config config_directory/fontserver.cfg -port port_number
```

Confdir_list 延伸特性識別 *config_directory*，而 Port_list 系統特性識別 *port_number*。當叢集管理員配置資料服務時，他會爲這些特性提供特定值。

xfnts_start 方法將這些特性宣告爲字串陣列，並使用 scds_get_ext_confdir_list() 與 scds_get_port_list() 函式 (在 scds_property_functions(3HA) 中有說明) 取得管理員設定的值，如下所示。

```
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);
   }
```

請注意，confdirs 變數指陣列的第一個元素 (0)。

xfnts_start 方法使用 sprintf 來產生 xfs 的指令行，如下所示。

```
/* 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);
```

請注意，他的輸出將重新導向至 dev/null，來抑制此常駐程式產生的訊息。

xfnts_start 方法將 xfs 指令行傳送至 scds_pmf_start()，以啓動 PMF 控制下的資料服務，如下所示。

```
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 ");
   }
```

請注意關於 scds_pmf_start() 呼叫的以下內容。

- SCDS_PMF_TYPE_SVC 參數將要啓動的程式識別爲資料服務應用程式 — 此方法也可以啓動故障監視器或某些其他類型的應用程式。
- SCDS_PMF_SINGLE_INSTANCE 參數將其識別爲單一實例資源。
- cmd 參數是先前產生的指令行。
- 最終參數 -1 指定子監視層次。-1 指定 PMF 監視所有子程序及原始程序。

在傳回前，svc_pmf_start() 將釋放爲 portlist 結構分配的記憶體空間，如下所示。

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

從 svc_start() 傳回

即使 svc_start() 成功傳回，基礎應用程式也可能無法啓動。因此，svc_start() 必須探測應用程式，以驗證該應用程式在傳回成功訊息之前已在執行。此探測也必須考慮此應用程式可能不會立即可用，因爲它要花些時間來啓動。svc_start() 方法將呼叫 svc_wait() (在 xfnts.c 中定義)，以確認此應用程式正在執行，如下所示。

```
/* 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() 函式呼叫 scds_get_netaddr_list(3HA)，來取得探測此應用程式所需的網路位址資源，如下所示。

```
/* 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);
   }
```

然後，svc_wait() 便可取得 start_timeout 與 stop_timeout 值，如下所示。

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

若要說明伺服器可能需要的啓動時間，svc_wait() 將呼叫 scds_svc_wait()，並傳送相當於 start_timeout 值 3% 的逾時值。然後，svc_wait() 將呼叫 svc_probe()，以確認此應用程式已經啓動。svc_probe() 方法在指定通訊埠上建立與伺服器的簡單套接字連接。如果無法連接至該通訊埠，svc_probe() 將傳回值 100，指示發生完全故障。如果連接成功，但與該通訊埠斷開連接失敗，則 svc_probe() 將傳回值 50。

svc_probe() 失敗或部分失敗時，svc_wait() 將以逾時值 5 呼叫 scds_svc_wait()。scds_svc_wait() 方法會將探測頻率限定爲每五秒一次。此方法也將計算嘗試啓動服務的次數。如果嘗試次數在資源的 Retry_interval 特性所指定的時間內，超出該資源 Retry_count 特性的值，則 scds_svc_wait() 函式將傳回故障。在此情況下，svc_start() 函式也將傳回故障。

```
#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);
```

注意 – 在結束之前，xfnts_start 方法將呼叫 scds_close() 以回收 scds_initialize() 分配的資源。請參閱第 119 頁的 「scds_initialize() 函式」與 scds_close(3HA) 線上援助頁，以取得詳細資訊。

---

xfnts_stop 方法

由於 xfnts_start 方法使用 scds_pmf_start() 來啓動 PMF 下的服務，因此 xfnts_stop 使用 scds_pmf_stop() 來停止此服務。

注意 – xfnts_stop 的第一次呼叫是呼叫 scds_initialize()，它執行某些必需的**事務性工作**功能 (第 119 頁的 「scds_initialize() 函式」與 scds_initialize(3HA) 線上援助頁包含更多詳細資訊)。

---

xfnts_stop 方法呼叫 svc_stop() 方法 (在 xfnts.c 中定義)，如下所示。

```
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 */
```

請注意下列關於 svc_stop() 呼叫 scds_pmf_stop() 函式的內容。

- SCDS_PMF_TYPE_SVC 參數將要停止的程式識別爲資料服務應用程式 — 此方法也可以停止故障監視器或其他類型的應用程式。
- SCDS_PMF_SINGLE_INSTANCE 參數識別訊號。
- SIGTERM 參數識別要用來停止資源實例的訊號。如果此訊號無法停止實例，scds_pmf_stop() 將發送 SIGKILL 來停止該實例，如果失敗，它將傳回逾時錯誤。請參閱 scds_pmf_stop(3HA) 線上援助頁以取得詳細資訊。
- 逾時值爲資源之 Stop_timeout 特性的值。

注意 – 在結束之前，xfnts_stop 方法將呼叫 scds_close() 以回收 scds_initialize() 分配的資源。請參閱第 119 頁的 「scds_initialize() 函式」與 scds_close(3HA) 線上援助頁，以取得詳細資訊。

---

xfnts_monitor_start 方法

在一個節點上啓動資源後，RGM 將在該節點上呼叫 Monitor_start 方法以啓動故障監視器。xfnts_monitor_start 方法使用 scds_pmf_start() 來啓動 PMF 下的監視器常駐程式。

注意 – xfnts_monitor_start 的第一次呼叫是呼叫 scds_initialize()，它執行某些必需的**事務性工作**功能 (第 119 頁的 「scds_initialize() 函式」與 scds_initialize(3HA) 線上援助頁包含更多詳細資訊)。

---

xfnts_monitor_start 方法呼叫 mon_start 方法 (在 xfnts.c 中定義)，如下所示。

```
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 */
}
```

請注意下列關於 svc_mon_start() 呼叫 scds_pmf_start() 函式的內容。

- SCDS_PMF_TYPE_MON 參數將要啓動的程式識別爲故障監視器 — 此方法也可以啓動資料服務或某些其他類型的應用程式。
- SCDS_PMF_SINGLE_INSTANCE 參數將其識別爲單一實例資源。
- xfnts_probe 參數識別要啓動的監視器常駐程式。假定爲監視器常駐程式與其他回呼程式位於同一目錄中。
- 最終參數 0 指定子監視層次 — 在此情況下，僅監視此監視器常駐程式。

注意 – 在結束之前，xfnts_monitor_start 方法將呼叫 scds_close() 以回收 scds_initialize() 分配的資源。請參閱第 119 頁的 「scds_initialize() 函式」與 scds_close(3HA) 線上說明手冊，以取得詳細資訊。

xfnts_monitor_stop 方法

由於 xfnts_monitor_start 方法使用 scds_pmf_start() 來啓動 PMF 下的監視器常駐程式，因此 xfnts_monitor_stop 使用 scds_pmf_stop() 來停止此監視器常駐程式。

注意 – xfnts_monitor_stop 的第一次呼叫是呼叫 scds_initialize()，它執行某些必需的**事務性工作**功能 (第 119 頁的 「scds_initialize() 函式」與 scds_initialize(3HA) 線上援助頁包含更多詳細資訊)。

xfnts_monitor_stop() 方法呼叫 mon_stop 方法 (在 xfnts.c 中定義)，如下所示。

```
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 */
}
```

請注意下列關於 svc_mon_stop() 呼叫 scds_pmf_stop() 函式的內容。

- SCDS_PMF_TYPE_MON 參數將要停止的程式識別爲故障監視器 — 此方法也可以停止資料服務或某些其他類型的應用程式。
- SCDS_PMF_SINGLE_INSTANCE 參數將其識別爲單一實例資源。

- SIGKILL 參數識別用來停止資源實例的訊號。如果此訊號無法停止實例，scds_pmf_stop() 將傳回逾時錯誤。請參閱 scds_pmf_stop(3HA)) 線上援助頁以取得詳細資訊。
- 逾時值爲資源之 Monitor_stop_timeout 特性的值。

注意 – 在結束之前，xfnts_monitor_stop 方法將呼叫scds_close() 以回收 scds_initialize() 分配的資源。請參閱第 119 頁的「scds_initialize() 函式」與 scds_close(3HA) 線上說明手冊，以取得詳細資訊。

---

xfnts_monitor_check 方法

每當故障監視器嘗試將包含資源的資源群組之故障轉移至其他節點時，RGM 均將呼叫 Monitor_check 方法。xfnts_monitor_check 方法呼叫 svc_validate() 方法，以確認正確配置已就緒可支援 xfs 常駐程式 (請參閱第 133 頁的「xfnts_validate 方法」以取得詳細資訊)。xfnts_monitor_check 的程式碼如下所示。

```
   /* 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 故障監視器

在一個節點上啓動資源後，RGM 將不直接呼叫 PROBE 方法，而是呼叫 Monitor_start 方法來啓動監視器。xfnts_monitor_start 方法啓動 PMF 控制下的故障監視器。xfnts_monitor_stop 方法停止該故障監視器。

SUNW.xfnts 故障監視器執行下列作業：

- 使用專門用來檢查基於 TCP 之簡單服務 (如 xfs) 的公用程式，定期監視 xfs 伺服器常駐程式的運作狀況。
- 在時間視窗內追蹤應用程式遇到的問題 (使用 Retry_count 與 Retry_interval 特性)，並決定當應用程式發生完全故障時，重新啓動資料服務還是對其進行故障轉移。scds_fm_action() 與 scds_fm_sleep() 函式爲此追蹤及決定機制提供內建支援。
- 使用 scds_fm_action() 來實施故障轉移或重新啓動的決定。
- 更新資源狀態，並使其可用於管理工具及圖形使用者介面。

xfonts_probe 主迴圈

xfonts_probe 方法實施迴圈。在實施迴圈之前，xfonts_probe

- 將爲 xfnts 資源擷取網路位址資源，如下所示。

```
/* 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);
   }
```

- 呼叫 scds_fm_sleep()，並將 Thorough_probe_interval 的值作爲逾時值傳送。對於探測之間的 Thorough_probe_interval 值，探測將靜止。

```
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 方法實施迴圈，如下所示。

```
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() 函式實施探測邏輯。來自 svc_probe() 的傳回值被傳送至 scds_fm_action()，該值將決定是否重新啓動應用程式，對資源群組進行故障轉移，或者不執行任何作業。

svc_probe() 函式

svc_probe() 函式透過呼叫scds_fm_tcp_connect()，建立與指定通訊埠的簡單套接字連接。如果連接失敗，svc_probe() 將傳回值 100，指示發生完全故障。如果連接成功但斷開連接失敗，svc_probe() 將傳回值 50，指示發生部分故障。如果連接與斷開連接均成功，svc_probe() 將傳回值 0，指示成功。

svc_probe() 的程式碼如下所示。

```
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);
}
```

完成後，`svc_probe()` 將傳回成功值 (`0`)、部分故障值 (`50`) 或完全故障值 (`100`)。`xfnts_probe` 方法將此值傳送至 `scds_fm_action()`。

決定故障監視器的動作

`xfnts_probe` 方法呼叫 `scds_fm_action()` 來決定要執行的動作。`scds_fm_action()` 的邏輯如下所示：

- 在 `Retry_interval` 特性值範圍內，保留累計故障歷史。
- 如果累計故障達到 100 (完全故障)，則重新啓動資料服務。如果超過 `Retry_interval`，則重設歷史。
- 如果在 `Retry_interval` 指定的時間內，重新啓動的次數超過 `Retry_count` 特性值，則對資料服務進行故障轉移。

例如，假定探測連接至 xfs 伺服器，但無法斷開連接。這指示該伺服器正在執行，但可能掛起或恰好在進行暫時載入。斷開連接的故障將部分 (50) 故障發送至 scds_fm_action()。此值低於重新啓動資料服務的臨界值，但該值保留在故障歷史中。

如果在下一次探測期間，伺服器仍無法結束連線，則值 50 將被新增至 scds_fm_action() 維護的故障歷史中。累計故障值現在爲 100，因此 scds_fm_action() 將重新啓動資料服務。

xfnts_validate 方法

當建立資源時，以及當管理動作更新該資源或其包含群組的特性時，RGM 將呼叫 Validate 方法。在套用建立或更新之前，RGM 將呼叫 Validate，並且在任何節點上此方法的故障退出碼將導致取消建立或更新。

僅當透過管理動作變更資源或群組特性時，RGM 才會呼叫 Validate，而在 RGM 設定特性或監視器設定資源特性 Status 與 Status_msg 時，RGM 不會呼叫該方法。

注意 – 每當 PROBE 方法嘗試將資料服務之故障轉移至新節點時，Monitor_check 方法都將明確地呼叫 Validate 方法。

RGM 藉由傳送至其他方法的引數以及其他引數 (包括正在更新的特性與值)，來呼叫 Validate。xfnts_validate 開始時呼叫 scds_initialize() 會剖析 RGM 傳送至 xfnts_validate 的所有引數，並在 scds_handle 參數中儲存資訊。xfnts_validate 呼叫的子常式將利用此資訊。

xfnts_validate 方法呼叫可確認下列內容的 svc_validate()。

- 已經爲資源設定了 Confdir_list 特性，該特性定義單一目錄。

```
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_list 指定的目錄包含 fontserver.cfg 檔案。

```
(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);
   }
```

- 在叢集節點上可以存取伺服器常駐程式二進位檔。

```
if (stat("/usr/openwin/bin/xfs", &statbuf) != 0) {
      scds_syslog(LOG_ERR,
          "Cannot access XFS binary : <%s> ", strerror(errno));
      return (1);
   }
```

- Port_list 特性指定單一通訊埠。

```
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
```

- 包含資料服務的資源群組至少還包含一個網路位址資源。

```
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;
}
```

在傳回之前，svc_validate() 將釋放所有分配的資源。

```
finished:
    scds_free_net_list(snrlp);
    scds_free_port_list(portlist);

    return (rc); /* return result of validation */
```

注意 – 在結束之前，xfnts_validate 方法將呼叫 scds_close() 以回收 scds_initialize() 分配的資源。請參閱第 119 頁的「scds_initialize() 函式」與 scds_close(3HA) 線上說明手冊，以取得詳細資訊。

---

xfnts_update 方法

RGM 呼叫 Update 方法來通知執行中資源其特性已經變更。可以爲 xfnts 資料服務變更的唯一特性關係到故障監視器。因此，每當變更特性時，xfnts_update 方法都將呼叫 scds_pmf_restart_fm() 來重新啓動故障監視器。

```
* 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.");
```

注意 – `scds_pmf_restart_fm()` 的第二個參數唯一識別要重新啓動的故障監視器實例 (如果有多個實例)。範例中的值 `0` 指示僅有一個故障監視器實例。

第 9 章

SunPlex Agent Builder

本章描述 SunPlex Agent Builder 以及用於 Agent Builder 的 Cluster Agent 模組，它們是自動化資源類型或資料服務建立的工具，使之可以在資源群組管理員 (RGM) 的控制下運行。資源類型是圍繞應用程式的包裝，使該應用程式在 RGM 控制下的叢集環境中運行。

本章說明下列主題：

- 第 137 頁的 「Agent Builder 簡介」
- 第 138 頁的 「使用 Agent Builder 之前」
- 第 139 頁的 「使用 Agent Builder」
- 第 152 頁的 「目錄結構」
- 第 152 頁的 「Agent Builder 輸出」
- 第 156 頁的 「Agent Builder 的 Cluster Agent 模組」

Agent Builder 簡介

Agent Builder 將提供一個基於螢幕的介面，用於輸入關於應用程式以及要建立之資源類型種類的資訊。

注意 – 如果無法存取 Agent Builder 的圖形化使用者介面版本，您可以透過指令行介面存取 Agent Builder。請參閱第 151 頁的 「如何使用 Agent Builder 的指令行版本」。

基於所輸入的資訊，Agent Builder 將產生以下軟體：

- 與資源類型的方法回呼對應的防故障備用或可延伸資源類型的 C shell、Korn shell (`ksh`) 或通用資料服務 (GDS) 源代碼檔案集，其適用於網路支援 (用戶端伺服器模型) 與非網路支援 (無用戶端) 應用程式
- 自訂資源類型註冊 (RTR) 檔案 (如果您產生 C shell 或 Korn shell 源代碼)

- 自訂公用程式程序檔與自訂線上說明手冊，前者用於啓動、停止以及移除資源類型的實例 (資源)，後者歸檔如何使用其中每個檔案
- Solaris 套裝軟體，包含二進位檔案 (如果您產生 C 源代碼)、RTR 檔案 (如果您產生 C shell 或 Korn shell 源代碼) 以及公用程式程序檔

Agent Builder 支援網路支援應用程式 (使用網路與用戶端進行通訊的應用程式) 以及非網路支援 (或獨立式) 應用程式。Agent Builder 也可讓您產生應用程式的資源類型，該資源類型具有程序監視設備 (PMF) 必須個別監視與重新啓動的多個獨立程序樹。請參閱第 138 頁的 「建立具有多個獨立程序樹的資源類型」。

使用 Agent Builder 之前

下節提供在使用 Agent Builder 之前需要瞭解的資訊。

建立具有多個獨立程序樹的資源類型

Agent Builder 可以建立具有多個獨立程序樹的應用程式資源類型。在 PMF 個別監視與啓動這些程序樹來看，它們是獨立的。PMF 透過每個程序樹自己的標籤來啓動它。

注意 – 僅當您指定的已產生的源代碼爲 C 或 GDS 時，Agent Builder 才可讓您建立具有多個獨立程序樹的資源類型。您無法使用 Agent Builder 爲 Korn shell 建立這些資源類型。若要爲 Korn shell 建立這些資源類型，您必須手動寫入程式碼。

至於具有多個獨立程序樹的基本應用程式，您無法指定一個單一指令行來啓動該應用程式。相反，您必須建立一個文字檔案，透過指定指令完整路徑的每一行，來啓動其中一個應用程式的程序樹。該檔案不得包含任何空白行。您在 [配置] 螢幕的 [Start 指令] 文字欄位中指定該文字檔。

確保此檔案沒有執行權限可讓 Agent Builder 辨別此檔案，其目的是從包含多個指令的簡單可執行程序檔啓動多個程序樹。如果爲該文字檔指定了執行權限，則叢集上的資源不會有任何問題或錯誤，但所有的指令會在一個PMF 標記下啓動，這會使 PMF 無法個別監視與重新啓動程序樹。

使用 Agent Builder

本節描述如何使用 Agent Builder，包括您可以使用 Agent Builder 之前必須完成的作業。本節還闡述了您在產生資源類型程式碼後可以利用 Agent Builder 的方法。

本章討論下列主題：

- 第 139 頁的 「分析應用程式」
- 第 139 頁的 「安裝與配置 Agent Builder」
- 第 140 頁的 「Agent Builder 螢幕」
- 第 140 頁的 「啓動 Agent Builder」
- 第 141 頁的 「導覽 Agent Builder」
- 第 144 頁的 「使用 [建立] 畫面」
- 第 146 頁的 「使用 [配置] 畫面」
- 第 148 頁的 「使用基於 Agent Builder Korn Shell 的變數 `$hostnames`」
- 第 148 頁的 「特性變數」
- 第 150 頁的 「如何複製現有的資源類型」
- 第 151 頁的 「編輯產生的來源代碼」
- 第 151 頁的 「如何使用 Agent Builder 的指令行版本」

分析應用程式

在使用 Agent Builder 之前，您必須決定您的應用程式是否符合可使之高度可用或可延伸的條件。Agent Builder 無法執行這個僅基於應用程式執行時間特性的分析。第 25 頁的 「分析應用程式的適當性」提供有關此主題的詳細資訊。

雖然在大多數情況下，Agent Builder 至少會提供一個部分解決方案，但它不可能總是能夠建立應用程式的完整資源類型。例如，更複雜的應用程式可能需要 Agent Builder 依預設不會產生的其他程式碼，如對其他特性加入驗證檢查的程式碼或調諧 Agent Builder 不接觸的參數之程式碼。在此類情況下，您必須對產生的來源代碼或 RTR 檔案進行變更。Agent Builder 就恰好是爲提供此種靈活性而設計的。

Agent Builder 會將註釋置於已產生源代碼中的特定位置上，您可以在這些地方新增自己的特定資源類型程式碼。對源代碼進行變更後，您可以使用 Agent Builder 產生的 `makefile` 重新編譯源代碼以及重新產生資源類型套裝軟體。

即使您寫入整個資源類型程式碼，而未使用由 Agent Builder 產生的任何程式碼，也可以使用 Agent Builder 提供的 `makefile` 與結構來建立資源類型的 Solaris 套裝軟體。

安裝與配置 Agent Builder

Agent Builder 不需要特殊安裝。Agent Builder 包含於 `SUNWscdev` 套裝軟體中，依預設，它作爲標準 Sun Cluster 軟體安裝的一部分來安裝。「*Sun Cluster* **軟體安裝指南 (適用於** *Solaris* **作業系統)**」包含更多資訊。

在使用 Agent Builder 之前，請驗證下列資訊：

- Java 運行時間環境包含於變數 `$PATH` 中。Agent Builder 依賴於 Java (Java 開發工具 1.3.1 版或更高版本)。如果 Java 未包含於 `$PATH` 中，`scdsbuilder` 則會傳回並顯示一則錯誤訊息。
- 您已安裝了 Solaris 8 或更高版本的開發者系統支援軟體群組。
- `cc` 編譯器包含於變數 `$PATH` 中。Agent Builder 使用變數 `$PATH` 中的第一個 `cc` 事件，識別用來產生資源類型之 C 二進位碼的編譯器。如果 `cc` 未包含於 `$PATH` 中，Agent Builder 將停用產生 C 程式碼的選項。請參閱第 144 頁的 「使用 [建立] 畫面」。

注意 – 除了標準 `cc` 編譯程式，您可以將其他編譯程式與 Agent Builder 搭配使用。若要使用其他編譯器，請在 `$PATH` 中建立一個從 `cc` 至其他編譯器 (如 `gcc`) 的符號連結。或者，將 `makefile` 中的編譯器規格 (目前爲 `CC=cc`) 變更爲其他編譯器的完整路徑。例如，在 Agent Builder 產生的 `makefile` 中，將 `CC=cc` 變更爲 `CC=`**路徑名稱**`/gcc`。在這種情況下，您無法直接運行 Agent Builder，而是必須使用 `make` 與 `make pkg` 指令來產生資料服務程式碼與套裝軟體。

Agent Builder 螢幕

Agent Builder 是含有兩個步驟的精靈，每個步驟對應一個螢幕。Agent Builder 將提供下列兩個螢幕，指導您完成建立新資源類型的程序。

1. **建立**。在此螢幕上，您要提供關於要建立的資源類型之基本資訊，例如其名稱以及所產生檔案的工作目錄。該工作目錄是您建立和配置資源類型範本的位置。您也要識別要建立的資源種類 (可延伸或防故障備用)、基本應用程式是否爲網路支援應用程式 (即是否使用網路與其用戶端進行通訊)，以及要產生的程式碼類型 (C、Korn shell [`ksh`] 或 GDS)。如需關於 GDS 的資訊，請參閱第 10 章。您必須提供關於此螢幕的所有資訊並選取 [建立] 以產生相應的輸出。然後，您可以顯示 [配置] 螢幕。
2. **配置**。在此螢幕上，您必須指定可以傳送至任何 UNIX® shell 以啓動基本應用程式的完整指令行。此外，您可以提供停止與測試應用程式的指令。如果您不指定這些指令，則產生的輸出將使用訊號來停止應用程式，並提供預設測試機制。請參閱第 146 頁的 「使用 [配置] 畫面」中對測試指令的描述。此畫面還可讓您變更這三個指令中每一個的逾時值。

啓動 Agent Builder

注意 – 如果無法存取 Agent Builder 的圖形化使用者介面版本，您可以透過指令行介面存取 Agent Builder。請參閱第 151 頁的 「如何使用 Agent Builder 的指令行版本」。

注意 – 如果您從現有資源類型的工作目錄啓動 Agent Builder，Agent Builder 會將 [建立] 畫面與 [配置] 畫面初始化爲現有資源類型的值。

透過鍵入下列指令來啓動 Agent Builder：

```
% /usr/cluster/bin/scdsbuilder
```

將顯示 [建立] 螢幕。

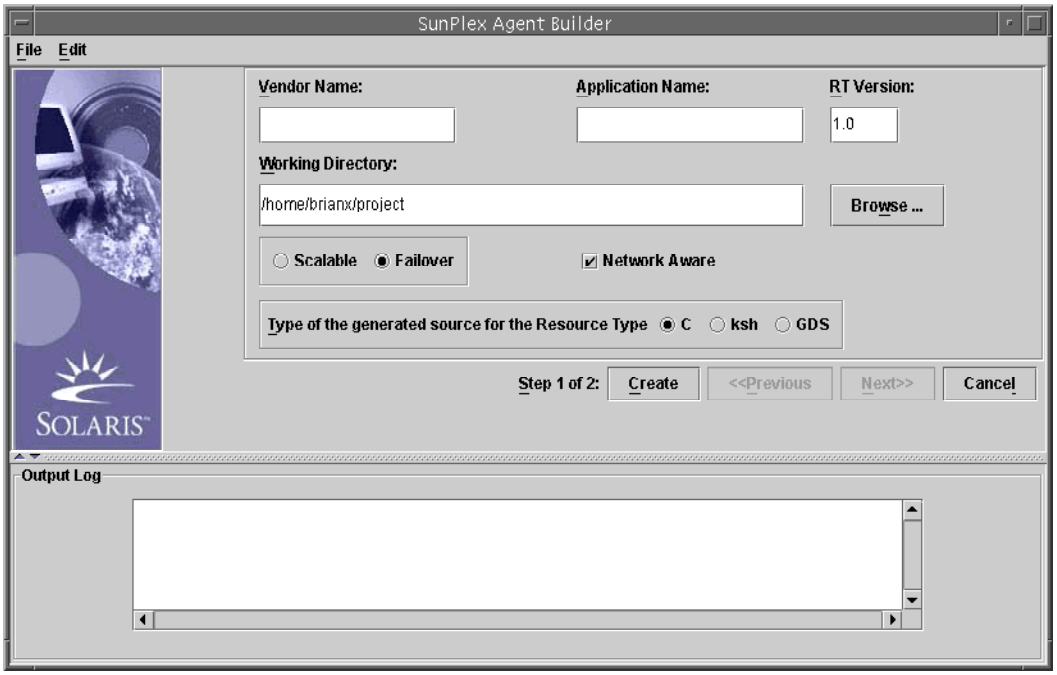


圖 9–1 [建立] 畫面

導覽 Agent Builder

透過執行下列作業，輸入關於 [建立] 和 [配置] 螢幕的資訊：

- 在欄位中鍵入資訊。
- 瀏覽目錄結構並選取檔案或目錄。
- 選取一組互斥單選按鈕中的一個，例如，按一下[可延伸] 或 [防故障備用]。
- 按一下 [開啓] 或 [關閉] 方塊。例如，按一下 [網路支援] 將基本應用程式識別爲網路支援的應用程式，而保留此方塊爲空白識別非網路支援的應用程式。

使用每個螢幕底部的選項可讓您完成作業、移至下一個螢幕或上一個螢幕，或結束 Agent Builder。Agent Builder 透過適當地反白顯示或灰色顯示這些選項來突出顯示。

例如，當您填入了 [建立] 螢幕上的欄位並核取了所需的選項後，請按一下螢幕底部的 [建立]。由於不存在上一個螢幕，並且您在完成此步驟之前無法移至下一步，因此 [上一個] 與 [下一個] 會呈灰色顯示。

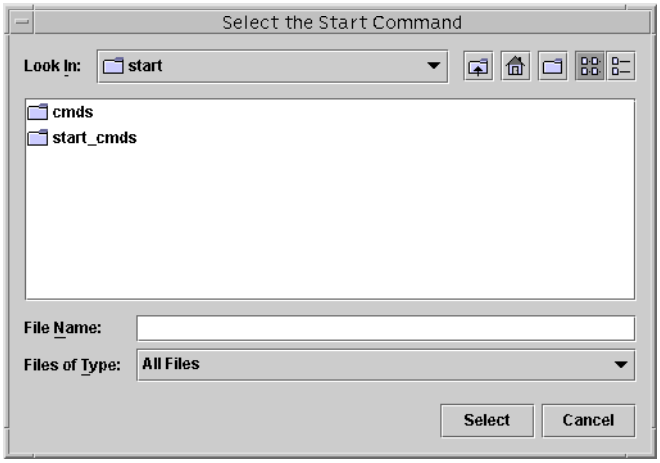
Agent Builder 在畫面底部的輸出登錄區域中顯示進度訊息。當 Agent Builder 完成作業時，它將顯示一則成功訊息，或者顯示一則警告，以便查看輸出日誌。將反白顯示 [下一個]，或者如果這是最後一個螢幕，則僅反白顯示 [取消]。

您可以隨時選取 [取消] 以結束 Agent Builder。

瀏覽

特定的 Agent Builder 欄位可讓您鍵入資訊或按一下 [瀏覽]，以瀏覽目錄結構並選取檔案或目錄。

當您按一下 [瀏覽] 時，將會顯示一個與下列螢幕相似的螢幕：



按兩下資料夾以開啓它。當您將游標移至某個檔案時，該檔案的名稱將顯示在 [檔案名稱] 方塊中。當您已找到所需的檔案並將游標移至該檔案時，請按一下 [選取]。

注意 – 如果您要瀏覽某個目錄，請將游標移至所需的目錄並按一下 [開啓]。如果沒有子目錄，Agent Builder 將關閉瀏覽視窗，並將游標所移至的目錄之名稱置於正確的欄位中。如果該目錄具有子目錄，請按一下 [關閉] 以關閉瀏覽視窗，並重新顯示上一個螢幕。Agent Builder 會將游標所移至的目錄之名稱置於正確的欄位中。

畫面右上角的圖示將執行以下作業：

此圖示將使您在目錄樹中上移一層。

此圖示將使您返回主資料夾。

此圖示將在目前選取的資料夾下建立一個新資料夾。

該圖示用於在不同檢視之間進行切換，且被保留以供將來使用。

功能表

Agent Builder 提供 [檔案] 和 [編輯] 下拉式功能表。

[檔案] 功能表

[檔案] 功能表包含兩個選項：

- **載入資源類型**。載入現有資源類型。Agent Builder 提供一個瀏覽畫面，從該畫面您可以選取現有資源類型的工作目錄。如果您啓動 Agent Builder 的目錄中存在資源類型，Agent Builder 將會自動載入該資源類型。[載入資源類型] 可讓您從任何目錄啓動 Agent Builder，並選取現有資源類型，作爲建立新資源類型的範本。請參閱第 150 頁的 「如何複製現有的資源類型」。
- **結束**。結束 Agent Builder。您也可以透過按一下 [建立] 或 [配置] 螢幕上的 [取消] 來結束。

[編輯] 功能表

[編輯] 功能表包含下列選項：

- **清除輸出日誌**。從輸出日誌中清除資訊。每當您選取 [建立] 畫面或 [配置] 畫面時，Agent Builder 均會將狀況訊息附加至輸出登錄。如果您正在進行反覆程序 (該程序可用於對來源代碼進行變更並在 Agent Builder 中重新產生輸出) 並要分離狀況訊息，您可以在每次使用日誌檔前儲存與清除日誌檔。
- **儲存日誌檔**。將日誌輸出儲存至檔案。Agent Builder 提供一個可讓您選擇目錄與指定檔案名稱的瀏覽螢幕。

使用 [建立] 畫面

[建立] 畫面

建立資源類型的第一步是填寫 [建立] 畫面，此畫面在您啓動 Agent Builder 時顯示。下圖將說明您在欄位中輸入資訊之後的 [建立] 畫面。

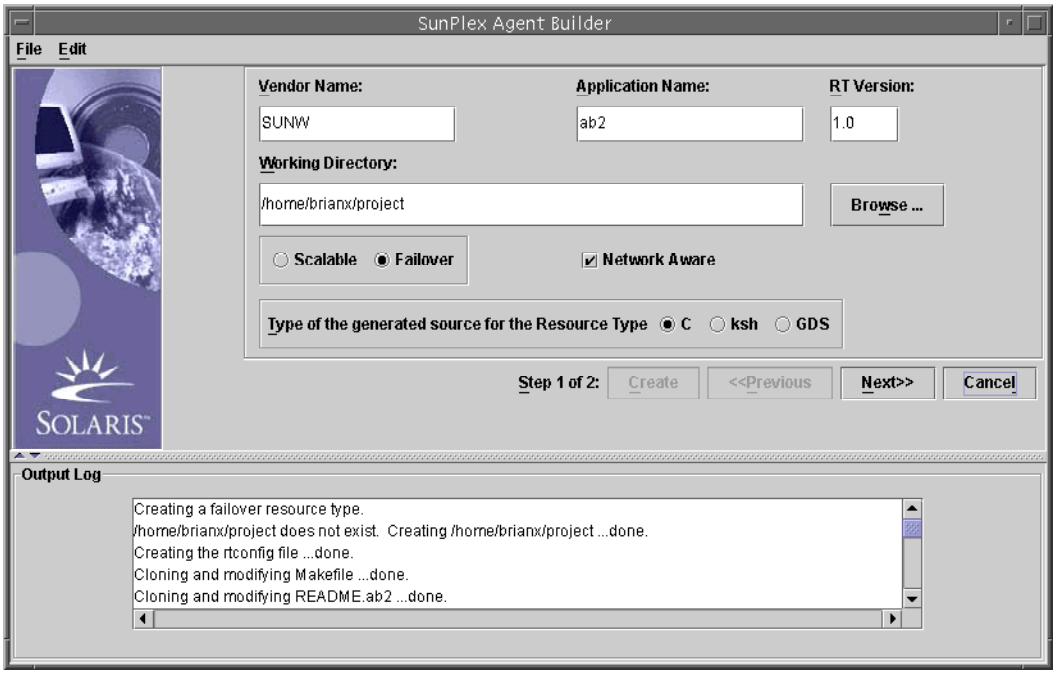


圖 9–2 [建立] 畫面

[建立] 畫面包含以下欄位、單選按鈕以及核取方塊：

- **供應商名稱**。用於識別資源類型供應商的名稱。您通常要指定供應商的證券代號，但可唯一識別供應商的任何名稱均有效。僅使用字母數字字元。
- **應用程式名稱**。資源類型的名稱。僅使用字母數字字元。

注意 – 供應商名稱與應用程式名稱共同構成了資源類型的完整名稱。完整名稱不得超出 9 個字元。

- **RT 版本**。所產生資源類型的版本。[RT 版本] 可辨別屬同一基本資源類型的多個註冊版本或升級版本。

  在 [RT 版本] 欄位中不能使用下列字元：空白、定位字元、斜線 (/)、反斜線 (\)、星號 (*)、問號 (?)、逗號 (,)、分號 (;)、左方括號 ([) 或右方括號 (])。
- **工作目錄**。在該目錄下，Agent Builder 將建立一個目錄結構，以包含爲目標資源類型建立的所有檔案。您可以在任何一個工作目錄中僅建立一個資源類型。Agent Builder 會將此欄位初始化爲您從中啓動 Agent Builder 的目錄之路徑，但是您可以鍵入其他名稱或使用 [瀏覽] 來尋找其他目錄。

  在工作目錄下，Agent Builder 將會以資源類型名稱建立子目錄。例如，如果 `SUNW` 是供應商名稱，`ftp` 是應用程式名稱，則 Agent Builder 會將該子目錄命名爲 `SUNWftp`。

  Agent Builder 會將目標資源類型的所有目錄與檔案都置於該子目錄下。請參閱第 152 頁的 「目錄結構」。
- **可延伸或防故障備用**。指定目標資源類型將爲防故障備用還是可延伸。
- **網路支援**。指定基本應用程式是否爲網路支援的應用程式，即是否使用網路與其用戶端進行通訊。選取 [網路支援] 核取方塊以指定網路支援，或者不選取該核取方塊以指定非網路支援。
- **C**、**ksh**。指定所產生源代碼的語言。雖然這些選項互斥，但是有了 Agent Builder，您仍可以使用 Korn shell 產生的程式碼來建立資源類型，然後重複使用相同資訊來建立 C 產生的程式碼。請參閱第 150 頁的 「如何複製現有的資源類型」。
- **GDS**。指定此服務爲通用資料服務。請參閱第 10 章，以取得有關建立與配置一般資料服務的資訊。

注意 – 如果 `cc` 編譯器不在 `$PATH` 中，則 Agent Builder 將以灰色顯示 C 單選按鈕，並允許您選取 ksh 單選按鈕。若要指定其他編譯程式，請參閱第 139 頁的 「安裝與配置 Agent Builder」結尾處的說明。

當您輸入所需的資訊後，請按一下 [建立]。螢幕底部的 [輸出日誌] 視窗顯示 Agent Builder 執行的動作。您可以從 [編輯] 功能表中選擇 [儲存輸出日誌]，將資訊儲存至輸出日誌。

完成以上作業後，Agent Builder 將顯示一則成功訊息或警告訊息。

- 如果 Agent Builder 無法完成此步驟，請檢查輸出日誌，以取得詳細資訊。

- 如果 Agent Builder 順利完成此步驟，請按 [下一個] 以顯示 [配置] 螢幕，此螢幕可讓您完成產生資源類型的程序。

注意 – 雖然產生完整的資源類型需要兩個步驟，但您可以在完成第一個步驟 (建立) 之後結束 Agent Builder，而不會遺失您已輸入的資訊或 Agent Builder 已完成的工作。請參閱第 150 頁的 「重複使用完成的工作」。

使用 [配置] 畫面

[配置] 畫面

如下圖所示，在 Agent Builder 完成資源類型的建立，並且您選取 [建立] 螢幕上的 [下一個] 之後，將顯示 [配置] 螢幕。您無法在已建立資源類型之前存取 [配置] 畫面。

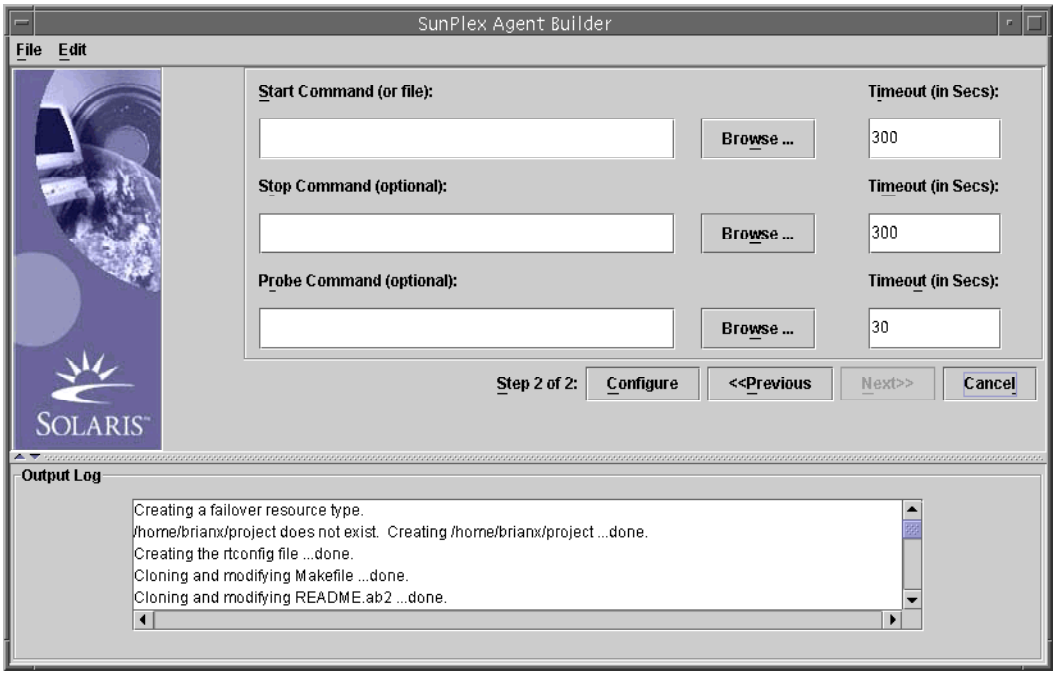


圖 9–3 [配置] 畫面

[配置] 畫面包含以下欄位：

- **Start 指令**。可傳送至任何 UNIX shell 以啓動基本應用程式的完整指令行。您必須指定 start 指令。您可以在提供的欄位中鍵入該指令，或使用 [瀏覽] 來尋找包含該指令的檔案，以啓動應用程式。

完整的指令行必須包含啓動應用程式所必需的所有項目，如主機名稱、連接埠號、配置檔案的路徑等。您還可以指定特性變數，在第 148 頁的 「特性變數」中對其進行了描述。如果基於 Korn shell 的應用程式要求在指令行上指定一個主機名稱，您可以使用 Agent Builder 定義的變數 `$hostnames`。請參閱第 148 頁的 「使用基於 Agent Builder Korn Shell 的變數 `$hostnames`」。

不要以雙引號 (“”) 括住指令。

注意 – 如果基本應用程式具有多個獨立的程序樹，其中每一個都在程序監視設備 (PMF) 的控制下以其自己的標記開始，則您無法指定單一指令。相反，您必須建立一個具有個別指令的文字檔案，以啓動每個程序樹，並在 [Start 指令] 文字欄位中指定該檔案的路徑。請參閱第 138 頁的 「建立具有多個獨立程序樹的資源類型」，其中列出了該檔案正常工作需要的某些特殊特性。

- **Stop 指令**。可傳送至任何 UNIX shell 以停止基本應用程式的完整指令行。您可以在提供的欄位中鍵入該指令，或使用 [瀏覽] 來尋找包含該指令的檔案，以停止應用程式。您還可以指定特性變數，在第 148 頁的 「特性變數」中對其進行了描述。如果基於 Korn shell 的應用程式要求在指令行上指定一個主機名稱，您可以使用 Agent Builder 定義的變數 `$hostnames`。請參閱第 148 頁的 「使用基於 Agent Builder Korn Shell 的變數 `$hostnames`」。

  此指令是可選用的。如果您不指定 stop 指令，則產生的程式碼將使用訊號 (位於 `Stop` 方法中) 來停止應用程式，如下所示：

  - `Stop` 方法傳送 `SIGTERM` 以停止應用程式，並等待逾時值之 80% 的時間以結束應用程式。
  - 如果 `SIGTERM` 訊號失敗，則 `Stop` 方法將傳送 `SIGKILL` 以停止應用程式，並等待逾時值之 15% 的時間以結束應用程式。
  - 如果 `SIGKILL` 失敗，則 `Stop` 方法也將以失敗結束。剩餘 5% 的逾時值被視爲經常性耗用時間。

注意 – 確定在應用程式完全停止之前 stop 指令不會傳回。

- **Probe 指令**。可定期運行以檢查應用程式運作狀態並傳回 `0` (成功) 至 `100` (完全失敗) 之間的適當結束狀態的指令。此指令是可選用的。您可以鍵入該指令的完整路徑，或使用 [瀏覽] 來尋找包含該指令的檔案，以測試應用程式。

  一般您將指定基本應用程式的簡單用戶端。如果您未指定 probe 指令，產生的程式碼會僅連接至資源所使用的通訊埠並僅從該通訊埠斷開連接，如果成功，則會宣告應用程式狀態良好。您還可以指定特性變數，在第 148 頁的 「特性變數」中對其進行了描述。如果基於 Korn shell 的應用程式要求您在 probe 指令行上指定一個主機名稱，您可以使用 Agent Builder 定義的變數 `$hostnames`。請參閱第 148 頁的 「使用基於 Agent Builder Korn Shell 的變數 `$hostnames`」。
- **逾時**。每個指令的逾時值 (以秒爲單位)。您可以指定一個新值，或接受 Agent Builder 提供的預設值 (對於啓動與停止爲 300 秒，對於探測爲 30 秒)。

使用基於 Agent Builder Korn Shell 的變數 $hostnames

對於許多應用程式，特別是網路支援的應用程式，應用程式在其上偵聽的主機名稱以及客戶所要求的服務，必須傳送至指令行上的應用程式。在許多情況下，主機名稱是您必須爲目標資源類型 (位於 [配置] 螢幕上) 的 start、stop 以及 probe 指令指定的參數。不過，應用程式在其上偵聽的主機名稱是叢集特定名稱。主機名稱是在叢集上運行資源時確定的，當Agent Builder 產生資源類型程式碼時，無法確定主機名稱。

爲了解決此問題，Agent Builder 將在指令行上提供您可以爲 start 指令、stop 指令以及 probe 指令指定的變數 `$hostnames`。

注意 – 僅支援變數 `$hostnames` 與基於 Korn shell 的服務配合使用。不支援變數 `$hostnames` 與基於 C 的服務和基於 GDS 的服務配合使用。

請完全按照指定實際主機名稱的方式來指定變數`$hostnames`，例如：

```
% /opt/network_aware/echo_server -p 連接埠號 -l $hostnames
```

在叢集上運行目標資源類型的資源時，將以變數 `$hostnames` 的值取代爲該資源配置的主機名稱 `LogicalHostname` 或 `SharedAddress` (在資源的 `Network_resources_used` 資源特性中)。

如果您使用多個主機名稱配置 `Network_resources_used` 特性，則變數 `$hostnames` 將包含所有主機名稱，每個名稱之間以逗號分隔。

特性變數

您可以使用特性變數，從 RGM 框架擷取所選 Sun Cluster 資源、資源類型以及資源群組特性的值。Agent Builder 將掃描特性變數的啓動、測試或停止程序檔，並在 Agent Builder 啓動該程序檔之前用它們的值取代這些變數。

注意 – 不支援特性變數與基於 Korn shell 的服務配合使用。

特性變數清單

以下清單包含您可以與程序檔配合使用的特性變數。在附錄 A 中對 Sun Cluster 資源、資源類型以及資源群組特性進行了描述。

以下清單包含資源特性變數：

- `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

以下清單包含資源類型特性變數：

- RT_API_VERSION
- RT_BASEDIR
- RT_FAILOVER
- RT_INSTALLED_NODES
- RT_NAME
- RT_RT_VERSION
- RT_SINGLE_INSTANCE

以下清單包含資源群組特性變數：

- 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

特性變數語法

在特性名稱之前加入百分比符號 (%) 以指示特性變數，如以下範例所示。

```
# /opt/network_aware/echo_server -t %RS_STOP_TIMEOUT -n %RG_NODELIST
```

考慮到前面的範例，Agent Builder 可能會解譯這些特性變數，並使用下列值啓動 `echo_server` 程序檔。

```
# /opt/network_aware/echo_server -t 300 -n phys-node-1,phys-node-2,phys-node-3
```

Agent Builder 如何取代特性變數

以下清單描述了 Agent Builder 如何解譯特性變數的類型：

- 整數被其實際值取代 (例如，`300`)。
- 布林值被字串 `TRUE` 或 `FALSE` 取代。
- 字串被實際字串取代 (例如，`phys-node-1`)。
- 字串清單被清單中的所有成員取代，每個成員以逗號分隔 (例如，`phys-node-1,phys-node-2,phys-node-3`)。
- 整數清單被清單中的所有成員取代，每個成員以逗號分隔 (例如，`1,2,3`)。
- 列舉類型被其字串形式的值取代。

重複使用完成的工作

Agent Builder 可讓您以下列方法重複使用已完成的工作：

- 您可以複製使用 Agent Builder 建立的現有資源類型。
- 您可以編輯 Agent Builder 產生的源代碼，然後重新編譯該代碼以建立一個新的套裝軟體。

▼ 如何複製現有的資源類型

依照此程序複製 Agent Builder 產生的現有資源類型。

1. **使用下列方法之一，將現有的資源類型載入 Agent Builder：**
   - 從現有資源類型 (使用 Agent Builder 建立) 的工作目錄 (包含 `rtconfig` 檔案) 啓動 Agent Builder。Agent Builder 會在 [建立] 和 [配置] 螢幕中載入該資源類型的值。
   - 使用 [檔案] 下拉式功能表中的 [載入資源類型] 選項。
2. **變更 [建立] 畫面上的工作目錄。**

   您必須使用 [瀏覽] 來選取目錄。鍵入新的目錄名稱還不夠。當您選取目錄後，Agent Builder 會重新啓動 [建立] 按鈕。
3. **進行變更。**

   您可以使用此程序來變更爲資源類型產生的程式碼之類型。例如，如果您初始建立 Korn shell 版本的資源類型，但稍後發現需要 C 版本，您可以載入現有的 Korn shell 資源類型，將輸出語言變更爲 C，然後使 Agent Builder 建立 C 版本的資源類型。
4. **建立複製的資源類型。**

   按一下 [建立] 以建立資源類型。按一下 [下一個] 以顯示 [配置] 螢幕。按一下 [配置] 以配置資源類型，然後按一下 [取消] 完成。

編輯產生的來源代碼

若要使建立資源類型的程序保持簡單，Agent Builder 將會限制輸入的數目，此數目會必要地限制所產生資源類型的範圍。因此，若要增加更複雜的功能 (如其他特性的驗證檢查) 或調諧 Agent Builder 不接觸的參數，您需要修改產生的來源代碼或 RTR 檔案。

來源檔位於 *install_directory* /*rt_name*/src 目錄中。Agent Builder 將在可新增程式碼的位置內嵌入來源代碼的註釋。這些註釋的格式爲 (對於 C 程式碼)：

```
/* User added code -- BEGIN vvvvvvvvvvvvvvv */
/* User added code -- END   ^^^^^^^^^^^^^^^ */
```

注意 – 除了井字號(#) 指示註釋的開始之外，這些註釋在 Korn shell 來源代碼方面是完全相同的。

例如，*rt_name*.h 宣告不同程式使用的所有公用程式常式。在宣告清單結尾處是可讓您宣告其他常式 (您可能已將其加入至某程式碼) 的註釋。

Agent Builder 也會在 *install_directory*/*rt_name*/src 目錄中產生具有適當目標的 makefile。使用 make 指令重新編譯源代碼，並使用 make pkg 指令重新產生資源類型套裝軟體。

RTR 檔案位於 *install_directory*/ *rt_name*/etc 目錄中。您可以使用標準文字編輯器編輯 RTR 檔案。請參閱第 29 頁的 「設定資源特性和資源類型特性」以取得關於 RTR 檔案的更多資訊，參閱附錄 A 以取得關於特性的資訊。

▼ 如何使用 Agent Builder 的指令行版本

Agent Builder 的指令行版本與圖形化使用者介面具有相同的基本程序。不過，與在圖形化使用者介面中輸入資訊不同，您要將參數傳送至指令 scdscreate 和 scdsconfig。請參閱 scdscreate(1HA) 和 scdsconfig(1HA) 線上說明手冊。

依照以下步驟使用 Agent Builder 的指令行版本：

1. **使用 scdscreate 建立 Sun Cluster 資源類型範本，以使應用程式高度可用或可延伸。**
2. **使用 scdsconfig 配置您使用 scdscreate 建立的資源類型範本。**
   您可以指定特性變數。第 148 頁的 「特性變數」中描述了特性變數。
3. **將目錄變更為工作目錄中的 pkg 子目錄。**
4. **使用 pkgadd 指令安裝您透過 scdscreate 建立的套裝軟體。**
5. **如果需要，請編輯產生的來源代碼。**
6. **執行啓動程序檔。**

目錄結構

Agent Builder 將建立一個目錄結構，以儲存其爲目標資源類型產生的所有檔案。在 [建立] 螢幕上指定工作目錄。您必須爲所開發的任何其他資源類型指定單獨的安裝目錄。在工作目錄下，Agent Builder 將建立一個子目錄，其名稱由供應商名稱與資源類型名稱 (位於 [建立] 螢幕) 串連而成。例如，如果您指定 SUNW 作爲供應商名稱，並建立名爲 ftp 的資源類型，則 Agent Builder 將在工作目錄下建立一個名爲 SUNWftp 的目錄。

在此子目錄下，Agent Builder 將建立並植入下表列出的目錄。

目錄名稱	目錄
bin	對於 C 輸出，包含從來源檔編譯的二進位檔。對於 Korn shell 輸出，與 src 目錄包含相同的檔案。
etc	包含 RTR 檔案。Agent Builder 將鏈結供應商名稱與應用程式名稱，以句點 (.) 分隔來形成 RTR 檔案名稱。例如，供應商名稱爲 SUNW，而資源類型的名稱爲 ftp，則 RTR 檔案的名稱將爲 SUNW.ftp。
man	包含 start、 stop 以及 remove 公用程式程序檔的自訂線上說明手冊。例如，startftp(1M)、stopftp(1M) 以及 removeftp(1M)。 若要檢視這些線上說明手冊，請使用 man -M 選項指定路徑。例如： # **man -M** *install_directory*/**SUNWftp/man removeftp**
pkg	包含最終套件。
src	包含 Agent Builder 產生的來源檔。
util	包含 Agent Builder 產生的 start、stop 以及 remove 公用程式程序檔。請參閱第 154 頁的 「公用程式程序檔與線上援助頁」。Agent Builder 會將應用程式名稱附加至這其中的每個程序檔名稱，例如 startftp、stopftp 以及 removeftp。

Agent Builder 輸出

本節說明了 Agent Builder 產生的輸出。

來源檔與二進位檔

管理資源群組並最終管理叢集上資源的資源群組管理員 (RGM) 在回呼模型上工作。當發生特定事件 (如節點故障) 時，RGM 將爲每個在受影響節點上執行的資源呼叫資源類型的方法。例如，RGM 將呼叫 `Stop` 方法以停止正在受影響節點上運行的資源，然後呼叫資源的 `Start` 方法以在其他節點上啓動資源。請參閱第 18 頁的 「RGM 模型」、第 20 頁的 「回呼方法」和`rt_callbacks`(1HA) 線上說明手冊，以取得關於該模型的更多資訊。

爲了支援該模型，Agent Builder 將在 *install_directory*/*rt_name*/`bin` 目錄中產生八個可執行 C 程式或 Korn shell 程序檔，作爲回呼方法。

注意 – 嚴格地說，實施故障監視器的 *rt_name* `_probe` 程式不是回呼程式。RGM 不會直接呼叫 *rt_name* `_probe`，但會呼叫 *rt_name*`_monitor_start` 與 *rt_name*`_monitor_stop`，其透過呼叫 *rt_name*`_probe` 來啓動與停止故障監視器。

Agent Builder 產生的八個方法列示如下：

- *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`

請參閱 `rt_callbacks`(1HA) 線上說明手冊，以取得關於其中每個方法的特定資訊。

在 *install_directory*/*rt_name*/`src` 目錄 (C 輸出) 中，Agent Builder 產生以下檔案：

- 標頭檔 (*rt_name*`.h`)
- 包含所有方法共用之程式碼的源代碼檔案 (*rt_name*`.c`)
- 共用程式碼的物件檔 (*rt_name*`.o`)
- 每個方法的源代碼檔案 (`*.c`)
- 每個方法的物件檔 (`*.o`)

Agent Builder 將連結 *rt_name*`.o` 檔案至每個方法 `.o` 檔案，以在 *install_directory*/*rt_name*/`bin` 目錄中建立可執行檔。

對於 Korn shell 輸出，*install_directory* /*rt_name*/`bin` 與 *install_directory*/*rt_name*/`src` 目錄完全相同。每個目錄均包含對應於七個回呼方法和 `Probe` 方法的上述八個可執行程序檔。

注意 – Korn shell 輸出包含兩個編譯的公用程式 (`gettime` 與 `gethostnames`)，特定的回呼方法需要這些程式取得時間並進行測試。

您可以編輯源代碼，執行 `make` 指令重新編譯代碼，完成時可以執行 `make pkg` 指令來產生一個新的套裝軟體。爲了支援對源代碼進行變更，Agent Builder 會在源代碼中可加入程式碼的適當位置嵌入註釋。請參閱第 151 頁的 「編輯產生的來源代碼」。

公用程式程序檔與線上援助頁

一旦您產生了資源類型並在叢集上安裝了資源類型的套件，則您仍必須取得正在叢集上執行的資源類型之實例 (資源)，一般透過使用管理指令或 SunPlex Manager 來完成此作業。然而，爲了方便，Agent Builder 將產生一個用於此目的的自訂公用程式程序檔 (啓動程序檔) 以及用於停止與移除目標資源類型的資源之程序檔。這三個位於 *install_directory* /*rt_name*/`util` 目錄中的程序檔將執行以下作業：

- **啓動程序檔**。註冊資源類型，並建立必要的資源群組與資源。該程序檔還將建立網路位址資源 (`LogicalHostname` 或 `SharedAddress`)，這些資源可讓應用程式透過網路與用戶端進行通訊。
- **停止程序檔**。停止並停用資源。
- **移除程序檔**。還原啓動程序檔的工作。也就是說，該程序檔停止並從系統移除資源、資源群組和目標資源類型。

注意 – 您僅可以對透過相應啓動程序檔啓動的資源使用移除程序檔，因爲這些程序檔使用內部慣例來命名資源與資源群組。

Agent Builder 透過將應用程式名稱附加至程序檔名稱來命名這些程序檔。例如，如果應用程式名稱爲 `ftp`，則程序檔將稱爲 `startftp`、`stopftp` 以及 `removeftp`。

Agent Builder 將提供每一個公用程式程序檔在 *install_directory*/*rt_name*/`man/man1m` 目錄中的線上援助頁。在啓動這些程序檔之前您應該讀取這些線上援助頁，因爲這些線上援助頁將歸檔您需要傳送至程序檔的參數。

若要檢視這些線上援助頁，請搭配使用 `-M` 選項與 `man` 指令來指定該線上援助頁目錄的路徑。例如，如果 `SUNW` 是供應商名稱，而 `ftp` 是應用程式名稱，則請使用以下指令來檢視 `startftp`(1M) 線上援助頁：

```
% man -M install_directory/SUNWftp/man startftp
```

叢集管理員也可使用線上援助頁公用程式程序檔。在叢集上安裝 Agent Builder 產生的套件時，公用程式程序檔的線上援助頁將置於 `/opt/` *rt_name*/`man` 目錄中。例如，使用以下指令來檢視 `startftp`(1M) 線上說明手冊：

```
% man -M /opt/SUNWftp/man startftp
```

支援檔案

Agent Builder 將支援檔案 (如pkginfo、postinstall、postremove 以及 preremove) 置於 *install_directory*/*rt_name*/etc 目錄中。該目錄還包含資源類型註冊 (RTR) 檔案，此檔案宣告可用於目標資源類型的資源與資源類型特性，並在向叢集註冊資源時初始化特性值。請參閱第 29 頁的 「設定資源特性和資源類型特性」，以取得更多資訊。RTR 檔案以 *vendor_name*.*resource_type_name* 的形式進行命名，例如，SUNW.ftp。

您可以使用標準文字編輯程式編輯此檔案並進行變更，而無需重新編譯來源代碼。然而，您必須使用 make pkg 指令重新建立套件。

套件目錄

install_directory/*rt_name*/pkg 目錄包含 Solaris 套件。套件的名稱是供應商名稱與應用程式名稱的串接，例如，SUNWftp。*install_directory*/*rt_name*/src 目錄中的 Makefile 支援新套件的建立。例如，如果您變更來源檔並重新編譯程式碼，或變更套件公用程式程序檔，請使用 make pkg 指令來建立新套件。

當您從叢集移除套件時，如果您嘗試同時從多個節點執行指令 pkgrm，則該指令可能會失敗。您可以透過以下兩種方法之一來解決此問題：

- 從任何節點運行 pkgrm 之前，請從叢集的某一個節點運行 remove *rt_name* 程序檔。
- 從叢集的某一個節點運行pkgrm (此操作會處理所有必要的清除作業)，並在必要時同時從剩餘的節點運行 pkgrm。

如果由於您嘗試同時從多個節點執行 pkgrm 而導致該指令失敗，請再次從一個節點執行此指令，然後再從剩餘的節點執行它。

rtconfig 檔案

如果您在工作目錄中產生 C 源代碼或 Korn shell 源代碼，Agent Builder 將產生配置檔案 rtconfig，該檔案包含您在 [建立] 螢幕和 [配置] 螢幕上輸入的資訊。如果您從現有資源類型的工作目錄啓動 Agent Builder (或從 [檔案] 下拉式功能表選擇 [載入資源類型] 來載入現有資源類型)，Agent Builder 會讀取 rtconfig 檔案，並在 [建立] 螢幕和 [配置] 螢幕中填入您爲現有資源類型提供的資訊。如果您要複製現有資源類型， 則此功能將十分有用。請參閱第 150 頁的 「如何複製現有的資源類型」。

Agent Builder 的 Cluster Agent 模組

用於 Agent Builder 的 Cluster Agent 模組是一個 NetBeans™ 模組。Cluster Agent 模組可讓 Sun Java Studio (以前爲 Sun ONE Studio) 產品的使用者透過整合式開發環境爲 Sun Cluster 軟體建立資源類型或資料服務。Agent Builder 提供一個基於螢幕的介面，用於說明您要建立的資源類型種類。

注意 – Sun Java Studio documentation 包含有關如何設定、安裝與使用 Sun Java Studio 產品的資訊。

▼ 如何安裝與設置 Cluster Agent 模組

Cluster Agent 模組在安裝 Sun Cluster 軟體時安裝。Sun Cluster 安裝工具會將 Cluster Agent 模組檔案 `scdsbuilder.jar` 置於 `/usr/cluster/lib/scdsbuilder` 中。爲了將 Cluster Agent 模組與 Sun Java Studio 軟體配合使用，您需要建立一個此檔案的符號連結。

注意 – 您必須安裝 Sun Cluster 與 Sun Java Studio 產品以及 Java 1.4，且它們必須可用於您要在其上運行 Cluster Agent 模組的系統。

---

1. **使所有使用者或僅使您自己可以使用 Cluster Agent 模組。**

   - 若要啓用所有使用者，請成爲超級使用者或擔任一個等效角色，並在全域模組目錄中建立符號連結。

     ```
     # cd /opt/s1studio/ee/modules
     # ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

     注意 – 如果您是在 `/opt/s1studio/ee` 目錄之外的目錄中安裝了 Sun Java Studio 軟體，請以您使用的路徑來取代此目錄路徑。

   - 若要僅使自己可以使用 Cluster Agent 模組，請在 `modules` 子目錄中建立符號連結。

     ```
     % cd ~your-home-dir/ffjuser40ee/modules
     % ln -s /usr/cluster/lib/scdsbuilder/scdsbuilder.jar
     ```

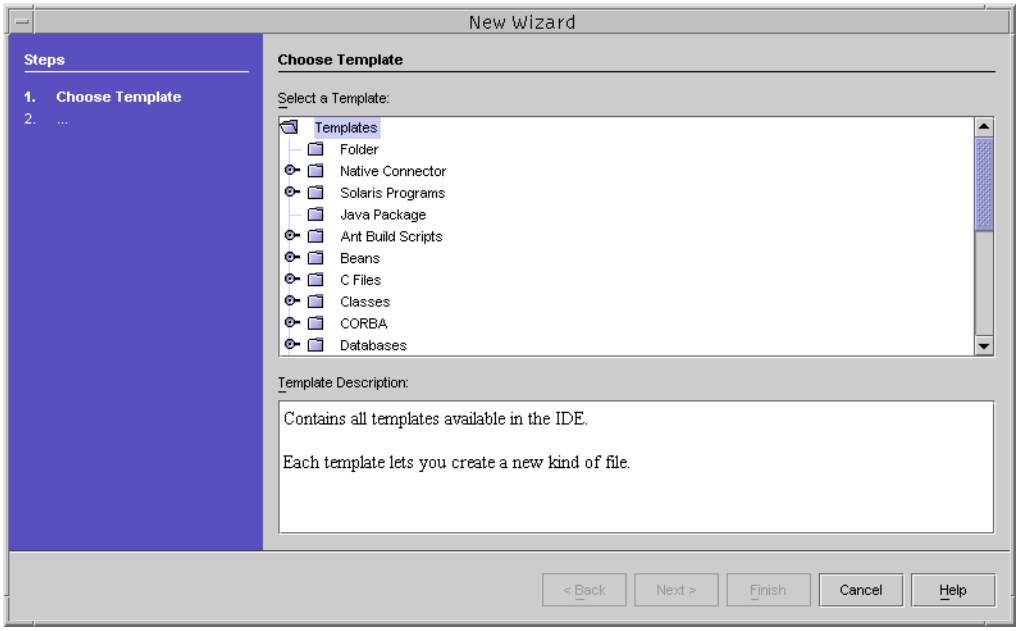
2. **停止與重新啓動 Sun Java Studio 軟體。**

▼ 如何啓動 Cluster Agent 模組

以下步驟描述如何從 Sun Java Studio 軟體啓動 Cluster Agent 模組。

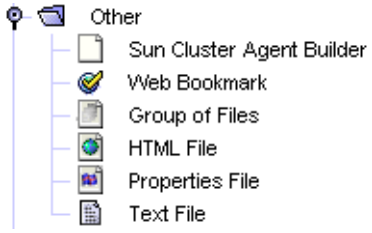
1. **從 Sun Java Studio [檔案] 功能表選擇 [新增]，或按一下工具列上的此圖示：**

螢幕上將顯示 [新增精靈] 畫面。

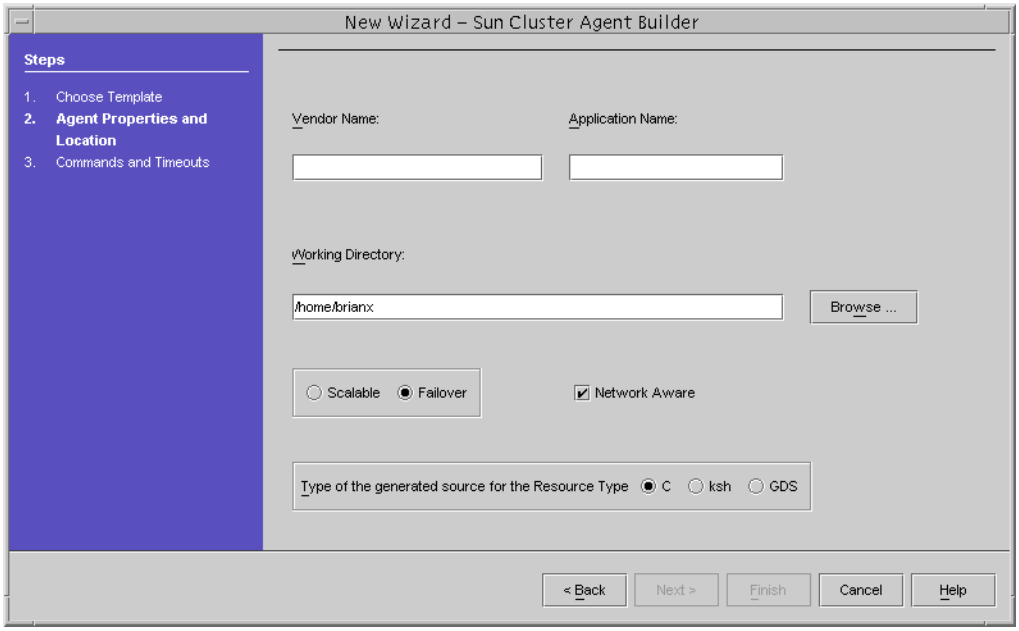


2. **在 [選取範本] 視窗中，請向下捲動 (如有必要) 並按一下 [其他] 資料夾旁的鍵。**

將開啓 [其他] 資料夾。



3. **從 [其他] 資料夾選取 Sun Cluster Agent Builder，並按一下 [下一個]。**
   Sun Java Studio 的 Cluster Agent 模組將會啓動。第一個 [新增精靈 - Sun Cluster Agent Builder] 畫面將會出現。



使用 Cluster Agent 模組

請以使用 Agent Builder 軟體的相同方式使用 Cluster Agent 模組。介面完全相同。例如，下圖說明 Agent Builder 中的 [建立] 畫面與 Cluster Agent 模組中的第一個 [新增精靈 - Sun Cluster Agent Builder] 畫面包含相同的欄位與選取。

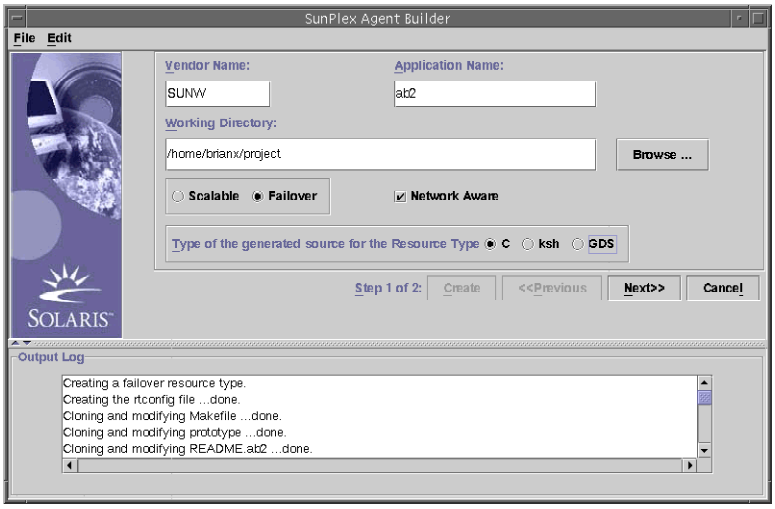


圖 9–4 Agent Builder 軟體中的 [建立] 畫面

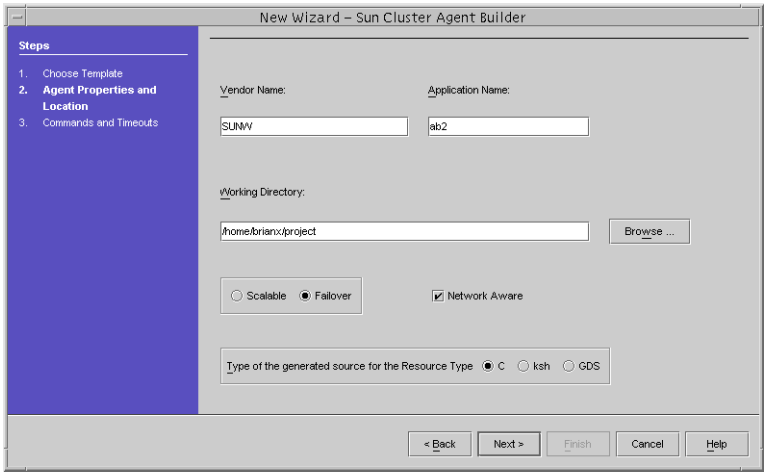


圖 9–5 Cluster Agent 模組中的 [新增精靈 - Sun Cluster Agent Builder] 畫面

Cluster Agent 模組與Agent Builder 之間的差異

儘管 Cluster Agent 模組與 Agent Builder 之間存在相似性，但也存在微小差異：

- 在 Cluster Agent 模組中，僅當您按一下第二個 [新增精靈 - Sun Cluster Agent Builder] 畫面上的 [完成] 之後，才會建立與配置資源類型。當您按一下第一個 [新增精靈 - Sun Cluster Agent Builder] 畫面上的 [下一步] 時，資源類型將**不**會建立。

  在 Agent Builder 中，當您按一下 [建立] 畫面上的 [建立] 時，將會立即建立資源類型，而當您按一下 [配置] 畫面上的 [配置] 時，將會立即配置資源類型。
- 顯示在 Agent Builder 的 [輸出登錄] 視窗中的資訊將會顯示在 Sun Java Studio 產品的獨立輸出視窗中。

第 10 章

一般資料服務

本章提供了關於通用資料服務 (GDS) 的資訊，並顯示如何建立使用 GDS 的服務。您可透過 SunPlex Agent Builder 或標準的 Sun Cluster 管理指令來建立此服務。

本章節包含下列主題：

- 第 161 頁的 「GDS 簡介」
- 第 166 頁的 「使用 SunPlex Agent Builder 建立使用 GDS 的服務」
- 第 170 頁的 「使用標準 Sun Cluster 管理指令建立使用 GDS 的服務」
- 第 172 頁的 「SunPlex Agent Builder 的指令行介面」

GDS 簡介

GDS 是一種機制，它透過將簡單的網路支援應用程式與非網路支援應用程式插入 Sun Cluster 資源群組管理 (RGM) 框架中，使其具有高度可用性或可延伸性。此機制不需要您爲代理程式編碼，而通常您必須這樣做，才能使應用程式具有高度可用性或可延伸性。

GDS 是單一且預先編譯的資料服務。您無法修改預先編譯的資料服務及其元件、回呼方法 (`rt_callbacks` (1HA)) 的實施以及資源類型註冊檔 (`rt_reg`(4))。

預先編譯的資源類型

一般資料服務資源類型 `SUNW.gds` 包含於 `SUNWscgds` 套件中。在叢集安裝期間，`scinstall` 公用程式會安裝此套裝軟體 (請參閱 `scinstall`(1M) 線上說明手冊)。此 `SUNWscgds` 套件包含下列檔案：

```
# 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
```

使用 GDS 的優勢和劣勢

與使用 SunPlex Agent Builder 產生的源代碼模型 (請參閱 `scdscreate`(1HA) 線上說明手冊) 或標準的 Sun Cluster 管理指令相比，GDS 具有以下優勢：

- GDS 容易使用。
- GDS 及其方法都是預先編譯的，因而無法修改。
- SunPlex Agent Builder 可用於爲應用程式產生程序檔，這些程序檔將置於一個可在多個叢集中重複使用的 Solaris 套裝軟體中。

雖然使用 GDS 有許多優勢，但也存在不使用該機制的情況。在以下情況下，將**不**使用 GDS 機制：

- 所要求的控制多於使用預先編譯的資源類型所可用的控制，例如當需要新增延伸特性或變更預設值時
- 需要修改源代碼以新增特殊功能

建立使用 GDS 的服務之方法

建立使用 GDS 的服務之方法有兩種：

- 使用 SunPlex Agent Builder
- 使用標準 Sun Cluster 管理指令

GDS 與 SunPlex Agent Builder

使用 SunPlex Agent Builder 並選取 GDS 作爲產生的源代碼類型。使用者輸入用來產生一組爲給定應用程式配置資源的程序檔。

GDS 與標準 Sun Cluster 管理指令

此方法使用 `SUNWscgds` 中預先編譯的資料服務程式碼，但要求系統管理員使用標準的 Sun Cluster 管理指令來建立和配置資源。請參閱 `scrgadm`(1M) 與 `scswitch`(1M) 線上說明手冊。

選取用來建立以 GDS 爲基礎的服務之方法

如程序第 171 頁的 「如何使用 Sun Cluster 管理指令建立使用 GDS 的高度可用服務」與第 171 頁的 「如何使用 Sun Cluster 管理指令建立使用 GDS 的可延伸服務」中所示，要發出適當的 `scrgadm` 與 `scswitch` 指令，需要進行大量的輸入動作。

將 GDS 與 SunPlex Agent Builder 搭配使用可簡化程序，因爲它會產生可發出 `scrgadm` 與 `scswitch` 指令的程序檔。

GDS 如何記錄事件

GDS 可讓您記錄從 GDS 傳送至它啓動的程序檔的相關資訊。此相關資訊包含 start、probe 和 stop 方法的狀態以及特性變數。您可以使用此資訊診斷程序檔中的問題或錯誤，或將其用於其他目的。

請使用第 166 頁的 「`Log_level` 特性」中描述的 `Log_level` 特性來指定 GDS 要記錄的訊息層級或類型。您可以指定 `NONE`、`INFO` 或 `ERR`。

GDS 日誌檔

以下兩個 GDS 日誌檔位於目錄 `/var/cluster/logs/DS/`*resource_group_name*`/`*resource_name* 中：

- `start_stop_log.txt`，包含由資源 start 方法和 stop 方法記錄的訊息
- `probe_log.txt`，包含由資源監視器記錄的訊息

以下範例顯示 `start_stop_log.txt` 所含資訊的類型：

```
10/20/2004 12:38:05 phys-node-1 START-INFO> Start succeeded. [/home/brianx/sc/start_cmd]
10/20/2004 12:42:11 phys-node-1 STOP-INFO> Successfully stopped the application
```

以下範例顯示 `probe_log.txt` 所含資訊的類型：

```
10/20/2004 12:38:15 phys-node-1 PROBE-INFO> The GDS monitor (gds_probe) has been started
10/20/2004 12:39:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2004 12:40:15 phys-node-1 PROBE-INFO> The probe result is 0
10/20/2004 12:41:15 phys-node-1 PROBE-INFO> The probe result is 0
```

必需的 GDS 特性

如果您的應用程式爲非網路支援應用程式，則必須提供 `Start_command` 延伸特性與 `Port_list` 特性。如果您的應用程式爲網路支援應用程式，則必須僅提供 `Port_list` 特性。

`Start_command` 延伸特性

您在 `Start_command` 延伸特性中指定的 Start 指令將啓動應用程式。它必須是一個 UNIX 指令，並且具有可直接傳送到 shell 以啓動應用程式的引數。

Port_list 特性

Port_list 特性可識別應用程式所偵聽連接埠的清單。必須在 SunPlex Agent Builder 建立的啓動程序檔中指定 Port_list 特性；如果使用的是標準 Sun Cluster 管理指令，則必須使用 scrgadm 指令指定。

可選擇的 GDS 特性

以下清單包含可選擇的 GDS 特性：

- Network_resources_used
- Stop_command (延伸特性)
- Probe_command (延伸特性)
- Start_timeout
- Stop_timeout
- Probe_timeout (延伸特性)
- Child_mon_level (僅與標準管理指令搭配使用的延伸特性)
- Failover_enabled (延伸特性)
- Stop_signal (延伸特性)
- Log_level (延伸特性)

Network_resources_used 特性

此特性的預設值爲空值。如果應用程式需要連結一個或多個特定位址，則必須指定此特性。如果省略此特性或將其指定爲 Null，則假定應用程式偵聽所有的位址。

建立 GDS 資源之前，必須先配置 LogicalHostname 或 SharedAddress 資源。請參閱「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」，以取得有關如何配置 LogicalHostname 或 SharedAddress 資源的資訊。

若要指定一個值，請指定一個或多個資源名稱。每個資源名稱可以包含一個或多個 LogicalHostname 或者一個或多個 SharedAddress。請參閱 r_properties (5) 線上說明手冊，以取得詳細資訊。

Stop_command 特性

Stop 指令必須停止應用程式，且僅在完全停止了應用程式之後才傳回。它必須是一個可直接傳送到 Shell 以停止應用程式的完整 UNIX 指令。

如果提供了 Stop_command 延伸特性，則 GDS stop 方法會在 80% 的 stop 逾時時間啓動 stop 指令。不管啓動 stop 指令的結果如何，GDS stop 方法都會在 15% 的 stop 逾時時間傳送 SIGKILL。剩餘的 5% 時間則爲事務性工作的經常性耗用時間。

如果 stop 指令被省略，GDS 會試圖使用 Stop_signal 指定的訊號來停止應用程式。

Probe_command 特性

Probe 指令將定期檢查給定應用程式的運作狀況。它必須是一個 UNIX 指令，並具有可直接傳送到 shell 以測試應用程式的引數。如果應用程式正常，probe 指令會傳回結束狀態 0。

probe 指令的結束狀態可用來判定應用程式的故障嚴重狀況。此結束狀態稱爲測試狀態，它必須是一個介於 0 (成功) 與 100 (完全故障) 之間的整數。此測試狀態也可以是特殊值 201，除非將 Failover_enabled 設定爲 FALSE，否則它會導致應用程式立即防故障備用。測試狀態用在 GDS 測試演算法中 (請參閱 scds_fm_action(3HA) 線上說明手冊)，以決定要在本機重新啓動應用程式，或者將故障轉移至另一個節點。如果結束狀態爲 201，應用程式會立即進行防故障備用。

如果忽略 probe 指令，GDS 將提供它自己的簡單測試方式，可透過 IP 位址集 (源自 Network_resources_used 特性或 scds_get_netaddr_list 的輸出) 連接至應用程式 (請參閱 scds_get_netaddr_list(3HA) 線上說明手冊)。如果連接成功，便會立即中斷。如果連接和中斷都順利完成，應用程式會被視爲運作良好。

注意 – 和 GDS 一起提供的探測功能，只用來取代正常運作的應用程式的特定探測功能。

Start_timeout 特性

此特性指定 start 指令的啓動逾時值。請參閱第 163 頁的 「Start_command 延伸特性」，以取得附加資訊。Start_timeout 的預設值爲 300 秒。

Stop_timeout 特性

此特性指定 stop 指令的停止逾時值。請參閱第 164 頁的 「Stop_command 特性」，以取得附加資訊。Stop_timeout 的預設值爲 300 秒。

Probe_timeout 特性

此特性指定 probe 指令的逾時值。請參閱第 165 頁的 「Probe_command 特性」，以取得附加資訊。Probe_timeout 的預設值爲 30 秒。

Child_mon_level 特性

注意 – 如果您使用標準的 Sun Cluster 管理指令，則可以使用該選項。如果您使用 SunPlex Agent Builder，則無法使用該選項。

此特性控制透過程序監視設備 (PMF) 監視的程序。它會指定叉狀子程序被監視到哪個層級。此特性所起的作用與 -C 引數對 pmfadm 指令所起的作用相似。請參閱 pmfadm(1M) 線上說明手冊。

省略此特性或將它設定爲預設值-1 所產生的效果，與省略 -C 選項對 pmfadm 指令所產生的效果相同。亦即，所有子程序 (及其子項) 都會受到監視。

Failover_enabled 特性

該布林延伸特性控制資源的防故障備用行爲。如果此延伸特性設定成 true，當重新啓動的數目在 retry_interval 秒之內超出 retry_count，則應用程式會故障轉移。

如果此特性設定成 false，當重新啓動的數目在 retry_interval 秒之內超出 retry_count 時，應用程式不會重新啓動或防故障備用至另一個節點。

此特性可用來防止應用程式資源啓動資源群組的防故障備用。此特性的預設值爲 true。

Stop_signal 特性

GDS 使用此整數延伸特性的值，以決定用於透過 PMF 停止應用程式的訊號。請參閱 signal(3HEAD) 線上說明手冊，以取得您可以指定的整數值清單。預設值爲 15 (SIGTERM)。

Log_level 特性

此特性指定 GDS 記錄的診斷訊息層級或類型。您可以爲該特性指定 NONE、INFO 或 ERR。指定 NONE 時，GDS 將不記錄診斷訊息。指定 INFO 時，將僅記錄資訊訊息。而指定 ERR 時，將僅記錄錯誤訊息。依預設，GDS 不記錄診斷訊息 (NONE)。

使用 SunPlex Agent Builder 建立使用 GDS 的服務

您可以使用 SunPlex Agent Builder 來建立使用 GDS 的服務。在第 9 章中將會更詳細地說明 SunPlex Agent Builder。

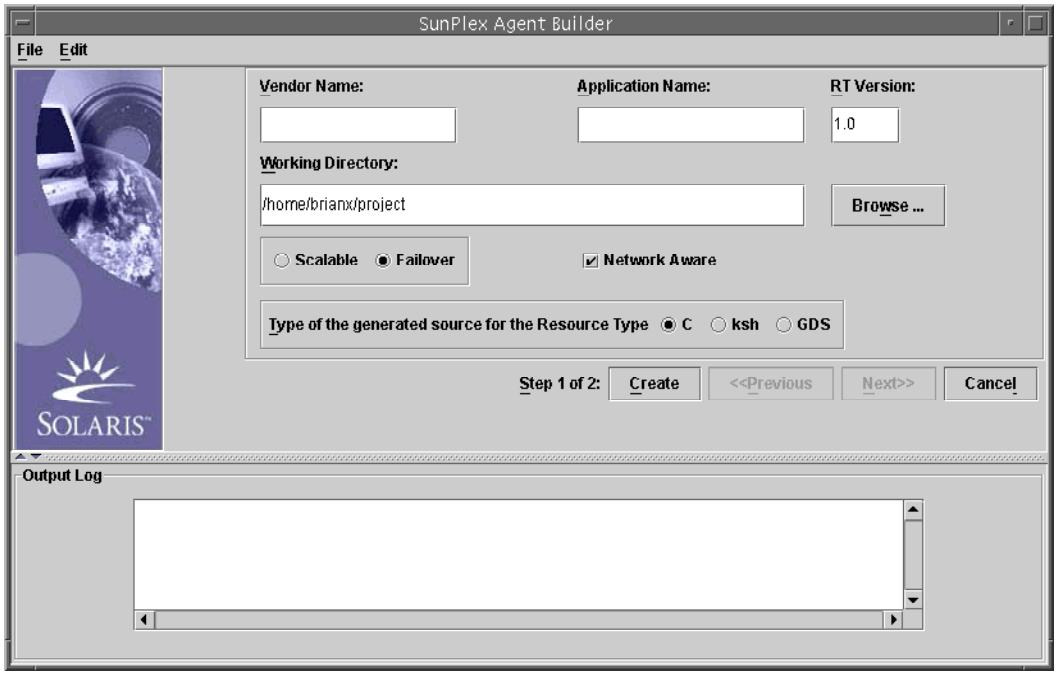
建立與配置程序檔

▼ 如何啓動 SunPlex Agent Builder 與建立程序檔

1. **成為超級使用者，或者假定一個對等身份。**
2. **啓動 SunPlex Agent Builder。**

```
# /usr/cluster/bin/scdsbuilder
```

3. **將顯示 SunPlex Agent Builder [建立] 螢幕。**



4. **鍵入供應商名稱。**
5. **鍵入應用程式名稱。**

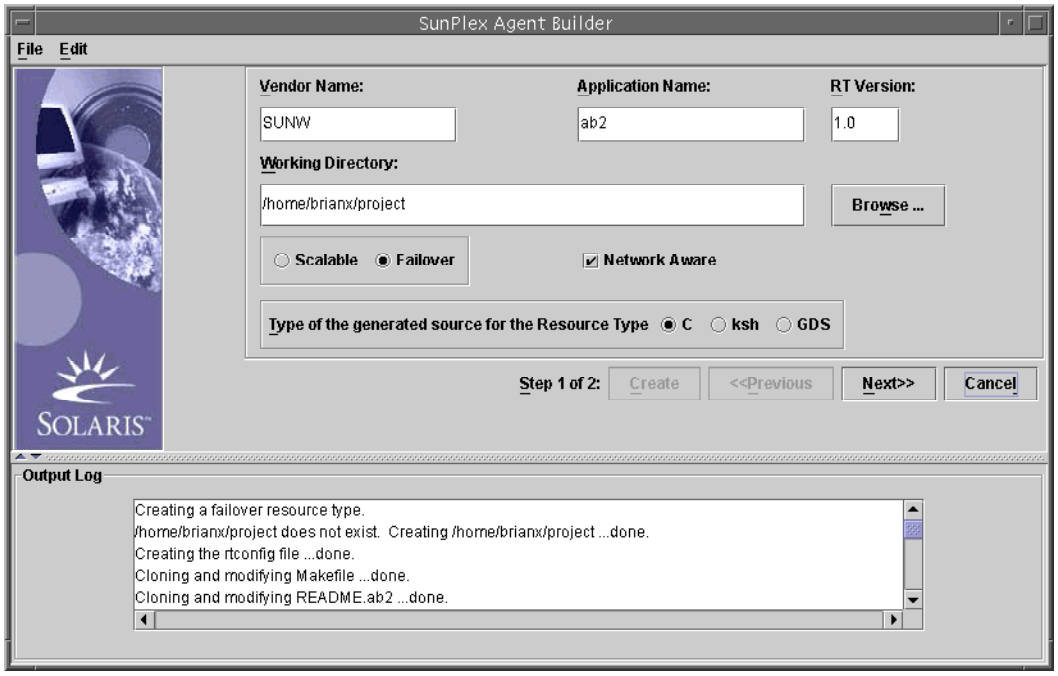
注意 – 供應商名稱與應用程式名稱的組合不得超過九個字元。它作爲程序檔的套裝軟體名稱。

---

6. **移至工作目錄。**
   您可以使用 [瀏覽] 下拉式功能表來選取目錄，而不必鍵入路徑。
7. **選取資料服務是否可延伸或故障轉移。**
   不需要選取 [網路支援]，因爲在建立 GDS 時它是預設值。
8. **選取 GDS。**
9. **(選擇性的) 變更顯示的 RT 版本預設值。**

注意 – 在 [RT 版本] 欄位中不能使用下列字元：空白、定位字元、斜線 (/)、反斜線 (\)、星號 (*)、問號 (?)、逗號 (,)、分號 (;)、左方括號 ([) 或右方括號 (])。

10. **按一下 [完成]。**
    Agent Builder 便會建立程序檔。在 [輸出日誌] 視窗中，將顯示服務的建立結果。



[建立] 將以灰色顯示。現在便可配置程序檔。

11. **按一下 [下一步]。**
    將顯示 [配置] 螢幕。

▼ 如何配置程序檔

建立程序檔後，需要配置新的服務。

1. **鍵入 start 指令的位置，或按一下 [瀏覽] 尋找 start 指令。**
   您可以指定特性變數。第 148 頁的 「特性變數」中描述了特性變數。

2. **(選擇性的) 鍵入 stop 指令，或按一下 [瀏覽] 尋找 stop 指令。**

您可以指定特性變數。第 148 頁的 「特性變數」中描述了特性變數。

3. **(選擇性的) 鍵入 probe 指令，或按一下 [瀏覽] 尋找 probe 指令。**
   您可以指定特性變數。第 148 頁的 「特性變數」中描述了特性變數。
4. **(選擇性的) 指定 start、stop 和 probe 指令的逾時值。**
5. **按一下 [配置]。**
   Agent Builder 開始配置程序檔。

注意 – Agent Builder 鏈結供應商名稱與應用程式名稱以建立套裝軟體名稱。

將建立程序檔的套裝軟體，並將其放置在以下目錄中：

working-dir/*vendor_nameapplication*/pkg
例如 /export/wdir/NETapp/pkg。

6. **以超級使用者身份，在叢集所有節點上安裝完成的套裝軟體。**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

7. **透過 pkgadd 安裝以下檔案：**

```
/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
```

注意 – 線上說明手冊及程序檔名稱與先前輸入的應用程式名稱 (以程序檔名稱開頭，如 startapp) 對應。

若要檢視線上援助頁，您需要指定到援助頁的路徑。例如，若要檢視 startapp (1M) 線上說明手冊，請鍵入：

```
# man -M /opt/NETapp/man startapp
```

8. **在叢集的一個節點上，配置資源並啓動應用程式。**

```
# /opt/NETapp/util/startapp -h logicalhostname -p port_and_protocol_list
```

依據資源類型的不同，startapp 程序檔的引數也會不同：故障轉移或可延伸。檢查自訂的線上說明手冊，或運行沒有引數的 startapp 程序檔，以顯示用法描述。

```
# /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]
```

SunPlex Agent Builder 的輸出

SunPlex Agent Builder 基於建立套裝軟體時您提供的輸入，產生三個程序檔和一個配置檔。配置檔會指定資源群組與資源類型的名稱。

程序檔爲：

- Start (啓動) 程序檔：用於配置資源，並啓動 RGM 控制下的應用程式。
- Stop (停止) 程序檔：用於停止應用程式及關閉資源與資源群組。
- Remove (移除) 程序檔：用於移除啓動程序檔所建立的資源與資源群組。

這些程序檔的介面及運作方式，與 SunPlex Agent Builder 爲非 GDS 型代理程式產生的公用程式程序檔相同。這些程序檔被置於可在多個叢集中重複使用的 Solaris 套裝軟體中。

您可以自訂配置檔，以自行提供資源群組名稱或通常輸入至 scrgadm 指令的參數。如果您不自訂程序檔，SunPlex Agent Builder 將爲 scrgadm 參數提供預設值。

使用標準 Sun Cluster 管理指令建立使用 GDS 的服務

本節描述如何輸入 GDS 的參數。可使用現有的 Sun Cluster 管理指令 (如 scrgadm 與 scswitch) 來使用並管理 GDS。

如果程序檔提供了足夠的功能性，則無需輸入本節顯示的較低層級管理指令。不過，如果您需要更精細地控制基於 GDS 的資源，則可以輸入較低層級的管理指令。這些指令由程序檔執行。

▼ 如何使用 Sun Cluster 管理指令建立使用 GDS 的高度可用服務

1. 成為超級使用者，或者假定一個對等身份。
2. 註冊 `SUNW.gds` 資源類型

```
# scrgadm -a -t SUNW.gds
```

3. 建立包含 `LogicalHostname` 資源與防故障備用服務本身的資源群組。

```
# scrgadm -a -g haapp_rg
```

4. 建立 `LogicalHostname` 資源的資源。

```
# scrgadm -a -L -g haapp_rs -l hhead
```

5. 建立故障轉移服務本身的資源。

```
# 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. 讓 `haapp_rg` 資源群組上線運作。

```
# scswitch -Z -g haapp_rg
```

▼ 如何使用 Sun Cluster 管理指令建立使用 GDS 的可延伸服務

1. 成為超級使用者，或者假定一個對等身份。
2. 註冊 `SUNW.gds` 資源類型。

```
# scrgadm -a -t SUNW.gds
```

3. 建立 `SharedAddress` 資源的資源群組。

```
# scrgadm -a -g sa_rg
```

4. 在 `sa_rg` 建立 `SharedAddress` 資源。

```
# scrgadm -a -S -g sa_rg -l hhead
```

5. 建立可延伸服務的資源群組。

```
# scrgadm -a -g app_rg -y Maximum_primaries=2 \
      -y Desired_primaries=2 -y RG_dependencies=sa_rg
```

6. **建立可延伸服務本身的資源群組。**

```
# 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. **讓包含網路資源的資源群組上線運作。**

```
 # scswitch -Z -g sa_rg
```

8. **讓 `app_rg` 資源群組上線運作。**

```
# scswitch -Z -g app_rg
```

SunPlex Agent Builder 的指令行介面

SunPlex Agent Builder 也包含一個指令行介面，此介面提供與圖形化使用者介面相同的功能性。此介面由指令 `scdscreate` 與 `scdsconfig` 組成。請參閱 `scdscreate` (1HA) 與 `scdsconfig`(1HA) 線上說明手冊。

▼ 如何透過 Agent Builder 的指令行版本建立使用 GDS 的服務

本節描述如何透過指令行介面來執行第 166 頁的 「使用 SunPlex Agent Builder 建立使用 GDS 的服務」中顯示的相同步驟集。

1. **成為超級使用者，或者假定一個對等身份。**
2. **建立服務。**

   對於防故障備用服務，請鍵入：

```
# scdscreate -g -V NET -T app -d /export/wdir
```

   對於可延伸服務，請鍵入：

```
# scdscreate -g -s -V NET -T app -d /export/wdir
```

注意 – `-d` 參數是可選擇的。如果您不指定此參數，目前的目錄將成爲工作目錄。

---

3. **配置服務。**

```
# scdsconfig -s "/export/app/bin/start" -t "/export/app/bin/stop" \
-m "/export/app/bin/probe" -d /export/wdir
```

您可以指定特性變數。第 148 頁的 「特性變數」中描述了特性變數。

注意 – 僅需要 `start` 指令。所有其他參數則均爲可選用參數。

---

4. **在叢集所有節點上安裝完成的套件。**

```
# cd /export/wdir/NETapp/pkg
# pkgadd -d . NETapp
```

5. **透過 `pkgadd` 安裝以下檔案：**

```
/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
```

注意 – 線上說明手冊及程序檔名稱與先前輸入的應用程式名稱 (以程序檔名稱開頭，如 `startapp`) 對應。

若要檢視線上援助頁，您需要指定到援助頁的路徑。例如，若要檢視 `startapp` (1M) 線上說明手冊，請鍵入：

```
# man -M /opt/NETapp/man startapp
```

6. **在叢集的一個節點上，配置資源並啓動應用程式。**

```
# /opt/NETapp/util/startapp -h logicalhostname -p port_and_protocol_list
```

依據資源類型的不同，`startapp` 程序檔的引數也會不同：故障轉移或可延伸。檢查自訂的線上說明手冊，或運行沒有引數的 `startapp` 程序檔，以顯示用法描述。

```
# /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]
```

第 11 章

資料服務開發程式庫參考

本章列出並簡明描述了資料服務開發程式庫 (DSDL) API 的函式。請參閱個別 3HA 線上援助頁，以取得每個 DSDL 函式的完整說明。DSDL 僅定義了 C 介面。基於程序檔的 DSDL 介面無法使用。

DSDL 提供下列類型的函式：

- 第 175 頁的 「通用函式」
- 第 176 頁的 「特性函式」
- 第 176 頁的 「網路資源存取函式」
- 第 178 頁的 「PMF 函式」
- 第 178 頁的 「故障監視器函式」
- 第 178 頁的 「公用程式函式」

DSDL 函式

以下子節簡明介紹了每種 DSDL 函式。不過，個別 3HA 線上援助頁是 DSDL 函式的決定性參考。

通用函式

本節中的函式提供廣泛功能。這些函式可讓您執行下列作業：

- 初始化 DSDL 環境
- 擷取資源、資源類型、資源群組名稱以及延伸特性值。
- 故障轉移並重新啓動資源群組以及重新啓動資源
- 將錯誤字串轉換爲錯誤訊息
- 在逾時控制下執行指令

下列函式初始化呼叫方法：

- `scds_initialize`(3HA) – 配置資源並初始化 DSDL 環境。
- `scds_close`(3HA) – 釋放 `scds_initialize` 配置的資源。

下列函式擷取關於資源、資源類型、資源群組與延伸特性的資訊：

- `scds_get_resource_name`(3HA) – 擷取呼叫程式的資源名稱。
- `scds_get_resource_type_name`(3HA) – 擷取呼叫程式的資源類型名稱。
- `scds_get_resource_group_name`(3HA) – 擷取呼叫程式的資源群組名稱。
- `scds_get_ext_property`(3HA) – 擷取指定延伸特性的值。
- `scds_free_ext_property`(3HA) – 釋放 `scds_get_ext_property` 配置的記憶體。

下列函式擷取關於某個資源所使用的 `SUNW.HAStoragePlus` 資源的狀態資訊。

- `scds_hasp_check`(3HA) – 擷取關於某個資源所使用的 `SUNW.HAStoragePlus`(5) 資源的狀態資訊。此項資訊是從所有 `SUNW.HAStoragePlus` 資源的狀態 (線上或其他狀態) 取得，其中資源視所採用爲其定義的 `Resource_dependencies` 或 `Resource_dependencies_weak` 系統特性而定。

下列函式重新啓動資源或資源群組，或對其進行防故障備用：

- `scds_failover_rg`(3HA) – 對資源群組進行防故障備用。
- `scds_restart_rg`(3HA) – 重新啓動資源群組。
- `scds_restart_resource`(3HA) – 重新啓動資源。

下列函式在逾時控制下執行指令，並將錯誤碼轉換爲錯誤訊息：

- `scds_timerun`(3HA) – 在逾時值控制下執行指令。
- `scds_error_string`(3HA) – 將錯誤碼轉譯爲錯誤字串。

特性函式

這些函式提供公用 API，便於存取相關資源、資源群組與資源類型的特定特性，包括某些常用的延伸特性。DSDL 提供了 `scds_initialize` 函式來剖析指令行引數。然後，程式庫便可以**快取**相關資源、資源群組與資源類型的各種特性。

`scds_property_functions`(3HA) 描述這些函式。這些函式包括：

- `scds_get_rs_`*property_name*
- `scds_get_rg_`*property_name*
- `scds_get_rt_`*property_name*
- `scds_get_ext_`*property_name*

網路資源存取函式

本節中列出的函式可擷取、列印與釋放資源及資源群組使用的網路資源。本節中的 `scds_get_*` 函式提供了擷取網路資源的便捷方式，無需使用 RMAPI 函式查詢特定特性 (如 `Network_resources_used` 與 `Port_list`)。`scds_print_`*name*`()` 函式列印 `scds_get_`*name*`()` 函式傳回的資料結構中的值。`scds_free_`*name*`()` 函式釋放 `scds_get_`*name*`()` 函式分配的記憶體。

處理主機名稱的函式包括：

- `scds_get_rg_hostnames`(3HA) – 擷取資源群組中網路資源所使用的主機名稱清單。
- `scds_get_rs_hostnames`(3HA) – 擷取資源所使用的主機名稱清單。
- `scds_print_net_list`(3HA) – 列印 `scds_get_rg_hostnames()` 或 `scds_get_rs_hostnames()` 傳回的主機名稱清單之內容。
- `scds_free_net_list`(3HA) – 釋放 `scds_get_rg_hostnames()` 或 `scds_get_rs_hostnames ()` 配置的記憶體。

處理連接埠清單的函式包括：

- `scds_get_port_list`(3HA) – 擷取資源所使用的連接埠協定組清單。
- `scds_print_port_list`(3HA) – 列印 `scds_get_port_list ()` 傳回的連接埠協定組清單之內容。
- `scds_free_port_list`(3HA) – 釋放 `scds_get_port_list()` 配置的記憶體。

處理網路位址的函式包括：

- `scds_get_netaddr_list`(3HA) – 擷取某個資源所使用的網路位址清單。
- `scds_print_netaddr_list`(3HA) – 列印 `scds_get_netaddr_list` 傳回的網路位址清單之內容。
- `scds_free_netaddr_list`(3HA) – 釋放 `scds_get_netaddr_list` 配置的記憶體。

使用 TCP 連接的故障監視

本節中的函式啓用基於 TCP 的監視。通常，故障監視器使用這些函式來建立其與服務的簡單套接字連接，從該服務讀取資料以及將資料寫入該服務以確定其狀況，然後與該服務斷開連接。

這些函式包括：

- `scds_fm_tcp_connect`(3HA) – 僅建立至使用 IPv4 定址之程序的 TCP 連線。
- `scds_fm_net_connect`(3HA) – 將建立 TCP 連線以連接至使用 IPv4 或 IPv6 定址的程序。
- `scds_fm_tcp_read`(3HA) – 使用 TCP 連線讀取受監視程序中的資料。
- `scds_fm_tcp_write`(3HA) – 使用 TCP 連線向受監視的程序寫入資料。
- `scds_simple_probe`(3HA) – 透過建立和終止至某個程序的 TCP 連線來測試該程序。該函式僅處理 IPv4 位址。
- `scds_simple_net_probe`(3HA) – 透過建立和終止至某個程序的 TCP 連線來測試該程序。該函式處理 IPv4 和 IPv6 位址。
- `scds_fm_tcp_disconnect`(3HA) – 終止與使用受監視之 IPv4 定址程序之間的連線。

- `scds_fm_net_disconnect`(3HA) – 終止連線至使用受監視之 IPv4 或 IPv6 定址的程序。

PMF 函式

這些函式封裝 PMF 功能性。透過 PMF 進行監視的 DSDL 模型可建立與使用 `pmfadm`(1M) 的隱含 *tag* 值。PMF 工具也使用 `Restart_interval`、`Retry_count` 與 `action_script` 的隱含值 (`pmfadm` 的 `-t`、`-n` 及 `-a` 選項)。最重要的是，DSDL 將程序失效歷史 (由 PMF 找到) 結合到應用程式故障歷史 (由故障監視器偵測) 中，以計算重新啓動或故障轉移決定。

這些函式包括：

- `scds_pmf_get_status`(3HA) – 確定指定的實例是否在 PMF 控制下被監視。
- `scds_pmf_restart_fm`(3HA) – 使用 PMF 重新啓動故障監視器。
- `scds_pmf_signal`(3HA) – 將指定訊號傳送至在 PMF 控制下運行的程序樹。
- `scds_pmf_start`(3HA) – 在 PMF 控制下執行指定程式 (包括故障監視器)。
- `scds_pmf_stop`(3HA) – 終止在 PMF 控制下運行的程序。
- `scds_pmf_stop_monitoring`(3HA) – 停止監視在 PMF 控制下運行的程序。

故障監視器函式

本節中的函式透過保留故障歷史並結合 `Retry_count` 與 `Retry_interval` 特性演算該歷史，提供故障監視的預先決定模型。

這些函式包括：

- `scds_fm_sleep`(3HA) – 等待故障監視器控制插槽上的訊息。
- `scds_fm_action`(3HA) – 完成測試後採取動作。
- `scds_fm_print_probes`(3HA) – 在系統日誌檔中寫入測試狀態資訊。

公用程式函式

本節中的函式可讓您將訊息與除錯訊息寫入系統日誌。這些函式包括：

- `scds_syslog`(3HA) – 在系統日誌檔中寫入訊息。
- `scds_syslog_debug`(3HA) – 在系統日誌檔中寫入除錯訊息。

第 12 章

CRNP

本章提供有關叢集重新配置通知協定 (CRNP) 的資訊。CRNP 使故障轉移應用程式與可延伸應用程式能夠「支援叢集」。更具體地來說，CRNP 提供了一種機制，可讓應用程式註冊 Sun Cluster 重新配置事件，並接收 Sun Cluster 重新配置事件之後續非同步的通知 。在叢集內執行的資料服務與在叢集外執行的應用程式可以註冊事件的通知。當叢集中的成員資格發生變更時，以及當資源群組或資源的狀態發生變更時，均會產生事件。

注意 – `SUNW.Event` 資源類型實施在 Sun Cluster 上提供了高度可用的 CRNP 服務。在 `SUNW.Event`(5) 線上說明手冊中對 `SUNW.Event` 資源類型實施進行了更加詳細地描述。

- 第 179 頁的 「CRNP 概觀」
- 第 182 頁的 「CRNP 使用的訊息類型」
- 第 183 頁的 「用戶端如何註冊到伺服器」
- 第 185 頁的 「伺服器回覆用戶端的方式」
- 第 186 頁的 「伺服器如何將事件發送給用戶端」
- 第 189 頁的 「CRNP 如何授權用戶端與伺服器」
- 第 190 頁的 「建立使用 CRNP 的 Java 應用程式」

CRNP 概觀

CRNP 提供一些機制與常駐程式，這些機制與常駐程式產生叢集重新配置事件、透過叢集路由這些事件，並將其發送給感興趣的用戶端。

`cl_apid` 常駐程式與用戶端互動。Sun Cluster 資源群組管理員 (RGM) 產生叢集重新配置事件。這些常駐程式使用 `sysevent`d(1M) 在每個本機節點上傳送事件。`cl_apid` 常駐程式在 TCP/IP 中使用可延伸標記語言 (XML) 與感興趣的用戶端通訊。

以下圖表展示了 CRNP 元件之間事件流程的概觀。在此圖表中，一個用戶端在叢集節點 2 上執行，另一個用戶端在叢集之外的電腦上執行。

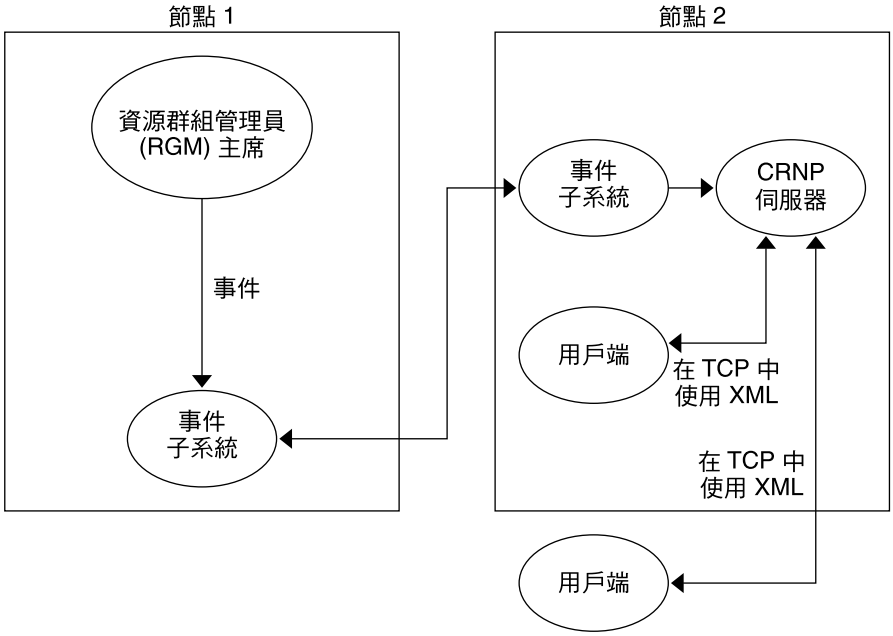


圖 12–1 CRNP 的工作方式

CRNP 協定概觀

CRNP 定義了標準七層開放式系統互連 (OSI) 協定堆疊的應用層、表示層和工作時段層。傳輸層必須是 TCP，網路層必須是 IP。CRNP 與資料連接層和實體層無關。在 CRNP 中所交換的所有應用層訊息均以 XML 1.0 爲基礎。

CRNP 協定的語義

用戶端透過將註冊訊息 (`SC_CALLBACK_RG`) 發送給伺服器來開始通訊。此註冊訊息指定用戶端要接收通知的事件類型，還指定可以將事件發送至的通訊埠。註冊連線的來源 IP 與指定的通訊埠一起形成回呼位址。

每當叢集內產生用戶端感興趣的事件，伺服器便會在其回呼位址 (IP 與通訊埠) 中聯絡用戶端，並將事件 (`SC_EVENT`) 發送給該用戶端。該伺服器是在叢集自身內執行的具有高度可用性的伺服器。該伺服器將用戶端註冊儲存在儲存體中，該儲存體在重新啓動叢集之後仍會持久性存在。

用戶端透過將註冊訊息 (`SC_CALLBACK_RG`，其中包含 `REMOVE_CLIENT` 訊息) 發送給伺服器來取消註冊。在用戶端接收到來自伺服器的 `SC_REPLY` 訊息之後會關閉連接。

以下圖表顯示用戶端與伺服器之間的通訊流程。

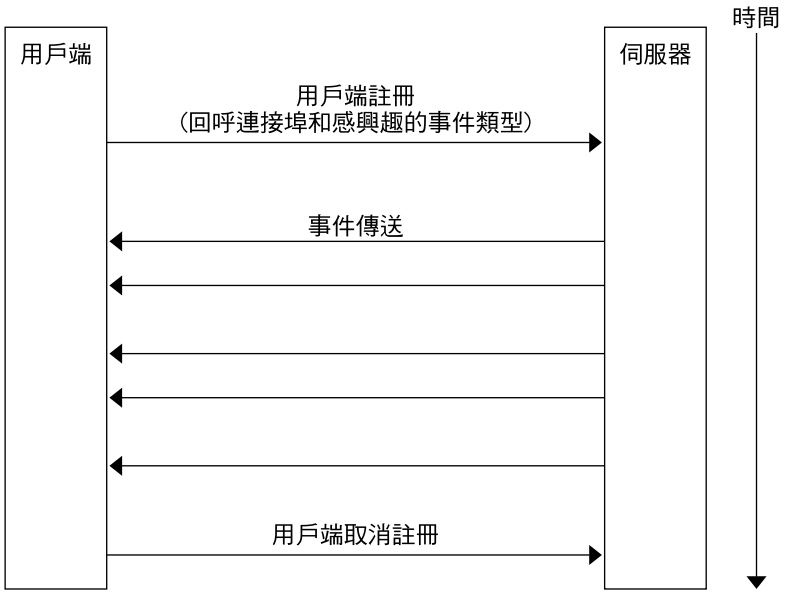


圖 12–2 用戶端與伺服器之間的通訊流程

CRNP 使用的訊息類型

CRNP 使用三種訊息類型，所有這些類型均基於 XML，如下表中所述。在本章後面對這些訊息類型進行了更加詳細的說明。在本章後面還對其用法進行了更加詳細的說明。

訊息類型	描述
SC_CALLBACK_REG	此訊息包括四種形式：ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS 與 REMOVE_EVENTS。其中每一種形式均包含下列資訊： ■ 協定版本 ■ ASCII 格式的回呼通訊埠 (非二進制格式) ADD_CLIENT、ADD_EVENTS 與 REMOVE_EVENTS 形式還包含不受限制的事件類型清單，其中每種形式均包括下列資訊： ■ 事件類別 ■ 事件子類別 (可選用的) ■ 名稱與值對的清單 (可選用的) 事件類別與事件子類別一起定義唯一的「事件類型」。產生類別 SC_CALLBACK_REG 所依照的 DTD (文件類型定義) 爲 SC_CALLBACK_REG。在附錄 F 中對此 DTD 進行了更詳細的說明。
SC_EVENT	此訊息包含下列資訊： ■ 協定版本 ■ 事件類別 ■ 事件子類別 ■ 供應商 ■ 出版商 ■ 名稱與值對清單 (0 個或更多個名稱與值對的資料結構) ■ 名稱 (字串) ■ 值 (字串或字串陣列) SC_EVENT 中的值尚未經過分類。產生類別 SC_EVENT 所依照的 DTD (文件類型定義) 是 SC_EVENT。在附錄 F 中對此 DTD 進行了更詳細的說明。
SC_REPLY	此訊息包含下列資訊： ■ 協定版本 ■ 錯誤碼 ■ 錯誤訊息 產生類別 SC_REPLY 所依照的 DTD (文件類型定義) 是 SC_REPLY。在附錄 F 中對此 DTD 進行了更詳細的說明。

用戶端如何註冊到伺服器

本節描述管理員如何設定伺服器、識別用戶端、資訊如何在應用層與工作時段層中發送以及錯誤狀況。

有關管理員如何設定伺服器的假定

系統管理員必須爲伺服器配置高度可用的 IP 位址 (即，IP 位址不固定給叢集中的某台特定機器使用) 及通訊埠編號。該管理員還必須將此網路位址發佈給預期的用戶端。CRNP 未定義如何使此伺服器名稱成爲用戶端可用的伺服器名稱。管理員將使用命名服務 (可使用戶端動態地尋找伺服器的網路位址)，或者將網路名稱加入配置檔供用戶端讀取。該伺服器將作爲故障轉移資源類型在叢集內執行。

伺服器識別用戶端的方式

每個用戶端由其回呼位址 (即，其 IP 位址與通訊埠編號) 唯一識別。該通訊埠是在 SC_CALLBACK_REG 訊息中指定的，IP 位址是從 TCP 註冊連接取得的。CRNP 假定具有同一回呼位址的後續 SC_CALLBACK_REG 訊息均來自同一用戶端，即使發送這些訊息的來源通訊埠不同亦如此。

在用戶端與伺服器之間傳送 SC_CALLBACK_REG 訊息的方式

用戶端透過開啓與伺服器的 IP 位址和埠號碼的 TCP 連線開始註冊。在建立了 TCP 連接並爲寫入做好準備之後，用戶端必須發送其註冊訊息。該註冊訊息必須是一條已正確格式化的 SC_CALLBACK_REG 訊息，在其前後都不包含額外位元組。

在將所有位元組寫入至串流之後，用戶端必須保持其連接的開啓狀態，才能接收到來自伺服器的回覆。如果用戶端沒有正確格式化該訊息，則伺服器不會註冊該用戶端，並且會將錯誤的回覆發送給該用戶端。如果該用戶端在伺服器發送回覆之前已關閉套接字連接，則伺服器會正常註冊該用戶端。

用戶端可以隨時聯絡伺服器。每當用戶端聯絡伺服器時，該用戶端必須發送 SC_CALLBACK_REG 訊息。如果該伺服器接收到形式錯誤的、順序紊亂的或者無效的訊息，便會將錯誤的回覆發送給該用戶端。

用戶端無法在發送 ADD_CLIENT 訊息之前發送 ADD_EVENTS、REMOVE_EVENTS 或者 REMOVE_CLIENT 訊息。用戶端無法在發送 ADD_CLIENT 訊息之前發送 REMOVE_CLIENT 訊息。

如果用戶端發送了 ADD_CLIENT 訊息，並且該用戶端已經註冊，則伺服器可以容許此訊息。在此情形下，伺服器會以靜音方式將舊的用戶端註冊取代爲在第二條 ADD_CLIENT 訊息中所指定的新的用戶端註冊。

在大多數情形下，用戶端透過在啓動時發送 ADD_CLIENT 訊息，在伺服器中註冊一次。並且用戶端會透過將 REMOVE_CLIENT 訊息發送給伺服器，僅取消註冊一次。然而，CRNP 會爲需要動態修改其事件類型清單的那些用戶端提供更多靈活性。

SC_CALLBACK_REG 訊息的內容

每個 ADD_CLIENT、ADD_EVENTS 與 REMOVE_EVENTS 訊息均包含事件的清單。下表說明 CRNP 接受的事件類型，其中包括必需的名稱與值對。

如果用戶端：

- 發送了指定一個或多個事件類型 (用戶端先前尚未註冊) 的 REMOVE_EVENTS 訊息，或者
- 兩次註冊同一事件類型

則該伺服器會以靜音方式忽略這些訊息。

類別與子類別	名稱與值對	描述
EC_Cluster ESC_cluster_membership	必需的：無 可選用的：無	註冊所有叢集成員資格變更事件 (節點失效或節點連結)
EC_Cluster ESC_cluster_rg_state	有一項是必需的，如下所示： rg_name 值類型：字串 可選用的：無	註冊資源群組 *name* 的所有狀態變更事件
EC_Cluster ESC_cluster_r_state	有一項是必需的，如下所示： r_name 值類型：字串 可選用的：無	註冊資源 *name* 的所有狀態變更事件
EC_Cluster 無	必需的：無 可選用的：無	註冊所有 Sun Cluster 事件

伺服器回覆用戶端的方式

處理註冊之後，伺服器會發送 SC_REPLY 訊息。該伺服器會透過從用戶端 (已接收到其註冊要求) 開啓的 TCP 連接發送此訊息。然後，便關閉該連接。在該用戶端接收到來自伺服器的 SC_REPLY 訊息之前，必須保持 TCP 連接的開啓狀態。

例如，該用戶端執行下列動作：

1. 開啓至伺服器的 TCP 連接
2. 等待連接變得「可寫入」
3. 發送 SC_CALLBACK_REG 訊息 (其中包含 ADD_CLIENT 訊息)
4. 等待 SC_REPLY 訊息
5. 接收 SC_REPLY 訊息
6. 接收表明該伺服器已關閉連接 (從套接字讀取 0 位元組) 的指示器
7. 關閉連接

稍後，用戶端便執行下列動作：

1. 開啓至伺服器的 TCP 連接
2. 等待連接變得「可寫入」
3. 發送 SC_CALLBACK_REG 訊息 (其中包含 REMOVE_CLIENT 訊息)
4. 等待 SC_REPLY 訊息
5. 接收 SC_REPLY 訊息
6. 接收表明該伺服器已關閉連接 (從套接字讀取 0 位元組) 的指示器
7. 關閉連接

每當伺服器接收到來自用戶端的 SC_CALLBACK_REG 訊息時，該伺服器均會透過開啓的同一連接發送 SC_REPLY 訊息。此訊息指定該作業是成功還是失敗。第 292 頁的「SC_REPLY XML DTD」包含 SC_REPLY 訊息的 XML 文件類型定義，以及此訊息可能包括的可能錯誤訊息。

SC_REPLY 訊息的內容

SC_REPLY 訊息指定作業是成功還是失敗。此訊息包含 CRNP 協定訊息的版本、狀況碼、詳細說明狀況碼的狀況訊息。下表說明狀況碼的可能值。

狀況碼	描述
OK	已成功處理訊息。
RETRY	伺服器由於暫態錯誤而拒絕了用戶端的註冊 (該用戶端應嘗試使用其他參數再次註冊)。

狀況碼	描述
LOW_RESOURCE	叢集資源不足，用戶端只能稍後再嘗試 (叢集的系統管理員還可以增加叢集上的資源)。
SYSTEM_ERROR	發生了嚴重的問題。請聯絡叢集的系統管理員。
FAIL	授權失敗，或者其他問題導致了註冊失敗。
MALFORMED	XML 要求的形式異常，因此無法進行剖析。
INVALID	XML 要求無效 (不符合 XML 規格)。
VERSION_TOO_HIGH	訊息的版本太高，以致無法成功處理該訊息。
VERSION_TOO_LOW	訊息的版本太低，以致無法成功處理該訊息。

用戶端處理錯誤狀況的方式

在正常狀況下，發送 SC_CALLBACK_REG 訊息的用戶端會接收到回覆，指示註冊成功或失敗。

然而，伺服器可能在用戶端註冊時，遇到禁止該伺服器將 SC_REPLY 訊息發送給用戶端的錯誤狀況。在此情況下，註冊在錯誤狀況發生之前可能已成功，或者已失敗，還可能尚未經過處理。

由於該伺服器在叢集上必須作爲故障轉移伺服器或者高度可用的伺服器來執行功能，此錯誤狀況並不意味著服務的結束。事實上，該伺服器不久就可以開始將事件發送給新註冊的用戶端。

若要修正這些狀況，您的用戶端應該進行以下兩項作業：

- 對等待 SC_REPLY 訊息的註冊連接強行加上應用層逾時，在此逾時後用戶端需要重試註冊。
- 先開始爲事件發送偵聽用戶端的回呼 IP 位址與通訊埠編號，然後再註冊事件回呼。該用戶端應該同時等待註冊確認訊息及事件發送。如果用戶端在接收到確認訊息之前就開始接收事件，則用戶端應該以靜音方式關閉註冊連接。

伺服器如何將事件發送給用戶端

由於事件在叢集內產生，CRNP 伺服器會將其發送給要求這些類型事件的所有用戶端。發送過程包括將 SC_EVENT 訊息發送給用戶端的回呼位址。每個事件的發送均發生在新的 TCP 連接中。

用戶端註冊事件類型之後，伺服器會立即透過包含 ADD_CLIENT 訊息或 ADD_EVENT 訊息的 SC_CALLBACK_REG 訊息將該類型的最新事件發送給該用戶端。然後，該用戶端就可以知曉發送後續事件的系統之目前狀態。

當伺服器開始與用戶端的 TCP 連接後，會透過此連接準確發送一條 SC_EVENT 訊息。然後，該伺服器發怖全雙工關閉。

例如，該用戶端執行下列動作：

1. 等待伺服器開始 TCP 連接
2. 接受從伺服器進來的連接
3. 等待 SC_EVENT 訊息
4. 讀取 SC_EVENT 訊息
5. 接收表明該伺服器已關閉連接 (從套接字讀取 0 位元組) 的指示器
6. 關閉連接

所有用戶端均已註冊時，必須一直爲進來的事件發送連接偵聽其回呼位址 (IP 位址與通訊埠編號)。

如果伺服器無法聯絡到用戶端，以致不能發送事件，則該伺服器會以您所定義的時間間隔與次數再次嘗試發送該事件。如果所有嘗試均失敗，則會從伺服器的用戶端清單中移除該用戶端。該用戶端還需要發送包含 ADD_CLIENT 訊息的另一個 SC_CALLBACK_REG 訊息來重新註冊，然後才可以接收更多事件。

如何保證事件的發送

叢集內事件的產生有一個總的排序，向每個用戶端發送事件的順序也會依照此順序。換句話說，如果在叢集內先產生事件 A 而後產生事件 B，則用戶端 X 會先接收到事件 A，然後接收到事件 B。然而，將事件發送給**所有**用戶端時**不會**保持總的排序。即，在用戶端 X 接收到事件 A 之前用戶端 Y 可以接收事件 A 與事件 B。以此方式，那些速度慢的用戶端不會阻礙向所有用戶端發送事件。

除了當伺服器遇到導致遺失叢集產生的事件之錯誤以外，該伺服器發送的所有事件 (除了子類別的第 一個事件與包含伺服器錯誤的事件) 均是回應叢集產生的實際事件。在此情況下，伺服器爲每個事件類型產生一個事件，表示該事件類型的系統之目前狀態。每個事件發送給表明對該事件類型感興趣的已註冊用戶端。

事件發送遵循「至少一次」的語義進行。即，允許伺服器將同一事件多次發送給某一用戶端。萬一伺服器暫時關閉，當它重新啓動後無法判斷該用戶端是否已接收到最新資訊，此時該允許是必要的。

SC_EVENT 訊息的內容

SC_EVENT 訊息包含叢集內所產生的實際訊息，該實際訊息已經過轉換以適應 SC_EVENT XML 訊息格式。下表說明 CRNP 發送的事件類型，包括名稱、值對、出版商及供應商。

類別與子類別	出版商與供應商	名稱與值對	註解
EC_Cluster ESC_cluster_membership	出版商：rgm 供應商：SUNW	名稱：node_list 值類型：字串陣列 名稱：state_list 值類型：字串陣列	state_list 的陣列元素位置與 node_list 的那些陣列元素位置是同步的。即，node_list 陣列中先列出的節點之狀態就是 state_list 陣列中的第一個狀態。 state_list 僅包含以 ASCII 表示的編號。每個編號表示叢集中該節點的目前典型編號。如果該編號與先前所接收到訊息中的編號相同，則該節點尚未變更其與叢集的關係 (分離、連結或者重新連結)。如果典型編號爲 –1，則該節點不是叢集的成員。如果典型編號是非負數的編號，則該節點是叢集的成員。 以 ev_ 開始的其他名稱及其關聯的值可能出現，但不是供用戶端使用的。
EC_Cluster ESC_cluster_rg_state	出版商：rgm 供應商：SUNW	名稱：rg_name 值類型：字串 名稱：node_list 值類型：字串陣列 名稱：state_list 值類型：字串陣列	state_list 的陣列元素位置與 node_list 的那些陣列元素位置是同步的。即，node_list 陣列中先列出的節點之狀態就是 state_list 陣列中的第一個狀態。 state_list 包含資源群組狀態的字串表示法。有效值爲可使用 scha_cmds(1HA) 指令擷取的那些值。 以 ev_ 開始的其他名稱及其關聯的值可能出現，但不是供用戶端使用的。

類別與子類別	出版商與供應商	名稱與值對	註解
EC_Cluster ESC_cluster_r_state	出版商：rgm 供應商：SUNW	有三項是必需的，如下所示： 名稱：r_name 值類型：字串 名稱：node_list 值類型：字串陣列 名稱：state_list 值類型：字串陣列	state_list 的陣列元素位置與 node_list 的那些陣列元素位置是同步的。即在 node_list 陣列中首先列出的節點之狀態就是 state_list 陣列中的第一個。 state_list 包含資源狀態的字串表示法。有效值爲可使用 scha_cmds(1HA) 指令擷取的那些值。 以 ev_ 開始的其他名稱及其關聯的值可能出現，但不是供用戶端使用的。

CRNP 如何授權用戶端與伺服器

伺服器使用 TCP 包裝函式的形式來授權用戶端。註冊訊息的來源 IP 位址 (也用來作爲發送事件的回呼 IP 位址) 必須位於伺服器上所允許的用戶端之清單中。來源 IP 位址與註冊訊息不能位於被拒用戶端的清單中。如果來源 IP 位址與註冊均不在此清單中，則伺服器會拒絕要求，並會向用戶端發出錯誤回覆。

當伺服器接收到 SC_CALLBACK_REG ADD_CLIENT 訊息時，則該用戶端的後續 SC_CALLBACK_REG 訊息包含的來源 IP 位址必須與第一條訊息中的來源 IP 位址相同。如果 CRNP 伺服器接收到不符合此需求的 SC_CALLBACK_REG，則該伺服器會：

- 忽略要求，並將錯誤回覆發送給用戶端，或者
- 假定該要求來自於新的用戶端 (依照 SC_CALLBACK_REG 訊息的內容)

此安全機制可協助阻止某人試圖取消註冊合法用戶端的服務干預被拒。

用戶端也應該以類似方式授權伺服器。用戶端僅需要接受其來源 IP 位址和通訊埠編號與該用戶端使用的註冊 IP 位址和通訊埠編號相同的伺服器之事件發送。

因爲預期 CRNP 服務的用戶端位於保護叢集的防火牆內，CRNP 不包括其他安全機制。

建立使用 CRNP 的 Java 應用程式

以下範例說明如何開發使用 CRNP 的名爲 `CrnpClient` 的簡單 Java 應用程式。該應用程式在叢集上的 CRNP 伺服器中註冊事件回呼、偵聽事件回呼、透過列印事件內容來處理這些事件。在終止之前，該應用程式會取消註冊其對事件回呼的要求。

檢查此範例時，請注意以下幾點。

- 應用程式範例執行 XML 的產生，並使用 JAXP (用於 XML 處理的 Java API) 進行剖析。此範例並不教您如何使用 JAXP。在 `http://java.sun.com/xml/jaxp/index.html` 上對 JAXP 進行了更詳細的說明。
- 此範例展示完整應用程式的片段，在附錄 G 中可以找到整個應用程式。爲了更有效地說明一些特殊概念，本章中所展示的範例與附錄 G 中所展示的完整應用程式稍有不同。
- 爲了簡潔，本章範例中的範例程式碼不包括註釋。附錄 G 中的完整應用程式包括註釋。
- 本範例中所顯示的應用程式僅需結束應用程式就可處理大多數錯誤狀況。實際的應用程式需要更牢固地處理錯誤。

▼ 設定環境

首先，您需要設定環境。

1. **下載並安裝 JAXP、正確版本的 Java 編譯器、虛擬機器。**
   您可以在 `http://java.sun.com/xml/jaxp/index.html` 上找到說明。

   注意 – 此範例需要 Java 1.3.1 或更高版本的 Java。
   ---

2. **確定您在編譯指令行中指定 `classpath`，以便編譯器可以找到 JAXP 類別。從來源檔所在的目錄，輸入：**

   ```
   % 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
   ```

   其中，*JAXP_ROOT* 是 JAXP 的 jar 檔案所在目錄的絕對路徑或相對路徑，*SOURCE_FILENAME* 是 Java 來源檔的名稱。

3. **執行應用程式時，請指定 `classpath`，以便該應用程式可以載入適當的 JAXP 類別檔案 (請注意，`classpath` 中的第一條路徑為目前目錄)：**

   ```
   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*

既然已配置了環境，您可以開發應用程式。

▼ 開始

在範例的此部分，您將使用剖析指令行引數與建構 CrnpClient 物件的主要方法，來建立名爲 CrnpClient 的基本類別。此物件將指令行引數傳送至類別、等待使用者終止該應用程式、對 CrnpClient 呼叫 shutdown，然後結束。

CrnpClient 類別的建構元需要執行下列作業：

- 設定 XML 處理物件。
- 建立偵聽事件回呼的執行緒。
- 聯絡 CRNP 伺服器與註冊事件回呼。

● **建立實施前導邏輯的 Java 程式碼。**

以下範例顯示 CrnpClient 類別的架構程式碼。建構元中所參考的四種輔助說明方法與關閉方法的實作將在以後加以說明。請注意，匯入您需要的所有套件之程式碼顯示如下。

```
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;
}
```

將在以後對成員變數進行更詳細的論述。

▼ 剖析指令行引數

- **若要查看剖析指令行引數的方式，請參閱附錄 G 中的程式碼。**

▼ 定義事件接收執行緒

在程式碼中，您需要確定事件接收是在獨立的執行緒中執行的，以便應用程式可以在事件執行緒封鎖並等待事件回呼時繼續進行其他工作。

注意 – 將在以後對設定 XML 加以論述。

---

1. **在程式碼中，定義稱為 `EventReceptionThread` 的 `Thread` 子類別 (用來建立 `ServerSocket` 並等待事件到達套接字)。**

   在範例程式碼的此部分，事件既未被讀取，也未經過處理。將在以後對讀取與處理事件加以論述。`EventReceptionThread` 在萬用字元網際網路協定位址建立 `ServerSocket`。`EventReceptionThread` 也保持對 `CrnpClient` 物件的參考，以便 `EventReceptionThread` 可以將事件發送給 `CrnpClient` 物件以進行處理。

```
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. **既然您看到了 `EventReceptionThread` 類別的工作方式，因此請建構 `createEvtRecepThr` 物件：**

```
private void createEvtRecepThr() throws Exception
{
        evtThr = new EventReceptionThread(this);
        evtThr.start();
}
```

▼ 註冊與取消註冊回呼

註冊作業包括：

- 開啓註冊網際網路協定與通訊埠的基本 TCP 套接字
- 建構 XML 註冊訊息
- 將 XML 註冊訊息發送給套接字
- 讀取來自套接字的 XML 回覆訊息
- 關閉套接字

1. **建立實施前導邏輯的 Java 程式碼。**

   以下範例顯示 `CrnpClient` 類別 (由 `CrnpClient` 建構元呼叫) 的 `registerCallbacks` 方法的實作。將在以後對 `createRegistrationString()` 與 `readRegistrationReply()` 的呼叫加以更詳細的說明。

   `regIp` 與 `regPort` 均爲由建構元設定的物件成員。

```
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. **實施 `unregister` 方法。此方法由 `CrnpClient` 的 `shutdown` 方法呼叫。將在以後對 `createUnregistrationString` 的實作加以更詳細的說明。**

```
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();
}
```

▼ 產生 XML

既然您已設定應用程式的結構，並已寫入所有網路程式碼，因此您可以寫入產生與剖析 XML 的程式碼。透過寫入產生 `SC_CALLBACK_REG` XML 註冊訊息的程式碼來啓動。

SC_CALLBACK_REG 訊息由註冊類型 (ADD_CLIENT、REMOVE_CLIENT、ADD_EVENTS 或者 REMOVE_EVENTS)、回呼通訊埠、使人感興趣的事件之清單所組成。每個事件由類別與子類別組成，其後跟隨名稱與值對的清單。

在範例的此部分，您將寫入儲存註冊類型、回呼通訊埠及註冊事件清單的 CallbackReg 類別。此類別還可以將自身串列化爲 SC_CALLBACK_REG XML 訊息。

此類別使人感興趣的方法爲 convertToXml 方法，該方法建立類別成員的 SC_CALLBACK_REG XML 訊息字串。在 http://java.sun.com/xml/jaxp/index.html 上的 JAXP 說明文件中對此方法中的程式碼進行了更加詳細的說明。

Event 類別的實作顯示如下。請注意，CallbackReg 類別使用儲存一個事件並可以將該事件轉換爲 XML Element 的 Event 類別。

1. 建立實施前導邏輯的 Java 程式碼。

```
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. **實施 `Event` 類別與 `NVPair` 類別。**

   請注意，`CallbackReg` 類別使用 `Event` 類別 (其自身使用 `NVPair` 類別)。

```
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;
}
```

▼ 建立註冊訊息與取消註冊訊息

既然您已建立了產生 XML 訊息的輔助說明類別，因此可以寫入 `createRegistrationString` 方法的實作。此方法由第 194 頁的 「註冊與取消註冊回呼」中說明的 `registerCallbacks` 方法呼叫。

`createRegistrationString` 建構 `CallbackReg` 物件，並設定其註冊類型與通訊埠。然後，`createRegistrationString` 使用 `createAllEvent`、`createMembershipEvent`、`createRgEvent` 與 `createREvent` 輔助說明方法建構各種事件。在建立 `CallbackReg` 物件之後，便會將每個事件加入該物件。最後，`createRegistrationString` 對 `CallbackReg` 物件呼叫 `convertToXml` 方法，以擷取 `String` 形式的 XML 訊息。

請注意，`regs` 成員變數儲存使用者提供給應用程式的指令行引數。第五個引數與後續引數指定應用程式應該註冊的事件。第四個引數指定註冊的類型，但在此範例中忽略。附錄 G 中的完整程式碼顯示如何使用此第四個引數。

1. **建立實施前導邏輯的 Java 程式碼。**

```
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. **建立取消註冊字串**。

   建立取消註冊字串要比建立註冊字串容易，因爲您無需考慮事件：

```
private String createUnregistrationString() throws Exception
{
        CallbackReg cbReg = new CallbackReg();
        cbReg.setPort("" + localPort);
        cbReg.setRegType(CallbackReg.REMOVE_CLIENT);
        String xmlStr = cbReg.convertToXml();
        return (xmlStr);
}
```

▼ 設定 XML 剖析器

您現在已經爲應用程式建立了網路程式碼與 XML 產生程式碼。最後一個步驟是剖析與處理註冊回覆與事件回呼。`CrnpClient` 建構元呼叫 `setupXmlProcessing` 方法。此方法建立 `DocumentBuilderFactory` 物件，並設定有關該物件的各種剖析特性。在 `http://java.sun.com/xml/jaxp/index.html` 上的 JAXP 說明文件中對此方法進行了更加詳細的說明。

- **建立實施前導邏輯的 Java 程式碼**。

```
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);
        }
```

▼ 剖析註冊回覆

若要剖析 CRNP 伺服器爲回應註冊訊息與取消註冊訊息所發送的 SC_REPLY XML 訊息，您需要 RegReply 輔助說明類別。您可以從 XML 文件建構此類別。此類別爲狀況碼與狀況訊息提供 accessor。若要剖析來自伺服器的 XML 串流，您需要建立新的 XML 文件，並使用該文件的剖析方法 (http://java.sun.com/xml/jaxp/index.html 上的 JAXP 說明文件中對此方法進行了更加詳細的說明)。

1. **建立實施前導邏輯的 Java 程式碼**。

   請注意，readRegistrationReply 方法使用新的 RegReply 類別。

```
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. **實施 RegReply 類別**。

   請注意，retrieveValues 方法在 XML 文件的 DOM 樹狀結構中行走，然後取出狀況碼與狀況訊息。http://java.sun.com/xml/jaxp/index.html 上的 JAXP 說明文件包含更多詳細資訊。

```
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;
}
```

▼ 剖析回呼事件

最後一個步驟是剖析與處理實際的回呼事件。若要輔助此作業，您要修改在第 194 頁的「產生 XML」中建立的 `Event` 類別，以便此類別可以從 XML 文件建構 `Event`，以及建立 XML `Element`。此變更需要其他建構元 (採用 XML 文件)、`retrieveValues` 方法、加入兩個成員變數 (`vendor` 與 `publisher`)、適用於所有欄位的 accessor 方法，最後還需要列印方法。

1. **建立實施前導邏輯的 Java 程式碼。**

   請注意，此程式碼類似於第 201 頁的 「剖析註冊回覆」中說明的 `RegReply` 類別之程式碼。

```
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. **針對支援 XML 剖析的 `NVPair` 類別實施其他建構元與方法。**

   變更步驟 1 中所顯示的 `Event` 類別需要對 `NVPair` 類別進行類似變更。

```
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. **在等待事件回呼的 `EventReceptionThread` 中實施 `while` 迴圈 (在第 193 頁的「定義事件接收執行緒」中有對 `EventReceptionThread` 的說明)。**

```
while(true) {
                Socket sock = listeningSock.accept();
                Document doc = db.parse(sock.getInputStream());
                Event event = new Event(doc);
                client.processEvent(event);
                sock.close();
        }
```

▼ 執行應用程式

- **執行應用程式。**

```
# java CrnpClient crnpHost crnpPort localPort ...
```

在附錄 G 中列出了 `CrnpClient` 應用程式的完整程式碼。

附錄 A

標準特性

本附錄說明了標準的資源類型、資源群組與資源特性，還說明了變更系統定義的特性與建立延伸特性可用的資源特性性質。

本附錄包含下列小節：

- 第 207 頁的 「資源類型特性」
- 第 213 頁的 「資源特性」
- 第 222 頁的 「資源群組特性」
- 第 228 頁的 「資源特性性質」

資源類型特性

以下資訊描述 Sun Cluster 所定義的資源類型特性。對特性值進行分類，如下所示 (在 [種類] 之後)：

- **必需的** – 在資源類型註冊 (RTR) 檔案中，特性必需有一個明確的值。否則無法建立特性所屬的物件。空白或空字串是不允許用來作爲值的。
- **條件式** – 此特性必須在 RTR 檔案中進行宣告才能存在。否則，RGM 不會建立此特性，管理公用程式也就無法使用它。允許空白或空字串。如果在 RTR 檔案中已宣告該特性，但是未指定任何值，則 RGM 將提供一個預設值。
- **條件式/明確** – 此特性必須在 RTR 檔案中以明確的值進行宣告才能存在。否則，RGM 不會建立此特性，管理公用程式也就無法使用它。不允許空白或空字串。
- **可選用的** — 可以在 RTR 檔案中宣告該特性。如果在 RTR 檔案中未宣告此特性，則 RGM 會建立它，並提供一個預設值。如果在 RTR 檔案中已宣告特性，但未指定任何值，則 RGM 將提供同一預設值，如同 RTR 檔案中尚未宣告該特性。

管理公用程式無法更新除 `Installed_nodes` 和 `RT_system` 之外的資源類型特性，這兩種特性無法在 RTR 檔案中進行宣告，必須由管理員來設定。

首先會顯示特性名稱，其後跟隨相關描述。

`API_version` (整數)
此資源類型實施所使用的資源管理 API 版本。

下列資訊概述各 Sun Cluster 發行版本所支援的 `API_version` 之最大值。

3.1 之前的版本及 3.1	2
3.1 10/03	3
3.1 4/04	4
3.1 9/04	5

在RTR 檔案中宣告大於 2 的 `API_version` 值可防止將該資源類型安裝在支援一個較低最高版本的 Sun Cluster 版本上。例如，如果您爲資源類型宣告 `API_version=5`，則該資源類型無法安裝於 3.1 9/04 之前發行的任何 Sun Cluster 版本上。

種類： 可選用的

預設值： 2

可調： 否

`Boot` (字串)
一個可選用的回呼方法：RGM 在節點 (已管理了此類型的資源後連結或重新連結叢集的節點) 上所呼叫的程式之路徑。在預期情況下，此方法會針對此類型資源執行類似於`Init` 方法的初始化動作。

種類： 條件式/明確

預設值： 無

可調： 否

`Failover` (布林值)
`TRUE` 指示不能於同時可在多個節點上連線的任何群組中配置此類型的資源。

種類： 可選用的

預設值： `FALSE`

可調： 否

`Fini` (字串)
一個可選用的回呼方法：在從 RGM 管理中移除此類型的資源時 RGM 所呼叫的程式之路徑。

種類： 條件式/明確

預設值： 無

可調： 否

`Init` (字串)
一個可選用的回呼方法：在此類型的資源變成由 RGM 管理的資源時 RGM 所呼叫的程式之路徑。

種類： 條件式/明確

預設值： 無

可調： 否

`Init_nodes` (列舉)
這些值可以是 `RG_primaries` (僅可以主控資源的節點) 或 `RT_installed_nodes` (安裝了資源類型的所有節點)。指示 RGM 要在其上呼叫 `Init`、`Fini`、`Boot`、`Validate` 方法的節點。

種類： 可選用的

預設值： `RG_primaries`

可調： 否

`Installed_nodes` (字串陣列)
允許資源類型在其上執行的叢集節點名稱的清單。RGM 自動建立此特性。叢集管理員可以設定值。您不能在 RTR 檔案中宣告這個特性。

種類： 可以由叢集管理員進行配置

預設值： 所有叢集節點

可調： 是

`Is_logical_hostname` (布林值)
`TRUE` 指示此資源類型是管理防故障備用網際網路協定 (IP) 位址的某一版本的 `LogicalHostname` 資源類型。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 否

`Is_shared_address` (布林值)
`TRUE` 指示此資源類型是管理防故障備用網際網路協定 (IP) 位址的某一版本的 `SharedAddress` 資源類型。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 否

`Monitor_check` (字串)
一個可選用的回呼方法：在對此類型的資源執行要求監視器的故障轉移之前，RGM 所呼叫的程式之路徑。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Monitor_start` (字串)
一個可選用的回呼方法：RGM 爲啓動此類型資源的故障監視器而呼叫的程式之路徑。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Monitor_stop` (字串)
已設定 `Monitor_start` 時所必需的回呼方法：RGM 爲停止此類型資源的故障監視器而呼叫的程式之路徑。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Pkglist` (字串陣列)
納入資源類型安裝中的選擇性套件清單。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Postnet_stop` (字串)
一個可選用的回呼方法：在呼叫此類型的資源所依賴的任何網路位址資源之 `Stop` 方法後，RGM 所呼叫的程式之路徑。取消配置網路介面之後，此方法必須執行 `Stop` 動作。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Prenet_start` (字串)
一個可選用的回呼方法：在呼叫此類型的資源所依賴的任何網路位址資源之 `Start` 方法前，RGM 所呼叫的程式之路徑。此方法預期將執行在配置網路介面之前所必須執行的 `Start` 動作。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Resource_type` (字串)
資源類型的名稱。若要檢視目前已註冊的資源類型之名稱，請使用：

```
scrgadm -p
```

在 Sun Cluster 3.1 及更高發行版本中，資源類型名稱包含版本 (這是必要的)：

vendor_id.resource_type:version

資源類型名稱的三個元件是在 RTR 檔案中指定爲 *Vendor_id*、*Resource_type* 與 *RT_version* 的特性。scrgadm 指令插入句點 (.) 和冒號 (:) 分割元。資源類型名稱的 RT_version 後綴與 RT_version 特性的值相同。若要確保 *Vendor_id* 是唯一的，建議您在建立資源類型時使用公司的證券代號。在 Sun Cluster 3.1 之前建立的資源類型名稱將繼續使用以下語法：

vendor_id.resource_type

種類： 必需的

預設值： 空字串

可調： 否

RT_basedir (字串)
用於完成回呼方法相對路徑的目錄路徑。此路徑預期會設定爲資源類型套件的安裝位置。此路徑必須爲完整的路徑，即必須以正斜線 (/) 開始。如果所有方法路徑名稱均是絕對的，則此特性不是必需的。

種類： 除非所有方法路徑名稱均是絕對的，否則是必需的

預設值： 沒有預設值

可調： 否

RT_description (字串)
資源類型的簡單說明。

種類： 條件式

預設值： 空字串

可調： 否

RT_system (布林值)
指示資源類型設定爲 TRUE 時，允許對該資訊類型執行的 scrgadm(1M) 作業會受到限制。其 RT_system 的值設定爲 TRUE 的資源類型稱爲系統資源類型。無論 RT_system 的目前值爲何，編輯 RT_system 特性本身永遠不會受到限制。

種類： 可選用的

預設值： FALSE

可調： 是

RT_version (字串)
從 Sun Cluster 3.1 開始，它是這個資源類型實作必需的版本字串。RT_version 是完整資源類型名稱的字尾部分。RT_version 特性 (在 Sun Cluster 3.0 中爲可選擇性) 在 Sun Cluster 3.1 及更高發行版本中是必要的。

種類： 可選擇性/明確或必需的

預設值： 沒有預設值

可調： 否

`Single_instance` (布林值)
如果爲 `TRUE`，則表明叢集中僅可以存在一個此類型的資源。RGM 一次僅允許一個此類型的資源在整個叢集範圍內執行。

種類： 可選用的

預設值： `FALSE`

可調： 否

`Start` (字串)
一個回呼方法：RGM 爲啓動此類型的資源而呼叫的程式之路徑。

種類： 除非 RTR 檔案宣告了 `Prenet_start` 方法，否則是必需的

預設值： 沒有預設值

可調： 否

`Stop` (字串)
一個回呼方法：RGM 爲停止此類型資源而呼叫的程式之路徑。

種類： 除非 RTR 檔案宣告了 `Postnet_start` 方法，否則是必需的

預設值： 沒有預設值

可調： 否

`Update` (字串)
一個可選用的回呼方法：變更此類型的執行資源之特性時，RGM 所呼叫的程式之路徑。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Validate` (字串)
一個可選用的回呼方法：爲檢查此類型的資源之特性值而要呼叫的程式之路徑。

種類： 條件式/明確

預設值： 沒有預設值

可調： 否

`Vendor_ID` (字串)
請參閱 `Resource_type` 特性。

種類： 條件式

預設值： 沒有預設值

可調： 否

資源特性

本小節描述 Sun Cluster 所定義的資源特性。對特性值進行分類，如下所示 (在 [種類] 之後)：

- **必需的** – 管理員必須在使用管理公用程式建立資源時指定一個值。
- **可選擇的** – 如果管理員在建立資源群組時未指定值，則系統將提供一個預設值。
- **條件式** – 僅當在 RTR 檔案中宣告了該特性時，RGM 才會建立該特性。否則，該特性不會存在，系統管理員也就無法使用它。在 RTR 檔案中宣告的條件式特性是可選擇的或者必需的，這取決於是否在 RTR 檔案中指定預設值。如需詳細資訊，請參閱每個條件式特性的說明。
- **僅限於查詢** – 不能由管理工具直接設定。

[可調] 列出您是否以及何時可以更新資源特性，如下所示：

`NONE` 或 `FALSE`	永遠不
`TRUE` 或 `ANYTIME`	任何時候
`AT_CREATION`	在將資源加入叢集時
`WHEN_DISABLED`	在資源被停用時

首先會顯示特性名稱，其後跟隨相關描述。

`Affinity_timeout` (整數)
時間長度 (以秒爲單位)，在此時間長度內，從資源中的任意服務之給定用戶端 IP 位址的連線被傳送到同一伺服器節點。

僅當 `Load_balancing_policy` 是 `Lb_sticky` 或者 `Lb_sticky_wild` 時，此特性才是有意義的。此外，必須將 `Weak_affinity` 設定爲 `FALSE` (預設值)。

此特性僅用於可延伸服務。

種類： 可選用的

預設值： 沒有預設值

可調： `ANYTIME`

`Cheap_probe_interval` (整數)
在兩次資源故障快速探測的呼叫之間的秒數。此特性由 RGM 建立，僅當在 RTR 檔案中宣告了此特性時，管理員才可以使用它。

如果在 RTR 檔案中指定了預設值，則此特性是可選用的。如果在資源類型檔案中未指定 `Tunable` 屬性，則此特性的 `Tunable` 值爲 `WHEN_DISABLED`。

如果在 RTR 檔案中宣告了此特性，且未指定 `Default` 屬性，則此特性是必需的。

種類： 條件式

預設值： 沒有預設值

可調： WHEN_DISABLED

延伸特性
在資源類型的 RTR 檔案中宣告的延伸特性。資源類型的實作定義這些特性。第 228 頁的 「資源特性性質」包含關於您可以爲延伸特性設定的個別特性之資訊。

種類： 條件式

預設值： 沒有預設值

可調： 取決於特定的特性

Failover_mode (列舉)
當 Start 方法 (Prenet_start 或 Start) 失敗時，NONE、SOFT 與 HARD 僅影響防故障備用行爲。然而，當資源成功啓動後，NONE、SOFT 與 HARD 不會影響資源監視器使用 scha_control(1HA) 或 scha_control(3HA) 所引發的後續資源的重新啓動或停止行爲。NONE (預設值) 指示 RGM 在方法失敗時設定資源狀態，並等待使用者介入。SOFT 指示在 Start 方法失敗時，RGM 重新將資源群組定位至其他節點。如果 Stop 或 Monitor_stop 方法失敗，RGM 會將資源設定爲 Stop_failed 狀態，並將資源群組設定爲 Error_stop_failed 狀態。然後，RGM 等待使用者介入。對於Stop 或 Monitor_stop 失敗，NONE 和 SOFT 是相同的。HARD 指示在 Start 方法失敗時，RGM 將重新定位群組。如果 Stop 或 Monitor_stop 方法失敗，則 RGM 會透過中斷叢集節點來停止資源。當 Start 或 Prenet_start 方法失敗時，HARD、NONE、SOFT 會影響防故障備用行爲。

與 NONE、SOFT 和 HARD 不同，RESTART_ONLY 和 LOG_ONLY 會影響所有防故障備用行爲，包括監視器引發的 (scha_control) 資源和資源群組的重新啓動，以及資源監視器 (scha_control) 引發的停止。RESTART_ONLY 指示監視器可以運行 scha_control 來重新啓動資源，但隨後使用 scha_control 嘗試執行資源群組的重新啓動或停止會失敗。RGM 允許在 Retry_interval 內重新啓動 Retry_count 次。如果超過 Retry_count 次，則不允許重新啓動資源。如果 Failover_mode 設定爲 LOG_ONLY，則不允許重新啓動或停止資源。將 Failover_mode 設定爲 LOG_ONLY，與將 Failover_mode 設定爲 RESTART_ONLY (Retry_count 設定爲零) 是等效的。如果 start 方法失敗，則 RESTART_ONLY 和 LOG_ONLY 與 NONE 是等效的：不會發生任何防故障備用，且資源轉爲 Start_failed 狀態。

種類： 可選用的

預設值： 沒有預設值

可調： ANYTIME

Load_balancing_policy (字串)
定義所使用的負載平衡策略之字串。此特性僅用於可延伸服務。如果在 RTR 檔案中宣告了 Scalable 特性，則 RGM 將自動建立此特性。Load_balancing_policy 可以採用以下值：

Lb_weighted (預設值)。負載是依照 Load_balancing_weights 特性中所設定的權重而在各節點中分布的。

Lb_sticky。總是將可延伸服務的給定用戶端 (由用戶端 IP 位址識別) 發送到叢集的同一個節點。

Lb_sticky_wild。總是將與萬用字元居留服務的 IP 位址連線的給定用戶端 IP 位址傳送到同一個叢集節點，而不管該 IP 位址將要導向至哪個連接埠號。

種類： 條件式/選用

預設值： Lb_weighted

可調： AT_CREATION

Load_balancing_weights (字串陣列)
僅用於可延伸資源。如果在 RTR 檔案中宣告了 Scalable 特性，則 RGM 將自動建立此特性。格式爲 *weight@node,weight@node*，其中 *weight* 是一個整數，反映分布到指定 *node* 的負載之相對部分。權重總量除以節點的權重即爲分配到該節點的負載分數值。例如，1@1,3@2 表明節點 1 接收到 1/4 的負載，節點 2 接收到 3/4 的負載。預設值空字串 ("") 設定均勻的分布。未指定有明確權重的任何節點均會接收到預設權重 1。

如果在資源類型檔案中未指定 Tunable 屬性，則此特性的 Tunable 值爲 ANYTIME。變更此特性將只修改新連接的分配。

種類： 條件式/選用

預設值： 空字串 ("")

可調： ANYTIME

適用於 Type (整數) 中的各種回呼方法之 *method*_timeout
時延 (以秒爲單位)，在此時延之後 RGM 會推斷出該方法的呼叫已失敗。

種類： 條件式/選用

預設值： 如果在 RTR 檔案中宣告了方法本身，則預設值爲 3,600 (1 小時)

可調： ANYTIME

Monitored_switch (列舉)
如果叢集管理員使用管理公用程式啓用或停用監視器，則由 RGM 設定爲 Enabled 或者 Disabled。如果設定爲 Disabled，則在再次啓用監視器之前，監視器的 Start 方法不會被呼叫。如果資源不具有監視器回呼方法，則此特性不存在。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 永遠不

Network_resources_used (字串陣列)
資源使用的邏輯主機名稱或共用位址網路資源之清單。對於可延伸服務，此特性必須是指存在於單獨資源群組的共用位址資源。對於故障轉移服務，此特性是指存在於同一資源群組之邏輯主機名稱或共用位址資源。如果在 RTR 檔案中宣告了 Scalable 特性，則 RGM 將自動建立此特性。如果在 RTR 檔案中未宣告 Scalable，則無法使用 Network_resources_used，除非在 RTR 檔案中明確宣告此特性。

如果在資源類型檔案中未指定 Tunable 屬性，則此特性的 Tunable 值爲 AT_CREATION。

注意 – SUNW.Event(5) 線上說明手冊描述了如何爲 CRNP 設置此特性。

種類： 條件式/必需的

預設值： 沒有預設值

可調： AT_CREATION

每個叢集節點上的 Num_resource_restarts (整數)
您無法直接設定此特性，因爲此特性由 RGM 設定爲在過去的 *n* 秒內在此節點上對此資源所進行的 scha_control、Resource_restart 或 Resource_is_restarted 呼叫之數目。*n* 爲此資源 Retry_interval 特性的值。每當此資源執行 scha_control 停止時，無論停止嘗試是成功還是失敗，RGM 均會將資源重新啓動計數器重設爲零。

如果資源類型未宣告 Retry_interval 特性，則 Num_resource_restarts 特性不能用於該類型的資源。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 否

每個叢集節點上的 Num_rg_restarts (整數)
您無法直接設定此特性，RGM 將其設定爲在過去的 *n* 秒內在此節點上由此資源對包含的資源群組所進行的 scha_control Restart 呼叫之數目，其中 *n* 爲此資源 Retry_interval 特性的值。如果資源類型未宣告 Retry_interval 特性，則 Num_rg_restarts 特性不能用於該類型的資源。

種類： 請參閱說明

預設值： 沒有預設值

可調： 否

On_off_switch (列舉)
如果叢集管理員使用管理公用程式啓用或停用資源，則由 RGM 設定爲 Enabled 或者 Disabled。如果停用，則會使資源離線，並在再次啓用此資源之前，不會呼叫任何回呼。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 永遠不

`Port_list` (字串陣列)
伺服器在其上進行偵聽的通訊埠編號之清單。附加至各連接埠號的是一條斜線 (/)，其後跟隨該連接埠所使用的協定，例如，`Port_list=80/tcp` 或 `Port_list=80/tcp6,40/udp6`。您可以指定下列協定值：

- `tcp` (用於 TCP IPv4)
- `tcp6` (用於 TCP IPv6)
- `udp` (用於 UDP IPv4)
- `udp6` (用於 UDP IPv6) 如果在 RTR 檔案中宣告了 `Scalable` 特性，則 RGM 會自動建立 `Port_list`。否則，此特性是不可用的，除非在 RTR 檔案中明確宣告該特性。

在「*Sun Cluster Data Service for Apache Guide for Solaris OS*」中，有爲 Apache 設置此特性的描述。

種類： 條件式/必需的

預設值： 沒有預設值

可調： `AT_CREATION`

`R_description` (字串)
資源的簡單說明。

種類： 可選用的

預設值： 空字串

可調： `ANYTIME`

`Resource_dependencies` (字串陣列)
相同群組或不同群組中的資源之清單，此資源對這些資源具有牢固依賴性。如果清單中的任一資源不在線上，則無法啓動此資源。如果此資源與清單中的一個資源同時啓動，則 RGM 會等到清單中的資源啓動之後再啓動此資源。如果此資源的 `Resource_dependencies` 清單中的資源未啓動，則此資源也會保持離線狀態。此資源的清單中的資源可能不啓動，原因是清單中資源的資源群組保持離線狀態，或清單中的資源處於 `Start_failed` 狀態。如果此資源因爲依賴於其他啓動失敗的資源群組中的資源而保持離線狀態，則此資源的群組會進入 `Pending_online_blocked` 狀態。

如果讓此資源與清單中的那些資源同時離線，則此資源會先於清單中的那些資源停止。然而，如果此資源保持離線狀態或停止失敗，則清單中位於其他資源群組中的資源無論如何均會停止。除非先停用此資源，否則無法停用清單中的資源。

依預設，在資源群組中，應用程式資源對網路位址資源有隱式牢固資源依賴性。第 222 頁的 「資源群組特性」中的 `Implicit_network_dependencies` 包含更多資訊。

在資源群組內，將以依賴性順序在運行 `Start` 方法之前運行 `Prenet_start` 方法。以依賴性順序在運行 `Stop` 方法之後運行 `Postnet_stop` 方法。在不同的資源群組中，依賴者等待所依賴的資源先完成 `Prenet_start` 和 `Start`，然後運行 `Prenet_start`。所依賴的資源等待依賴者先完成 `Stop` 和 `Postnet_stop`，然後運行 `Stop`。

種類： 可選用的

預設值： 空清單

可調： ANYTIME

`Resource_dependencies_restart` (字串陣列)
相同群組或不同群組中的資源之清單，此資源對這些資源具有重新啓動依賴性。

如果重新啓動依賴性清單的任何資源重新啓動，則此資源會重新啓動，除此之外，此特性與 `Resource_dependencies` 的工作方式相同。在清單中的資源重新上線之後，此資源會重新啓動。

種類： 可選用的

預設值： 空清單

可調： ANYTIME

`Resource_dependencies_weak` (字串陣列)
相同群組或不同群組中的資源之清單，此資源對這些資源具有弱依賴性。弱依賴性決定方法呼叫的順序。RGM 先呼叫清單中資源的 `Start` 方法，然後呼叫此資源的 `Start` 方法。RGM 先呼叫此資源的 `Stop` 方法，然後呼叫清單中那些資源的 `Stop` 方法。如果清單中的那些資源啓動失敗或保持離線狀態，則此資源仍可以啓動。

如果此資源與其 `Resource_dependencies_weak` 清單中的某個資源同時啓動，則 RGM 會等到清單中的資源啓動之後再啓動此資源。如果清單中的資源未啓動 (例如，清單中資源的資源群組保持離線狀態，或清單中的資源處於 `Start_failed` 狀態)，則此資源會啓動。當此資源的 `Resource_dependencies_weak` 清單中的資源啓動時，此資源的資源群組可能會暫時進入 `Pending_online_blocked` 狀態。當清單中的所有資源已啓動或啓動失敗時，此資源會啓動，且其群組會重新進入 `Pending_online` 狀態。

如果讓此資源與清單中的那些資源同時離線，則此資源會先於此清單中的那些資源停止。如果此資源保持離線狀態或停止失敗，則清單中的資源無論如何均會停止。除非先停用此資源，否則無法停用清單中的資源。

在資源群組內，將以依賴性順序在運行 `Start` 方法之前運行 `Prenet_start` 方法。以依賴性順序在運行 `Stop` 方法之後運行 `Postnet_stop` 方法。在不同的資源群組中，依賴者等待所依賴的資源先完成 `Prenet_start` 和 `Start`，然後運行 `Prenet_start`。所依賴的資源等待依賴者先完成 `Stop` 和 `Postnet_stop`，然後運行 `Stop`。

種類： 可選用的

預設值： 空清單

可調： ANYTIME

`Resource_name` (字串)
資源實例的名稱。此名稱在叢集配置中必須是唯一的，並且在建立資源後，無法對其進行變更。

種類： 必需的

預設值： 沒有預設值

可調： 永遠不

`Resource_project_name` (字串)
與資源關聯的 Solaris 專案名稱。使用此特性可將 Solaris 資源管理功能 (例如 CPU 共用和資源集區) 套用至叢集資料服務。當 RGM 使資源上線運作時，此特性便會啓動此專案名稱下的相關程序。如果未指定此特性，則會將專案名稱從包含此資源的資源群組之 `RG_project_name` 特性中取出 (請參閱 `rg_properties` (5))。如果未指定兩種特性中的任一特性，則 RGM 會使用預先定義的專案名稱 `default`。指定的專案名稱必須存在於專案資料庫中，還必需將使用者 `root` 配置爲命名專案的成員。僅在 Solaris 9 及更高版本的 Solaris 中才支援此特性。

注意 – 對此特性的變更將在下次啓動此資源時生效。

種類： 可選用的

預設值： Null (空)

可調： `ANYTIME`

每個叢集節點上的 `Resource_state` (列舉)
在每個叢集節點上由 RGM 決定的資源狀態。可能的狀態爲 `Online`、`Offline`、`Start_failed`、`Stop_failed`、`Monitor_failed`、`Online_not_monitored`、`Starting` 與 `Stopping`。

您無法配置這個特性。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 永遠不

`Retry_count` (整數)
資源失敗時監視器可以嘗試重新啓動資源的次數。此特性由 RGM 建立，僅當在 RTR 檔案中宣告了此特性時，管理員才可以使用它。如果在 RTR 檔案中指定了預設值，則 `Retry_count` 是可選擇的。

如果在資源類型檔案中未指定 `Tunable` 屬性，則此特性的 `Tunable` 值爲 `WHEN_DISABLED`。

如果在 RTR 檔案中宣告了此特性，且未指定 `Default` 屬性，則此特性是必需的。

種類： 條件式

預設值： 沒有預設值

可調： `WHEN_DISABLED`

`Retry_interval` (整數)
試圖重新啓動失敗資源的間隔秒數。資源監視器將此特性與 `Retry_count` 配合使用。此特性由 RGM 建立，僅當在 RTR 檔案中宣告了此特性時，管理員才可以使用它。如果在 RTR 檔案中指定了預設值，則 `Retry_interval` 是可選擇的。

如果在資源類型檔案中未指定 `Tunable` 屬性，則此特性的 `Tunable` 值爲 `WHEN_DISABLED`。

如果在 RTR 檔案中宣告了此特性，且未指定 `Default` 屬性，則此特性是必需的。

種類： 條件式

預設值： 沒有預設值

可調： `WHEN_DISABLED`

`Scalable` (布林值)
指示資源是否是可延伸的，也就是說，資源是否使用 Sun Cluster 的網路負載平衡功能。

如果在 RTR 檔案中宣告了此特性，則 RGM 將爲該類型的資源自動建立以下可延伸服務特性：`Affinity_timeout`、`Load_balancing_policy`、`Load_balancing_weights`、`Network_resources_used`、`Port_list`、`UDP_affinity` 與 `Weak_affinity`。這些特性均有其預設值，除非在 RTR 檔案中對它們進行了明確宣告。在 RTR 檔案中宣告 `Scalable` 時，其預設值爲 `TRUE`。

如果在 RTR 檔案中宣告此特性，則不允許爲其指定除 `AT_CREATION` 之外的 Tunable 屬性。

如果在 RTR 檔案中未宣告此特性，則資源是不可延伸的，您無法調校此特性，並且 RGM 不會設定任何可延伸服務特性。然而，您可以在 RTR 檔案中明確宣告 `Network_resources_used` 特性與 `Port_list` 特性，因爲這兩種特性在非可延伸服務與及可延伸服務中均有用。

您可以將此資源特性與 `Failover` 資源類型特性結合使用，如下所示：

在 `r_properties`(5) 中更加詳細地描述了如何將此資源特性與 `Failover` 資源類型特性結合使用。

種類： 可選用的

預設值： 沒有預設值

可調： `AT_CREATION`

每個叢集節點上的 `Status` (列舉)
由資源監視器使用 `scha_resource_setstatus`(1HA) 或 `scha_resource_setstatus`(3HA) 所設定。可能的值爲 `OK`、`degraded`、`faulted`、`unknown` 與 `offline`。使資源上線或離線時，如果此資源的監視器或方法未設定 `Status` 的值，則 RGM 會自動設定 `Status` 的值。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 永遠不

每個叢集節點上的 Status_msg (字串)
資源監視器設定 Status 特性的同時進行設定。使資源上線或離線時，如果此資源的方法未設定此特性，則 RGM 會自動將此特性重設爲空字串。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 永遠不

Thorough_probe_interval (整數)
在兩次資源高耗用時間故障探測的呼叫之間的秒數。此特性由 RGM 建立，僅當在 RTR 檔案中宣告了此特性時，管理員才可以使用它。如果在 RTR 檔案中指定了預設值，則 Thorough_probe_interval 是可選擇的。

如果在資源類型檔案中未指定 Tunable 屬性，則此特性的 Tunable 值爲 WHEN_DISABLED。

如果在 RTR 檔案的特性宣告中未指定 Default 性質，則此特性是必需的。

種類： 條件式

預設值： 沒有預設值

可調： WHEN_DISABLED

Type (字串)
此資源的資源類型就是一個實例。

種類： 必需的

預設值： 沒有預設值

可調： 永遠不

Type_version (字串)
指定當前與此資源相關聯的資源類型之版本。RGM 自動建立這個無法在 RTR 檔案中宣告的特性。此特性的值與資源類型的 RT_version 特性相等。建立資源時，Type_version 特性未明確指定，儘管它可能作爲資源類型名稱的後綴出現。編輯資源時，可以將 Type_version 變更爲新的值。

此特性的可調性源自下列來源：

- 目前版本的資源類型
- RTR 檔案中的 #$upgrade_from 指令

種類： 請參閱說明

預設值： 沒有預設值

可調： 請參閱說明

UDP_affinity (布林值)
如果爲 True，則會將給定用戶端的所有 UDP 通訊發送到目前處理用戶端的所有 TCP 通訊之同一伺服器節點。

僅當 Load_balancing_policy 是 Lb_sticky 或者 Lb_sticky_wild 時，此特性才是有意義的。此外，必須將 Weak_affinity 設定爲 FALSE (預設值)。

此特性僅用於可延伸服務。

種類： 可選用的

預設值： 沒有預設值

可調： WHEN_DISABLED

Weak_affinity (布林值)
如果爲 True，則啓用用戶端相似性的弱化形式。用戶端相似性的弱化形式允許將自給定用戶端的連線傳送到同一伺服器節點，以下情況除外：

- 例如，當伺服器偵聽程式由於故障監視器重新啓動、資源的防故障備用或切換保護移轉或者節點在發生故障後重新連結叢集而啓動時
- 當可延伸資源的 Load_balancing_weights 由於某個管理動作而變更時

弱相似性在記憶體消耗與處理器週期方面爲預設形式提供了低耗用時間的選擇。

僅當 Load_balancing_policy 是 Lb_sticky 或者 Lb_sticky_wild 時，此特性才是有意義的。

此特性僅用於可延伸服務。

種類： 可選用的

預設值： 沒有預設值

可調： WHEN_DISABLED

資源群組特性

下列資訊描述 Sun Cluster 所定義的資源群組特性。對特性值進行分類，如下所示 (在 [種類] 之後)：

- **必需的** — 管理員必須在使用管理公用程式建立資源群組時指定一個值。
- **可選擇的** – 如果管理員在建立資源群組時未指定值，則系統將提供一個預設值。
- **僅限於查詢** – 不能由管理工具直接設定。

每個描述說明特性在初始設定之後是可更新 (Y) 還是不可更新 (N)。

首先會顯示特性名稱，其後跟隨相關描述。

Auto_start_on_new_cluster (布林值)
在新的叢集形成時，此特性不允許自動啓動資源群組。

如果已設定爲 TRUE，則資源群組管理員會在重新啓動叢集時，試圖自動啓動該資源群組以達到 Desired_primaries。如果設定爲 FALSE，則叢集重新啓動時，資源群組並不會自動啓動。

種類： 可選用的

預設值： TRUE

可調： 是

Desired_primaries (整數)
群組需要同時在其上連線的節點之數目。

如果 RG_mode 特性爲 Failover，則此特性的值不得大於 1。如果 RG_mode 特性爲 Scalable，則允許值大於 1。

種類： 可選用的

預設值： 1

可調： 是

Failback (布林值)
一個布林值，指示當叢集成員資格發生變化時，是否重新計算其上群組已連線的節點集。重新計算將導致 RGM 使群組在喜好程度低的節點上離線，在喜好程度高的節點上上線。

種類： 可選用的

預設值： FALSE

可調： 是

Global_resources_used (字串陣列)
指示叢集檔案系統是否由此資源群組中的任何資源使用。管理員可以指定的合法值是星號 (*) (指示所有全域資源) 以及空字串 (“”) (指示沒有全域資源)。

種類： 可選用的

預設值： 所有全域資源

可調： 是

Implicit_network_dependencies (布林值)
一個布林值，當爲 TRUE 時，指示 RGM 應該在群組內執行非網路位址資源對網路位址資源的隱式牢固依賴性。網路位址資源包括邏輯主機名稱和共用位址資源類型。

在可延伸資源群組中，由於可延伸資源群組不包含任何網路位址資源，因此此特性不起作用。

種類： 可選用的

預設值： TRUE

可調： 是

Maximum_primaries (整數)
群組一次可以連線的最大節點數。

如果 RG_mode 特性爲 Failover，則此特性的值不得大於 1。如果 RG_mode 特性爲 Scalable，則允許值大於 1。

種類： 可選用的

預設值： 1

可調： 是

Nodelist (字串陣列)
叢集節點的清單，在這些節點上可以依喜好程度使群組上線。這些節點被稱爲資源群組的潛在主節點或主控者。

種類： 可選用的

預設值： 所有叢集節點的清單

可調： 是

Pathprefix (字串)
叢集檔案系統中的目錄；群組中的資源可在其中寫入必要的管理檔案。某些資源可能需要此特性。使 Pathprefix 針對每個資源群組均是唯一的。

種類： 可選用的

預設值： 空字串

可調： 是

Pingpong_interval (整數)
一個非負整數值 (以秒爲單位)，RGM 使用它來確定在何處使資源群組上線。可能需要此特性的情況如下所示：

- 如果發生重新配置
- 執行 scha_control -O GIVEOVER 指令或者包含 SCHA_GIVEOVER 引數的 scha_control() 函式 發生重新配置時，如果資源群組在過去的 Pingpong_interval 秒內於特定節點上多次上線失敗，則該節點被視爲無資格託管資源群組，RGM 將尋找其他主控者。資源群組上線失敗的原因是此資源的 Start 方法或 Prenet_start 方法以非零結束或逾時。

如果呼叫資源的 scha_control 指令或函式導致資源群組在過去的 Pingpong_interval 秒內於特定節點上離線，則會由於從其他節點接著呼叫 scha_control() 而使該節點無資格託管資源群組。

種類： 可選用的

預設值： 3,600 (1 小時)

可調： 是

Resource_list (字串陣列)
群組中包含的資源之清單。管理員不會直接設定這個特性。相反，RGM 在管理員於資源群組內新增或移除資源時更新此特性。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 否

`RG_affinities` (字串)
RGM 嘗試在其他給定資源群組的目前主節點 (正相似性) 上尋找資源群組，或者在給定資源群組的非目前主節點 (負相似性) 上尋找資源群組。

您可以將 `RG_affinities` 設定爲下列字串：

- ++ 或強的正相似性
- + 或弱的正相似性
- - 或弱的負相似性
- -- 或強的負相似性
- +++ 或具有防故障備用委派的強的正相似性 例如，

`RG_affinities=+RG2,--RG3` 指示此資源群組的 RG2 具有弱的正相似性，RG3 具有強的負相似性。

「*Sun Cluster Data Services Planning and Administration Guide for Solaris OS*」的「Administering Data Service Resources」中對使用 `RG_affinities` 進行了描述。

種類： 可選用的

預設值： 空字串

可調： 是

`RG_dependencies` (字串陣列)
可選擇的資源群組清單，這些資源群組指示在同一節點上使其他群組上線或離線的喜好排序。所有強的 `RG_affinities` (正、負) 及 `RG_dependencies` 的圖形均不可包含循環。

例如，假定資源群組 RG2 列於資源群組 RG1 的 `RG_dependencies` 清單中。換言之，假定 RG1 對 RG2 具有資源群組依賴性。以下清單概述了此資源群組依賴性的效果：

- 當某節點連結叢集時，直到該節點的所有 `Boot` 方法在 RG2 的資源上完成之後，才會於 RG1 的資源上運行該節點的 `Boot` 方法。
- 如果 RG1 和 RG2 在同一節點上同時處於 `Pending_online` 狀態，則直到 RG2 中的所有資源完成其啓動方法之後，才會於 RG1 中的任何資源上運行啓動方法 (`Prenet_start` 或 `Start`)。
- 如果 RG1 和 RG2 在同一節點上同時處於 `Pending_offline` 狀態，則直到 RG1 中的所有資源完成其停止方法之後，才會於 RG2 中的任何資源上運行停止方法 (`Stop` 或 `Postnet_stop`)。
- 如果在嘗試切換 RG1 或 RG2 的主節點時，使 RG1 在任一節點上線，而 RG2 在所有節點離線，則切換主節點會失敗。`scswitch`(1M) 和 `scsetup`(1M) 包含更多資訊。
- 如果在 RG2 上將 `Desired_primaries` 特性設定爲零，則不允許在 RG1 上將 `Desired_primaries` 特性設定爲大於零的值。
- 如果在 RG2 上將 `Auto_start_on_new_cluster` 特性設定爲 `FALSE`，則不允許在 RG1 上將 `Auto_start_on_new_cluster` 特性設定爲 `TRUE`。

種類： 可選用的

預設值： 空清單

可調： 是

RG_description (字串)
資源群組的簡單說明。

種類： 可選用的

預設值： 空字串

可調： 是

RG_is_frozen (布林值)
一個布林值，指示是否對資源群組所依賴的全域裝置進行故障保護移轉。如果此特性設定爲 TRUE，則對全域裝置進行故障保護移轉。如果此特性設定爲 FALSE，則不對全域裝置進行故障保護移轉。資源群組依賴全域裝置，如其 Global_resources_used 特性所指示。

您不會直接設定 RG_is_frozen 特性。在全域裝置的狀態變更時，RGM 會更新 RG_is_frozen 特性。

種類： 可選用的

預設值： 沒有預設值

可調： 否

RG_mode (列舉)
指示資源群組是故障轉移群組還是可延伸群組。如果該值爲 Failover，則 RGM 將群組的 Maximum_primaries 特性設定爲 1，並限制資源群組由單一節點主控。

如果此特性的值爲 Scalable，則 RGM 會允許 Maximum_primaries 特性的值大於 1。因此，群組可由多個節點同時主控。RGM 不允許將其 Failover 特性爲 TURE 的資源加入其 RG_mode 爲 Scalable 的資源群組。

如果 Maximum_primaries 爲 1，則預設值爲 Failover。如果 Maximum_primaries 大於 1，則預設值爲 Scalable。

種類： 可選用的

預設值： 依賴於 Maximum_primaries 的值

可調： 否

RG_name (字串)
資源群組的名稱。此名稱在叢集中必須是唯一的。

種類： 必需的

預設值： 沒有預設值

可調： 否

RG_project_name (字串)
與資源群組關聯的 Solaris 專案名稱。使用此特性可將 Solaris 資源管理功能 (例如 CPU 共用和資源集區) 套用至叢集資料服務。當 RGM 使資源群組上線時，此特性會

針對未設定 Resource_project_name 特性的資源啓動此專案名稱下的相關程序。指定的專案名稱必須存在於專案資料庫中，還必需將使用者 root 配置爲命名專案的成員。

僅在 Solaris 9 及更高版本的 Solaris 中才支援此特性。

注意 – 對此特性的變更將在下次啓動此資源時生效。

種類： 可選用的

預設值： 文字字串「default」

可調： ANYTIME

每個叢集節點上的 RG_state (列舉)

由 RGM 設定爲 Unmanaged、Online、Offline、Pending_online、Pending_offline、Pending_online_blocked、Error_stop_failed、Online_faulted 或 Pending_online_blocked，描述每個叢集節點上群組的狀態。

您無法配置這個特性。然而，您可以透過呼叫 scswitch(1M) 或者使用等效的 scsetup(1M) 或 SunPlex Manager 指令來間接設定此特性。

種類： 僅限於查詢

預設值： 沒有預設值

可調： 否

RG_system (布林值)

如果資源群組的 RG_system 特性爲 TRUE，則會限制針對該資源群組及其包含的資源之特定作業。此限制旨在協助防止對重要資源群組和資源的意外修改或刪除。僅有 scrgadm(1M) 和 scswitch(1M) 指令受此特性的影響。針對 scha_control(1HA) 和 scha_control(3HA) 的作業不受影響。

在對資源群組 (或資源群組的資源) 執行限制作業之前，您必須先將資源群組的 RG_system 特性設定爲 FALSE。當您修改或刪除支援叢集服務的資源群組，或者修改或刪除此類資源群組包含的資源時，請小心操作。

其 RG_system 值設定爲 TRUE 的資源群組稱爲系統資源群組。無論 RG_system 的目前值爲何，編輯 RG_system 特性本身永遠不會受到限制。rg_properties(5) 線上說明手冊更加詳細地描述了這些限制。

種類： 可選用的

預設值： FALSE

可調： 是

資源特性性質

下列資訊描述可用來變更系統定義的特性或建立延伸特性的資源特性特性。

注意 – 您不能指定 `Null` 或者空字串 (“”) 作爲 `boolean`、`enum` 或 `int` 類型的預設值。

首先會顯示特性名稱，其後跟隨相關描述。

`Array_maxsize`
對於 `stringarray` 類型，爲所允許的陣列元素之最大數目。

`Array_minsize`
對於 `stringarray` 類型，爲所允許的陣列元素之最小數目。

`Default`
指示特性的預設值。

`Description`
一個字串附註，用於對特性作簡短描述。對於系統定義的特性，不能在 RTR 檔案中設定 `Description` 特性。

`Enumlist`
對於 `enum` 類型，爲此特性所允許的一個字串值集。

`Extension`
如果使用，則表明 RTR 檔案項目宣告一種由資源類型實施所定義的延伸特性。否則，此項目爲系統定義的特性。

`Max`
對於 `int` 類型，爲此特性所允許的最大值。

`Maxlength`
對於 `string` 類型與 `stringarray` 類型，爲所允許的最大字串長度。

`Min`
對於 `int` 類型，爲此特性所允許的最小值。

`Minlength`
對於 `string` 類型與 `stringarray` 類型，爲所允許的最小字串長度。

`Property`
資源特性的名稱。

Tunable
: 指示叢集管理員何時可以在資源中設定此特性的值。可以設定爲 NONE 或者 FALSE，以免管理員設定此特性。允許管理員調校的值爲 TRUE 或者 ANYTIME (任何時候)、AT_CREATION (僅當建立資源時) 或者 WHEN_DISABLED (當資源離線時)。若要建立其他可調情況 (例如，「停用監視時」或「離線時」)，請將此特性設定爲 ANYTIME，並使用 Validate 方法驗證資源的狀態。

  對於每個標準的資源特性，預設值均有所不同，如以下區段所示。如果在 RTR 檔案中未另行指定，則調校延伸特性的預設設定爲 TRUE (ANYTIME)。

特性的類型
: 可允許的類型包括：string、boolean、int、enum 與 stringarray。對於系統定義的特性，您不能在 RTR 檔案項目中設定類型性質。類型決定了可接受的特性值和在 RTR 檔案項目中所允許的類型特定性質。enum 類型是一個字串值集。

附錄 B

範例資料服務程式碼清單

此附錄提供了範例資料服務中每種方法的完整程式碼。還列出了資源類型註冊檔案的內容。

此附錄包含下列程式碼清單。

- 第 231 頁的 「資源類型註冊檔案清單」
- 第 234 頁的 「Start 方法」
- 第 237 頁的 「Stop 方法」
- 第 239 頁的 「gettime 公用程式」
- 第 240 頁的 「PROBE 程式」
- 第 245 頁的 「Monitor_start 方法」
- 第 247 頁的 「Monitor_stop 方法」
- 第 248 頁的 「Monitor_check 方法」
- 第 250 頁的 「Validate 方法」
- 第 253 頁的 「Update 方法」

資源類型註冊檔案清單

RTR (資源類型註冊) 檔案包含叢集管理員註冊資料服務時定義資料服務初始配置的資源和資源類型特性宣告。

範例 B–1 SUNW.Sample RTR 檔案

```
#
# Copyright (c) 1998-2004 by Sun Microsystems, Inc.
# All rights reserved.
#
# Registration information for Domain Name Service (DNS)
#
```

範例 B–1 SUNW.Sample RTR 檔案 （續）

```
#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;
```

範例 **B–1** SUNW.Sample RTR 檔案 *(續)*

```
}
{
        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 = "";
}

#
```

範例 B–1 SUNW.Sample RTR 檔案 （續）

```
# Extension Properties
#

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

# Time out value in seconds before declaring the probe as failed.
{
        PROPERTY = Probe_timeout;
        EXTENSION;
        INT;
        DEFAULT = 30;
        TUNABLE = ANYTIME;
        DESCRIPTION = "Time out value for the probe (seconds)";
}
```

Start 方法

當包含資料服務資源的資源群組在某叢集節點上線時或當啓用資源時，RGM 將在此叢集節點上呼叫 Start 方法。在範例應用程式中，Start 方法將在此節點上啓動 in.named (DNS) 常駐程式。

範例 B–2 dns_svc_start 方法

```
#!/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"
```

範例 B–2 dns_svc_start 方法　（續）

```
#############################################################################
# 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 "$@"
```

範例 B–2 dns_svc_start 方法 *(續)*

```
PMF_TAG=$RESOURCE_NAME.named
SYSLOG_TAG=$RESOURCETYPE_NAME,$RESOURCEGROUP_NAME,$RESOURCE_NAME

# Get the value of the Confdir property of the resource in order to start
# DNS. Using the resource name and the resource group entered, find the value of
# Confdir value set by the cluster administrator when adding theresource.
config_info=scha_resource_get -O Extension -R $RESOURCE_NAME \
-G $RESOURCEGROUP_NAME Confdir`
# scha_resource_get returns the "type" as well as the "value" for the extension
# properties. Get only the value of the extension property.
CONFIG_DIR=`echo $config_info | awk '{print $2}'`

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

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

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

# Get the value for Retry_count from the RTR file.
RETRY_CNT=`scha_resource_get -O Retry_Count -R $RESOURCE_NAME \
-G $RESOURCEGROUP_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"
```

範例 B–2 dns_svc_start 方法　（續）

```
fi
exit 0
```

Stop 方法

當包含 HA-DNS 資源的資源群組在叢集節點上處於離線狀態時或停用資源時，將在此節點上呼叫 Stop 方法。此方法將在該節點上停止 in.named (DNS) 常駐程式。

範例 B–3 dns_svc_stop 方法

```
#!/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] \
```

範例 B–3 dns_svc_stop 方法 (續)

```
                    "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] \
```

範例 B–3 dns_svc_stop 方法　（續）

```
          "${ARGV0} Failed to stop HA-DNS; Exiting UNSUCCESFUL"

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

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

# Successfully stopped DNS. Log a message and exit success.
logger -p ${SYSLOG_FACILITY}.info -t [$SYSLOG_TAG] \
    "HA-DNS successfully stopped"
exit 0
```

gettime 公用程式

gettime 公用程式是由 PROBE 程式使用的 C 程式，用來追蹤在探測重新啓動之間的持續時間。您必須編譯此程式，並將其置於與回呼方法相同的目錄中，即 RT_basedir 特性指向的目錄。

範例 B–4 gettime.c 公用程式

```
#
# 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 程式

PROBE 程式使用 nslookup(1M) 指令檢查資料服務的可用性。Monitor_start 回呼方法啓動此程式，而 Monitor_start 回呼方法停止此程式。

範例 B–5 dns_probe 程式

```
#!/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
        done
}
```

範例 B–5 dns_probe 程式 （續）

```
#############################################################################
# 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
                # implies that the dataservice has already
                # exceeded the maximum retries allowed under PMF.
```

範例 B–5 dns_probe 程式　(續)

```
                # 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
      elif [ $retries -ge $RETRY_COUNT ]; then
```

範例 B–5 dns_probe 程式 （續）

```
         # 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}'`
```

範例 B–5 dns_probe 程式　(續)

```
# 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
               else
```

範例 B–5 dns_probe 程式　(續)

```
                        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 方法

此方法啓動用於資料服務的 PROBE 程式。

範例 B–6 dns_monitor_start 方法

```
#!/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

        while getopts 'R:G:T:' opt
```

範例 B–6 dns_monitor_start 方法 *(續)*

```
        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 \
    -T $RESOURCETYPE_NAME
```

範例 B–6 dns_monitor_start 方法　(續)

```
# 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 方法

此方法停止用於資料服務的 PROBE 程式。

範例 B–7 dns_monitor_stop 方法

```
#!/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] \
                    "ERROR: Option $OPTARG unknown"
```

範例 B–7 dns_monitor_stop 方法 *(續)*

```
                    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 方法

此方法驗證 Confdir 特性指向之目錄的存在。每當 PROBE 方法將資料服務故障轉移至新節點和故障轉移至爲潛在主控者的檢查節點時，RGM 將呼叫 Monitor_check。

範例 B–8 dns_monitor_check 方法

```
#!/bin/ksh#
# Monitor check  Method for DNS.
#
```

範例 B–8 dns_monitor_check 方法 (續)

```
# 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`

# Parse the arguments that have been passed to this method.
```

範例 B–8 dns_monitor_check 方法 *(續)*

```
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 方法

此方法驗證 Confdir 特性指向之目錄的存在。建立資料服務時和叢集管理員更新資料服務特性時，RGM 將呼叫此方法。每當故障顯示器將資料服務故障轉移至新節點時，Monitor_check 即會呼叫此方法。

範例 B–9 dns_validate 方法

```
#!/bin/ksh
# Validate method for HA-DNS.
```

範例 B–9 dns_validate 方法　（續）

```
# 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
                        # properties, so this is a no-op.
                        ;;
```

範例 B–9 dns_validate 方法 *(續)*

```
                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

# Parse the arguments that have been passed to this method.
```

範例 B–9 dns_validate 方法　(續)

```
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 方法

RGM 呼叫 Update 方法以通知執行中的資源其特性已變更。

範例 B–10 dns_update 方法

```
#!/bin/ksh
# Update method for HA-DNS.
```

範例 B–10 dns_update 方法　(續)

```
# 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.
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.
```

範例 B–10 dns_update 方法 *(續)*

```
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
```

附錄 C

資料服務開發程式庫資源類型程式碼範例清單

本附錄列出 SUNW.xfnts 資源類型中每個方法的完整程式碼。它包含 xfnts.c 的清單，該清單包含回呼方法呼叫的子常式程式碼。本附錄中的程式碼清單如下所示。

- 第 257 頁的 「xfnts.c」
- 第 269 頁的 「xfnts_monitor_check 方法」
- 第 270 頁的 「xfnts_monitor_start 方法」
- 第 271 頁的 「xfnts_monitor_stop 方法」
- 第 272 頁的 「xfnts_probe 方法」
- 第 275 頁的 「xfnts_start 方法」
- 第 276 頁的 「xfnts_stop 方法」
- 第 277 頁的 「xfnts_update 方法」
- 第 279 頁的 「xfnts_validate 方法程式碼清單」

xfnts.c

該檔案實施 SUNW.xfnts 方法呼叫的子常式。

範例 C–1 xfnts.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts.c - Common utilities for HA-XFS
 *
 * This utility has the methods for performing the validation, starting and
 * stopping the data service and the fault monitor. It also contains the method
 * to probe the health of the data service. The probe just returns either
 * success or failure. Action is taken based on this returned value in the
 * method found in the file xfnts_probe.c
 *
```

範例 **C–1** xfnts.c　(續)

```
 */

#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():
 *
 * Do HA-XFS specific validation of the resource configuration.
```

範例 C–1 xfnts.c （續）

```
 *
 * 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
       * is missing void arg
```

範例 C–1 xfnts.c (續)

```
        */
       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.");
       rc = 1;
```

範例 C–1 xfnts.c （續）

```
      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;

   /* get the configuration directory from the confdir_list property */
```

範例 C–1 xfnts.c (續)

```
   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.
 *
```

範例 C–1 xfnts.c （續）

```
 * 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.");
      return (1);
```

範例 C–1 xfnts.c (續)

```
}

/*
 * 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);
   }

/* We rely on RGM to timeout and terminate the program */
```

範例 C–1 xfnts.c （續）

```
   } 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
mon_stop(scds_handle_t scds_handle)
```

範例 **C–1** xfnts.c　(續)

```
{

   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
    * in probe_timeout property before concluding that the probe failed.
```

範例 C–1 xfnts.c （續）

```
 */

/*
 * 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"
```

範例 **C–1** xfnts.c　(續)

```
            " %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);
```

範例 C–1 xfnts.c （續）

```
   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 方法

此方法確認基本資源類型配置有效。

範例 C–2 xfnts_monitor_check.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_check.c - Monitor Check method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_check.c 1.11 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * just make a simple validate check on the service
 */

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

   /* Process the arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
```

範例 C–2 xfnts_monitor_check.c (續)

```
    }

    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 方法

此方法啓動 xfnts_probe 方法。

範例 C–3 xfnts_monitor_start.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_start.c - Monitor Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_start.c 1.10 01/01/18
SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method starts the fault monitor for a HA-XFS resource.
 * This is done by starting the probe under PMF. The PMF tag
 * is derived as RG-name,RS-name.mon. The restart option of PMF
 * is used but not the "infinite restart". Instead
 * interval/retry_time is obtained from the RTR file.
 */

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

範例 C–3 xfnts_monitor_start.c (續)

```
   /* 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 方法

此方法停止 xfnts_probe 方法。

範例 C–4 xfnts_monitor_stop.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_monitor_stop.c - Monitor Stop method for HA-XFS
 */

#pragma ident "@(#)xfnts_monitor_stop.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * This method stops the fault monitor for a HA-XFS resource.
 * This is done via PMF. The PMF tag for the fault monitor is
 * constructed based on RG-name_RS-name.mon.
 */

int
main(int argc, char *argv[])
{

   scds_handle_t   scds_handle;
   int    rc;
```

範例 C–4 xfnts_monitor_stop.c　(續)

```
   /* 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 方法

xfnts_probe 方法檢查應用程式的可用性，並決定是故障轉移資料服務還是重新啓動資料服務。xfnts_monitor_start 回呼方法啓動該程式，而 xfnts_monitor_stop 回呼方法停止該程式。

範例 C–5 xfnts_probe.c+

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_probe.c - Probe for HA-XFS
 */

#pragma ident "@(#)xfnts_probe.c 1.26 01/01/18 SMI"

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <strings.h>
#include <rgm/libdsdev.h>
#include "xfnts.h"


/*
 * main():
 * Just an infinite loop which sleep()s for sometime, waiting for
```

範例 C–5 xfnts_probe.c+ （續）

```
 * the PMF action script to interrupt the sleep(). When interrupted
 * It calls the start method for HA-XFS to restart it.
 *
 */

int
main(int argc, char *argv[])
{
   int         timeout;
   int         port, ip, probe_result;
   scds_handle_t      scds_handle;

   hrtime_t      ht1, ht2;
   unsigned long      dt;

   scds_netaddr_list_t *netaddr;
   char   *hostname;

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* Get the ip addresses available for this resource */
   if (scds_get_netaddr_list(scds_handle, &netaddr)) {
      scds_syslog(LOG_ERR,
          "No network address resource in resource group.");
      scds_close(&scds_handle);
      return (1);
   }

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

   /*
    * Set the timeout from the X props. This means that each probe
    * iteration will get a full timeout on each network resource
    * without chopping up the timeout between all of the network
    * resources configured for this resource.
    */
   timeout = scds_get_ext_probe_timeout(scds_handle);

   for (;;) {

      /*
       * sleep for a duration of thorough_probe_interval between
       *  successive probes.
       */
```

範例 C–5 xfnts_probe.c+ (續)

```
      (void) scds_fm_sleep(scds_handle,
          scds_get_rs_thorough_probe_interval(scds_handle));

      /*
       * Now probe all ipaddress we use. Loop over
       * 1. All net resources we use.
       * 2. All ipaddresses in a given resource.
       * For each of the ipaddress that is probed,
       * compute the failure history.
       */
      probe_result = 0;
      /*
       * Iterate through the all resources to get each
       * IP address to use for calling svc_probe()
       */
      for (ip = 0; ip < netaddr->num_netaddrs; ip++) {
         /*
          * Grab the hostname and port on which the
          * health has to be monitored.
          */
         hostname = netaddr->netaddrs[ip].hostname;
         port = netaddr->netaddrs[ip].port_proto.port;
         /*
          * HA-XFS supports only one port and
          * hence obtain the port value from the
          * first entry in the array of ports.
          */
         ht1 = gethrtime(); /* Latch probe start time */
         scds_syslog(LOG_INFO, "Probing the service on "
             "port: %d.", port);

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

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

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

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

xfnts_start 方法

當包含資料服務資源的資源群組在某叢集節點上線時或當啓用資源時，RGM 將在此叢集節點上呼叫 Start 方法。xfnts_start 方法在該節點上啓動 xfs 常駐程式。

範例 C–6 xfnts_start.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_svc_start.c - Start method for HA-XFS
 */

#pragma ident "@(#)xfnts_svc_start.c 1.13 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * The start method for HA-XFS. Does some sanity checks on
 * the resource settings then starts the HA-XFS under PMF with
 * an action script.
 */

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

   /*
    * Process all the arguments that have been passed to us from RGM
    * and do some initialization for syslog
    */

   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }

   /* 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);
```

範例 C–6 xfnts_start.c　*(續)*

```
   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 方法

當在某個叢集節點上使包含 HS-XFS 資源的資源群組離線時或當停用資源時，RGM 將在該叢集節點上呼叫 Stop 方法。此方法在該節點上停止 xfs 常駐程式。

範例 C–7 xfnts_stop.c

```
/*
 * Copyright (c) 1998-2004 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"

/*
```

範例 C–7 xfnts_stop.c （續）

```
 * 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 方法

RGM 呼叫 Update 方法以通知正在執行的資源其特性已變更。在管理動作成功地設定了資源或資源群組的特性之後，RGM 將呼叫 Update。

範例 C–8 xfnts_update.c

```
#pragma ident "@(#)xfnts_update.c  1.10     01/01/18 SMI"

/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_update.c - Update method for HA-XFS
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <rgm/libdsdev.h>
```

範例 **C–8** xfnts_update.c　*(續)*

```
/*
 * 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);

   return (0);
}
```

xfnts_validate 方法程式碼清單

此方法驗證 Confdir_list 特性指向的目錄是否存在。建立資料服務時和叢集管理員更新資料服務特性時，RGM 將呼叫此方法。每當故障監視器將資料服務故障轉移至新節點時，Monitor_check 方法即會呼叫此方法。

範例 C–9 xfnts_validate.c

```
/*
 * Copyright (c) 1998-2004 by Sun Microsystems, Inc.
 * All rights reserved.
 *
 * xfnts_validate.c - validate method for HA-XFS
 */

#pragma ident "@(#)xfnts_validate.c 1.9 01/01/18 SMI"

#include <rgm/libdsdev.h>
#include "xfnts.h"

/*
 * Check to make sure that the properties have been set properly.
 */

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

   /* Process arguments passed by RGM and initialize syslog */
   if (scds_initialize(&scds_handle, argc, argv) != SCHA_ERR_NOERR)
{
      scds_syslog(LOG_ERR, "Failed to initialize the handle.");
      return (1);
   }
   rc = svc_validate(scds_handle);

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

   /* Return the result of validate method */
   return (rc);

}
```

附錄 D

合法的 RGM 名稱和值

本附錄列出針對用於Resource Group Manager (RGM) 名稱和值的合法字元之需求。

RGM 的合法名稱

RGM 的名稱分爲以下種類。

- 資源群組名稱
- 資源類型名稱
- 資源名稱
- 特性名稱
- 列舉文字列名稱

適用於除資源類型名稱以外的名稱之規則

除了資源類型名稱以外，所有名稱必須遵守下列規則。

- 必須爲 ASCII
- 必須以字母開始
- 可以包含大寫字母、小寫字母、數字、破折號 (-) 和底線 (_)
- 不得超過 255 個字元

資源類型名稱的格式

完整的資源類型名稱之格式依資源類型而定，如下所示：

- 如果資源類型的資源類型註冊 (RTR) 檔案包含 `#$upgrade` 指令，則格式如下所示：

  vendor-id.*base-rt-name*:*version*

- 如果資源類型的 RTR 檔案**不**包含 `#$upgrade` 指令，則格式如下所示：

  vendor-id.*base-rt-name*

使用句點將 *vendor-id* 與 *base-rt-name* 分隔。使用冒號將 *base-rt-name* 與 *version* 分隔。

此格式的可變項目如下所示：

vendor-id	指定供應商 ID 的前綴。供應商 ID 的前綴爲 RTR 檔案中 `Vendor_id` 資源類型特性的值。
base-rt-name	指定基本資源類型名稱。基本資源類型名稱爲 RTR 檔案中 `Resource_type` 資源類型特性的值。
version	指定版本的後綴。版本的後綴爲 RTR 檔案中 `RT_version` 資源類型特性的值。如果 RTR 檔案包含 `#$upgrade` 指令，則版本的後綴**僅僅**是完整的資源類型名稱的一部分。`#$upgrade` 指令是在 Sun Cluster 的 3.1 版中引入的。

注意 – 如果僅註冊基本資源類型名稱的一個版本，則您不必在 `scrgadm`(1M) 指令中使用完整的名稱。您可以忽略供應商 ID 的前綴、版本編號的後綴，或者前綴和後綴皆忽略。

如需關於資源類型特性的更多資訊，請參閱第 207 頁的 「資源類型特性」。

範例 D–1 包含 `#$upgrade` 指令的資源類型之完整名稱

此範例顯示 RTR 檔案中的特性如下設定的資源類型之完整名稱：

- `Vendor_id=SUNW`
- `Resource_type=sample`
- `RT_version=2.0`

此 RTR 檔案定義的資源類型之完整名稱如下所示：

```
SUNW.sample:2.0
```

範例 D–2 不包含 `#$upgrade` 指令的資源類型之完整名稱

此範例顯示 RTR 檔案中的特性如下設定的資源類型之完整名稱：

- `Vendor_id=SUNW`
- `Resource_type=nfs`

範例 D–2 不包含 `#$upgrade` 指令的資源類型之完整名稱　**(續)**

此 RTR 檔案定義的資源類型之完整名稱如下所示：

```
SUNW.nfs
```

RGM 的值

RGM 的值分爲兩個種類：特性值與描述值。兩個種類共用相同的規則，如下所示。

- 值必須是 ASCII。
- 值的最大長度爲 4 百萬位元組減 1，即 4,194,303 位元組。
- 值不能包含下列字元中的任何一個：
  - Null (空)
  - 換行
  - 逗號
  - 分號

附錄 E

不支援叢集的應用程式的要求

一般而言，非叢集支援應用程式必須滿足特定的需求，才能成爲具有高度可用性 (HA) 的候選應用程式。第 25 頁的 「分析應用程式的適當性」一節列出了這些要求。附錄提供了關於該清單中特定項目的附加詳細資訊。

透過將應用程式的資源配置到資源群組，可以使該應用程式具有高可用性。應用程式的資料置於一個具有高可用性的全域檔案系統中，在一個伺服器發生故障時，可以透過尚存的伺服器存取資料。請參閱「*Sun Cluster* **概念指南** *(***適用於** *Solaris* **作業系統***)*」中有關叢集檔案系統的資訊。

爲了網路上的用戶端進行網路存取，在邏輯主機名稱資源中配置一個邏輯網路 IP 位址，邏輯主機名稱資源與資料服務資源處於相同的資源群組。資料服務資源與網路位址資源一起進行故障轉移，導致資料服務的網路用戶端在其新主機上存取資料服務資源。

多重主機資料

高度可用的全域檔案系統的裝置是多重主機式的，便於實體主機當機時，尚存的主機之一可以存取該裝置。要使應用程式具有高可用性，其資料必須具有高可用性，因而其資料必須常駐於全域 HA 檔案系統中。

全域檔案系統掛載於作爲獨立實體建立的裝置群組上。您可以選擇將某些裝置群組作爲掛載的全域檔案系統，將其他裝置群組作爲原始裝置與資料服務 (例如 HA Oracle) 搭配使用。

應用程式可能具有指令行切換或指向資料檔位置的配置檔案。如果應用程式使用固定連線的路徑名稱，則可將路徑名稱變更爲指向全域檔案系統中檔案的符號連結，無需變更應用程式代碼。請參閱第 286 頁的 「使用多重主機資料放置的符號連結」以取得關於使用符號連結的更多詳細說明。

在最嚴重的情況下，必須修改應用程式的來源代碼，以提供某種用於指向實際資料位置的機制。您可以透過執行附加的指令行切換來完成此作業。

Sun Cluster 支援使用 UNIX® UFS 檔案系統以及在容體管理程式中配置的 HA 原始裝置。安裝和配置時，系統管理員必須指定哪些磁碟資源用於 UFS 檔案系統，哪些用於原始裝置。通常，僅資料庫伺服器和多媒體伺服器使用原始裝置。

使用多重主機資料放置的符號連結

有時，應用程式會使其資料檔的路徑名稱固定連線，而沒有覆寫固定連線路徑名稱的機制。若要避免修改應用程式代碼，有時您可以使用符號連結。

例如，假定應用程式用固定連線路徑名稱 `/etc/mydatafile` 命名其資料檔。您可以將此路徑從檔案變更爲符號連結，此符號連結的值指向其中一個邏輯主機檔案系統中的某檔案。例如，您可以將它變更爲連結至 `/global/phys-schost-2/mydatafile` 的符號連結。

如果應用程式或其管理程序之一修改資料檔名稱及其內容，則使用符號連結將會出現問題。例如，假定應用程式透過首先建立一個新的暫存檔案 `/etc/mydatafile.new` 來執行更新。然後，透過使用 `rename(2)` 系統呼叫 (或 `mv(1)` 程式) 重新命名此暫存檔案具有實際檔案名稱。透過建立暫存檔案並將其重新命名爲實際檔案，資料服務試圖確保其資料檔內容始終保持正確的格式。

但很遺憾，`rename(2)` 動作破壞了符號連結。名稱 `/etc/mydatafile` 現在是一個規則檔案，並與 `/etc` 目錄位於相同的檔案系統，不在叢集的全域檔案系統中。由於 `/etc` 檔案系統對於每個主機都是專用的，因此在故障轉移或切換保護移轉之後資料將不可用。

引起此情形的潛在問題是：現有的應用程式不知道符號連結，並且撰寫時未考慮符號連結。若要使用符號連結將資料存取重新導向邏輯主機的檔案系統，必須以不會刪除符號連結的方式來執行應用程式。因此，符號連結不能完全糾正在叢集全域檔案系統上放置資料的問題。

主機名稱

您必須確定資料服務是否需要知道執行它的伺服器的主機名稱。如果是，則可能需要修改資料服務以使用邏輯主機名稱 (即配置至邏輯主機名稱資源的主機名稱，此邏輯主機名稱資源常駐於與應用程式資源相同的資源群組)，而不是實體主機名稱。

有時，在用於資料服務的主從式協定中，伺服器會將其主機名稱作爲至用戶端的部分訊息內容傳回至用戶端。對於這樣的協定，用戶端可能會依賴此傳回主機名稱作爲聯絡伺服器時使用的主機名稱。對於故障轉移後或切換保護移轉後要使用的傳回主機名稱，主機名稱應該是資源群組的邏輯主機名稱，而不是實體主機的名稱。在這種情況下，必須修改資料服務代碼，以將邏輯主機名稱傳回用戶端。

多重主目錄主機

多重主目錄主機一詞指處於一個以上公用網路上的主機。這樣的主機具有多個主機名稱和 IP 位址。針對每個網路， 它都有一個主機名稱 - IP 位址配對。Sun Cluster 被設計爲允許主機出現在任意數目的網路上，包括只出現在一個網路上 (非多重主目錄的的情況)。正如實體主機名稱具有多個主機名稱 - IP 位址配對，每個資源群組也可以具有多個主機名稱 - IP 位址配對，每個公用網路對應一個配對。Sun Cluster 將資源群組從一個實體主機移動至另一個實體主機時，該資源群組的完整主機名稱 - IP 位址配對集也將移動。

資源群組的主機名稱 - IP 位址配對集被配置爲包含在資源群組中的邏輯主機名稱資源。建立和配置資源群組時，將由系統管理員指定這些網路位址資源。Sun Cluster 資料服務 API 包含查詢這些主機名稱 - IP 位址配對的工具。

爲 Solaris 作業系統撰寫的大多數現成資料服務常駐程式已經能夠正確處理多重主目錄主機。許多資料服務透過連結至 Solaris 萬用字元位址 `INADDR_ANY` 來進行所有的網路通訊。該連結自動使資料服務處理所有網路介面的全部 IP 位址。`INADDR_ANY` 有效地連結至機器上目前配置的所有 IP 位址。使用 `INADDR_ANY` 的資料服務常駐程式一般無須變更，即可處理 Sun Cluster 邏輯網路位址。

連結至 INADDR_ANY 與連結至特定 IP 位址

即使使用非多重主目錄主機，Sun Cluster 邏輯網路位址概念也可使機器具有多個 IP 位址。機器具有一個其實體主機的 IP 位址和目前它所主控的每個網路位址 (邏輯主機名稱) 資源的其他 IP 位址。當機器成爲網路位址資源的主控者後，它將動態地取得其他 IP 位址。當機器放棄網路位址資源的主控地位後，它將動態地放棄 IP位址。

連結至 `INADDR_ANY` 時，某些資料服務無法在 Sun Cluster 環境中正常運作。當資源群組被主控或取消主控時，這些資料服務必須將 IP 位址集動態地變更至其所連結的 IP 位址集。實現重新連結的一個策略是終止這些資料服務的啓動和停止方法，並重新啓動資料服務的常駐程式。

`Network_resources_used` 資源特性允許一般使用者配置應用程式資源應該連結到的特定網路位址資源集。對需要此功能的資源類型，必須在資源類型的 RTR 檔案中宣告 `Network_resources_used` 特性。

當 RGM 將資源群組置於線上或離線時，它依照特定的次序探測、取消探測網路位址，並根據它呼叫資料服務資源方法的時間向上或向下配置網路位址。請參閱第 39 頁的「決定要使用的 `Start` 和 `Stop` 方法」。

到資料服務的 `Stop` 方法返回時，資料服務必須已經停止使用資源群組的網路位址。同樣地，到 `Start` 方法返回時，資料服務必須已經開始使用網路位址。

如果資料服務連結至 INADDR_ANY 而非個別 IP 位址，則呼叫資料服務資源方法的次序與呼叫網路位址方法的次序無關。

如果資料服務的停止和開始方法透過終止和重新啓動資料服務的常駐程式完成其工作，則資料服務將在適當的時間停止和開始使用網路位址。

用戶端重試

對於網路用戶端，故障轉移和切換保護移轉表現爲邏輯主機當機之後快速重新啓動。理想做法爲，用戶端應用程式和主從式協定合力進行一定量的重試作業。如果應用程式和協定已經處理了單一伺服器當機和重新啓動的情況，則它們也將處理接管和切換保護移轉資源群組的情況。某些應用程式可能選擇不斷重試。較複雜的應用程式將通知使用者長時間的重試正在進行中，並讓使用者選擇是否繼續。

附錄 F

CRNP 的文件類型定義

本附錄列出了叢集重新配置通知協定 (CRNP) 的 DTD (文件類型定義)。

SC_CALLBACK_REG XML DTD

注意 – SC_CALLBACK_REG 與 SC_EVENT 使用的 NVPAIR 資料結構僅定義一次。

```
<!— SC_CALLBACK_REG XML format specification
        Copyright 2001-2004 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:

A client of the Cluster Reconfiguration Notification Protocol should use this xml format
to register initially with the service, to subsequently register for more events, to
subsequently remove registration of some events, or to remove itself from the service
entirely.

A client is uniquely identified by its callback IP and port.  The port is defined in the
SC_CALLBACK_REG element, and the IP is taken as the source IP of the registration
connection.  The final attribute of the root SC_CALLBACK_REG element is either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS, depending on which form of the
message the client is using.

The SC_CALLBACK_REG contains 0 or more SC_EVENT_REG sub-elements.

One SC_EVENT_REG is the specification for one event type.  A client may specify only the
CLASS (an attribute of the SC_EVENT_REG element), or may specify a SUBCLASS (an optional
attribute) for further granularity.  Also, the SC_EVENT_REG has as subelements 0 or more
NVPAIRs, which can be used to further specify the event.
```

```
Thus, the client can specify events to whatever granularity it wants.  Note that a client
cannot both register for and unregister for events in the same message.  However a client
can subscribe to the service and sign up for events in the same message.

Note on versioning: the VERSION attribute of each root element is marked "fixed", which
means that all message adhering to these DTDs must have the version value specified.  If a
new version of the protocol is created, the revised DTDs will have a new value for this
fixed" VERSION attribute, such that all message adhering to the new version must have the
new version number.
—>

<!— SC_CALLBACK_REG definition

The root element of the XML document is a registration message.  A registration message
consists of the callback port and the protocol version as attributes, and either an
ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, or REMOVE_EVENTS attribute, specifying the
registration type.  The ADD_CLIENT, ADD_EVENTS, and REMOVE_EVENTS types should have one or
more SC_EVENT_REG subelements.  The REMOVE_CLIENT should not specify an SC_EVENT_REG
subelement.
        ATTRIBUTES:
               VERSION        The CRNP protocol version of the message.
               PORT           The callback port.
               REG_TYPE       The type of registration.  One of:
                       ADD_CLIENT, ADD_EVENTS, REMOVE_CLIENT, REMOVE_EVENTS

        CONTENTS:
SUBELEMENTS: SC_EVENT_REG (0 or more)
—>
<!ELEMENT SC_CALLBACK_REG (SC_EVENT_REG*)>
<!ATTLIST SC_CALLBACK_REG
       VERSION        NMTOKEN                                                  #FIXED
       PORT           NMTOKEN                                                  #REQUIRED
       REG_TYPE       (ADD_CLIENT|ADD_EVENTS|REMOVE_CLIENT|REMOVE_EVENTS)      #REQUIRED

>
<!— SC_EVENT_REG definition

The SC_EVENT_REG defines an event for which the client is either registering or
unregistering interest in receiving event notifications.  The registration can be for any
level of granularity, from only event class down to specific name/value pairs that must be
present.  Thus, the only required attribute is the CLASS.  The SUBCLASS attribute, and the
NVPAIRS sub-elements are optional, for higher granularity.

Registrations that specify name/value pairs are registering interest in notification of
messages from the class/subclass specified with ALL name/value pairs present.
Unregistrations that specify name/value pairs are unregistering interest in notifications
that have EXACTLY those name/value pairs in granularity previously specified.
Unregistrations that do not specify name/value pairs unregister interest in ALL event
notifications of the specified class/subclass.

        ATTRIBUTES:
               CLASS:         The event class for which this element is registering
                              or unregistering interest.
               SUBCLASS:      The subclass of the event (optional).
```

```
        CONTENTS:
               SUBELEMENTS: 0 or more NVPAIRs.
—>

<!ELEMENT SC_EVENT_REG (NVPAIR*)>
<!ATTLIST SC_EVENT_REG
       CLASS          CDATA                #REQUIRED
       SUBCLASS       CDATA                #IMPLIED
>
```

NVPAIR XML DTD

```
<!— NVPAIR XML format specification

        Copyright 2001-2004 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        Intended Use:
              An nvpair element is meant to be used in an SC_EVENT or SC_CALLBACK_REG
              element.
—>

<!— NVPAIR definition

       The NVPAIR is a name/value pair to represent arbitrary name/value combinations.
       It is intended to be a direct, generic, translation of the Solaris nvpair_t
       structure used by the sysevent framework.  However, there is no type information
       associated with the name or the value (they are both arbitrary text) in this xml
       element.

       The NVPAIR consists simply of one NAME element and one or more VALUE elements.
       One VALUE element represents a scalar value, while multiple represent an array
       VALUE.

       ATTRIBUTES:

       CONTENTS:
               SUBELEMENTS: NAME(1), VALUE(1 or more)
—>

<!ELEMENT NVPAIR (NAME,VALUE+)>
<!— NAME definition

        The NAME is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
               Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
```

```
                parsing inside.
—>
<!ELEMENT NAME (#PCDATA)>

<!— VALUE definition
        The VALUE is simply an arbitrary length string.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary text data.  Should be wrapped with <![CDATA[...]]> to prevent XML
                parsing inside.
—>

<!ELEMENT VALUE (#PCDATA)>
```

SC_REPLY XML DTD

```
<!— SC_REPLY XML format specification

        Copyright 2001-2004 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.
—>

<!— SC_REPLY definition

        The root element of the XML document represents a reply to a message.  The reply
        contains a status code and a status message.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                STATUS_CODE:    The return code for the message.  One of the
                                following: OK, RETRY, LOW_RESOURCES, SYSTEM_ERROR, FAIL,
                                MALFORMED, INVALID_XML, VERSION_TOO_HIGH, or
                                VERSION_TOO_LOW.

                CONTENTS:
                        SUBELEMENTS: SC_STATUS_MSG(1)
—>

<!ELEMENT SC_REPLY (SC_STATUS_MSG)>
<!ATTLIST SC_REPLY
        VERSION         NMTOKEN                                         #FIXED   "1.0"
        STATUS_CODE     OK|RETRY|LOW_RESOURCE|SYSTEM_ERROR|FAIL|MALFORMED|INVALID,\
                        VERSION_TOO_HIGH, VERSION_TOO_LOW) #REQUIRED
>

<!— SC_STATUS_MSG definition
        The SC_STATUS_MSG is simply an arbitrary text string elaborating on the status
```

```
        code.  Should be wrapped with <![CDATA[...]]> to prevent XML parsing inside.

        ATTRIBUTES:

        CONTENTS:
                Arbitrary string.
—>

<!ELEMENT SC_STATUS_MSG (#PCDATA)>
```

SC_EVENT XML DTD

注意 – SC_CALLBACK_REG 與 SC_EVENT 使用的 NVPAIR 資料結構僅定義一次。

```
<!— SC_EVENT XML format specification

        Copyright 2001-2004 Sun Microsystems, Inc. All rights reserved.
        Use is subject to license terms.

        The root element of the XML document is intended to be a direct, generic,
        translation of the Solaris sysseventd message format.  It has attributes to
        represent the class, subclass, vendor, and publisher, and contains any number of
        NVPAIR elements.

        ATTRIBUTES:
                VERSION:        The CRNP protocol version of the message.
                CLASS:          The sysevent class of the event
                SUBCLASS:       The subclass of the event
                VENDOR:         The vendor associated with the event
                PUBLISHER:      The publisher of the event
        CONTENTS:
                SUBELEMENTS: NVPAIR (0 or more)
—>

<!ELEMENT SC_EVENT (NVPAIR*)>
<!ATTLIST SC_EVENT
        VERSION         NMTOKEN         #FIXED "1.0"
        CLASS           CDATA           #REQUIRED
        SUBCLASS        CDATA           #REQUIRED
        VENDOR          CDATA           #REQUIRED
        PUBLISHER       CDATA           #REQUIRED
>
```

附錄 G

CrnpClient.java 應用程式

本附錄顯示完整的 CrnpClient.java 應用程式，在第 12 章中對該應用程式進行了詳細論述。

CrnpClient.java 的內容

```
/*
 * 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;
}
```

索引

編號和符號

#$upgrade_from 指令，51
  ANYTIME，52
  AT_CREATION，52
  WHEN_DISABLED，52
  WHEN_OFFLINE，52
  WHEN_UNMANAGED，52
  WHEN_UNMONITORED，52
  可調性值，52
#$upgrade 指令，282
[建立] 畫面, Agent Builder，144
[配置] 螢幕, Agent Builder，146

A

Affinity_timeout, 資源特性，213
Agent Builder
  [建立] 畫面，144
  [配置] 螢幕，146
  Cluster Agent 模組，156
    差異，159
  rtconfig檔案，155
  二進位檔案，153
  分析應用程式，139
  支援檔案，155
  用來建立 GDS，162
  用來建立使用 GDS 的服務，166
  目錄結構，152
  安裝，139
  使用，139
  使用指令行版本建立使用 GDS 的服務，172
  指令行版本，151
Agent Builder (續)
  重複使用完成的工作，150
  套件目錄，155
  配置，139
  啓動，141, 166
  描述，18, 22
  程序檔，154
  源代碼檔案，153
  編輯產生的來源代碼，151
  線上援助頁，154
  複製現有的資源類型，150
  導覽，141
    [編輯] 功能表，144
    [檔案] 功能表，143
    功能表，143
    瀏覽，142
  輸出，170
ANYTIME, #$upgrade_from 指令，52
API, 資源管理, 參閱RMAPI
API_version, 資源類型特性，208
Array_maxsize, 資源特性特性，228
Array_minsize, 資源特性特性，228
arraymax, 資源類型遷移，49
arraymin, 資源類型遷移，49
AT_CREATION, #$upgrade_from 指令，52
Auto_start_on_new_cluster, 資源群組特性，222

B

Boot, 資源類型特性，208
Boot 方法, 使用，40, 70

C

C 程式函式, RMAPI, 65
CCR (叢集配置儲存庫), 55
Cheap_probe_interval, 資源特性, 213
Cluster Agent 模組
  Agent Builder 差異, 159
  安裝, 156
  使用, 158
  啓動, 157
  設定, 156
  描述, 156
CRNP
  Java 應用程式範例, 190
  SC_CALLBACK_REG 訊息, 183
  用戶端, 183
  用戶端與伺服器的註冊, 183
  用戶端識別程序, 183
  伺服器, 183
  伺服器回覆, 185
  伺服器事件發送, 186
  函式, 179
  協定, 180
  協定的語義, 180
  訊息類型, 182
  授權, 189
  通訊, 180
  說明, 179
  錯誤狀況, 186

D

Default, 資源特性特性, 228
Description, 資源特性特性, 228
Desired_primaries, 資源群組特性, 223
DSDL
  libdsdev.so, 17
  元件, 22
  公用程式函式, 178
  存取網路位址, 105
  何處實現, 17
  函式, 175
  故障監視, 177
  故障監視器函式, 178
  特性函式, 176
  停止資料服務, 104
  啓用 HA 本機檔案系統, 106
  啓動資料服務, 104
DSDL (續)
  通用函式, 175
  描述, 103
  程序監視設備 (PMF) 函式, 178
  概觀, 17
  資源類型實作範例
    svc_probe() 函式, 130
    X 字型伺服器, 117
    xfnts_start 方法, 120
    xfnts_stop 方法, 124
    xfnts_update 方法, 135
    決定故障監視器的動作, 132
    從 svc_start() 傳回, 122
  資源類型實施範例
    TCP 通訊埠編號, 118
  資源類型實現範例
    scds_initialize() 函式, 119
    SUNW.xfnts RTR 檔案, 119
    SUNW.xfnts 故障監視器, 128
    X 字型伺服器配置檔案, 118
    xfnts_monitor_check 方法, 127
    xfnts_monitor_start 方法, 125
    xfnts_monitor_stop 方法, 126
    xfnts_probe 主迴圈, 128
    xfnts_validate 方法, 133
    啓動服務, 120
    驗證服務, 120
  實施故障監視器, 105
  對資源類型除錯, 106
  網路資源存取函式, 176

E

Enumlist, 資源特性特性, 228
Extension, 資源特性特性, 228

F

Failback, 資源群組特性, 223
Failover, 資源類型特性, 208
Failover_mode, 資源特性, 214
Fini, 資源類型特性, 208
Fini 方法, 使用, 40, 70

G

GDS
- Child_mon_level 特性，165
- Failover_enabled 特性，166
- Log_level 特性，166
- Network_resources_used 特性，164
- Port_list 特性，164
- Probe_command 特性，165
- Probe_timeout 特性，165
- Start_command 延伸特性，163
- Start_timeout 特性，165
- Stop_command 特性，164
- Stop_signal 特性，166
- Stop_timeout 特性，165
- SUNW.gds 資源類型，161
- 必需的特性，163
- 何時使用，162
- 使用 Agent Builder 的指令行版本建立服務，172
- 使用 SunPlex Agent Builder 建立服務使用，166
- 使用的方法，162
- 使用指令來建立服務使用，170
- 定義，38
- 爲何使用，162
- 描述，161
- 與 Sun Cluster 管理指令搭配使用，162
- 與 SunPlex Agent Builder 搭配使用，162

Global_resources_used, 資源群組特性，223

H

HA 資料服務, 測試，46
halockrun, 描述，42
hatimerun, 描述，42

I

Implicit_network_dependencies, 資源群組特性，223
Init, 資源類型特性，208
Init_nodes, 資源類型特性，209
Init 方法, 使用，40
Init方法, 使用，70
Installed_nodes, 資源類型特性，209
Is_logical_hostname, 資源類型特性，209
Is_shared_address, 資源類型特性，209

J

Java, 使用 CRNP 的應用程式範例，190

L

libdsdev.so, DSDL，17
libscha.so, RMAPI，17
Load_balancing_policy, 資源特性，214
Load_balancing_weights, 資源特性，215

M

Max, 資源特性特性，228
max, 資源類型遷移，49
Maximum_primaries, 資源群組特性，223
Maxlength, 資源特性特性，228
*method*_timeout, 資源特性，215
Min, 資源特性特性，228
min, 資源類型遷移，49
Minlength, 資源特性特性，228
Monitor_check, 資源類型特性，209
Monitor_check 方法
- 使用，71
- 相容性，52

Monitor_start, 資源類型特性，209
Monitor_start 方法, 使用，71
Monitor_stop, 資源類型特性，210
Monitor_stop 方法, 使用，71
Monitored_switch, 資源特性，215

N

Network_resources_used, 資源特性，215
Nodelist, 資源群組特性，224
Num_resource_restarts, 資源特性，216
Num_rg_restarts, 資源特性，216

O

On_off_switch, 資源特性, 216

P

Pathprefix, 資源群組特性, 224
Pingpong_interval, 資源群組特性, 224
Pkglist, 資源類型特性, 210
PMF
  用途, 42
  函式, DSDL, 178
Port_list, 資源特性, 216
Postnet_start 方法, 使用, 71
Postnet_stop
  相容性, 52
  資源類型特性, 210
Prenet_start, 資源類型特性, 210
Prenet_start 方法, 使用, 71
Property, 資源特性特性, 228

R

R_description, 資源特性, 217
Resource_dependencies, 資源特性, 217
Resource_dependencies_restart, 資源特性, 218
Resource_dependencies_weak, 資源特性, 218
Resource_list, 資源群組特性, 224
Resource_name, 資源特性, 218
Resource_project_name, 資源特性, 219
Resource_state, 資源特性, 219
Resource_type, 資源類型特性, 210
Resource_type, 遷移, 50
Retry_count, 資源特性, 219
Retry_interval, 資源特性, 219
RG_affinities, 資源群組特性, 225
RG_dependencies, 資源群組特性, 225
RG_description, 資源群組特性, 226
RG_is_frozen, 資源群組特性, 226
RG_mode, 資源群組特性, 226
RG_name, 資源群組特性, 226
RG_project_name, 資源群組特性, 226
RG_state, 資源群組特性, 227
RG_system, 資源群組特性, 227
RGM
  參閱資源群組管理員
  用途, 18
  處理資源, 18
  處理資源類型, 18
  描述, 20
RMAPI, 17
  C 程式函式, 65
  libscha.so, 17
  shell 指令, 64
  元件, 21
  公用程式函式, 67
  方法引數, 68
  回呼方法, 68
  何處實現, 17
  退出碼, 69
  資源函式, 65
  資源指令, 64
  資源群組函式, 67
  資源群組指令, 65
  資源類型函式, 66
  資源類型指令, 65
  叢集函式, 67
  叢集指令, 65
RT_basedir, 資源類型特性, 211
RT_description, 資源類型特性, 211
RT_system, 資源類型特性, 211
RT_Version
  目的, 51
  何時不變更, 51
  何時變更, 51
RT_version, 資源類型特性, 211
RT_version, 遷移, 50
rtconfig 檔案, 155
RTR
  描述, 21
  檔案
    SUNW.xfnts, 119
    說明, 107
    遷移, 50
    變更, 60

S

Scalable, 資源特性, 220
scds_initialize() 函式, 119
shell 指令, RMAPI, 64

Single_instance, 資源類型特性, 211
Start, 資源類型特性, 212
Start 方法, 使用, 39, 69
Status, 資源特性, 220
Status_msg, 資源特性, 221
Stop, 資源類型特性, 212
Stop 方法
  使用, 39
  相容性, 52
Stop方法, 使用, 69
Sun Cluster
  指令, 23
  搭配使用 GDS, 162
SunPlex Agent Builder
  參閱Agent Builder
  用來建立 GDS, 162
  用來建立使用 GDS 的服務, 166
  使用指令行版本建立使用 GDS 的服務, 172
  啓動, 166
  輸出, 170
SunPlex Manager, 說明, 22
SUNW.xfnts
  RTR 檔案, 119
  故障監視器, 128
svc_probe() 函式, 130

T

TCP 連接, 使用 DSDL 故障監視, 177
Thorough_probe_interval, 資源特性, 221
Tunable, 資源特性特性, 228
Type, 資源特性, 221
Type_version, 資源類型, 221

U

UDP_affinity, 資源特性, 221
Update, 資源類型特性, 212
Update 方法
  使用, 42, 70
  相容性, 52

V

Validate, 資源類型特性, 212
Validate 方法
  升級, 53
  使用, 42, 70
  檢查要升級的屬性值, 55
Vendor_id
  區別, 50
Vendor_ID, 資源類型特性, 212
Vendor_id
  遷移, 50

W

Weak_affinity, 資源類型, 222
WHEN_DISABLED, #$upgrade_from 指令, 52
WHEN_OFFLINE, #$upgrade_from 指令, 52
WHEN_UNMANAGED, #$upgrade_from 指令, 52
WHEN_UNMONITORED, #$upgrade_from 指令, 52

X

X 字型伺服器
  定義, 117
  配置檔案, 118
xfnts_monitor_check, 127
xfnts_monitor_start, 125
xfnts_monitor_stop, 126
xfnts_start, 120
xfnts_stop, 124
xfnts_update, 135
xfnts_validate, 133
xfs伺服器, 通訊埠編號, 118

—

一般資料服務, 參閱GDS

—
—

二進位檔案, Agent Builder, 153

介
介面, 指令行，23

元
元件, RMAPI，21

公
公用程式函式
  DSDL，178
  RMAPI，67

升
升級
  資源類型的範例，57
  預設屬性值，55
  說明文件需求，56

引
引數, RMAPI 方法，68

支
支援升級, 定義的，49
支援檔案, Agent Builder，155

方
方法
  Boot，40, 70, 113
  Fini，40, 70, 113
  Init，40, 70, 113
  Monitor_check，71, 112
  Monitor_check 回呼，71
  Monitor_start，71, 111
  Monitor_start 回呼，71
  Monitor_stop，71, 111
  Monitor_stop 回呼，71
  Postnet_start，71
方法 (續)
  Postnet_start 回呼，71
  Prenet_start，71
  Prenet_start 回呼，71
  Start，39, 69, 109
  Stop，39, 69, 110
  Update，42, 70, 112
  Update 回呼，70
  Validate，42, 70, 108
  Validate回呼，70
  xfnts_monitor_check，127
  xfnts_monitor_start，125
  xfnts_monitor_stop，126
  xfnts_start，120
  xfnts_stop，124
  xfnts_update，135
  xfnts_validate，133
  回呼，42
    初始化，69
    控制，69
  等冪性，37
方法引數, RMAPI，68
方法程式碼, 變更，60

主
主要節點，20
主控者, 描述，20
主節點，20

代
代碼, RMAPI 退出，69

功
功能表
  Agent Builder，143
  Agent Builder 編輯，144
  Agent Builder 檔案，143

可
可延伸服務, 驗證，45

可延伸資源, 實施, 43
可調性限制, 說明文件需求, 56
可調性選項, 50
  ANYTIME, 52
  AT_CREATION, 52
  WHEN_DISABLED, 52
  WHEN_OFFLINE, 52
  WHEN_UNMANAGED, 52
  WHEN_UNMONITORED, 52

用
用戶端, CRNP, 183

目
目錄, Agent Builder, 155
目錄結構, Agent Builder, 152

列
列舉文字列名稱, 規則, 281

合
合法名稱, 資源群組管理員, 281-283

回
回呼方法
  Monitor_check, 71
  Monitor_start, 71
  Monitor_stop, 71
  Postnet_start, 71
  Prenet_start, 71
  RMAPI, 68
  Update, 70
  Validate, 70
  使用, 42
  初始化, 69
  控制, 69
  概觀, 17
  說明, 20

存
存取網路位址, 使用 DSDL, 105

安
安裝 Agent Builder, 139
安裝要求, 資源類型套件, 59

伺
伺服器
  CRNP, 183
  X 字型
    定義, 117
    配置檔案, 118
  xfs
    通訊埠編號, 118

完
完全合格的名稱, 如何獲取, 50

使
使用 DSDL 停止資料服務, 104
使用 DSDL 啓動資料服務, 104
使用 DSDL 對資源類型除錯, 106

函
函式
  DSDL, 175
  DSDL 公用程式, 178
  DSDL 故障監視器, 178
  DSDL 特性, 176
  DSDL 程序監視設備 (PMF), 178
  DSDL 網路資源存取, 176
  RMAPI C 程式, 65
  RMAPI 公用程式, 67
  RMAPI 資源, 65
  RMAPI 資源群組, 67
  RMAPI 資源類型, 66
  RMAPI 叢集, 67

函式 (續)
scds_initialize(), 119
svc_probe(), 130
通用 DSDL, 175

協

協定, CRNP, 180

延

延伸, 資源特性, 214
延伸特性, 宣告, 35

持

持續連接, 使用, 46

指

指令
#$upgrade, 282
#$upgrade_from, 51
halockrun, 42
hatimerun, 42
RMAPI 資源類型, 65
RTR 檔案中的放置, 51
Sun Cluster, 23
可調性限制, 51
用來建立 GDS, 162
用來建立使用 GDS 的服務, 170
預設可調性, 51
指令行
Agent Builder, 151
指令, 23

故

故障監視器
SUNW.xfnts, 128
函式, DSDL, 178
常駐程式
設計, 113
故障轉移資源, 實施, 43

相

相依性, 協調資源之間的相依性, 47

重

重複使用完成的工作, Agent Builder, 150

值

值
資源群組管理員, 283
預設屬性, 55

套

套件目錄, Agent Builder, 155

格

格式, 資源類型名稱, 282-283

特

特性
Child_mon_level, 165
Failover_enabled, 166
GDS, 必需的, 166
Log_level, 166
Network_resources_used, 164
Port_list, 164
Probe_command, 165
Probe_timeout, 165
Start_command 延伸, 163
Start_timeout, 165
Stop_command, 164
Stop_signal, 166
Stop_timeout, 165
宣告延伸, 35
宣告資源, 32

特性 (續)
宣告資源類型，30
設定資源，29, 42
設定資源類型，29
資源，213
資源特性，228
資源群組，222
資源類型，207
變更資源，42
特性名稱, 規則，281
特性函式, DSDL，17
特性值, 規則，283
特性特性, 資源，228
特性變數，148
Agent Builder 如何取代類型，150
清單，148
資源清單，148
資源群組清單，149
資源類型清單，149
語法，149

記

記錄, 加入資源，41

訊

訊息, SC_CALLBACK_REG CRNP，183
訊息記錄, 加入資源，41

退

退出碼, RMAPI，69

配

配置, Agent Builder，139

區

區別多個註冊版本, *RT_version*，50
區別供應商, *Vendor_id*，50

常

常駐程式, 設計故障監視器，113

規

規則
列舉文字列名稱，281
特性名稱，281
特性值，283
描述值，283
資源名稱，281
資源群組名稱，281

通

通用資料服務, 參閱GDS

透

透過 DSDL 啓用 HA 本機檔案系統，106

描

描述值, 規則，283

測

測試
HA 資料服務，46
資料服務，46

畫

畫面, 建立，144

程

程式設計架構，18
程式碼
變更方法，60
變更監視器，60

程序監視設備, 參閱PMF
程序管理，41
程序管理工具, 概觀，17
程序檔
Agent Builder，154
建立，166
配置，168

等

等冪性, 方法，37

註

註冊 CRNP 用戶端與伺服器，183

源

源代碼, 編輯產生的 Agent Builder，150
源代碼檔案, Agent Builder，153

資

資料服務
建立
分析適當性，25
確定介面，27
設定開發環境，28
測試，46
測試 HA，46
傳送至叢集進行測試，29
寫入，46
範例，73
Monitor_check 方法，95
Monitor_start 方法，93
Monitor_stop 方法，93
RTR 檔案，74
RTR 檔案中的延伸特性，78
RTR 檔案中的資源特性，76
Start 方法，83
Stop 方法，85
Update 方法，100
Validate 方法，96
共用功能性，79
資料服務, 範例 (續)
定義故障監視器，88
控制資料服務，83
探測程式，88
產生錯誤訊息，82
處理特性更新，96
獲取特性資訊，82
資料服務開發程式庫, 參閱DSDL
資料服務範例
Monitor_check 方法，95
Monitor_start 方法，93
Monitor_stop 方法，93
RTR 檔案，74
RTR 檔案中的延伸特性，78
RTR 檔案中的範例特性，76
Start 方法，83
Stop 方法，85
Update 方法，100
Validate 方法，96
共用功能性，79
定義故障監視器，88
控制資料服務，83
探測程式，88
產生錯誤訊息，82
處理特性更新，96
獲取特性資訊，82
資源
協調相依性，47
停止，38
將訊息記錄加入，41
啓動，38
描述，19
實施可延伸，43
實施故障轉移，43
監視，40
遷移至其他版本，53
資源 Type_version 屬性
可調性，52
編輯，52
資源Type_version 屬性，52
資源名稱, 規則，281
資源函式, RMAPI，65
資源指令, RMAPI，64
資源相依性, 協調，47
資源特性，213
Affinity_timeout，213
Cheap_probe_interval，213
Failover_mode，214

資源特性 (續)
Load_balancing_policy, 214
Load_balancing_weights, 215
*method*_timeout, 215
Monitored_switch, 215
Network_resources_used, 215
Num_resource_restarts, 216
Num_rg_restarts, 216
On_off_switch, 216
Port_list, 216
R_description, 217
Resource_dependencies, 217
Resource_dependencies_restart, 218
Resource_dependencies_weak, 218
Resource_name, 218
Resource_project_name, 219
Resource_state, 219
Retry_count, 219
Retry_interval, 219
Scalable, 220
Status, 220
Status_msg, 221
Thorough_probe_interval, 221
Type, 221
UDP_affinity, 221
存取資訊, 37
延伸, 214
宣告, 32
設定, 29, 42
變更, 42
資源特性特性, 228
Array_maxsize, 228
Array_minsize, 228
Default, 228
Description, 228
Enumlist, 228
Extension, 228
Max, 228
Maxlength, 228
Min, 228
Minlength, 228
Property, 228
Tunable, 228
類型, 229
資源群組
可延伸, 20
故障轉移, 20
特性, 20
資源群組 (續)
說明, 19
資源群組名稱, 規則, 281
資源群組函式, RMAPI, 67
資源群組指令, RMAPI, 65
資源群組特性, 222
Auto_start_on_new_cluster, 222
Desired_primaries, 223
Failback, 223
Global_resources_used, 223
Implicit_network_dependencies, 223
Maximum_primaries, 223
Nodelist, 224
Pathprefix, 224
Pingpong_interval, 224
Resource_list, 224
RG_affinities, 225
RG_dependencies, 225
RG_description, 226
RG_is_frozen, 226
RG_mode, 226
RG_name, 226
RG_project_name, 226
RG_state, 227
RG_system, 227
存取資訊, 37
資源群組管理員
參閱RGM
合法名稱, 281-283
值, 283
資源管理 API, 參閱RMAPI
資源類型
Type_version, 221
Weak_affinity, 222
升級, 53
多版本, 49
使用 DSDL 除錯, 106
說明, 19
遷移要求, 49
資源類型升級, 範例, 57
資源類型名稱
Sun Cluster 3.0, 51
不包含版本後綴, 51
版本字尾, 50
限制, 50
規則, 282-283
實作, 56
範例, 282

資源類型函式, RMAPI, 66
資源類型指令, RMAPI, 65
資源類型套件, 安裝要求, 59
資源類型特性, 207
  API_version, 208
  Boot, 208
  Failover, 208
  Fini, 208
  Init, 208
  Init_nodes, 209
  Installed_nodes, 209
  Is_logical_hostname, 209
  Is_shared_address, 209
  Monitor_check, 209
  Monitor_start, 209
  Monitor_stop, 210
  Pkglist, 210
  Postnet_stop, 210
  Prenet_start, 210
  Resource_type, 210
  RT_basedir, 211
  RT_description, 211
  RT_system, 211
  RT_version, 211
  Single_instance, 211
  Start, 212
  Stop, 212
  Update, 212
  Validate, 212
  Vendor_ID, 212
  宣告, 30
  設定, 29
資源類型註冊, 參閱RTR
資源類型監視器, 實作, 56

預

預設特性值
  要升級的新值, 55
  繼承時, 55
預設屬性值
  Sun Cluster 3.0, 56
  升級, 55
  叢集配置儲存庫, 55

實

實作
  資源類型名稱, 56
  資源類型監視器, 56
實施, 故障監視器針對 DSDL, 105
實現, RMAPI, 17

監

監視器程式碼, 變更, 60

管

管理指令, 用來建立使用 GDS 的服務, 170

網

網路資源存取函式, DSDL, 176

語

語法
  列舉文字列名稱, 281
  特性名稱, 281
  特性值, 283
  描述值, 283
  資源名稱, 281
  資源群組名稱, 281
  資源類型名稱, 282-283

說

說明文件需求
  可調性限制, 56
  要升級, 56

寫

寫入資料服務, 46

範

範例
- 使用 CRNP 的 Java 應用程式，190
- 資料服務，73
- 資源類型升級，57

範例 DSDL
- `scds_initialize()` 函式，119
- `SUNW.xfnts` RTR 檔案，119
- `SUNW.xfnts` 故障監視器，128
- `svc_probe()` 函式，130
- TCP 通訊埠編號，118
- X 字型伺服器，117
- X 字型伺服器配置檔案，118
- `xfnts_monitor_check` 方法，127
- `xfnts_monitor_start` 方法，125
- `xfnts_monitor_stop` 方法，126
- `xfnts_probe` 主迴圈，128
- `xfnts_start` 方法，120
- `xfnts_stop` 方法，124
- `xfnts_update` 方法，135
- `xfnts_validate` 方法，133
- 決定故障監視器的動作，132
- 從 `svc_start()` 傳回，122
- 啓動服務，120
- 驗證服務，120

編

編輯產生的 Agent Builder 代碼，151

線

線上援助頁, Agent Builder，154

複

複製現有的資源類型, Agent Builder，150

遷

遷移資源類型，49

導

導覽 Agent Builder，141

螢

螢幕, 配置，146

選

選項, 可調性，50

錯

錯誤狀況, CRNP，186

檔

檔案
- Agent Builder 中的二進位，153
- Agent Builder 中的支援，155
- Agent Builder 中的源，153
- `rtconfig`，155

檢

檢查, 驗證可延伸服務，45

獲

獲取完全合格的名稱，50

叢

叢集函式, RMAPI，67
叢集指令, RMAPI，65
叢集重新配置通知協定, 參閱CRNP
叢集配置儲存庫，55

瀏
瀏覽, Agent Builder，142

類
類型, 資源特性特性，229

屬
屬性值, 預設，55

變
變數
Agent Builder 如何取代特性的類型，150
特性，148
特性清單，148
特性語法，149
資源特性清單，148
資源群組特性清單，149
資源類型特性清單，149

驗
驗證檢查, 可延伸服務，45