

Understanding Serviceguard Package Dependencies

April 2009

Introduction	2
Scope.....	2
Package priority	2
Configuration.....	3
Configuration examples	4
Same node dependency example:.....	4
Different node dependency example	4
Any node dependency example	4
Same node exclusionary (down) dependency example:.....	4
All nodes exclusionary (down) dependency example	5
Multiple dependency example	5
Dependency behavior	6
Using the cmrunpkg command	6
Using the cmhaltpkg command.....	7
Using the cmapplyconf command	7
Using other commands.....	7
Automatic mode.....	8
Automatic mode examples	8
Assigning priorities.....	10
For more information.....	13

Introduction

Dependencies define runtime relationships between packages. Serviceguard evaluates dependencies while making package placement decisions. The dependency parameters in the package configuration file specify under what conditions the dependency is satisfied. For example, you can define a dependency for package A requiring that package B is running before package A is started on the same node, or requiring that package B must be down on all nodes before package A is run.

The following dependency types are supported:

- Same node dependency: A package can require that another package be UP on the same node. This means that the package will not start on a node, or continue to run, if the package that it depends on is not UP on the same node.
- Different node dependency: A package can require that another package be UP on a different node in the cluster. This means that the package will not start on a node, or continue to run, if the package that it depends on is not UP on a different node in the same cluster.
- Any node dependency: A package can require that another package be UP on any node in the cluster. This means that the package will not start, or continue to run, if the package that it depends on is not UP on a node in the same cluster.
- Same node exclusionary dependency: A package can require that another package be DOWN on the same node, although that package can be UP on another node.
- All nodes exclusionary dependency: A package can require that another package be DOWN on all nodes in the cluster.

Scope

This document describes how you can configure and use package dependencies. It provides a brief description and several examples. For a complete description of rules and restrictions, please refer to the manpage for the `cmmakepkg(1m)` command and the Managing Serviceguard manual.

Package priority

`Priority` is a parameter defined in the package configuration file. You can use it to influence the behavior of packages involved in dependency relationships with other packages. The legal values for `priority` are "no_priority" or a positive integer. Priority is a ranking, so a smaller number represents higher priority; for example a package with a priority of 10 has a higher priority than a package with a priority of 20. A package with a numeric priority has higher priority than a package with "no_priority". No two packages in a cluster can have the same numeric priority (though any number of packages can have "no_priority"). HP recommends you use priorities in increments of 10 so you can add new packages without having to reassign priorities for the existing packages.

Configuration

The package parameters `dependency_name`, `dependency_condition`, and `dependency_location` define package dependencies. You configure these parameters in the configuration file of the dependent package. If you want `pkg1` to depend on `pkg2`, you specify the dependency in the configuration file for `pkg1`.

- `dependency_name` specifies the name of the dependency. This must be unique among dependency names specified by the package. For example, a dependency on `pkg2` can be called "pkg2dep".
- `dependency_condition` describes what must be true for the dependency to be satisfied. The syntax is:

```
<package name> = up
```

OR

```
<package name> = down
```

- "up" means this package requires the package identified by "package name" to be up (i.e. status reported by `cmviewcl` is "up").
- "down" means that this package requires the package identified by "package name" to be down (i.e. status reported by `cmviewcl` is "down"). This is known as an exclusionary dependency.
- `dependency_location` describes where the dependency condition must be satisfied. This parameter is optional. The default value is `same_node`. The legal values depend on the dependency condition:
 - if dependency condition is "up", the legal values are: `same_node`, `any_node`, and `different_node`.
 - if dependency condition is "down", the legal values are: `same_node` and `all_nodes`.

`dependency_name` and `dependency_condition` are required and `dependency_location` is optional. These parameters must be specified in the order listed above.

A package can define only one dependency on another package. However a package can specify dependencies on multiple packages. To specify a dependency on more than one package, you must repeat these parameters for each dependency.

The names of these parameters are case insensitive (i.e. `dependency_name` and `DEPENDENCY_NAME` are the same).

Notes:

An exclusionary dependency must be mutual. If `pkg1` depends on `pkg2` to be down on the same node, `pkg2` must also depend on `pkg1` to be down on the same node. This means that an exclusionary dependency must be defined in the configuration files of both packages, and you must apply both packages to the cluster configuration at the same time.

If a package has an exclusionary dependency on another package, at least one of these packages must be assigned a numerical priority; the priority of both packages cannot be "no_priority".

Configuration examples

Same node dependency example:

Pkg1 requires pkg2 to be up on the same node before it can start and in order to continue running.

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=up
Dependency_location  same_node
```

Different node dependency example

Pkg1 requires pkg2 to be up on a different node before it can start and in order to continue running.

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=up
Dependency_location  different_node
```

Any node dependency example

Pkg1 requires pkg2 to be up on any node in the same cluster before it can start and in order to continue running.

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=up
Dependency_location  any_node
```

Same node exclusionary (down) dependency example:

Pkg1 requires pkg2 to be down on same node before it can start and in order to continue running.

However pkg2 can be up on another node in the cluster. This dependency is mutual. Pkg1 needs to define a same node "down" dependency on pkg2 and pkg2 needs to define a same node "down" dependency on pkg1. Both configuration files must be applied at the same time. Note that at least one of these packages must be assigned a numerical priority; the priority of both packages cannot be "no_priority".

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=down
Dependency_location  same_node
```

Include the following in the configuration file for pkg2:

```
Dependency_name      pkg1dep
Dependency_condtion  pkg1=down
Dependency_location  same_node
```

All nodes exclusionary (down) dependency example

Pkg1 requires pkg2 to be down in the cluster before it can start and in order to continue running. This dependency is mutual. Pkg1 needs to define an all nodes "down" dependency on pkg2 and pkg2 needs to define an all nodes "down" dependency on pkg1. Both configuration files must be applied at the same time. Note that at least one of these packages must be assigned a numerical priority; the priority of both packages cannot be "no_priority".

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=down
Dependency_location  all_nodes
```

Include the following in the configuration file for pkg2:

```
Dependency_name      pkg1dep
Dependency_condtion  pkg1=down
Dependency_location  all_nodes
```

Multiple dependency example

Say you have the following requirement:

- Pkg1 requires pkg2 to be down on the same node before it can start and in order to continue running. However pkg2 can be up on another node in the cluster. This dependency is mutual. Pkg1 needs to define a "down" dependency on pkg2 and pkg2 needs to define a "down" dependency on pkg1. Both configuration files must be applied at the same time.
- Pkg1 requires pkg3 to be up on any node in the cluster before it can start and in order to continue running.

Include the following in the configuration file for pkg1:

```
Dependency_name      pkg2dep
Dependency_condtion  pkg2=down
Dependency_location  same_node
```

```
Dependency_name      pkg3dep
Dependency_condtion  pkg3=up
Dependency_location  any_node
```

Include the following in the configuration file for pkg2:

```
Dependency_name      pkg1dep
Dependency_condtion  pkg1=down
Dependency_location  same_node
```

Dependency behavior

This section describes the rules Serviceguard uses in managing packages with dependencies.

Using the `cmrunpkg` command

The following rules apply when you run packages that have dependencies using the `cmrunpkg` command:

- `cmrunpkg` will fail if running the packages specified on the command line will cause another package to halt. This means if you attempt to run a higher priority package using `cmrunpkg`, and this will lead to a lower priority package going down, the command will fail. Say you have the following configuration:
 - `pkg1` has an all nodes exclusionary dependency on `pkg2`
 - `pkg1` has higher priority than `pkg2`
 - `pkg1` is currently down and `pkg2` is runningThe command `cmrunpkg pkg1` will fail, because it will cause `pkg2` to halt.
- You can use `cmrunpkg` to run a package configured with dependencies only if all of its non-exclusionary dependencies (all packages named in a dependency condition attribute with an “up” dependency) are either running or specified on the `cmrunpkg` command line. The order in which you specify these packages on the command line does not matter. For example:
 - if `pkg1` has a `same_node` “up” dependency on `pkg2`, you can start `pkg1` using `cmrunpkg` if `pkg2` is already up; otherwise you must specify it on the `cmrunpkg` command line:
`cmrunpkg pkg1 pkg2`
- If a package has an all nodes exclusionary (down) dependency on another package, both packages cannot run at the same time. This means that you cannot include both packages on the `cmrunpkg` command line.
- If a package has a same node exclusionary (down) dependency or a different node (up) dependency on another package, you need to use `cmrunpkg` with the `-a` option in order to include both packages on the `cmrunpkg` command line,. Examples:
 - if `pkg1` has a `same_node` “down” dependency on `pkg2`:
 - `cmrunpkg pkg1` is allowed
 - `cmrunpkg pkg1 pkg2` is not allowed
 - `cmrunpkg -a pkg1 pkg2` is allowed
 - if `pkg1` has a `different_node` dependency on `pkg2`:
 - `cmrunpkg pkg1` is allowed if `pkg2` is already running on a different node
 - `cmrunpkg pkg1 pkg2` is not allowed
 - `cmrunpkg -a pkg1 pkg2` is allowed

NOTE:

You can use the `-t` option to the `cmrunpkg` command to preview the effect on packages of the command before running the actual command.

Using the `cmhaltpkg` command

The following rules apply when halting packages with dependencies using the `cmhaltpkg` command:

- You can halt a package using `cmhaltpkg` command only if its non-exclusionary dependent packages are either down or specified on the `cmhaltpkg` command line. Non-exclusionary dependents are packages that have defined an "up" dependency on this package. The order in which you specify these packages on the command line does not matter. For example:
 - if `pkg1` has a `same_node` "up" dependency on `pkg2`, you can halt `pkg2` using `cmhaltpkg`, if `pkg1` is already down; otherwise you must specify it on the `cmhaltpkg` command line:
`cmhaltpkg pkg1 pkg2`
 - if `pkg1` has a `same_node` "down" dependency on `pkg2`, you can halt `pkg2` without halting `pkg1`.
- `cmhaltpkg` will fail if halting the packages specified on the command line will cause a package not specified on the command line to halt. Say you have the following configuration:
 - `pkg1` (with priority 10) has an all nodes exclusionary dependency on `pkg2` (with priority 20)
 - `pkg3` (with priority 30) also has an all nodes exclusionary dependency on `pkg2`
 - `pkg1` and `pkg3` are currently running and `pkg2` is down because `pkg1` with higher priority is up.The command `cmhaltpkg pkg1` will fail, because it will allow `pkg2` with priority 20 to start which will force the lower priority `pkg3` down. You need to halt `pkg3` before running the command.

Note:

You can use the `-t` option to the `cmhaltpkg` command to preview the effect on packages of the command before running the actual command.

Using the `cmapplyconf` command

Modifying the cluster or a package configuration while the cluster and packages are running may cause some packages to start, halt or move due to package dependencies. For example if you increase the priority of a package with a same-node exclusionary dependency on another package which is running, it may cause that package to go down. The `cmapplyconf` command (also `cmcheckconf`) will issue a warning if the change in configuration causes a package to halt or move.

Using other commands

Using commands like `cmmodpkg`, `cmrunnode`, and `cmhaltnode` may cause some packages to start, halt or move due to package dependencies. You can use the `-t` option to these commands to preview the effect the command will have on packages before running the actual command.

Automatic mode

Serviceguard uses the following rules in managing packages with dependencies in automatic mode (that is, when Serviceguard itself is managing packages, rather than responding to commands that run and halt packages).

1. If pkg1 has a same-node, any-node, or different-node dependency on pkg2, and pkg2 fails over to another node, pkg1 will restart. If the dependency is same-node, pkg1 will follow pkg2 to the new node. If the dependency is any-node or different node, pkg1 may restart on the current node or go to another node where the dependency is met.
2. Serviceguard gives preference to running higher-priority packages, and the node order of the higher-priority package dominates the node order of the lower-priority package. This means that if necessary Serviceguard halts or moves a lower priority package in order to meet the dependency of a higher priority package. This halting or moving of the lower priority package is called "dragging".

Note:

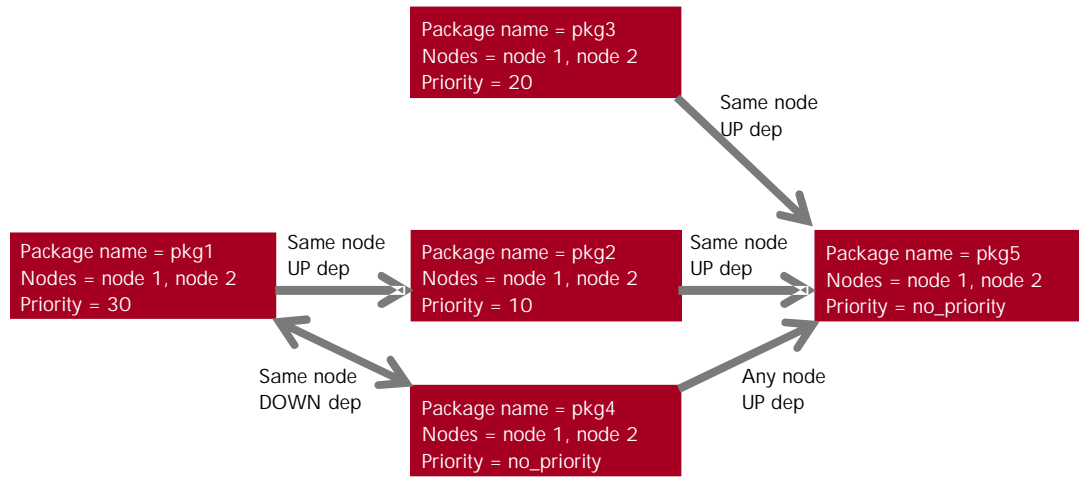
The rule for dragging applies to the entire dependency graph. In order to drag, the package must have higher priority than all the packages that it directly or indirectly needs to halt or move.

Automatic mode examples

The following examples use Figure 1 below. This figure shows five packages:

- Pkg1 can run on two nodes, node1 and node2. This package depends on pkg2 to be up on the same node and it has a same-node exclusionary dependency on pkg4. pkg1's priority is 30.
- Pkg2 can run on two nodes, node1 and node2. This package depends on pkg5 to be up on the same node. Pkg2's priority is 10. This is the highest priority package.
- Pkg3 can run on two nodes, node1 and node2. This package depends on pkg5 to be up on the same node. Pkg3's priority is 20.
- Pkg4 can run on two nodes, node1 and node2. This package depends on pkg5 to be up on any node in the cluster and it depends on pkg1 to be down on the same node. Pkg4's priority is no_priority.
- Pkg5 can run on two nodes, node1 and node2. pkg5's priority is no_priority.

Figure 1: Automatic mode example



Initial startup example

Let's assume all the packages in Figure 1 above are down. If they all have `auto_run` enabled, `pkg1`, `pkg2`, `pkg3` and `pkg5` start on `node1` and `pkg4` starts on `node2`. `pkg2` is the highest priority package and prefers to run on `node1`. It requires `pkg5` to run on the same node. So `pkg2` and `pkg5` will run on `node1`. `pkg1` requires `pkg2` to be up on the same node and it requires `pkg4` to be down on the same node. Since `pkg1` has higher priority than `pkg4`, and `pkg2` is running on `node1`, `pkg1` will run on `node1`. `pkg3` will start on `node1` where `pkg5` is running. `pkg4` prefers to run on `node1`. But it requires `pkg1` to be down on the same node. Since it has lower priority than `pkg1` and `pkg1` is running on `node1`, `pkg4` cannot run on `node1`. Since `pkg5` is running on `node1`, `pkg4`'s `any_node` dependency is satisfied on `node2`. So it will run on `node2`.

Package and node failure examples

In Figure 1 above, say `pkg1`, `pkg2`, `pkg3`, and `pkg5` are running on `node1` and `pkg4` is running on `node2`. The following sections provide examples of how Serviceguard responds to package and node failures.

What if `pkg1` fails?

If `pkg1` fails, it will go down and not fail over, because it requires `pkg2` to be running on the same node. Since `pkg2` has higher priority than `pkg1`, `pkg1` cannot drag `pkg2` to another node.

What if `pkg2` fails?

`pkg2` is the highest-priority package on the dependency graph. So if both `pkg2` and the package that it depends on (`pkg5`) can run on `node2`, Serviceguard will halt or move any package in order to run `pkg2` on `node2`. That means that `pkg1`, `pkg2` and `pkg5` will have to be halted and moved. Since `pkg3` and `pkg4` depend on `pkg5`, they will have to be halted also. Serviceguard will halt `pkg1` before `pkg2` and will halt `pkg2`, `pkg3` and `pkg4` before `pkg5` (this assumes the default value for the `successor_halt_timeout` parameter; see the `cmmakepkg` manpage for the description of this parameter). Once `pkg5` is halted, Serviceguard will start `pkg5` on `node2`. Then it will start `pkg2` and `pkg3`. `pkg1` will start on `node node2` after `pkg2` is up. Since `pkg1` requires `pkg4` to be down on the same node and it has higher priority than `pkg4`, `pkg4` starts on `node1`.

What if pkg3 fails?

If pkg3 fails on node1, it will halt and will not be failed over. In order for pkg3 to fail over to node2, it requires pkg5 on the same node. So it needs to drag pkg5 to node2. Pkg3 has higher priority than pkg5 and it can drag pkg5. Moving pkg5 to node2 means pkg2 will need to move to node2. So, pkg3 will have to drag pkg2 also. Pkg3 cannot drag pkg2 because pkg2 has higher priority.

This shows that the rule for dragging applies to the entire dependency graph. If a package fails, it cannot drag other packages unless it has higher priority than all the packages that it directly or indirectly halts or moves. In this example, in order to start pkg3 on node2, we need to move or restart pkg1, pkg2, pkg4, and pkg5.

What if pkg4 fails?

If pkg4 fails on node2, it will halt. It cannot be started on node1 because pkg1 has a higher priority and is running on node1. However, if you use the `cmhaltpkg` command to halt pkg1 on node1, pkg4 will start on node1.

What if pkg5 fails?

If pkg5 fails on node1, this will cause pkg2, pkg3 and pkg4 to fail. Since pkg1 depends on pkg2, pkg1 will fail also. Serviceguard will halt these packages in the order described for pkg2's failure and will restart pkg1, pkg2, pkg3, pkg4 and pkg5 as described for pkg2's failure above.

What if node1 fails?

If node1 fails, pkg2, pkg3 and pkg5 will start on node2. Since pkg1 has higher priority than pkg4 and requires pkg4 to be down on node2, pkg4 will be halted and pkg1 will start on node2.

What if node2 fails?

If node2 fails, pkg4 will go down. It will not be restarted because of the exclusionary dependency with the higher priority package pkg1.

Assigning priorities

In order to determine what priority to assign to packages with dependencies, you need to decide if a dependent package should be able to halt or move other packages in order to meet its dependencies. This section shows how to assign priorities in different scenarios:

1. You have three packages. PkgA requires that pkgB run on the same node, and pkgB requires that pkgC run on the same node. PkgA, pkgB, and pkgC should act as a group. If any of them fails, they should go where all can run.
 - Assign the highest priority to pkgA and 2nd highest priority to pkgB.
 - Set pkgA's priority to 10, pkgB's to 20 and pkgC's to 30.
2. You have six packages. PkgA, pkgB, and pkgC should act as a group as in previous scenario. PkgD, pkgE, and pkgF should act as group as in previous scenario.
 - Assign the highest priority to pkgA and 2nd highest priority to pkgB in their group.
 - Assign the highest priority to pkgD and 2nd highest priority to pkgE in their group.
 - Set pkgA's priority to 10, pkgB's to 20 and pkgC's to 30.
 - Set pkgD's priority to 40, pkgE's to 50 and pkgF's to 60
3. You have three packages. PkgA requires that pkgB run on the same node, and pkgB requires that pkgC run on the same node. If pkgA fails, it should not affect pkgB and pkgC (pkgA should not drag pkgB and pkgC). But pkgB and pkgC should act as a group. If pkgB fails, it should drag pkgC.

- Assign the highest priority to pkgB. If pkgB is the highest priority, the priorities assigned to pkgA and pkgC do not matter.
 - Set pkgB's priority to 10, pkgA's to 20 and pkgC's to 30.
4. You have three packages. PkgA requires that pkgB run on the same node and pkgB requires that pkgC run on the same node. If pkgA fails, it should not affect pkgB. If pkgB fails, it should not affect pkgC. Neither pkgA nor pkgB should drag pkgC.

Note: This is the default behavior if you do not assign priority to packages with the same node up dependencies.

- Solution 1:
 - Set priority of all three packages to “no_priority”
- Solution 2:
 - Assign the highest priority to pkgC. If pkgC is the highest priority, the priorities assigned to pkgA and pkgB do not matter.
 - Set pkgC's priority to 10, pkgA's to 20 and pkgB's to 30.

5. You have 3 packages. Both pkgA and pkgB require pkgC to be UP on the same node. If pkgB fails, it should not affect pkgA. If pkgA fails, it should be able to drag pkgC to a node where both can run. If pkgA is not up, pkgB should be able to drag pkgC.

- Assign the highest priority to pkgA.
- Assign the 2nd highest priority to pkgB.
- Set pkgA's priority to 10, pkgB's to 20 and pkgC's to 30.

6. You have three packages. PkgA depends on pkgC on the same node. PkgB depends on pkgC on any node. If pkgA fails, it should be able to drag pkgC to a node where both can run.

- Assign the highest priority to pkgA
- Since pkgB has an "any node" dependency on pkgC, its priority must be the same or lower than pkgC.
- Assign 10, 30, and 20 to pkgA, pkgB and pkgC respectively. Another solution would be to assign say priority of 10 to pkgA and "no_priority" to pkgB and pkgC.

7. You have three packages. PkgA and pkgB depend on pkgC to be UP on the same node. If either pkgA or pkgB fails, they should not affect each other.

Note: this is the default behavior if you do not assign priority to packages with the same node up dependencies.

- Solution 1:
 - Set priority of all three packages to "no_priority"
- Solution 2:
 - Assign the highest priority to pkgC. Since pkgC is the highest priority, the priorities of pkgA and pkgB do not matter.
 - Assign priority 10 to pkgC. Assign 20 and 30 to pkgA and pkgB respectively.

8. You have three packages. PkgA requires both pkgB and pkgC to be UP on the same node. If A fails, it should drag the others to a node where all can run.

- Make pkgA the highest priority package. The relative priorities of pkgB and pkgC do not matter.
- Assign 10, 20, and 30 to pkgA, pkgB, and pkgC respectively.

For more information

To learn more about HP Serviceguard Solutions for high availability and disaster recovery, please visit:

www.hp.com/go/serviceguardsolutions

For technical documentation:

www.docs.hp.com/hpux/ha

© Copyright 2005-2009 Hewlett-Packard Development Company, L.P. The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Linux is a U.S. registered trademark of Linus Torvalds. Microsoft and Windows are U.S. registered trademarks of Microsoft Corporation. UNIX is a registered trademark of The Open Group.

April 2009

