Developing System Services in Oracle® Solaris 11.4

Part No: E61677
November 2020

ORACLE®

Developing System Services in Oracle Solaris 11.4

Part No: E61677

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Référence: E61677

Restrictions de licence/Avis d'exclusion de responsabilité en cas de dommage indirect et/ou consécutif

Ce logiciel et la documentation qui l'accompagne sont protégés par les lois sur la propriété intellectuelle. Ils sont concédés sous licence et soumis à des restrictions d'utilisation et de divulgation. Sauf stipulation expresse de votre contrat de licence ou de la loi, vous ne pouvez pas copier, reproduire, traduire, diffuser, modifier, accorder de licence, transmettre, distribuer, exposer, exécuter, publier ou afficher le logiciel, même partiellement, sous quelque forme et par quelque procédé que ce soit. Par ailleurs, il est interdit de procéder à toute ingénierie inverse du logiciel, de le désassembler ou de le décompiler, excepté à des fins d'interopérabilité avec des logiciels tiers ou tel que prescrit par la loi.

Exonération de garantie

Les informations fournies dans ce document sont susceptibles de modification sans préavis. Par ailleurs, Oracle Corporation ne garantit pas qu'elles soient exemptes d'erreurs et vous invite, le cas échéant, à lui en faire part par écrit.

Avis sur la limitation des droits

Si ce logiciel, ou la documentation qui l'accompagne, est livré sous licence au Gouvernement des Etats-Unis, ou à quiconque qui aurait souscrit la licence de ce logiciel pour le compte du Gouvernement des Etats-Unis, la notice suivante s'applique :

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

Avis sur les applications dangereuses

Ce logiciel ou matériel a été développé pour un usage général dans le cadre d'applications de gestion des informations. Ce logiciel ou matériel n'est pas conçu ni n'est destiné à être utilisé dans des applications à risque, notamment dans des applications pouvant causer un risque de dommages corporels. Si vous utilisez ce logiciel ou matériel dans le cadre d'applications dangereuses, il est de votre responsabilité de prendre toutes les mesures de secours, de sauvegarde, de redondance et autres mesures nécessaires à son utilisation dans des conditions optimales de sécurité. Oracle Corporation et ses affiliés déclinent toute responsabilité quant aux dommages causés par l'utilisation de ce logiciel ou matériel pour des applications dangereuses.

Marques

Oracle et Java sont des marques déposées d'Oracle Corporation et/ou de ses affiliés. Tout autre nom mentionné peut correspondre à des marques appartenant à d'autres propriétaires qu'Oracle.

Intel et Intel Inside sont des marques ou des marques déposées d'Intel Corporation. Toutes les marques SPARC sont utilisées sous licence et sont des marques ou des marques déposées de SPARC International, Inc. AMD, Epyc, et le logo AMD sont des marques ou des marques déposées d'Advanced Micro Devices. UNIX est une marque déposée de The Open Group.

Avis d'exclusion de responsabilité concernant les services, produits et contenu tiers

Ce logiciel ou matériel et la documentation qui l'accompagne peuvent fournir des informations ou des liens donnant accès à des contenus, des produits et des services émanant de tiers. Oracle Corporation et ses affiliés déclinent toute responsabilité ou garantie expresse quant aux contenus, produits ou services émanant de tiers, sauf mention contraire stipulée dans un contrat entre vous et Oracle. En aucun cas, Oracle Corporation et ses affiliés ne sauraient être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'accès à des contenus, produits ou services tiers, ou à leur utilisation, sauf mention contraire stipulée dans un contrat entre vous et Oracle.

Date de publication et mention de la version préliminaire de Disponibilité Générale ("Pre-GA")

Version préliminaire de Disponibilité Générale ("Pre-GA") : 15.01.2020

Avis sur la version préliminaire de Disponibilité Générale ("Pre-GA") de la documentation

Si ce document est fourni dans la Version préliminaire de Disponibilité Générale ("Pre-GA") à caractère public ou privé :

Cette documentation est fournie dans la Version préliminaire de Disponibilité Générale ("Pre-GA") et uniquement à des fins de démonstration et d'usage à titre préliminaire de la version finale. Celle-ci n'est pas toujours spécifique du matériel informatique sur lequel vous utilisez ce logiciel. Oracle Corporation et ses affiliés déclinent expressément toute responsabilité ou garantie expresse quant au contenu de cette documentation. Oracle Corporation et ses affiliés ne sauraient en aucun cas être tenus pour responsables des pertes subies, des coûts occasionnés ou des dommages causés par l'utilisation de cette documentation.

Mention sur les informations confidentielles Oracle

INFORMATIONS CONFIDENTIELLES ORACLE. Destinées uniquement à un usage autorisé. Ne pas distribuer à des tiers.

Avis sur la reconnaissance du revenu

Si ce document est fourni dans la Version préliminaire de Disponibilité Générale ("Pre-GA") à caractère privé :

Les informations contenues dans ce document sont fournies à titre informatif uniquement et doivent être prises en compte en votre qualité de membre du customer advisory board ou conformément à votre contrat d'essai de Version préliminaire de Disponibilité Générale ("Pre-GA") uniquement. Ce document ne constitue en aucun cas un engagement à fournir des composants, du code ou des fonctionnalités et ne doit pas être retenu comme base d'une quelconque décision d'achat. Le développement, la commercialisation et la mise à disposition des fonctions ou fonctionnalités décrites restent à la seule discrétion d'Oracle.

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité de la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse : http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info ou le site http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs si vous êtes malentendant.

Contents

Examples

Using This Documentation

- **Overview** – Describes how to use the Oracle Solaris Service Management Facility (SMF) feature. SMF is one of the components of the wider Oracle Solaris Predictive Self Healing capability.
- **Audience** – System administrators and application developers who create custom services to configure applications
- **Required knowledge** – Experience administering Oracle Solaris systems

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E37838-01.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Introduction to Developing Service Management Facility Services

The Oracle Solaris Service Management Facility (SMF) framework manages system and application services. SMF manages critical system services essential to the working operation of the system and manages application services such as a database or Web server.

SMF replaces the use of configuration files for managing services and is the recommended mechanism to use to start applications. SMF replaces the `init` scripting start-up mechanism, `inetd.conf` configurations, and most `rc?.d` scripts.

This chapter describes:

- Where to get more information about SMF
- New features in this release
- How to gain privileges you need to use some SMF commands

SMF Documentation

This guide describes how to develop an SMF service to provide service support for your application, including the following topics:

- Using the service creation tool.
- Converting `inetd.conf` configurations to SMF services.
- Converting SMF service properties to configuration files. This mechanism provides a bridge for services that are managed by SMF but interact with applications that still require configuration files.
- Creating a service that runs periodically rather than continuously, similar to a `cron` job.

See *Managing System Services in Oracle Solaris 11.4* for the following information:

- Information for system administrators such as inspecting and changing service property values, enabling service instances, troubleshooting installed services.

- Descriptions of concepts and components such as service states, service models, service restarters, service properties, service bundles, and service configuration repository.
- Descriptions of different types of dependencies and their attributes and effects.
- How to create a new instance of an existing service or modify an existing service instance.

The following resources provide additional examples of creating and delivering services to perform tasks such as application configuration:

- Chapter 6, "Automating System Change as Part of Package Operations" in *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.4*
- Chapter 7, "Modifying Packages" in *Packaging and Delivering Software With the Image Packaging System in Oracle Solaris 11.4*

New Features in This Release

The following SMF features are new in this release. In many cases, more information is available in *Managing System Services in Oracle Solaris 11.4*.

Additional repository layers

Three new repository layers enable finer grained specification of configuration.

1. `enterprise-profile` layer – Configuration that applies across all of the Oracle Solaris systems for an enterprise.
2. `node-profile` layer – Configuration that is specific to a particular Oracle Solaris instance.

   The existing `site-profile` layer had been used for any configuration that was not delivered by Oracle Solaris, such as configuration delivered by `sysconfig` or by user profiles. In Oracle Solaris 11.4, the `site-profile` layer is for configuration that is common to many systems at the same location or site. Most configuration that previously belonged to the `site-profile` layer now belongs to the `node-profile` layer.

   When a system is initially updated to Oracle Solaris 11.4, all profiles in the `/etc/svc/profile/site/` directory and the `/etc/svc/profile/site.xml` profile, if it exists, are moved to the `/etc/svc/profile/node/` directory. The configuration these profiles describe is then part of the `node-profile` layer. Moved profiles are named to indicate that they were moved and where they were previously located. For example, the `/etc/svc/profile/site/sc_profile.xml` profile might be moved to `/etc/svc/profile/node/migrated_etc_svc_profile_site_sc_profile.xml`, and `/etc/svc/profile/site.xml` might be moved to `/etc/svc/profile/node/migrated_etc_svc_profile_site.xml`.

Any file in the `/etc/svc/profile/site/` directory that was delivered by an IPS package is not moved, and that configuration remains part of the `site-profile` layer.

3. `sysconfig-profile` layer – Configuration that is specified by using the `sysconfig` command interactively, and any configuration that is specified in a profile that is installed by an AI client or by the `zoneadm` (`install`, `attach`, or `clone`) command.

   When a system is initially updated to Oracle Solaris 11.4, all `.xml` files in the `/etc/svc/profile/sysconfig/` directory are backed up into the `/etc/svc/profile/backup/`*timestamp*`/profiles.tar` file. Some of the profiles in this directory might contain configuration performed by `sysconfig` or the Oracle Solaris installer and might contain active configuration at the `admin` layer. Profiles that contain active configuration are left in place, effectively migrating the configuration from the `admin` layer to the `sysconfig-profile` layer. Any profile in the `/etc/svc/profile/sysconfig` directory that is not referenced by the SMF repository is removed after being backed up into the `/etc/svc/profile/backup/`*timestamp*`/profiles.tar` file.

Nested property groups

The parent of a property group can be a service or service instance or another property group. Another way to describe this feature is a property group can have properties and can have any number of child property groups. Nesting property groups enables finer definition of relationships among configuration data. To identify properties in a nested property group, name all the parent property groups as well.

Property group name and property name options

The `-G` option can be used with the `svcprop` and `svccfg` commands to specify a property group. The `-P` option can be used to specify a property. The `-T` option can be used with the `svccfg` command to specify a property type.

Special characters in property group names and property names

Property group names and property names can contain any character defined in Uniform Resource Identifier (URI) Generic Syntax RFC 3986. See "Naming Property Groups and Properties" on page 26.

Stencil defines

If your application requires multiple configuration files that have the same syntax, you can use the stencil defines feature to use a single service to define all configuration files.

Goal services

A goal service provides a single point of monitoring for a configurable set of dependent services. Most services that cannot reach the `online` state remain silently in the `offline` state. A goal service transitions to the `maintenance` state and generates an FMA alert if any of its dependencies cannot be satisfied without administrative intervention.

`svcadm goals` command

The `svcadm goals` command configures goal dependencies for a goal service. As with other subcommands of the `svcadm` command, the `svcadm goals` command can take the `-s` option to request synchronous behavior.

`svcbundle` options

Starting with SRU 27, you can specify `delay`, `jitter`, specific method timeout (*method-name*`-timeout`), and general timeout (`timeout`) on the `svcbundle` command line.

Service Management Privileges

Exporting and developing service manifests and profiles does not require special privilege. Using the `svccfg` and `svcadm` commands to modify service state and configuration requires increased privilege. Use one of the following methods to gain the privilege you need. See *Securing Users and Processes in Oracle Solaris 11.4* for more information about roles and profiles, including how to determine which role or profile you need and how to assign privileges.

Roles

Use the `roles` command to list the roles that are assigned to you. Use the `su` command with the name of the role to assume that role. As this role, you can execute any commands that are permitted by the rights profiles that are assigned to that role. For example, if the role is assigned the Service Configuration rights profile, you can execute the `svccfg` and `svcadm` commands modify service properties and change service state.

Rights profiles

Use the `profiles` command to list the rights profiles that are assigned to you. Use one of the following methods to execute commands that your rights profiles permit you to execute:

- Use a profile shell such as `pfbash` or `pfksh`.
- Use the `pfexec` command in front of the command that you want to execute. In general, you must specify the `pfexec` command with each privileged command that you execute.

Authorizations

See the `smf_security(7)` man page for detailed information about authorizations required for SMF operations. If the Service Configuration rights profile is not sufficient to manage a particular service, inspect the service for the following properties:

- The `action_authorization`, `modify_authorization`, `read_authorization`, and `value_authorization` properties specify required authorizations. Individual services can require their own particular authorizations.
- Properties of the `method` property group can specify requirements to run the method such as the user and privilege set.

sudo command

Depending on the security policy at your site, you might be able to use the `sudo` command with your user password to execute a privileged command.

If you need to require specific privileges of administrators who want to use the service you develop, see "Securing Service Tasks" on page 31.

♦ ♦ ♦ CHAPTER 2

Using SMF to Control Your Application

This chapter presents the following information that applies to all SMF services:

- Naming rules and data type definitions
- Best practices such as storing service files in standard locations
- Service development troubleshooting information

This chapter also describes how to use the `svcbundle` tool to get started creating a new service and how to convert a run control script to an SMF service.

Subsequent chapters discuss creating specific types of services for specific purposes.

Creating an SMF Service

An SMF service consists of one or more service manifests and zero or more profile files. Service instances define methods to perform the work of the instance.

A service manifest contains the complete set of properties associated with a specific service, including instances, dependencies, application configuration properties, and methods to run when the service starts and stops. Manifests also provide template information such as a description of the service.

Profiles can define instances for a service that is already defined in a manifest. Profiles can define new properties for these service instances and new values for properties that are defined in the service manifest. Profiles cannot define template elements.

See the `service_bundle(5)` man page and the `/usr/share/lib/xml/dtd/service_bundle.dtd.1` service bundle DTD for a complete description of the contents and format of SMF manifests and profiles. See also "Naming Services and Instances" on page 26 for naming rules and information about assigning property group types, and see "Property Group Types and Property Types" on page 28 for information about the values of different property types.

A method can be a daemon, other binary executable, or an executable script. See "Creating Service Instance Methods" on page 29 for more information.

You can use multiple manifests to describe a single service. This method can be useful, for example, to define a new instance of a service without modifying the existing manifest for the service. See "Creating a Service Using Multiple Manifests" on page 35 for more information.

Use the following best practices when creating a custom service:

- Use the `site` prefix in the service name as described in "Service Names" in *Managing System Services in Oracle Solaris 11.4*. The `site` prefix is reserved for site-specific customizations. A service named `svc:/site/`*service-name* will not conflict with the services delivered in an Oracle Solaris release.
- Add name and description metadata to your manifests so that users can get information about this service from the `svcs` and `svccfg describe` commands. You can also add descriptions of property values. See the `value`, `values`, and `template` elements in the DTD.
- Use the `svccfg validate` command to validate your service manifest file or service instance FMRI.
- Use the `smf_method_exit()` function to document the successful or unsuccessful exit of a method script in the log file of the service instance.
- Store your manifest, profile, and method files in the standard locations shown in the following table.

TABLE 1 Standard Locations of Service Development Files

File	Standard Location
manifest	`/lib/svc/manifest/site`
profile	`/etc/svc/profile/site`
method	`/lib/svc/method`

Manifests and profiles stored in these locations are imported into the service configuration repository by the `svc:/system/early-manifest-import:default` service during the boot process before any services start. Running the import process early ensures that the repository will contain information from the latest manifests before the services are started. Manifests and profiles stored in these standard locations are also imported when the `svc:/system/manifest-import` service is restarted.

With your manifest, profile, and method files in standard locations, restart the `manifest-import` service to install and configure your service instances. Use the `svcs` command to check the status of your service instances.

Creating an SMF Service Using the Service Bundle Generator Tool

You can use the svcbundle service bundle generator tool to create a simple service or to start a more complex service. For more information, see the svcbundle(8) man page. You can use the service bundle DTD and other service manifests to complete a more complex service.

▼ How to Create an SMF Service Using svcbundle

Note - Do not use this procedure if you are creating a periodic service. See "Creating a Periodic Service Using the Service Bundle Generator Tool" on page 45 and "Creating a Scheduled Service Using the Service Bundle Generator Tool" on page 56.

1. **Determine the service model.**
   By default, svcbundle creates a transient service. Determine whether the start method script for this service starts any long-running daemon and therefore this service is a contract service. See "Service Models" in *Managing System Services in Oracle Solaris 11.4* and the model and startd/duration properties in the svcbundle(8) man page for information about service models.

2. **Copy the script to the standard location.**
   The service in this example uses a custom script named ex-svc as the start method. Copy this script to /lib/svc/method/ex-svc.

3. **Create an initial manifest.**
   Because this is a custom service, the service name should start with site. The name of this service is site/ex-svc.
   This service is a transient service and does not need a stop method.
   The default method timeout is 60 seconds. The start method in this example takes longer than 60 seconds to complete. To prevent the service from failing simply because the start method has not finished, this example specifies a longer timeout for the start method. Specifying the timeout option instead of start-timeout would set the timeout for all methods. Specifying timeout in addition to start-timeout would set the timeout for the refresh and stop methods.

```
$ svcbundle -o /tmp/ex-svc.xml -s service-name=site/ex-svc \
> -s start-method=/lib/svc/method/ex-svc -s start-timeout=120
```

   If this service were a contract service, you would specify contract or daemon as the value of the model or duration property, as in -s model=contract.

By default, the instance that is created is named `default` and is enabled. If you want the instance to have a different name, specify the `instance-name` property. You can also specify `instance-property`, `service-property`, `enabled`, and other properties on the `svcbundle` command line. See the `svcbundle`(8) man page for descriptions.

4. **Make any necessary changes to the manifest.**

   The `/tmp/ex-svc.xml` manifest is shown in Example 1, “Example Service Manifest,” on page 24.

   Verify that the content of the manifest is what you need. You might need to edit the manifest to add a dependency or make some other change.

   Edit the manifest to add `common_name` and `description` information in the `template` data area. You can also add `documentation` and other template data.

5. **Verify that the manifest is valid.**

   ```
   $ svccfg validate /tmp/ex-svc.xml
   ```

6. **Copy the manifest to the standard directory.**

   ```
   $ cp /tmp/ex-svc.xml /lib/svc/manifest/site/ex-svc.xml
   ```

7. **Import the manifest and start the service.**

   ```
   $ svcadm restart manifest-import
   ```

8. **List the new service.**

   Verify that the new service exists and is in the expected state.

   ```
   $ svcs ex-svc
   ```

Example 1 Example Service Manifest

The example in this procedure produced the following manifest:

```
<?xml version="1.0" ?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<!--
    Manifest created by svcbundle (2020-Oct-01 18:16:58-0700)
-->
<service_bundle name="site/ex-svc" type="manifest">
    <service name="site/ex-svc" version="1" type="service">
        <!--
            The following dependency keeps us from starting until the
```

```
            multi-user milestone is reached.
        -->
        <dependency name="multi_user_dependency" grouping="require_all"
            restart_on="none" type="service">
            <service_fmri value="svc:/milestone/multi-user"/>
        </dependency>
        <exec_method name="start" type="method" timeout_seconds="120"
            exec="/lib/svc/method/ex-svc"/>
        <!--
            The exec attribute below can be changed to a command that SMF
            should execute to stop the service.  Use svcbundle -s
            stop-method to set the attribute.
        -->
        <exec_method name="stop" type="method" timeout_seconds="60"
            exec=":true"/>
        <!--
            The exec attribute below can be changed to a command that SMF
            should execute when the service is refreshed.  Use svcbundle -s
            refresh-method to set the attribute.
        -->
        <exec_method name="refresh" type="method" timeout_seconds="60"
            exec=":true"/>
        <property_group name="startd" type="framework">
            <propval name="duration" type="astring" value="transient"/>
        </property_group>
        <instance name="default" enabled="true"/>
        <template>
            <common_name>
                <!--
                    Replace loctext content with a short name for the
                    service.
                -->
                <loctext xml:lang="C">
                        site/ex-svc
                </loctext>
            </common_name>
            <description>
                <!--
                    Replace loctext content with a brief description of the
                    service
                -->
                <loctext xml:lang="C">
                        The site/ex-svc service.
                </loctext>
            </description>
        </template>
    </service>
</service_bundle>
```

Example 2 Automatically Installing a Generated Manifest

If you do not need to make any changes to the new service manifest, you can use the `-i` option to install the manifest as soon as it is created. The `svcbundle` command will write the manifest to `/lib/svc/manifest/site` and restart the `manifest-import` service. Any existing file with the same name in the `/lib/svc/manifest/site` directory will be overwritten.

```
$ svcbundle -i -s service-name=site/ex-svc -s start-method=/lib/svc/method/ex-svc
```

Naming Services and Instances

Instance names must match the following expression:

```
([A-Za-z0-9][_A-Za-z0-9.-]*,)?[A-Za-z0-9][_A-Za-z0-9.-]*
```

Service names must match the following expression, where *name* is an instance name:

name`(/`*name*`)*`

Instance names have the following characteristics:

- Are case sensitive
- Must begin with an alphanumeric character
- Can contain alphanumeric characters, the underscore (_), the hyphen (-), and the dot (.)
- Can have a single comma (,) between the first and last character

Except for the comma, this specification is a subset of the characters defined in the Uniform Resource Identifier (URI) Generic Syntax RFC 3986.

A provider name can be included at the beginning of the service or instance name. A provider name has the following characteristics:

- Is separated from the rest of the service or instance name by a comma (,)
- Must begin with an alphanumeric character
- Can contain one or more periods (.)

Naming Property Groups and Properties

Property group and property names must match the following expression. Notice the space between the = and the `%`:

```
[A-Za-z0-9-._~:/?#[]@!$&'()*+,;= %]+
```

A property group or property name has the following characteristics:

- Is case sensitive
- Can contain any character defined in Uniform Resource Identifier (URI) Generic Syntax RFC 3986

In an FMRI, the following characters are unreserved and will appear unencoded:

```
A-Z
a-z
0-9
-
.
_
~
```

The following characters are reserved and will appear encoded when they are part of an FMRI or fragment of an FMRI. A percent (%) character must be encoded when it is used as part of a property group name or property name. See also "Using Reserved Characters" on page 27.

```
:   %3A          !   %21          +   %2B
/   %2F          $   %24          ,   %2C
?   %3F          &   %26          ;   %3B
#   %23          '   %27          =   %3D
[   %5B          (   %28      <space>   %20
]   %5D          )   %29          %   %25
@   %40          *   %2A
```

The following example shows a full FMRI for a property: the FMRI of the service instance, followed by `/:properties/`, followed by the name of the property.

```
svc:/application/pkg/server:default/:properties/pkg/port
```

You can use the `-f` option of the `svcprop` command to show the full FMRI of a property.

```
$ svcprop -fp pkg/port pkg/server:default
svc:/application/pkg/server:default/:properties/pkg/port count 80
```

Using Reserved Characters

When you use `svcbundle`, you must encode reserved characters, as shown in the following example:

```
# svcbundle -s service-name=site/enchars-example \
```

```
> -s start-method=true \
> -s instance-property=config:start%3Aend:count:10 \
> -s instance-property=config:students%2Fteachers:count:20 \
> -s instance-property=config:maximum%20%23:count:9 \
> -s instance-property=config:%25%20increase:count:10 > enchars_example.xml
```

When you edit a manifest or profile directly or use the `editprop` tool, you do not need to encode reserved characters. The following `config` property group is from the `enchars_example.xml` manifest output by the above `svcbundle` command:

```
<property_group type="application" name="config">
    <propval type="count" name="start:end" value="10"/>
    <propval type="count" name="students/teachers" value="20"/>
    <propval type="count" name="maximum #" value="9"/>
    <propval type="count" name="% increase" value="10"/>
</property_group>
```

Reserved characters always appear encoded in an FMRI.

```
$ svcprop -fp config enchars-example:default
svc:/site/enchars-example:default/:properties/config/%25%20increase count 10
svc:/site/enchars-example:default/:properties/config/maximum%20%23 count 9
svc:/site/enchars-example:default/:properties/config/start%3Aend count 10
svc:/site/enchars-example:default/:properties/config/students%2Fteachers count 20
```

Property Group Types and Property Types

A property group type is a category for the property group. Property group types include the following:

```
application
configfile
dependency
framework
implementation
method
template
```

You can introduce a new property group type. The name of a property group type follows the same rules as stated in “Naming Property Groups and Properties” on page 26. The name of a property group type is no longer than 140 characters.

When you create a property group, the type of the property group should be either `application` or a new type that you created. The property group types `configfile`, `dependency`, `framework`, `implementation`, `method`, and `template` have special use in SMF. Property groups of type

application are expected to be of interest only to the service to which this property group is attached.

The following table describes the possible values for properties of various types. This information is also available from the scf_value_create(3SCF) man page.

TABLE 2 Service Property Type Value Descriptions

Property Type	Value Description
boolean	Single bit: true or false
count	Unsigned 64-bit quantity
integer	Signed 64-bit quantity
time	Signed 64-bit seconds or signed 32-bit nanoseconds (*ns*) in the following range: 0 <= *ns* < 1,000,000,000
astring	An 8-bit NULL-terminated string
ustring	An 8-bit UTF-8 string
uri	A URI string
fmri	A Fault Management Resource Identifier
host	A host name, an IPv4 address, or an IPv6 address
hostname	A fully qualified domain name
net_addr	A valid net_addr_v4 or net_addr_v6 address
net_addr_v4	A dotted-quad IPv4 address with optional network portion
net_addr_v6	A legal IPv6 address with optional network portion

Creating Service Instance Methods

A start method performs the work of the service instance. Other methods perform tasks necessary to disable or refresh a service instance, for example.

In a service manifest or profile, a method is defined in an exec_method element that includes name and exec attributes. Possible values for the name attribute are provided in the restarter man page. For example, the master restarter, /lib/svc/bin/svc.startd, supports start, stop, and refresh methods as described in the svc.startd(8) man page.

There is no restart method. The svcadm restart and svccfg restart commands run the stop method and then the start method.

The exec attribute defines what the method will execute. Possible values for the exec attribute include a custom method script, an existing executable, or a special token defined in SMF, as shown in the following examples:

`exec='/lib/svc/method/tcsd.sh start'`

By convention, a custom script as shown in this example takes an argument that specifies the value of the `name` attribute of the method. In this way, the same script can be used for all methods of that service instance.

`exec='/usr/lib/zones/zonestatd'`

This example specifies an existing executable.

`exec=':true'`
`exec=':kill'`

The tokens `:kill` and `:true` are explained in the `smf_method(7)` man page. The `:true` token should be used for methods that are required by the restarter but that are not necessary for the particular service instance implementation.

The `:kill` token causes all processes in the primary service contract to be terminated and therefore is most appropriate for a `stop` method. In general, a `refresh` method should not be `:kill` unless the processes in the contract are programmed to handle those signals gracefully.

Service Method Scripts

A method that is a script can be a Bourne shell compatible script or a Python script, for example.

- The file `/lib/svc/share/smf_include.sh` defines many helper functions for Bourne shell compatible method scripts.
- The file `/usr/lib/`*python-version*`/vendor-packages/smf_include.py` defines many helper functions for Python method scripts, including the following functions that are unique for Python:
  - `smf_subprocess()` – Starts the specified executable in a subprocess. The process can return immediately, enabling the instance to act as a contract service.
  - `smf_main()` – Calls the appropriate function from the method script using frame inspection. See the comments in the `/usr/lib/`*python-version*`/vendor-packages/smf_include.py` file.
- The file `/lib/svc/share/smf_exit_codes.sh` defines method exit codes.

Use the `smf_method_exit()` function to document the exit of a method script in the log file of the service instance. The `smf_method_exit()` function takes an exit code, a token that summarizes the exit reason, and a string that can describe the exit in greater detail. See the `smf_method_exit(3SCF)` man page for the syntax of the `smf_method_exit()` function. See `/lib/svc/share/smf_exit_codes.sh` for the list of exit codes.

Securing Service Tasks

Use any of the following options to restrict which users can run or manage a service or which privileges a user must have to run or manage a service:

- Use *_authorization properties to specify authorizations that are required to read property values, modify service properties and property values, and perform actions on services. See the smf_security(7) man page for more information.
- Use the method_credential element of the method_context property group to specify requirements as values of the following properties. All of these properties are optional, but at least one of these properties must be set if you specify a method_credential element:
  - user. The user ID in numeric or text form. If absent or :default, uid 0 and default home directory / are used.
  - group. The group ID in numeric or text form. If absent or :default, the group associated with the user in the passwd database is used.
  - supp_groups. Supplementary group IDs to be associated with the method, separated by commas or spaces. If absent or :default, initgroups(3C) is used.
  - privileges. A comma-separated list of privileges. See the privileges(7) man page.
  - limit_privileges. A comma-separated list of privileges. See the privileges(7) man page.
  - clearance. The process clearance. The value can be ADMIN_HIGH, ADMIN_LOW, or a hexadecimal label. The default value is ADMIN_LOW.
  - trusted_path. If the value is true, the process runs in the Trusted Path Domain (TPD). The default value is false. For more information and examples, see "SMF Services in Immutable Zones" in *Creating and Using Oracle Solaris Zones*.

  In Chapter 5, "Creating Services to Manage Oracle Database Instances", the instance control service and the listener service specify that the service must be run by user oracle in group oinstall.

  The network/ntp service shows a list of required authorizations as the value of the privileges property.

  The network/physical service shows the use of the supp_groups and trusted_path properties.

  The system/console-login service shows the use of the clearance property to require ADMIN_HIGH privilege both to start and stop the service.
- Specify a rights profile for the method to use. See the MySQL and Apache examples in "Locking Down Resources by Using Extended Privileges" in *Securing Users and Processes in Oracle Solaris 11.4*.

Service Development Best Practices

Follow the guidelines described in this section as you develop your service.

Service Method Best Practices

Follow the guidelines described in this section as you develop your service start method or other service methods.

Use SMF Method Exit and Useful Exit Reason

Services are expected to return a successful status when they have completed initialization and are ready to provide the service.

To exit your start method, use `smf_method_exit()`; do not use `exit()`. The `smf_method_exit()` interface requires the following arguments:

- One of the method exit codes defined in `/lib/svc/share/smf_exit_codes.sh` or in the `smf_method(7)` man page
- A short explanation of the reason for exiting
- A longer explanation of the reason for exiting

Make sure that error messages are informative, including guidance for resolving the problem. You might need to capture messages or other information from commands called by your method. If your method is an existing executable, you might want to call that executable inside a method script to improve the exit messaging. The system administrator will see these messages in the service log file.

See the `smf_method_exit(3SCF)` man page for more information.

Use Dependencies, Avoid Using Timeouts

Do not exit your start method until initialization of the service is complete. If you exit your start method before service initialization is complete, services that depend on this service cannot be started.

Set appropriate values for `timeout_seconds` properties to avoid failing solely because more time is needed to complete the method tasks.

Do not use `timeout_seconds` values or any other kind of timeout or wait to allow enough time for dependencies to reach the `online` state. Instead, declare dependencies appropriately. In

addition to allowing enough time for dependencies to start, if a service on which this service has a `require` dependency fails, then this service should fail with appropriate messages and not continue to wait for the failed dependency to start. Again, appropriately declared `dependency` elements are the correct implementation. See "Showing Service Dependencies" in *Managing System Services in Oracle Solaris 11.4* and the "Dependencies" section of the `smf(7)` man page.

Provide Documentation

Provide appropriate template information as described in the `smf_template(7)` man page. Administrators can use the `svccfg describe` command to view this information.

- Provide a short common name for the service as described in "Service and Instance Common Names" in the `smf_template(7)` man page.
- Reference appropriate man pages (`manpage` element) or stable URLs (`doc_link` element) for more information.
- Provide names, descriptions, choices, and constraints for property groups and properties that are specific to this service.

Validate the Service Manifest

Use the `svccfg validate` command to validate your service manifest.

If the `svccfg validate` command fails with the error "Required property group missing," you might be attempting to validate a partial manifest. Specification of a single service can be spread across multiple manifests. To avoid this error, make sure all manifests for a multiple-manifest service specify the `include` property in the `service_bundle` element as described in "Creating a Service Using Multiple Manifests" on page 35.

Use Standard Locations

Copy your service manifests and profiles to standard locations with standard ownership and permissions. Do not use non-standard locations for manifest and profile files. See "Service Bundles" in *Managing System Services in Oracle Solaris 11.4* or Table 1, "Standard Locations of Service Development Files," on page 22 for manifest and profile standard locations.

When you create a service for your own use, use `site` at the beginning of the service name (`svc:/site/`*service-name*`:`*instance-name*), and place the manifest in `/lib/svc/manifest/site`.

To test your service, place the manifest and any associated profiles in the correct standard locations and restart the `manifest-import` service. See Chapter 5, "Configuring Multiple Systems" in *Managing System Services in Oracle Solaris 11.4* and "Repairing an Instance That Is Degraded, Offline, or in Maintenance" in *Managing System Services in Oracle Solaris 11.4* for related information.

Converting a Run Control Script to an SMF Service

This section describes how to replace a run control script with an SMF service manifest so that the run control service can be managed by SMF.

▼ How to Convert a Run Control Script to an SMF Service

This procedure describes how to use the `rc-script` property with the `svcbundle` command to convert a run control script to an SMF service.

1. **Determine the service model.**

   By default, `svcbundle` creates a `transient` service. Determine whether this run control script starts any long-running daemon and therefore this service is a `contract` service. See "Service Models" in *Managing System Services in Oracle Solaris 11.4* and the `model` and `startd/duration` properties in the `svcbundle`(8) man page for information about service models.

2. **Create an initial manifest.**

   To convert a run control script, use the `rc-script` property name with the `-s` option of the `svcbundle` command. See the `rc-script` property in the `svcbundle`(8) man page for more information or enter `svcbundle help rc-script`.

   In this example, the service name is `ex-con` and is a contract service that runs at level 2. The run level is specified after a colon after the script name in the `rc-script` property value.

   ```
   $ svcbundle -o /tmp/ex-con.xml -s service-name=ex-con \
   > -s rc-script=/etc/init.d/ex-con:2 -s model=contract
   ```

3. **Make any necessary changes to the manifest.**

   Verify that the content of the `/tmp/ex-con.xml` manifest is what you need. You might need to add a dependency or adjust the method timeout, for example. Add comments to describe what the service does and how the properties of the service are used.

4. **Verify that the manifest is valid.**

   Use the `svccfg validate` command to ensure the service manifest is valid.

   ```
   $ svccfg validate /tmp/ex-con.xml
   ```

5. **Copy the manifest to the standard directory.**

   ```
   $ cp /tmp/ex-con.xml /lib/svc/manifest/site/ex-con.xml
   ```

6. **Stop the existing service.**

   ```
   $ /etc/init.d/ex-con stop
   ```

7. **Disable the run control script.**

   Remove any links to the run control script from the appropriate `rc`*n*`.d` directories.

8. **Import the manifest and start the service.**

   ```
   $ svcadm restart manifest-import
   ```

9. **List the new service.**

   Verify that the new service exists and is in the expected state.

   ```
   $ svcs ex-con
   ```

Creating a Service Using Multiple Manifests

If a new package requires a custom configured instance of a service, that package can deliver just the custom instance without modifying any other service.

Using multiple manifests to define a service enables you to deliver service instances only as needed and without modifying the parent service. For example, if instance I of service S is only needed by tool T, then tool T can deliver instance I without redelivering service S. Then if tool T is not installed on the system, instance I also is not installed, even if service S is installed. Similarly, if tool T is uninstalled, instance I is uninstalled, leaving service S still installed and unmodified. Specify service S in one manifest and instance I in a separate manifest. Deliver the service S manifest in one package, and deliver the instance I manifest in the tool T package. The tool T package has a dependency on the package that delivers service S.

If you use multiple manifests to specify a single service, use the following service manifest design:

- In one manifest, include the service definition, template data, and a default instance.

- In each manifest that defines additional instances of the service, include the following:
  - Specify the same `service_bundle` element as in the manifest that contains the service definition, and then add the `include` attribute. The value of the `include` attribute is the full file name of the manifest that contains the service definition.
  - Specify the same `service` element as in the manifest that contains the service definition. The value of the `name` attribute in the `service` element is exactly the same as in the manifest that contains the service definition.
- Do not deliver the same service or instance in multiple manifests, When this type of conflict is detected, SMF cannot determine which definitions to use, and the instance is placed in the maintenance state.

An example of a service that is delivered in multiple manifests is the `svc:/system/console-login` service. The `console-login` service includes the following instances and manifests:

`svc:/system/console-login:default`

The manifest `/lib/svc/manifest/system/console-login.xml` delivers the service definition, the templates, and the `default` instance.

`svc:/system/console-login:terma`

The manifest `/lib/svc/manifest/system/console-login-terma.xml` delivers the `terma` instance of the `console-login` service.

`svc:/system/console-login:termb`

The manifest `/lib/svc/manifest/system/console-login-termb.xml` delivers the `termb` instance of the `console-login` service.

`svc:/system/console-login:vt?`

The manifest `/lib/svc/manifest/system/console-login-vts.xml` delivers the `vts` instances of the `console-login` service.

The manifest that contains the service definition, `/lib/svc/manifest/system/console-login.xml`, contains the following lines:

```
<service_bundle type="manifest" name="SUNWcs:console">

<service
        name="system/console-login"
        type="service"
        version="1">

<instance name='default' enabled='true'>
</instance>
```

The manifest that defines additional instances `vt2`, `vt3`, `vt4`, `vt5`, and `vt6`, `/lib/svc/manifest/system/console-login-vts.xml`, contains the following lines:

```
<service_bundle type="manifest" name="SUNWcs:console"
    include="/lib/svc/manifest/system/console-login.xml">

<service
        name="system/console-login"
        type="service"
        version="1">

<instance name='vt2' enabled='true'>

        <dependency
                name='system-console'
                grouping='require_all'
                restart_on='none'
                type='service'>
                <service_fmri value='svc:/system/console-login:default' />
        </dependency>

</instance>
```

The definitions of instances `vt3`, `vt4`, `vt5`, and `vt6` contain the same dependency on `console-login:default` as shown for the `vt2` instance.

The `svcs` command output displays the dependency relationships, as shown in the following examples:

```
$ svcs 'svc:/system/console-login:vt*'
STATE          STIME    FMRI
online         Dec_04   svc:/system/console-login:vt3
online         Dec_04   svc:/system/console-login:vt5
online         Dec_04   svc:/system/console-login:vt2
online         Dec_04   svc:/system/console-login:vt4
online         Dec_04   svc:/system/console-login:vt6
$ svcs -D console-login:default
STATE          STIME    FMRI
online         Dec_04   svc:/system/vtdaemon:default
online         Dec_04   svc:/system/console-login:vt3
online         Dec_04   svc:/system/console-login:vt5
online         Dec_04   svc:/system/console-login:vt2
online         Dec_04   svc:/system/console-login:vt4
online         Dec_04   svc:/system/console-login:vt6
online         Dec_04   svc:/system/console-reset:default
$ svcs -d 'svc:/system/console-login:vt*'
STATE          STIME    FMRI
...
```

```
online         Dec_04   svc:/system/console-login:default
online         Dec_04   svc:/system/vtdaemon:default
```

The `default`, `terma`, and `termb` instances are delivered by the `system/core-os` package. The `vts` instances are delivered by the `system/virtual-console` package. The `system/virtual-console` package contains a `require` dependency on the `system/core-os` package to ensure that the `console-login:default` instance exists.

If a service has only one instance, best practice is to use a single manifest to specify the service: Define the one instance in the manifest that contains the service definition.

♦ ♦ ♦ CHAPTER 3

Creating a Service to Run Periodically

This chapter describes how to create a service that runs a relatively short task at regular intervals. This chapter describes:

- Periodic service definition
- How to create a periodic service
- How to specify the execution schedule of a periodic service

Periodic Services

Most system services are implemented as long-running daemons and run until an administrator intervenes. A *periodic* or *scheduled* service runs a relatively short task at regular intervals. A scheduled service is a type of periodic service. See Chapter 4, "Creating a Service to Run on a Specific Schedule" for more information about scheduled services.

You might want to create a periodic service to perform a task that you previously would have configured `cron` to perform. One advantage of using an SMF service and delivering the service in an IPS package is that you can take advantage of SMF and IPS dependency features to ensure the task only runs when other required software is installed and running. Another advantage is that when the user uninstalls the package, the periodic task is removed and does not need to be separately removed from a `crontab` file.

A periodic service instance is managed by the periodic restarter, `svc.periodicd`, which is invoked by the `svc:/system/svc/periodic-restarter` service at system startup. The periodic restarter runs the start method for the instances it manages at scheduled intervals whenever the instance is in the `online` state. A periodic instance that is enabled transitions to the `online` state as soon as all of the dependencies of the instance are met. If no errors or administrative interventions occur, the periodic service remains in the `online` state between runs of the start method, when no processes associated with the method are running. See the `svc.periodicd`(8) man page for more information.

Creating a Periodic Service

The start method and scheduling parameters for a periodic service instance are defined in the `periodic_method` element. When a `periodic_method` element exists, the service instance is automatically delegated to the periodic restarter.

For an example of a simple periodic service manifest, see Example 4, “Example Periodic Service Manifest,” on page 46 in “Creating a Periodic Service Using the Service Bundle Generator Tool” on page 45.

To view real-world examples, you can use the following command to list periodic and scheduled services that are on your system:

```
$ svcs -R svc:/system/svc/periodic-restarter:default
```

To view a manifest, use the `svcs -l` command to show the full path to the manifest. Do not modify the manifest.

Specifying the `periodic_method` Element

The `periodic_method` element can be specified within a `service` element or within an `instance` element. The `periodic_method` element specifies both start method and scheduling information for periodic services and can have the attributes and elements that are described in this section. The `period` and `exec` attributes are required.

EXAMPLE 3 Periodic Service `periodic_method` Element

The following example shows the `periodic_method` element in a periodic service. In this example, the periodic restarter executes `/usr/bin/periodic-ex` every day: every 86400-86460 seconds.

```
<periodic_method
        period='86400'
        jitter='60'
        persistent='true'
        exec='/usr/bin/periodic-ex'
        timeout_seconds='0'>
        <method_context>
                <method_credential user='root' group='root' />
        </method_context>
```

```
</periodic_method>
```

Periodic Service Scheduling Constraints Attributes

For `svcprop` and `svccfg`, the following attributes of the `periodic_method` element are accessed as properties in the `periodic` property group.

The units for a `time` value are seconds, as shown in Table 2, “Service Property Type Value Descriptions,” on page 29.

`delay` attribute

Optional. Value type: `time`. Default value: 0. The fixed number of seconds after the service has transitioned to the `online` state before the first invocation of the start method.

`period` attribute

Required. Value type: `time`. The number of seconds between invocations of the start method.

`jitter` attribute

Optional. Value type: `time`. Default value: 0. The maximum of a random number of seconds after `period` before the start method is run. The final number of seconds that is used ranges between 0 and the value of this property.

Other Periodic Service Scheduling Attributes

For `svcprop` and `svccfg`, the following attributes of the `periodic_method` element are accessed as properties in the `periodic` property group.

`persistent` attribute

Optional. Value type: `boolean`. Default value: `false`. Specifies whether scheduling should be maintained across system downtime.

If the value is `false`, scheduling of the start method restarts as if the periodic service instance has just transitioned to the `online` state after being enabled.

If the value of the `persistent` attribute is `true` and the value of the `recover` attribute (below) is `false`, scheduling of the start method for the instance emerging from downtime continues on the same schedule that was defined before the downtime occurred.

If the value of the `persistent` attribute is `true` and the value of the `recover` attribute is `true`, see the description of the `recover` attribute below.

`recover` attribute

Optional. Value type: `boolean`. Default value: `false`. Specifies whether the instance should have a recovery execution if an invocation was lost during system downtime.

This value has effect only if the value of the `persistent` attribute for this instance is `true`.

If the value of the `persistent` attribute is `true` and the value of the `recover` attribute is `true`, the periodic restarter invokes the start method for the instance as soon as possible as the instance emerges from system downtime. Subsequent invocations occur according to the `period` and `jitter` values.

If the value of the `persistent` property is `true` and the value of the `recover` property is `false`, see the description of the `persistent` property above.

Periodic Service Start Method Attributes and Context

For `svcprop` and `svccfg`, the following attributes of the `periodic_method` element and properties specified in the `method_context` element are accessed as properties in the `start` property group.

`exec` attribute

Required. Value type: `astring`. The start method for the periodic service. The start method actions must be suitable to pass to the `exec` system call. See the `smf_method(7)` man page and the `exec(2)` man page.

This start method should perform a task and then terminate within the time specified by the `period` attribute.

The `SMF_EXIT_TEMP_TRANSIENT` exit code does not apply to periodic service start methods because the periodic restarter does not implement transient services. When used in a periodic service start method, the `SMF_EXIT_TEMP_TRANSIENT` exit code is treated the same as the `SMF_EXIT_ERR_OTHER` exit code.

Periodic service instances use only a start method. If any refresh or stop method is defined, a warning message is issued at manifest import and the refresh or stop method is ignored. When a periodic instance is refreshed, the periodic restarter rereads the values of the properties in the `periodic` property group described in “Storing Periodic Service Data in the Service Configuration Repository” on page 43. The periodic restarter does not need a stop method because processes contracted by a periodic instance do not run persistently. Periodic instances run short-lived processes and then wait until the next scheduled time to run.

`timeout_seconds` attribute

Optional. Value type: `integer`. The number of seconds to wait for the method action to complete. Use a value of `0` or `-1` to specify an infinite timeout.

The `timeout_seconds` attribute value is required by the `periodic_method` element. If you do not specify a value for the `timeout_seconds` attribute in the `periodic_method` element, the value for `timeout_seconds` is assumed to be infinite. See "Scheduling After Start Method Problems" on page 49 for a description of start method scheduling if the start method runs longer than the specified `timeout_seconds` value or if a contracted process still exists when the periodic restarter attempts to invoke the start method for the next period.

`method_context` element

Optional. The context in which the start method will run. See the `method_credential` element in "Securing Service Tasks" on page 31, and see the Method Context section of the `smf_method`(7) man page.

Storing Periodic Service Data in the Service Configuration Repository

As described in "Specifying the `periodic_method` Element" on page 40, the `periodic_method` element provides both method information and scheduling information for periodic services. When a manifest with a `periodic_method` element is imported, the data described in this section is stored in the service configuration repository.

Restarter Properties

The restarter for the service is set to `svc:/system/svc/periodic-restarter:default`. Administrators can use the `svcs -l` command to show the restarter or use the `svccfg` or `svcprop` command to view the `general/restarter` property.

Because a periodic service instance remains online between invocations of the start method, the instance can be in the `online` state with no associated contracted processes running on the system. For an online periodic service instance, the `auxiliary_state` property can have one of the following values to distinguish whether the method action is running:

`running`	The instance is online and has associated contracted processes.
`scheduled`	The instance is online but has no associated contracted processes.

To check this state, administrators can use the `svcs -o astate` command to show the ASTATE column or use the `svccfg` or `svcprop` command to view the `restarter/auxiliary_state` property.

`periodic` Property Group

The scheduling attributes of the `periodic_method` element (`period`, `delay`, `jitter`, `persistent`, and `recover`) are stored as properties of a property group named `periodic`. See "Specifying the `periodic_method` Element" on page 40 for definitions of these attributes and see "Scheduling Executions of a Periodic Service Start Method" on page 47 for examples of how they are used. Administrators can use the `svcprop` and `svccfg` commands to show and modify these `periodic` property group properties.

Last and Next Start Method Invocations

The service configuration repository also stores the following two pieces of scheduling information for the instance, which are properties of the `periodic_restarter` property group:

`last_run`
: The absolute time of the last attempt to run the start method of this instance. If the start method has never run, this property does not exist. To check this time, administrators can use the `svcs -o lrun` command to show the LRUN column.

`next_run`
: The absolute time of the next attempt to run the start method of this instance. If an absolute time does not exist, this property might be absent or have no value. The value of `next_run` is computed at the time `last_run` is set and is the next scheduled start method invocation as described in "Scheduling Executions of a Periodic Service Start Method" on page 47, including the specific `RAND(`*jitter*`)` value for that invocation. To check this time, administrators can use the `svcs -o nrun` command to show the NRUN column. This time is managed by the `periodic-restarter` service for each periodic service instance. Administrators cannot modify this value.

`start` Property Group

The `exec` and `timeout_seconds` values and `method_context` information are stored as properties of a property group named `start`. This `start` property group represents the start method for the periodic service and is defined in the same way as the start method for any other service.

Creating a Periodic Service Using the Service Bundle Generator Tool

When you use the `svcbundle` command to create a periodic service, you must specify the `period` property as well as both the `service-name` and `start-method` properties. By default, `svcbundle` creates a `transient` service. When you specify `-s period`, `svcbundle` creates a periodic service.

▼ How to Create a Periodic Service Using `svcbundle`

1. **Copy the start method to the standard location.**

   In this example, the start method for this service is named `periodic-ex`. Copy this executable to `/lib/svc/method/periodic-ex`.

2. **Create an initial manifest.**

   In this example, the service name is `site/periodic-example`. Specify a period for the start method scheduling. Do *not* specify any of the following properties: `bundle-type`, `duration`, `model`, `rc-script`, `refresh-method`, or `stop-method`.

   ```
   $ svcbundle -o /tmp/periodic-example.xml -s service-name=site/periodic-example \
   > -s start-method=/lib/svc/method/periodic-ex -s timeout=180 \
   > -s period=3600 -s delay=15 -s jitter=5
   ```

   When you specify the `period` property, `svcbundle` creates a `periodic_method` element, which causes the restarter for the service to be set to the periodic restarter when the manifest is imported. The value of the `period` property becomes the value of the `period` attribute of the `periodic_method` element. The value of the `start-method` property becomes the value of the `exec` attribute of the `periodic_method` element.

3. **Make any necessary changes to the manifest.**

   The `/tmp/periodic-example.xml` manifest is shown in Example 4, “Example Periodic Service Manifest,” on page 46.

   Verify that the content of the `/tmp/periodic-example.xml` manifest is what you need. You might want to make changes such as add a `method_context` element in the `periodic_method` element.

   Add comments and fill out `template` elements to describe what the service does and how the properties of the service are used.

4. **Verify that the service manifest is valid.**

```
$ svccfg validate /tmp/periodic-example.xml
```

5. **Copy the manifest to the standard directory.**

```
$ cp /tmp/periodic-example.xml /lib/svc/manifest/site/periodic-example.xml
```

6. **Import the manifest and start the service.**

```
$ svcadm restart manifest-import
```

7. **List the new service.**

   Verify that the new service exists and is in the expected state.

```
$ svcs periodic-example
$ svcs -l periodic-example
$ svcprop -p periodic periodic-example:default
```

Example 4 Example Periodic Service Manifest

The example in this procedure produced the following manifest:

```
<?xml version="1.0" ?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<!--
    Manifest created by svcbundle (2020-Oct-01 16:28:06-0700)
-->
<service_bundle name="site/periodic-example" type="manifest">
    <service name="site/periodic-example" version="1" type="service">
        <!--
            The following dependency keeps us from starting until the
            multi-user milestone is reached.
        -->
        <dependency name="multi_user_dependency" grouping="require_all"
            restart_on="none" type="service">
            <service_fmri value="svc:/milestone/multi-user"/>
        </dependency>
        <periodic_method delay="15" jitter="5" recover="false"
            persistent="false" timeout_seconds="180"
            exec="/lib/svc/method/periodic-ex" period="3600"/>
        <instance name="default" enabled="true"/>
        <template>
            <common_name>
                <!--
                    Replace loctext content with a short name for the
                    service.
```

```
                -->
                <loctext xml:lang="C">
                        site/periodic-example
                </loctext>
            </common_name>
            <description>
                <!--
                    Replace loctext content with a brief description of the
                    service
                -->
                <loctext xml:lang="C">
                        The site/periodic-example service.
                </loctext>
            </description>
        </template>
    </service>
</service_bundle>
```

Scheduling Executions of a Periodic Service Start Method

Scheduling executions of the start method of a periodic service instance (specified by the `exec` attribute of the `periodic_method` element) always requires the value of the `period` attribute of the `periodic_method` element. Other attributes of the `periodic_method` element might also be used, and the `next_run` value might be used.

The `period` property must have a valid value. The `jitter`, `delay`, `persistent`, and `recover` properties have default values and are not required to be explicitly set. The units for a `time` value are seconds, as shown in Table 2, “Service Property Type Value Descriptions,” on page 29.

The scheduling attributes of the `periodic_method` element (`period`, `delay`, `jitter`, `persistent`, and `recover`) are stored as properties of the `periodic` property group. For example, the `period` attribute in the manifest becomes the `periodic/period` property when the periodic service is imported. Administrators can use the `svcprop`and `svccfg` commands to view property values in the `periodic` property group and can use the `svccfg` command to modify these values.

In the following scheduling descriptions, italic type indicates the value of the property. For example, *jitter* is the value of the `periodic/jitter` property in the service configuration repository, which is the same as the value of the `jitter` attribute of the `periodic_method` element in the service manifest.

Scheduling After the Instance is Initially Enabled

When a periodic service instance is initially enabled (for example at system boot), the first execution of the start method of the instance occurs at the following number of seconds after the instance transitions to the `online` state:

delay + `RAND(`*jitter*`)`

Subsequent executions of the start method of the instance occur at the following relative time:

period + `RAND(`*jitter*`)`

To prevent schedule drift, subsequent executions ignore the *jitter* value of previous executions. For example, the second execution of the start method of an instance occurs at the following number of seconds after the instance transitioned to the `online` state:

delay + *period* + `RAND(`*jitter*`)`

The *n*th execution of the start method of an instance occurs at the following number of seconds after the instance transitioned to the `online` state:

delay + `(`*n*`-1)`*period* + `RAND(`*jitter*`)`

Scheduling After System Downtime

The `persistent` and `recover` properties can modify the time the periodic instance method runs when a periodic instance emerges from system downtime. The value of the `recover` property has effect only if the value of the `persistent` property is `true`.

If the value of the `persistent` property is `false`, the next execution of the start method for a periodic instance emerging from system downtime occurs at the following number of seconds after the instance transitions to the `online` state:

delay + `RAND(`*jitter*`)`

If the value of the `persistent` property is `true` and the value of the `recover` property is `false`, the next execution of the start method for the instance emerging from system downtime is scheduled for the following absolute time:

next_run + `(`*n*`)`*period*

If the value of the `next_run` property is in the future, *n* is 0. If the value of the `next_run` property is in the past, the smallest value of *n* is used that results in a future time. Subsequent

invocations occur from that time according to the `period` and `jitter` values. The `next_run` property is described in "Last and Next Start Method Invocations" on page 44.

If the value of the `persistent` property is `true` and the value of the `recover` property is `true`, the periodic restarter invokes the start method for the instance as soon as possible as the instance emerges from system downtime. Subsequent invocations occur from that time according to the `period` and `jitter` values.

Scheduling After Service Restart

The periodic restarter service (`svc:/system/svc/periodic-restarter`) automatically attempts to restart if it terminates. When the periodic restarter service restarts after failure, the start method of each periodic instance is scheduled for the following absolute time:

next_run + (*n*)*period*

If the value of the `next_run` property is in the future, *n* is 0. If the value of the `next_run` property is in the past, the smallest value of *n* is used that results in a future time. Subsequent invocations occur from that time according to the `period` and `jitter` values. The `next_run` property is described in "Last and Next Start Method Invocations" on page 44.

If a periodic service instance is restarted, the start method for the instance is invoked at the following number of seconds after the instance transitions to the `online` state:

delay + RAND(*jitter*)

Scheduling After Start Method Problems

The start method should perform a task and then terminate within the time specified by the `period` property. If a contracted process still exists when the periodic restarter attempts to invoke the start method for the next period, then the invocation for that period is skipped and the periodic restarter attempts to invoke the start method again at the following period. The periodic restarter invokes the method again at the following number of seconds after the instance transitioned to the `online` state, where *n* is the smallest value that results in a future time:

delay + (*n*)*period* + RAND(*jitter*)

If the start method runs longer than the number of seconds specified by the `timeout_seconds` value, all processes in the contract are terminated and the invocation is a non-fatal fault. The first time the start method terminates in any non-fatal fault, the instance is placed into the

`degraded` state and start method invocations continue as scheduled. If one of the following two invocations succeeds, the instance is placed back into the `online` state. After transitioning from the `degraded` state to the `online` state, the periodic restarter invokes the method at the following number of seconds after the instance initially transitioned from the `offline` to the `online` state, where *n* is the smallest value that results in a future time:

delay + (*n*)*period* + RAND(*jitter*)

After three successive non-fatal faults of the start method, the instance is moved from the `degraded` state to the `maintenance` state. On the first fatal fault of the start method, the service is placed into the `maintenance` state. See "Repairing an Instance That Is Degraded, Offline, or in Maintenance" in *Managing System Services in Oracle Solaris 11.4*.

♦ ♦ ♦ CHAPTER 4

Creating a Service to Run on a Specific Schedule

This chapter describes how to create a service that runs a relatively short task at a specified regular time. This chapter describes:

- Scheduled service definition
- How to create a scheduled service
- How to specify the execution schedule of a scheduled service

Scheduled Services

A *scheduled* service is a type of *periodic* service, described in Chapter 3, "Creating a Service to Run Periodically". Each invocation of a periodic service instance start method occurs at a time relative to the last invocation. Each invocation of a scheduled service instance start method occurs at a specific absolute time. Use a scheduled service when the task must run at a certain time, such as during off-peak hours.

Creating a Scheduled Service

The start method and scheduling parameters for a scheduled service instance are defined in the `scheduled_method` element. When a `scheduled_method` element exists, the service instance is automatically delegated to the periodic restarter, `svc:/system/svc/periodic-restarter`.

The start method scheduling parameters are stored in property groups of type `schedule` in the service configuration repository. The periodic restarter combines the values of each property of each property group of type `schedule` to schedule invocations of the start method of the scheduled service instance.

For an example of a simple periodic service manifest, see Example 6, “Example Scheduled Service Manifest,” on page 58 in “Creating a Scheduled Service Using the Service Bundle Generator Tool” on page 56.

Specifying the `scheduled_method` Element

The `scheduled_method` element can be specified within a `service` element or within an `instance` element. The `scheduled_method` element specifies both start method and scheduling information for scheduled services and can have the attributes that are described in this section. The `interval` and `exec` attributes are required. The method information is the same for a `scheduled_method` element as for a `periodic_method` element.

EXAMPLE 5 Scheduled Service `scheduled_method` Element

The following example shows the `scheduled_method` element in a scheduled service. In this example, the periodic restarter executes `/usr/bin/scheduled-ex` every Sunday at 3:15am and does not perform an off-schedule execution if a scheduled execution is missed, for example due to system downtime.

```
<scheduled_method
    interval='week'
    day='Sunday'
    hour='3'
    minute='15'
    recover='false'
    exec='/usr/bin/scheduled-ex'
    timeout_seconds='0'>
        <method_context>
                <method_credential user='root' group='root' />
        </method_context>
</scheduled_method>
```

Scheduled Service Scheduling Constraints Attributes

When specifying values of scheduling constraints for a scheduled service, note the following:

- Specify the fewest scheduling constraints that are required to schedule the task. Constraints that represent shorter time periods than the `interval` value specify more precisely when in the interval to invoke the start method for the instance.
- Specify a continuous set of scheduling constraints from the constraint that represents the longest time to the constraint that represents the shortest time. For example, you can specify

just a `week_of_year` value, but you cannot specify a `week_of_year` and an `hour` unless you also specify a `day_of_month` or `day`.

- You can specify constraints that represent time periods that are equal to or longer than the `interval` value if the value of `frequency` is greater than `1`. See "Scheduling Executions of a Scheduled Service Start Method" on page 59 for more information about how to use these constraints.

For `svcprop` and `svccfg`, the following attributes of the `scheduled_method` element are accessed as properties in the `scheduled` property group.

`interval`

Required. Value type: `astring`. The base length of time between invocations of the service start method, depending on the value of `frequency`. The value of `interval` is one of the following: `year`, `month`, `week`, `day`, `day_of_month`, `hour`, `minute`.

`frequency`

Optional. Value type: `count`. Default value: 1. The number of intervals that must occur before the start method is executed by the periodic restarter. For example, if `interval` is `week` and `frequency` is `4`, the start method will be invoked every fourth week. Note that every fourth week is different from the fourth week of every month because some months have five weeks.

If the value of `frequency` is greater than `1`, all constraints from `year` down to the same length of time as the `interval` value must be specified. For example, if `interval` is `week`, values must be specified for `year` and `week_of_year`. See "Scheduling One Invocation Per Multiple Intervals" on page 60 for examples.

`timezone`

Optional. Value type: `astring`. Default value: the system time zone. The time zone to use to create and interpret schedules. Use this attribute to define a schedule relative to a time zone other than the time zone configured for the system.

`year`

Value type: `count`. A Gregorian calendar year. The year in which to invoke the start method.

`week_of_year`

Value type: `integer`. The ordinal number of the week in an ISO 8601 week date year. Valid values are `1` through `53` and `-1` through `-53`. A negative number specifies weeks backward from the last week of the year. For a year that has 52 weeks, `-1` is the same as `52`, and for a year that has 53 weeks, `-1` is the same as `53`. If you specify `53` or `-53`, the start method will not run in years that have only 52 weeks.

The `week_of_year` and `month` attributes are mutually exclusive.

`month`

Value type: `astring`. A month of a Gregorian year. Valid values are `1` through `12`, `-1` through `-12`, the C Locale full name of the month, or the C Locale three-character abbreviation for the name of the month. A negative number specifies months backward from the last month of the year. C Locale names are not case sensitive.

The `week_of_year` and `month` attributes are mutually exclusive.

`day_of_month`

Value type: `integer`. A day in a Gregorian calendar month. Valid values are `1` through `31` and `-1` through `-31`. A negative number specifies days backward from the last day of the month. If you specify a day that does not exist in a particular month, for example `31` or `-1` for April, the start method will run on the last day of that month.

The `day_of_month` and `day` attributes are mutually exclusive.

`weekday_of_month`

Value type: `integer`. The ordinal number of the week in a month in which the specified day (see `day`) occurs. Valid values are `1` through `5` and `-1` through `-5`. A negative number specifies weeks backward from the last week of the month. For example, the third Thursday in the month (`3`) is different from the next-to-last Thursday in the month (`-2`) if the month has five Thursdays. If you specify `5` or `-1`, the start method will not run in months that have only four of the specified `day`.

If `weekday_of_month` is specified, `day` must also be specified.

`day`

Value type: `astring`. A day in an ISO 8601 standard week. Valid values are `1` through `7` and `-1` through `-7`, the C Locale full name of the day, or the C Locale three-character abbreviation of the name of the day. A negative number specifies days backward from the end of the week. C Locale names are not case sensitive.

The `day_of_month` and `day` attributes are mutually exclusive.

`hour`

Value type: `integer`. An hour of an ISO 8601 standard day. Valid values are `0` through `23` and `-1` through `-24`. A negative number specifies hours backward from the end of the day. If the scheduled day includes a transition between standard time and daylight saving time, the specified hour might occur two times that day or might not occur at all that day. If the specified hour occurs more than one time on the scheduled day, the start method will run only on the first occurrence of the hour. If the specified hour does not occur on the scheduled day, the start method will run in the following hour.

`minute`

Value type: `integer`. A minute of an hour. Valid values are `0` through `59` and `-1` through `-60`. A negative number specifies minutes backward from the end of the hour.

Other Scheduled Service Scheduling Attributes

For `svcprop` and `svccfg`, the `recover` attribute of the `scheduled_method` element is accessed as a property in the `scheduled` property group.

`recover`

Optional. Value type: `boolean`. Default value: `false`. Specifies whether the instance should have a recovery execution if an invocation was lost during system downtime.

If the value of the `recover` attribute is `true`, the periodic restarter invokes the start method for the instance as soon as possible as the instance emerges from system downtime. Subsequent invocations occur according to the specified scheduling constraints.

If the value of the `recover` property is `false`, no recovery invocation is executed. Invocations continue according to the specified scheduling constraints.

Scheduled Service Start Method Attributes and Context

`exec`

Required. Value type: `astring`. The action to take, which must be suitable to pass to the `exec` system call. See the `smf_method(7)` man page. See additional description in "Specifying the `periodic_method` Element" on page 40.

`timeout_seconds`

Optional. Value type: `integer`. The number of seconds to wait for the start method action to complete. Use a value of `0` or `-1` to specify an infinite timeout.

The `timeout_seconds` attribute value is required by the `exec_method` element. If you do not specify a value for the `timeout_seconds` attribute in this `scheduled_method` element, the value for the `exec_method` element is assumed to be infinite. See "Scheduling After Start Method Problems" on page 63 for a description of start method scheduling if the start method runs longer than the specified `timeout_seconds` value or if a contracted process still exists when the periodic restarter attempts to invoke the start method in the next scheduled interval.

`method_context` element

Optional. See the Method Context section of the `smf_method(7)` man page.

Storing Scheduled Service Data in the Service Configuration Repository

As described in "Specifying the `scheduled_method` Element" on page 52, the `scheduled_method` element provides both method information and scheduling information for scheduled services. When a manifest with a `scheduled_method` element is imported, the data described in this section is stored in the service configuration repository.

- **Restarter properties.** See "Restarter Properties" on page 43 for information about the `restarter` and `auxiliary_state` properties. Recall that a scheduled service is a type of periodic service.
- `schedule` **property groups.** The scheduling attributes of the `scheduled_method` element are stored as properties of a property group of type `schedule`. Property groups of type `schedule` can have the properties described in "Scheduled Service Scheduling Constraints Attributes" on page 52 and "Other Scheduled Service Scheduling Attributes" on page 55. See "Scheduling Invocations at Irregular Intervals" on page 60 and "Scheduling Executions of a Scheduled Service Start Method" on page 59 for examples of how these properties are used. Administrators can use the `svcprop` and `svccfg` commands to show and modify these properties. Use the `svcprop -g schedule` command to list all the properties of property groups of type `schedule`, as described in "Showing Properties in a Property Group Type" in *Managing System Services in Oracle Solaris 11.4*.
- **Last and next start method invocations.** See "Last and Next Start Method Invocations" on page 44 for descriptions of the `last_run` and `next_run` properties. The value of `next_run` is the next scheduled start method invocation as described in "Scheduling Executions of a Scheduled Service Start Method" on page 59.
- `start` **property group.** The `exec` and `timeout_seconds` values and `method_context` information are stored as properties of a property group named `start`. This `start` property group represents the start method for the scheduled service and is defined in the same way as the start method for any other service.

Creating a Scheduled Service Using the Service Bundle Generator Tool

When you use the `svcbundle` command to create a scheduled service, you must specify the `interval` property as well as both the `service-name` and `start-method` properties. By default, `svcbundle` creates a `transient` service. When you specify `-s period`, `svcbundle` creates a periodic service.

▼ How to Create a Scheduled Service Using svcbundle

1. **Copy the start method to the standard location.**

   In this example, the start method for this service is named `scheduled-ex`. Copy this executable to `/lib/svc/method/scheduled-ex`.

2. **Create an initial manifest.**

   In this example, the service name is `site/scheduled-example`. Specify an interval for the start method scheduling. Do *not* specify any of the following properties: `bundle-type`, `duration`, `model`, `rc-script`, `refresh-method`, or `stop-method`.

   A scheduled service should use the fewest scheduling constraints required to schedule the task. In the following example, the periodic restarter will execute the start method for the service every Sunday between 02:00 and 03:00. To invoke the method between 02:00 and 02:01, add the constraint `minute='0'`.

   ```
   $ svcbundle -o /tmp/scheduled-example.xml -s service-name=site/scheduled-example \
   > -s start-method=/lib/svc/method/scheduled-ex -s interval=week -s day=Sunday -s hour=2
   ```

   When you specify the `interval` property, `svcbundle` creates a `scheduled_method` element, which causes the restarter for the service to be set to the periodic restarter when the manifest is imported. The value of the `interval` property becomes the value of the `interval` attribute of the `scheduled_method` element. The value of the `start-method` property becomes the value of the `exec` attribute of the `scheduled_method` element.

3. **Make any necessary changes to the manifest.**

   The `/tmp/scheduled-example.xml` manifest is shown in Example 6, “Example Scheduled Service Manifest,” on page 58.

   Verify that the content of the manifest is what you need. You might want to make changes such as the following:

   - Specify additional constraints in the `scheduled_method` element.
   - Specify additional property groups of type `scheduled`, as described in “Scheduling Invocations at Irregular Intervals” on page 60.
   - Add a `method_context` element in the `scheduled_method` element.

   Add comments and fill out `template` elements to describe what the service does and how the properties of the service are used.

4. **Verify that the manifest is valid.**

   ```
   $ svccfg validate /tmp/scheduled-example.xml
   ```

5. **Copy the manifest to the standard directory.**

```
$ cp /tmp/scheduled-example.xml /lib/svc/manifest/site/scheduled-example.xml
```

6. **Import the manifest and start the service.**

```
$ svcadm restart manifest-import
```

7. **List the new service.**

Verify that the new service exists and is in the expected state.

```
$ svcs scheduled-example
$ svcs -l scheduled-example
$ svcprop -p scheduled scheduled-example:default
```

Example 6 Example Scheduled Service Manifest

The example in this procedure produced the following manifest:

```
<?xml version="1.0" ?>
<!DOCTYPE service_bundle
  SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>
<!--
    Manifest created by svcbundle (2020-Oct-01 17:21:24-0700)
-->
<service_bundle name="site/scheduled-example" type="manifest">
    <service name="site/scheduled-example" version="1" type="service">
        <!--
            The following dependency keeps us from starting until the
            multi-user milestone is reached.
        -->
        <dependency name="multi_user_dependency" grouping="require_all"
            restart_on="none" type="service">
            <service_fmri value="svc:/milestone/multi-user"/>
        </dependency>
        <scheduled_method recover="false"
            exec="/lib/svc/method/scheduled-ex" interval="week"
            day="Sunday" hour="2" timeout_seconds="0"/>
        <instance name="default" enabled="true"/>
        <template>
            <common_name>
                <!--
                    Replace loctext content with a short name for the
                    service.
                -->
                <loctext xml:lang="C">
                        site/scheduled-example
                </loctext>
```

```
            </common_name>
            <description>
                <!--
                    Replace loctext content with a brief description of the
                    service
                -->
                <loctext xml:lang="C">
                        The site/scheduled-example service.
                </loctext>
            </description>
        </template>
    </service>
</service_bundle>
```

Scheduling Executions of a Scheduled Service Start Method

Scheduling executions of the start method of a scheduled service instance (specified by the `exec` attribute of the `scheduled_method` element) always requires the values of the `interval` and `frequency` attributes of the `scheduled_method` element. The `frequency` attribute has a default value of `1`. The value of the `interval` attribute must be explicitly set. Other attributes of the `scheduled_method` element are also used, and the `next_run` value might be used.

The scheduling attributes of the `scheduled_method` element are stored as properties of the `scheduled` property group. For example, the `interval` attribute in the manifest becomes the `scheduled/interval` property when the periodic service is imported. Administrators can use the `svcprop`and `svccfg` commands to view property values in the `scheduled` property group and can use the `svccfg` command to modify these values.

Scheduling One Invocation Per Interval

If the value of `frequency` is `1`, the start method of a scheduled service instance is initially invoked at a random time within the smallest specified constraint. Subsequent invocations occur within the same unit of the next shorter constraint in the next execution interval. For example, if the `interval` is `week` and no other scheduling constraints are specified, the start method will initially be invoked at a random time during the week. Subsequent invocations will occur within the same day in future weeks so that the start method will not be invoked on two consecutive days, for example. Similarly, if the `interval` is `week`, and `day` and `hour` are also specified, the start method will initially be invoked at a random time during the specified hour on the

specified day; subsequent invocations will occur within the same minute during the specified hour and day in future weeks.

Scheduling One Invocation Per Multiple Intervals

If the value of `frequency` is greater than `1`, scheduling constraints that are the same length of time or longer than the `interval` value are used to set the time of the initial invocation. All constraints from `year` down to the same length of time as the `interval` value must be specified. For example, if `interval` is `week`, values must be specified for `year` and `week_of_year`.

If `interval` is `week`, `frequency` is `2`, `year` is `2014`, and `week_of_year` is any even number from 2 through 52, then the start method will be invoked every even-numbered week. If `week_of_year` is any odd number from 1 through 53, then the start method will be invoked every odd-numbered week. Because the year 2014 has already passed, the value of the `week_of_year` attribute in this example does not indicate the particular week when the start method will initially be invoked, but only whether it will be invoked in the next even week or the next odd week. Subsequent invocations will occur every two weeks, so that after a year that has 53 weeks, such as 2015, the start method that was initially invoked in an even-numbered week will be invoked in odd-numbered weeks.

If `interval` is `week`, `frequency` is `4`, `year` is `2014`, and `week_of_year` is `1`, the start method will be invoked in weeks 1, 5, 9, and so forth in 2015. Because 2015 has 53 weeks, the start method will be invoked in week 53, and in 2016 the start method will be invoked in weeks 4, 8, 12, and so forth.

Scheduling Invocations at Irregular Intervals

Sometimes one set of scheduling constraints is not enough to define the schedule. A scheduled service can have multiple property groups of type `schedule`. The periodic restarter combines the values of each property of each property group of type `schedule` to schedule invocations of the start method of the scheduled service instance.

Just as a scheduled service should have only as many scheduling constraints as necessary to define the schedule, a scheduled service should have only as many `schedule` property groups as necessary to define the required schedule for the task.

EXAMPLE 7 Invoking Every Twelve Hours

In this example, the start method is invoked at 06:00 and 18:00 every day. The value of the `hour` attribute could be either `6` or `18`.

```
<scheduled_method
    interval='hour'
    frequency='12'
    hour='6'
    minute='0'
    exec='/usr/bin/scheduled_service_method'
    timeout_seconds='0'>
</scheduled_method>
```

EXAMPLE 8 Invoking Daily at 03:00 and 23:00

In this example, an additional `schedule` property group is specified to invoke the start method at 03:00 and 23:00 every day.

```
<instance name='default' enabled='false'>

    <scheduled_method
        interval='day'
        hour='3'
        minute='0'
        exec='/usr/bin/scheduled_service_method'
        timeout_seconds='0'>
            <method_context>
                <method_credential user='root' group='root' />
            </method_context>
    </scheduled_method>

    <property_group name='run2' type='schedule'>
        <propval name='interval' type='astring' value='day' />
        <propval name='hour' type='integer' value='23' />
        <propval name='minute' type='integer' value='0' />
    </property_group>

</instance>
```

EXAMPLE 9 Invoking at 03:00 and 23:00 on Tuesday and Thursday

To invoke the start method at 03:00 and 23:00 every Tuesday and Thursday requires three additional `schedule` property groups, as shown in this example.

```
<instance name='default' enabled='false'>
```

```
        <scheduled_method
            interval='week'
            day='3'
            hour='3'
            minute='0'
            exec='/usr/bin/scheduled_service_method'
            timeout_seconds='0'>
                <method_context>
                    <method_credential user='root' group='root' />
                </method_context>
        </scheduled_method>

        <property_group name='run2' type='schedule'>
            <propval name='interval' type='astring' value='week' />
            <propval name='day' type='astring' value='3' />
            <propval name='hour' type='integer' value='23' />
            <propval name='minute' type='integer' value='0' />
        </property_group>

        <property_group name='run3' type='schedule'>
            <propval name='interval' type='astring' value='week' />
            <propval name='day' type='astring' value='5' />
            <propval name='hour' type='integer' value='3' />
            <propval name='minute' type='integer' value='0' />
        </property_group>

        <property_group name='run4' type='schedule'>
            <propval name='interval' type=astring'' value='week' />
            <propval name='day' type='astring' value='5' />
            <propval name='hour' type='integer' value='23' />
            <propval name='minute' type='integer' value='0' />
        </property_group>

    </instance>
```

If the service is already imported, you can specify these additional property groups by using a site profile or by using the `svccfg` subcommands `addpg` and `setprop`.

Resolving Multiple Possible Invocations in One Interval

If scheduling constraints are specified such that multiple start method invocation times are possible in a single interval, the start method will be invoked at the earliest time that matches all constraints. An example of multiple possible invocation times in a single interval is when

switching from daylight saving time to standard time, when times between 01:00 and 02:00 occur twice.

Scheduling After System Downtime

The `recover` property can modify the time the scheduled instance method runs when a scheduled instance emerges from system downtime.

If the value of the `recover` property is `true`, the periodic restarter invokes the start method for the scheduled service instance as soon as possible as the instance emerges from system downtime. Subsequent invocations occur according to the `interval` and `frequency` values and any other specified constraints.

If the value of the `recover` property is `false`, the periodic restarter invokes the start method for the scheduled service instance at the time specified by the `next_run` property. If the value of the `next_run` property is in the past, future start method invocations occur according to the `interval` and `frequency` values and any other specified constraints. The `next_run` property is described in "Last and Next Start Method Invocations" on page 44.

Scheduling After Service Restart

The periodic restarter service (`svc:/system/svc/periodic-restarter`) automatically attempts to restart if it terminates. When the periodic restarter service restarts after failure, the start method of each scheduled instance is invoked at the time specified by the `next_run` property. If the value of the `next_run` property is in the past, future start method invocations occur according to the `interval` and `frequency` values and any other specified constraints. The `next_run` property is described in "Last and Next Start Method Invocations" on page 44.

If a scheduled service instance is restarted, the start method for the instance is invoked according to the `interval` and `frequency` values and any other specified constraints.

Scheduling After Start Method Problems

The start method should perform a task and then terminate within the period specified by the `interval` and `frequency` properties. If a contracted process still exists when the periodic restarter attempts to invoke the start method in the next scheduled interval, then the invocation for that period is skipped and the periodic restarter attempts to invoke the start method again at the following period.

If the start method runs longer than the number of seconds specified by the `timeout_seconds` value, all processes in the contract are terminated and the invocation is a non-fatal fault. The first time the start method terminates in any non-fatal fault, the instance is placed into the `degraded` state and start method invocations continue as scheduled according to the `interval` and `frequency` values and any other specified constraints. If one of the following two invocations succeeds, the instance is placed back into the `online` state.

After three successive non-fatal faults of the start method, the instance is moved from the `degraded` state to the `maintenance` state. On the first fatal fault of the start method, the service is placed into the `maintenance` state. See "Repairing an Instance That Is Degraded, Offline, or in Maintenance" in *Managing System Services in Oracle Solaris 11.4*.

♦ ♦ ♦ CHAPTER 5

Creating Services to Manage Oracle Database Instances

This chapter presents the following services that help manage the Oracle Database:

- A database service that starts or stops an Oracle Database instance
- A listener service that starts the listener, which is a process that manages the incoming traffic of client connection requests to the database instance

Configuring the Environment

The examples in this chapter use file-backed storage. An alternative to using file-backed storage is to use the Automatic Storage Management (ASM) feature. ASM is a volume manager and a file system for Oracle Database files.

The following environment variables must be set for each installation of the Oracle Database:

`ORACLE_HOME` The location where the database is installed. In the example in this chapter, the location of the database installation is `/opt/oracle/product/home`.

`ORACLE_SID` The systems ID to uniquely identify a particular database on a system.

In the examples in this chapter, `ORACLE_HOME` is set in the service manifest and then used in the method script. `ORACLE_SID` is set in the method script.

Creating a Service to Start or Stop an Oracle Database Instance

This section describes the Oracle Database instance control service manifest and the start/stop method script that is used in that manifest.

Database Instance Control Service Manifest

The following are some features to note about the Oracle Database instance control service. See the service manifest following this list.

- In this example, the service name and manifest name are the same. The service is named `site/oracle/db/database`. The manifest file name is `database.xml` and is located at `/lib/svc/manifest/site/oracle/db/database.xml`.
- No default instance is defined for this service. Add instances by using the `svccfg add` command. The name of each instance must match the name of an Oracle Database instance.
- Two dependencies are defined.
  - The dependency on `svc:/system/filesystem/local` requires all local file systems to be mounted. If you are using a file-backed database, the database service should depend on the local filesystem. If you are using ASM, the database service should depend on the service that manages ASM.
  - The dependency on `svc:/milestone/multi-user` requires the system to reach the multi-user milestone before this database service starts.
- The `method_context` element specifies credentials and resources required to run the method script. In addition to the attributes shown in this example, the `method_context` element can specify attributes such as `project`, `working_directory`, and `resource_pool`. See the `smf_method(7)` man page and the DTD for more information.
  - The `method_credential` element in the `method_context` element specifies that, except for the super user, only the user `oracle` in the group `oinstall` can execute the methods of this service. The `method_credential` element can specify other attributes such as `supp_groups` and `limit_privileges`. You can also define a `method_profile` element instead of the `method_credential` element shown in this example.
  - The `method_environment` element in the `method_context` element sets the `ORACLE_HOME` environment variable. This value is used in the method script to find Oracle libraries and commands.
- The start/stop method script is `/lib/svc/method/svc-oracle-database`.

- `db` specifies to call the `database` start or stop function instead of the `listener` start or stop function. This same method script is used by the `listener` service.
- `%m` is the name of the method: either `start` or `stop`. This specifies whether to call the start function or the stop function in the method script.
- `%i` is the service instance name. This value is assigned to `ORACLE_SID` in the method script, so be sure to give the service instance the same name as the database instance.
- `timeout_seconds="0"` means this method execution has no time constraint.

See the `smf_method(7)` man page for more information about `%m`, `%i`, and `timeout_seconds`.

- The `action_authorization` property requires that a user must be assigned the `solaris.smf.manage.oracle` authorization to perform tasks such as enable this service or modify properties. See the `smf_security(7)` man page for more information.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
 Copyright (c) 2015, 2018, Oracle and/or its affiliates. All rights reserved.

 Define a service to start or stop an Oracle Database instance.
-->

<service_bundle type="manifest" name="site/oracle/db/database">
  <service name="site/oracle/db/database" type="service" version="1">

    <dependency type="service"
        name="filesystem_dependency"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/system/filesystem/local"/>
    </dependency>

    <dependency type="service"
        name="multi_user_dependency"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/milestone/multi-user"/>
    </dependency>

    <method_context>
      <method_credential user="oracle" group="oinstall" />
      <method_environment>
        <envvar name="ORACLE_HOME" value="/opt/oracle/product/home" />
      </method_environment>
    </method_context>

    <exec_method type="method"
```

```
        name="start"
        exec="/lib/svc/method/svc-oracle-database db %m %i"
        timeout_seconds="0"/>

    <exec_method type="method"
        name="stop"
        exec="/lib/svc/method/svc-oracle-database db %m %i"
        timeout_seconds="0"/>

    <property_group name="general" type="framework">
      <propval type="astring"
        name="action_authorization"
        value="solaris.smf.manage.oracle"/>
    </property_group>

    <stability value="Evolving"/>
  </service>
</service_bundle>
```

Add name and description metadata to the manifest so that users can get information about this service from the `svcs` and `svccfg describe` commands. See the `template` element in the DTD.

Ensure the service manifest is valid:

```
# svccfg validate database.xml
```

To install the service, copy the manifest to `/lib/svc/manifest` and restart the `manifest-import` service:

```
# mkdir -p /lib/svc/manifest/site/oracle/db
# cp database.xml /lib/svc/manifest/site/oracle/db
# svcadm restart manifest-import
```

Note - Before you enable the service, create and install the method script (“Start/Stop Method Script for the Oracle Database Instance Control Service” on page 68) and add service instances (“Add Database Service Instances” on page 70).

Start/Stop Method Script for the Oracle Database Instance Control Service

The following is the start/stop method script, `svc-oracle-database`, for both the `database` service and the `listener` service. This method uses the `sqlplus` command to start and stop `database` service instances, and uses the `lsnrctl` command to start and stop `listener` service instances.

```
#!/usr/bin/bash
#
# Copyright (c) 2015, 2018, Oracle and/or its affiliates. All rights reserved.
#

# Load SMF constants and functions
. /lib/svc/share/smf_include.sh

if [[ -z "$SMF_FMRI" ]]; then
 echo "this script can only be invoked by smf(7)"
 exit $SMF_EXIT_ERR_NOSMF
fi

[[ -d "$ORACLE_HOME" || -d "$ORACLE_HOME/lib" ]] || \
 smf_method_exit $SMF_EXIT_ERR_CONFIG dir_failed "ORACLE_HOME: $ORACLE_HOME: directory
 is not set properly"

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib

stop_listener()
{
 "$ORACLE_HOME/bin/lsnrctl" stop "$1" || exit $SMF_EXIT_ERR_FATAL
}

start_listener()
{
 if "$ORACLE_HOME/bin/lsnrctl" status "$1" >/dev/null; then
  echo "Listener '$1' has already been started - restart it now ..."
  "$ORACLE_HOME/bin/lsnrctl" stop "$1" || exit $SMF_EXIT_ERR_FATAL
 fi

 "$ORACLE_HOME/bin/lsnrctl" start "$1" || exit $SMF_EXIT_ERR_FATAL
}

stop_database()
{
 export ORACLE_SID="$1"

 "$ORACLE_HOME/bin/sqlplus" /nolog <<-EOF || exit $SMF_EXIT_ERR_FATAL
  connect / as sysdba
  shutdown immediate
  quit
 EOF
}

start_database()
{
 export ORACLE_SID="$1"
```

```
 "$ORACLE_HOME/bin/sqlplus" /nolog <<-EOF || exit $SMF_EXIT_ERR_FATAL
  connect / as sysdba
  startup
  quit
 EOF
}

case "$1:$2" in
 "listener:start") start_listener "$3";;
 "listener:stop") stop_listener "$3";;
 "db:start") start_database "$3";;
 "db:stop") stop_database "$3";;
 *)
  echo "Usage: $0 {db|listener} {start|stop} {ORACLE_SID}"
  exit $SMF_EXIT_ERR_CONFIG
  ;;
esac
```

Install the method script:

```
# cp svc-oracle-database /lib/svc/method
```

Typically, method scripts have the following ownership:

```
# chown root:bin /lib/svc/method/svc-oracle-database
```

Ensure the script is executable. Typically, method scripts have the following access:

```
# chmod 555 /lib/svc/method/svc-oracle-database
```

Add Database Service Instances

Verify that the start/stop method script is installed and executable.

Add instances by using the `svccfg add` command and specifying the name of a database instance as the name of the service instance. In the following example, `sales_db` is the name of an Oracle Database instance:

```
$ svccfg -s site/oracle/db/database add sales_db
$ svccfg -s database:sales_db
svc:/site/oracle/db/database:sales_db> addpropvalue general/complete astring: dev
svc:/site/oracle/db/database:sales_db> addpropvalue general/enabled boolean: true
svc:/site/oracle/db/database:sales_db> refresh
svc:/site/oracle/db/database:sales_db> exit
```

The service method uses the service instance name to select the Oracle Database on which to operate. In this example, the following command starts the `sales_db` Oracle Database instance:

```
# svcadm enable database:sales_db
```

Note that the preceding `svccfg` command set this service instance to be enabled by default.

Verify that the `database` service is installed and the `sales_db` instance is online:

```
# svcs database
```

The following command stops the `sales_db` Oracle Database instance:

```
# svcadm disable database:sales_db
```

Creating an Oracle Database Listener Service

This section describes the Oracle Database listener service manifest and the start/stop method script that is used in that manifest.

The listener is a process that manages the incoming traffic of client connection requests to the database instance.

Listener Service Manifest

The following are some features to note about the Oracle Database listener service. See the service manifest following this list.

- As in the Oracle Database instance control service (`database` service), the service name and manifest name are the same. The service is named `site/oracle/db/listener`. The manifest file name is `listener.xml` and is located at `/lib/svc/manifest/site/oracle/db/listener.xml`.
- No default instance is defined for this service. Add instances by using the `svccfg add` command. The name of each instance must match the name of an Oracle Database instance.
- Three dependencies are defined. In addition to the two dependencies defined for the `database` service, the `listener` service depends on the `database` service.
- The `method_credential` element is the same as the `method_credential` element in the `database` service manifest except that the user `oracle` is in the group `oinstall`.
- The method script is the same as the `database` service method script: `/lib/svc/method/svc-oracle-database`. The `listener` argument calls the listener start or stop function

instead of the database start or stop function. The `%m`, `%i`, and `timeout_seconds` arguments are the same as for the `database` service.

- The `action_authorization` property is the same as for the `database` service.

```
<?xml version="1.0"?>
<!DOCTYPE service_bundle SYSTEM "/usr/share/lib/xml/dtd/service_bundle.dtd.1">
<!--
 Copyright (c) 2015, 2018, Oracle and/or its affiliates. All rights reserved.

 Define a service to start or stop an Oracle Database instance.
-->

<service_bundle type="manifest" name="site/oracle/db/listener">
  <service name="site/oracle/db/listener" type="service" version="1">

    <dependency type="service"
        name="filesystem_dependency"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/system/filesystem/local"/>
    </dependency>

    <dependency type="service"
        name="multi_user_dependency"
        grouping="require_all"
        restart_on="none">
        <service_fmri value="svc:/milestone/multi-user"/>
    </dependency>

    <dependency type="service"
        name="database_dependency"
        grouping="require_all"
        restart_on="none">
      <service_fmri value="svc:/site/oracle/db/database"/>
    </dependency>

    <method_context>
      <method_credential user="oracle" group="oinstall" />
      <method_environment>
        <envvar name="ORACLE_HOME" value="/opt/oracle/product/home" />
      </method_environment>
    </method_context>

    <exec_method type="method"
        name="start"
        exec="/lib/svc/method/svc-oracle-database listener %m %i"
        timeout_seconds="0"/>
```

```
    <exec_method type="method"
        name="stop"
        exec="/lib/svc/method/svc-oracle-database listener %m %i"
        timeout_seconds="0"/>

    <property_group name="general" type="framework">
      <propval type="astring"
        name="action_authorization"
        value="solaris.smf.manage.oracle"/>
    </property_group>

    <stability value="Evolving"/>
  </service>
</service_bundle>
```

Add name and description metadata to the manifest so that users can get information about this service from the `svcs` and `svccfg describe` commands. See the `template` element in the DTD.

Ensure the service manifest is valid:

```
# svccfg validate listener.xml
```

To install the service, copy the manifest to `/lib/svc/manifest` and restart the `manifest-import` service:

```
# cp listener.xml /lib/svc/manifest/site/oracle/db/listener.xml
# svcadm restart manifest-import
```

Note - Before you enable the service, create and install the method script ("Start/Stop Method Script for the Oracle Database Instance Control Service" on page 68) and add service instances ("Add Listener Service Instances" on page 73).

Add Listener Service Instances

Verify that the start/stop method script is installed and executable.

Add instances by using the `svccfg add` command and specifying `LISTENER` as the name of the service instance, as described in Listener Control Utility in the *Database Net Services Reference documentation*.

```
$ svccfg -s site/oracle/db/listener add LISTENER
$ svccfg -s listener:LISTENER
svc:/site/oracle/db/listener:LISTENER> addpropvalue general/complete astring: dev
svc:/site/oracle/db/listener:LISTENER> addpropvalue general/enabled boolean: true
svc:/site/oracle/db/listener:LISTENER> refresh
```

```
svc:/site/oracle/db/listener:LISTENER> exit
```

The following command starts the listener:

```
# svcadm enable listener:LISTENER
```

Note that the preceding `svccfg` command set this service instance to be enabled by default.

Verify that the `listener` service is installed and the `LISTENER` instance is online:

```
# svcs listener
```

The following command stops the listener:

```
# svcadm disable listener:LISTENER
```

♦ ♦ ♦ CHAPTER 6

Using a Stencil to Create a Configuration File

If your application cannot use `libscf` library interfaces to read properties, you can use a stencil to create a configuration file. A *stencil service* creates configuration files by using a stencil file and property values defined in the stencil service. A *stencil file* contains a structural definition of a configuration file that is required by an application that is not integrated with SMF but stores its configuration in SMF. Stencil services enable you to take advantage of SMF configuration management with no change to the existing application.

If an application requires multiple configuration files, specify each configuration file in a separate property group.

Configuration files are generated immediately before the service instance `start` or `refresh` method is executed. If you update the stenciled property values, refresh or restart the service to incorporate the changes into the configuration file before the application starts and reads the configuration file.

This chapter describes:

- How to create a stencil service
- How to create a stencil service to generate multiple configuration files
- The Puppet stencil service in Oracle Solaris
- The WebUI stencil service in Oracle Solaris

Creating a Stencil Service

A stencil file contains a structural definition of a configuration file that is required by an application that is not integrated with SMF, but stores its configuration in SMF. The `svcio` utility generates the configuration file from the definitions in the stencil file and properties in the SMF service. See the `svcio(1)` man page for more information about the `svcio` utility and the `smf_stencil(5)` man page for information about stencil file format.

How to Create a Stencil Service

1. **Add a `configfile` type property group to the service.**
   Property groups of type `configfile` tell SMF how to generate configuration files. This stencil service property group tells the `svcio` utility the path and ownership to use to create the configuration file. SMF regenerates configuration for stencil-aware services before running the start or refresh methods.
   Each `configfile` type property group describes a single configuration file for the service and tells `svcio` how to generate these files from other properties stored in the SMF repository.
   A `configfile` type property group has the following properties:

`path`	The path to which to write the configuration file, for example `/etc/svc.conf`.
`stencil`	The path of the stencil file to use, relative to `/lib/svc/stencils`. For example, if the value of the `stencil` property is `svc.stencil`, the `/lib/svc/stencils/svc.stencil` file will be used.
`mode`	The mode to use for the configuration file (`path`), for example 644.
`owner`	The owner to set for the configuration file (`path`). If this property is not set, the owner of the file is the user who invokes `svcio`.
`group`	The group to set for the configuration file (`path`). If this property is not set, the group will be the default group for `path`.

2. **Create a stencil file.**
   The stencil file tells the `svcio` utility the format to use to create the configuration file. The `svcio` utility converts SMF properties into application-specific configuration files based on the stencil. See the `smf_stencil(5)` man page for information about stencil file format.
   Put your stencil file in the `/lib/svc/stencils` directory, and specify the name of the stencil file as the value of the `stencil` property in the `configfile` type property group.

How to Create a Stencil Service to Generate Multiple Configuration Files

1. **Define multiple `configfile` type property groups.**

Each `configfile` type property group describes a single configuration file for the service. To create multiple configuration files, define a separate `configfile` type property group for each configuration file.

Each `configfile` type property group in a single service must specify a different configuration file: a different value for the `path` property. Values of the `stencil`, `mode`, `owner`, and `group` properties can be the same or different.

2. Add a `defines` property group to each `configfile` type property group.

To add configuration that is unique to each configuration file, specify the unique configuration in a `defines` property group in each `configfile` type property group. A `defines` property group is type `application`. A `defines` property group is a nested property group: a child of the `configfile` type property group.

A `defines` property group defines name/value pairs to preload before parsing the stencil. See Example 10, “Creating Multiple Configuration Files Using a Single Stencil File,” on page 77 below.

3. Create a stencil file.

The stencil file can be the same or different in each `configfile` type property group. If the structure of two configuration files is the same, you can use the same stencil file for each configuration file, including a line such as the following in the stencil file to add unique values to each configuration file:

```
# $%[property-name], printing one value per line if multiple values are defined
$%{$%[property-name]:,\n}
```

In Example 10, “Creating Multiple Configuration Files Using a Single Stencil File,” on page 77, you can see how *property-name* selects different values for each configuration file defined in each `defines` property group.

Put your stencil file in the `/lib/svc/stencils` directory, and specify the name of the stencil file as the value of the `stencil` property in the `configfile` type property group.

Example 10 Creating Multiple Configuration Files Using a Single Stencil File

This example has the following features:

- Properties named `host_keys/dsa` and `host_keys/rsa` are defined for the service.
- Two `configfile` type property groups specify two different configuration files: `/etc/ssh/hostkey_1` and `/etc/ssh/hostkey_2`.
- Within each `configfile` type property group, a `defines` property group defines the `key` property that provides the appropriate `host_keys` property value to the configuration file.

```
<property_group type="application" name="host_keys">
    <propval type="astring" name="dsa" value="key1" />
    <property type="astring" name="rsa">
        <astring_list>
            <value_node value="key2" />
            <value_node value="key3" />
        </astring_list>
    </property>

    <property_group type="configfile" name="host_key_1">
        <propval type="astring" name="mode" value="0444" />
        <propval type="astring" name="path" value="/etc/ssh/hostkey_1" />
        <propval type="astring" name="stencil" value="sshhostkey.stencil" />

        <property_group type="application" name="defines">
            <propval type="astring" name="key" value="host_keys/dsa" />
        </property_group>
    </property_group>

    <property_group type="configfile" name="host_key_2">
        <propval type="astring" name="mode" value="0444" />
        <propval type="astring" name="path" value="/etc/ssh/hostkey_2" />
        <propval type="astring" name="stencil" value="sshhostkey.stencil" />

        <property_group type="application" name="defines">
            <propval type="astring" name="key" value="host_keys/rsa" />
        </property_group>
    </property_group>
</property_group>
```

The stencil file includes the following line, where `key` is the property defined in each `defines` property group:

```
# $%[key], printing one key per line if multiple are defined
$%{$%[key]:,\n}
```

When the `svcio` utility reads the stencil file and service property values, the following configuration files are created:

```
$ cat /etc/ssh/hostkey_1
# host_keys/dsa, printing one key per line if multiple are defined
key1
$ cat /etc/ssh/hostkey_2
# host_keys/rsa, printing one key per line if multiple are defined
key2
key3
```

Puppet Stencil Service

Puppet is a toolkit for managing the configuration of many systems. For information about using Puppet on Oracle Solaris, see *Using Puppet to Perform Configuration Management in Oracle Solaris 11.4*.

On Oracle Solaris, the Puppet application is managed by SMF.

High Level View of Puppet Services

When you install the `system/management/puppet` package, you get the following SMF service instances:

```
$ svcs puppet
STATE          STIME    FMRI
disabled        8:17:58 svc:/application/puppet:agent
disabled        8:17:58 svc:/application/puppet:main
disabled        8:17:58 svc:/application/puppet:master
disabled        8:17:58 svc:/application/puppet:user
online          8:36:42 svc:/application/puppet:upgrade
```

- The `master` instance serves configuration to agents.
- The `agent` instance controls configuration of a particular node.
- The `main` instance holds shared configuration values.
- The `user` instance is used by the `puppet apply` command, as well as many of the less common `puppet` subcommands.
- The `upgrade` instance performs migration and cleanup steps if needed.

Puppet writes to log files in the following locations:

```
$ svcprop -p config/logdest puppet:master
/var/log/puppetlabs/puppet/puppet-master.log
$ svcprop -p config/logdest puppet:agent
/var/log/puppetlabs/puppet/puppet-agent.log
```

Puppet Configuration File

Puppet expects to use a configuration file named `/etc/puppetlabs/puppet/puppet.conf`. The `/usr/sbin/puppet` application reads configuration information from `/etc/puppetlabs/puppet/puppet.conf` and not from properties set in the `application/puppet` service instances.

To provide the required configuration file, each `puppet` instance provides a `configfile` type property group. The `configfile` type property group tells the `svcio` utility to run and create the specified configuration file. The stencil file is used to write data from service property values to the configuration file in the correct format.

The following command shows all `puppet` service properties that are in a property group of type `configfile`. This output shows that all instances of the `puppet` service have the same `configfile` properties with the same values. Each `puppet` service instance provides the path to the configuration file, the mode of the configuration file, and the path to the stencil file.

```
$ svcprop -g configfile puppet
```

The following command shows the values set in the parent service:

```
$ svccfg -s puppet listprop puppet_stencil
puppet_stencil          configfile
puppet_stencil/mode     astring     0444
puppet_stencil/path     astring     /etc/puppetlabs/puppet/puppet.conf
puppet_stencil/stencil astring     puppet.stencil
```

If you run the following command for each instance, you see that the service instance properties are inherited from the parent service. The instances do not set custom values for any of these properties:

```
$ svccfg -s puppet:master listprop puppet_stencil
```

Though every instance specifies the same stencil file, stencil `defines` property groups are not used because every instance also specifies the same configuration file: Puppet expects to find all configuration in the one file, not in multiple configuration files. In addition, other services also use values from this `puppet.conf` configuration file.

If your site requires more instances, for example `puppet:agent1` and `puppet:agent2`, add instances and customize property values and add properties for each instance as shown in “Modifying the Puppet Configuration File” on page 82.

Puppet Stencil File

The content of the stencil file tells you what properties and other information are written to the configuration file. The `puppet.stencil` path that is the value of the `puppet_stencil/stencil` property is relative to `/lib/svc/stencils`. The following is the content of the stencil file, `/lib/svc/stencils/puppet.stencil`:

```
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppet.com/guides/configuring.html
```

```
# for details.
#
#
# service: puppet:main
# main is the global section used by all commands and services. It can be
# overridden by the other sections.
#
; walk the main instance and extract all properties from the config PG
$%/(svc:/$%s:(main)/:properties)/ {
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}

#
# service: puppet:master
# master is used by the Puppet master service and the Puppet cert command.
#
; walk the master instance and extract all properties from the config PG
$%/(svc:/$%s:(master)/:properties)/ {
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}

#
# service: puppet:agent
# agent is used by the Puppet agent service.
#
; walk the agent instance and extract all properties from the config PG
$%/(svc:/$%s:(agent)/:properties)/ {
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}

#
# service: puppet:user
# user is used by the Puppet apply command, as well as many of the less common
# Puppet subcommands.
#
; walk the user instance and extract all properties from the config PG
$%/(svc:/$%s:(user)/:properties)/ {
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}
```

Expressions used in this stencil file have the following meanings. See the `smf_stencil(5)` man page for more information.

TABLE 3 Expressions Used in Stencil Files

Expression	Meaning
`$s`	`application/puppet`
`%1`	`svc:/$%s:(.`*instance-name*`)/:properties`
`%2`	Service instance name: `main`, `master`, `agent`, `user`
`%3`	Each property name in the specified property group, `config`

For example, given the following `config` property group properties, the following section in the stencil file produces the following block in the configuration file.

```
$ svcprop -p config puppet:main
config/confdir astring /etc/puppetlabs/puppet
config/rundir astring /var/run/puppetlabs
config/vardir astring /var/cache/puppetlabs
```

Stencil file:

```
$%/(svc:/$%s:(main)/:properties)/ {
[$%2]
$%/$%1/config/(.*)/ {
$%3 = $%{$%1/config/$%3} }
}
```

Configuration file:

```
[main]

confdir = /etc/puppetlabs/puppet
rundir = /var/run/puppetlabs
vardir = /var/cache/puppetlabs
```

Modifying the Puppet Configuration File

As you can see in “Puppet Stencil File” on page 80, all the property values in the `config` property group for each instance except `puppet:upgrade` are written to the configuration file. To modify the configuration file, modify property values in the `config` property group.

Configuration in the `[main]` block applies to both the agent and the master. For the Puppet `master`, `agent`, and `user`, configuration in the corresponding block overrides the same configuration in the `[main]` block.

The following commands show how to add configuration to your Puppet configuration file for the `puppet:agent` service instance.

```
$ svccfg -s puppet:agent
svc:/application/puppet:agent> setprop config/report=true
svc:/application/puppet:agent> setprop config/pluginsync=true
svc:/application/puppet:agent> refresh
svc:/application/puppet:agent> exit
$ svcadm restart puppet:agent
```

Following the restart, the `/etc/puppetlabs/puppet/puppet.conf` configuration file contains the following lines in the `agent` block:

```
[agent]

logdest = /var/log/puppetlabs/puppet/puppet-agent.log
pluginsync = true
report = true
```

Similar commands can be used to remove properties and change property values. See Chapter 4, "Configuring Services" in *Managing System Services in Oracle Solaris 11.4*.

WebUI Stencil Service

The WebUI service, `svc:/system/webui/server:default`, uses `config` property group properties and the `/lib/svc/stencils/webui.conf.stencil` file to create the `/var/webui/conf/webui.conf` configuration file.

```
$ svcprop -g configfile webui/server:default
webui_conf/group astring sys
webui_conf/mode integer 644
webui_conf/owner astring root
webui_conf/path astring /var/webui/conf/webui.conf
webui_conf/stencil astring webui.conf.stencil
```

The following properties are set in the `conf` property group:

```
$ svcprop -p conf webui/server:default
conf/cipher_suite astring HIGH:\!aNULL:\!eNULL:\!MD5:\!DES:\!RC4:\!3DES:\!PSK
conf/default_landing_page astring /solaris/apps/analytics
conf/listen_addresses astring 6787
conf/protocol astring all\ -SSLv2\ -SSLv3\ -TLSv1\ -TLSv1.1
conf/redirect_from_https boolean false
conf/redirect_root_url boolean true
```

```
conf/server_name astring 127.0.0.1
```

You can search the `/lib/svc/stencils/webui.conf.stencil` stencil file to see how the `conf` properties are used.

The stencil file and resultant configuration file contain many lines that do not use any properties from any service. The configuration file is an Apache configuration file with many directives such as `LoadModule` and `RewriteCond`.

♦ ♦ ♦ CHAPTER 7

Creating a Service that Notifies if Conditions are not Satisfied

The services described in this chapter notify you if any of their required dependencies is not online.

This chapter discusses:

- How to create a goal service
- Best practices for creating a goal service

Creating a Goal Service

Most services that cannot reach the `online` state remain silently in the `offline` state. A *goal* service transitions to the `maintenance` state if its dependencies cannot be satisfied without administrative intervention. Transitioning to the `maintenance` state enables the administrator to be notified through the usual SMF fault notification mechanisms. For example, SNMP traps are sent on `maintenance` transitions. Using a goal service, you can specify a set of services that must be running, and if those services are not running, you are notified and do not need to separately check the state of each service in the set. You can also specify a goal service as a dependency of another service as a more compact way to specify do not start this service until all of these other services are ready. A goal service provides a single point to report failure of any of the specified dependency services.

The `svc:/milestone/goals:default` service is a goal service instance that indicates whether a system has booted according to your criteria. The default dependency of the `milestone/goals` service is `svc:/milestone/multi-user-server:default`. The dependencies of `milestone/goals` should be the mission critical services for the system. Then you will be notified if any of these services cannot come online at system startup without administrative intervention.

To specify services that are critical to your systems at startup as dependencies of a goal service, use one of the following methods:

- Add dependencies to `svc:/milestone/goals:default`. Use the `svcadm goals` command as described in "Changing the Goals of a Goal Service" in *Managing System Services in Oracle Solaris 11.4* to add dependencies to the `milestone/goals` instance or to another goal service instance.
- Create your own custom goal service. For example, if you want NFS, NTP, and Apache all running at system startup, create a goal service that specifies NFS, NTP, and Apache services as dependencies of the goal service. On another system, you might want a goal service that specifies database services as dependencies.

Goal services that are in the `maintenance` state automatically leave that state once their dependencies are satisfiable.

You can modify and extend notification configuration as described in "Configuring Notification of State Transition and FMA Events" in *Managing System Services in Oracle Solaris 11.4*.

To define a service to be a goal service, set the `goal-service` boolean property in the `general` property group to `true`:

```
general/goal-service=true
```

To determine whether a service is a goal service, use the `svccfg listprop` command to show the value of the `general/goal-service` property.

Best Practices for Designing Goal Services

The following best practices are recommended for goal services:

- Do not define a service that performs work to be a goal service. A goal service aggregates the state of a set of services. A service that performs work can be a dependency of a goal service. Because a goal service does not perform work, the value of the `exec` attribute of the `exec_method` element for start and stop methods can be simply `:true`.
- In general, in the `dependency` element of a goal service, set the value of the `grouping` attribute to `require_all`, and set the value of the `restart_on` attribute to `restart`. See "Dependency Groupings" in *Managing System Services in Oracle Solaris 11.4* and "Showing Whether a Service Will Automatically Restart" in *Managing System Services in Oracle Solaris 11.4* for more information.
- Dependencies of a goal service should be persistently enabled services, not dynamically enabled services. The goal service will remain in the `maintenance` state until the dynamically enabled dependency is enabled by another service or by an administrator. If you must include a dynamically-enabled dependency, set the value of the `grouping` attribute of the dependency to `optional_all`. Use the `svcs -l` command to check whether

a dependency is persistently enabled (enabled across reboots). In the `svcs -l` output, the value of `enabled` is `true` for persistently enabled services.

Index

R

S

T

W