Hewlett Packard Enterprise

HPE Serviceguard Developer's Toolbox

Part Number: 766172-002
Published: June 2017

Contents

Notices

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise 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. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Links to third-party websites take you outside the Hewlett Packard Enterprise website. Hewlett Packard Enterprise has no control over and is not responsible for information outside the Hewlett Packard Enterprise website.

Overview

HPE Serviceguard is a specialized software package for protecting mission-critical applications from a wide variety of hardware and software failures. Serviceguard monitors the health of each node in the cluster and rapidly responds to failures in a way that minimizes application downtime.

Hewlett Packard Enterprise provides Serviceguard Toolkits and Toolbox for integration of popular applications with Serviceguard. The Toolbox can be used for integrating any application with Serviceguard, whereas the Toolkit is used for integrating specific applications.

This document discusses how to install and configure Serviceguard Developer's Toolbox in Linux and HP-UX environments. The Toolbox includes a standardized integration template written in Posix Shell, and validation guidelines.

This user's guide describes the modular Toolkit framework, which minimizes the effort required for deploying an application in an Serviceguard cluster environment. This document covers the following topics:

- How to integrate an application with Serviceguard
- HPE Serviceguard Toolkits
- Modular Toolkit framework

Audience

This guide is useful for developers who use Serviceguard for high availability. The audience include software vendors (ISVs), partner organizations in Hewlett Packard Enterprise, and customers.

Advantages

The Toolbox framework leverages the design from Serviceguard Toolkits, and provides the following benefits:

- Reduction in application integration effort
- Application integration modularization
- Ease of deployment
- Simplification of application maintenance and upgrades
- Appropriate failover decisions
- Reduction in testing effort

To effectively use the Toolbox, you must have a working knowledge of Serviceguard. The template can then be customized for the application, and integrated with Serviceguard, implementing functions to allow Serviceguard to start, stop, and monitor the application.

Dependencies

You must ensure the following on all the cluster nodes:

- Serviceguard, supported version is installed.
- The Toolbox is installed on all the target nodes, where the application package is configured to run.
- The application is installed and configured on all the target nodes.

Prerequisites

In the cluster, depending on the operating system, HPE Serviceguard Linux or Serviceguard HP-UX must be installed on all the nodes that will be configured.

Downloading the Toolbox

Hewlett Packard Enterprise partners and ISVs, can access Toolbox from **http://www.hpe.com/dspp**—> Resources & Downloads —> Technologies & Tools —> High Availability -> Software/Products —> HPE Serviceguard Developer's Toolbox.

This Toolbox is also available as a free download from software depot at **http://www.hpe.com/downloads/sgdtoolbox**.

Contact **ha-toolkits@hpe.com** for questions or feedback about Toolbox.

Installing Serviceguard developer's Toolbox

This section describes the procedure to install Serviceguard developer's Toolbox.

1. Untar and copy all the files into a directory.

   NOTE:

   The Toolbox contains two files in a tar file. Ensure that all the files are copied into the same directory.

2. Run `setup.sh`, and when prompted choose appropriate selection to install or uninstall.

   The following files are installed on the successful completion of the setup:

   - `tkit_module.sh`: The main Toolkit script that is called by the master control script of Serviceguard. It is installed in the following location:
     - **HP-UX** — /etc/cmcluster/scripts/ecmt/toolbox/tkit_module.sh
     - **RHEL** — /usr/local/cmcluster/conf/scripts/tkit/toolbox/tkit_module.sh
     - **SLES** — /opt/cmcluster/conf/scripts/tkit/toolbox/tkit_module.sh
   - `toolbox.1`: Attribute Definition File that defines the Toolbox attributes included in the package configuration file. It is installed in the following location.
     - **HP-UX** — /etc/cmcluster/modules/ecmt/toolbox/toolbox.1
     - **RHEL** — /usr/local/cmcluster/conf/modules/tkit/toolbox/toolbox.1
     - **SLES** — /opt/cmcluster/conf/modules/tkit/toolbox/toolbox.1

       A soft link named *Toolbox* pointing to the ADF file `toolbox.1` is created in the respective directories.

Application integration with HPE Serviceguard

To manage an application with Serviceguard using the Toolbox, you must create an application package, which is the basic entity being managed and moved within the Serviceguard cluster. To initiate package creation, you must generate a package configuration file using the `cmmakepkg` command.

NOTE:

Serviceguard 11.18.05 or later in Linux, Serviceguard A.11.18 and additional platform specific patches `PHSS_38423` (on HP-UX 11iv2), `PHSS_38424` (on HP-UX 11iv3) must be installed before creating the package. Toolbox version A.12.00.00 supports only modular packages.

To create a package:

HP-UX — `#cmmakepkg -m ecmt/toolbox/toolbox pkg.conf`

RHEL — `#cmmakepkg -m tkit/toolbox/toolbox pkg.conf`

SLES — `#cmmakepkg -m tkit/toolbox/toolbox pkg.conf`

While configuring a package configuration file, you must define a set of application services that are run by the package manager when a package starts up on a node in the cluster.

The configuration includes cluster nodes on which the package can run, together with definitions of the acceptable types of failover allowed for the package.

For more information, see the sample configuration files provided in Appendix A:**MySQL Toolkit scripts** on page 17

The Package configuration file contains the information necessary to start the application processes in the package, monitor the packages during operation, react to a failure, and halt the package when necessary.

For more information about package configuration and management in HP-UX and Linux, see the latest version of Managing Serviceguard user guide at **http://www.hpe.com/info/hpux-serviceguard-docs** —> Serviceguard and **http://www.hpe.com/info/linux-serviceguard-docs** respectively.

High Availability (HA) Toolkits

HA Toolkits are a set of scripts that integrate popular applications with Serviceguard. Toolkits are available for both HP-UX and Linux platforms.

ECMT is a bundle of Toolkits for the following popular applications on HP-UX:

- Database Toolkits (Oracle's single instance database application including Oracle 9i, Oracle 10g R1, Oracle 10g R2 with ASM, Oracle 11gR1 and 11gR2, Postgre SQL, and MySQL)
- Internet Toolkits (including HPE Apache, HPE Tomcat).

HA Toolkits on Linux (RHEL and SLES) include:

- File system Toolkits (NFS and Samba)
- Database Toolkits (Oracle's single instance database application including Oracle 10g R1, Oracle 10g R2 with ASM, Oracle 11g R1 and 11g R2, Postgre SQL, and MySQL)
- Internet Toolkits (Apache, Tomcat, and Sendmail)

The Toolbox framework provides a set of scripts that is installed on top of Serviceguard's package manager. In this framework, the application interaction is separated from the control script of Serviceguard. This allows the same Toolkit to be used across OS platforms as well as Serviceguard versions. This framework reduces the work required "up front", and reduces the overall maintenance and testing required for supporting the Toolkit.

A Toolkit, created from the template determines when the application is to failover to a standby node in the cluster. It interprets the exit code from the application, logs meaningful messages and enables Serviceguard to take appropriate action (failover/restart) based on the package configuration.

Modular Toolkit framework

The Toolkit framework described in this section is intended to simplify the integration of applications with Serviceguard. The framework is designed to have the following features:

- Starting and stopping an application
- Monitoring the health of an application
- Easy deployment and maintenance of an application
- Reusing code components

The Toolkit framework consists of the following modules:

- **Toolkit module script (tkit_module.sh)**

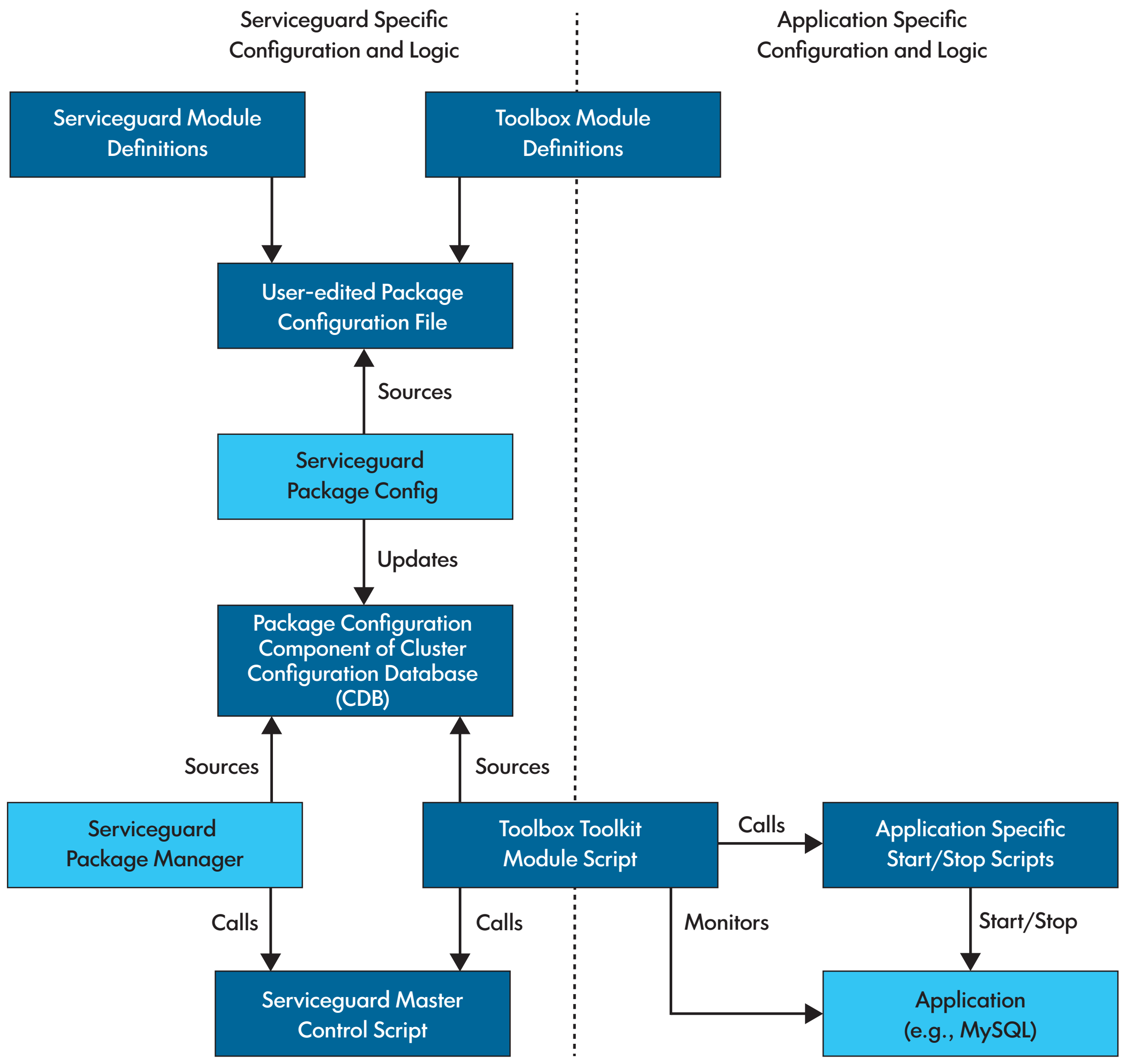
  This is the main Toolkit script that is called by Serviceguard's master control script. This script has fixed entry points independent of the application being integrated and as such provides an abstraction of the monitored applications to Serviceguard.

  The `tkit_module.sh` script validates and loads the application configuration parameters, and performs start and stop of application services. It also contains monitoring functions for the application monitored. The commonly used monitoring criteria include monitoring the process through the `ps` command and monitoring the listening port by using the `netstat` command.

  You can enhance the Toolbox monitor function by adding functionality. For example, you might add the functionality to query a database application or you might add the functionality to read or write to a file system application.

  When the monitor function detects an application failure, it returns fail status that is a non-zero number to Serviceguard, thereby enabling it to take appropriate action (failover, restart and so on) based on the package configuration.
- **Attribute Definition File (toolbox.1)**

The `cmmakepkg` command uses the `toolbox.1` file to include the application specific configuration variables into the package configuration file. The ADF also defines various properties of the variables specified such as type, legal values, comments, and so on. When the package configuration file is validated or applied using the `cmcheckconf` and `cmapplyconf` command respectively, the values assigned to the application specific variables are verified for conformity with the properties defined in the ADF file.



Serviceguard Specific Configuration and Logic
Application Specific Configuration and Logic
Serviceguard Module Definitions
Toolbox Module Definitions
User-edited Package Configuration File
Sources
Serviceguard Package Config
Updates
Package Configuration Component of Cluster Configuration Database (CDB)
Sources
Sources
Serviceguard Package Manager
Toolbox Toolkit Module Script
Calls
Application Specific Start/Stop Scripts
Calls
Calls
Monitors
Start/Stop
Serviceguard Master Control Script
Application (e.g., MySQL)

Figure 1: Toolbox architecture

As shown in in the figure, when the package configuration file is created including the Toolbox module, edited, and applied, the configuration variables are updated in the cluster configuration database (CDB). Serviceguard package manager sources these values from the CDB which includes details of scripts to be executed in a sequence to bring up or bring down the package. The sequence includes the execution of the `tkit_module.sh` script. After sourcing, it calls the Serviceguard master control script to execute the sequence of the scripts.

The Serviceguard master control script executes each script in the sequence passed by the Serviceguard master control script by passing start, stop, or validate arguments.

When the `tkit_module.sh` is called with the `start` argument by the Serviceguard master control script, it executes the application start script. The application start script is written by the user for the application intended to be integrated with Serviceguard. The application start script must return a zero if the application starts successfully, and a non-zero value if the application fails to start.

When the `tkit_module.sh` is called with `stop` argument, it executes the application stop script. Application stop script is written by the user for the application intended to be integrated with Serviceguard. The application start script must return a zero if the application starts successfully, and a non-zero value if the application fails to start.

Apart from these arguments, the user is required to call the `tkit_module.sh` script with the `toolbox_monitor` argument in the `service_cmd` of the package configuration file. This call is automatically generated in the package configuration file when it is created using the Toolbox module. When the `tkit_module.sh` script is called with the `toolbox_monitor` argument, it runs a monitor_function within which it monitors the health of the application.

This Toolbox contains the following:

◦ The Toolbox template
◦ The test tool

To integrate an application with Serviceguard, customize the Toolkit template for the specific application. Copy the customized scripts to the package directory on each of the nodes in the cluster that will be configured to run the application.

Configuring Toolkits

The following section describes the implementation details of the Toolkit framework modules and modifications required for easily incorporating the application specific details. For more information, see the sample Toolkit scripts in Appendix A: **MySQL Toolkit scripts** on page 17.

Toolkit package configuration file (pkg.conf)

You must generate the `pkg.conf` using the `cmmakepkg` command by including the Toolbox module as described in the **Application integration with HPE Serviceguard** on page 6section. This file contains all the configuration variables required for a package including user configured variables. The following section describes only the Toolbox specific variables.

NOTE:

In Serviceguard Version 11.19 and later, all the Toolbox specific configuration variables mentioned in this section are prefixed with `tkit/toolbox/toolbox/` in Linux and `ecmt/toolbox/toolbox/` in HP-UX.

- *Toolkit Configuration Directory (TKIT_CONF_DIR)*

  The Toolkit configuration directory is the place where the package configuration file resides. For ease of use, you can consider placing the user written application start and application stop scripts in this directory.
- *Maintenance Flag (TKIT_MAINTENANCE_FLAG)*

  You can set the value of this parameter to either **yes** or **no** to enable or disable maintenance operations respectively.

  When set to **yes**, the `tkit_module.sh` determines the existence of the `MAINTENANCE_FILE` file in the Toolkit configuration directory, where MAINTENANCE_FILE= `toolbox.debug`.

  If the `toolbox.debug` file exists and TKIT_MAINTENANCE_FLAG is set to **yes**, the application monitoring is suspended, and application maintenance can be performed without halting the package. This means the application can be halted or started without bringing down the package, so that package resources (including volume groups, relocatable IPs, and so on.) are intact and available.

To continue monitoring by exiting the maintenance mode, remove the `toolbox.debug` file. Ensure that the application instance is running properly before exiting the maintenance mode, because the monitoring of the instance resumes after the package exits the maintenance mode.

The default setting for this variable in the Toolkit template is **yes**. When the package is running, you can initiate the maintenance mode by starting the file `toolbox.debug` in the Toolkit configuration directory.

To resume application monitoring after maintenance, manually remove the file `toolbox.debug` from the Toolkit configuration directory.

- *Monitoring Interval (TKIT_MONITORING_INTERVAL)*

  This parameter specifies the time interval, in seconds, between two successive monitoring verifications to make sure that the application services are up and running. This indicates how frequently the application must be monitored.

  > **NOTE:**
  >
  > In Linux, you can specify the monitoring interval in microseconds. For this, you must modify monitor_function of the `tkit_module.sh` script to use `usleep` (currently supported only in Linux) instead of `sleep`.

- *Monitoring Criteria (TKIT_MONITORING_TYPE/TKIT_MONITORING_ELEMENT)*

  The variable TKIT_MONITORING_TYPE specifies how the application is to be monitored. You can set TKIT_MONITORING_TYPE to one of the following values, or specify an appropriate value based on your own implementation:

  - *PID File (pid_file)*

    Some applications store the process ID in a file. For such applications, you can choose the monitoring criteria (TKIT_MONITORING_TYPE) as “pid_file” (the location of the PID file is to be assigned to the variable, TKIT_MONITORING_ELEMENT). In this case, the Toolkit will monitor the process corresponding to the process ID stored in this file. The name of this process must be assigned to the variable APP_SERVER_PROC.

    > **NOTE:**
    >
    > The APP_SERVER_PROC variable is only used in PID_FILE monitoring. For the rest of the TKIT_MONITORING_TYPE options it is commented out.

  - *Monitoring Process Names (process_monitor)*

    You can choose to monitor a process by name by setting the variable TKIT_MONITORING_TYPE to the “process_monitor”. If multiple processes must be monitored for a single application, list all the processes by name in the TKIT_MONITORING_ELEMENT.
  - *Monitoring Port (port_monitor)*

    Some applications are monitored based on the port to which they listen. You may choose to monitor a port by setting the variable TKIT_MONITORING_TYPE to the “port_monitor”. An example of this type of application is the Tomcat web server. For these types of applications, you must set the port number to the variable, TKIT_MONITORING_ELEMENT.

    The variable TKIT_MONITORING_ELEMENT specifies the actual object being monitored for the selected monitoring type. For example, if you choose to monitor the PID file of an application (

    `/var/run/app/app.pid`

    ), then the configuration would be as follows:

TKIT_MONITORING_TYPE	pid_file
TKIT_MONITORING_ELEMENT	/var/run/app/app.pid
APP_SERVER_PROC	mysqld

If you choose to monitor the process name of an application, then the configuration would be as follows:

TKIT_MONITORING_TYPE	process_monitor
TKIT_MONITORING_ELEMENT	app_proc_name_1
TKIT_MONITORING_ELEMENT	app_proc_name_2
TKIT_MONITORING_ELEMENT	app_proc_name_3

If you choose to monitor the port of an application on which it is listening, then the configuration would be as follows for port number 8080:

TKIT_MONITORING_TYPE	port_monitor
TKIT_MONITORING_ELEMENT	8080

NOTE:

You can provide your own monitoring function. For example,

– Querying the application periodically
– Monitoring a log file for a specific keyword
– For file systems, reading from or writing to, or creating or deleting a file

If you provide your own monitoring function, you must add it to the `tkit_module.sh`, and appropriately set the parameters TKIT_MONITORING_TYPE and TKIT_MONITORING_ELEMENT in the package configuration file. Please note that the monitor function must return the failure status (1) to the Toolkit when it detects the failure of the application, to enable Serviceguard to take appropriate action.

◦ *Application start script (TKIT_APP_START)*

  This variable must be set to the user implemented application start script. The absolute path of the start script must be specified, for example:

```
TKIT_APP_START /etc/cmcluster/pkg_dir/start_app.sh
```

◦ *Application stop script (TKIT_APP_STOP)*

  This variable must be set to the user implemented application stop script. The absolute path of the stop script must be specified, for example:

```
TKIT_APP_STOP /etc/cmcluster/pkg_dir/stop_app.sh
```

◦ *Server Process Name (APP_SERVER_PROC)*

  This variable must be set only if you choose the monitoring criteria as PID file (pid_file). This variable contains the name of the main application daemon, the process id of which is stored in the PID file. That is, if the application for which this Toolkit is developed is MySQL, then

```
APP_SERVER_PROC mysqld
```

.

- *Toolkit Module Script (tkit_module.sh)*

  The Toolkit module script contains the following functions that support validation, starting, stopping, and monitoring of application services:

  - **validate_function**: validates parameters in the package environment.
  - **start_function**: invokes the `TKIT_APP_START` script (to be developed by user) to start the application.
  - **monitor_function**: monitors the application based on the monitoring criteria provided. When an application process that is being monitored fails, the function will set its status to "failure" to be returned to Serviceguard so that appropriate action can be taken.

    This function can monitor application integration using Serviceguard Developer's Toolbox version A. 12.00.00 based on the `pid_file`, `process_monitor`, or `port_monitor`. You can choose any of these or implement your own monitoring function.
  - **stop_function**: invokes the `TKIT_APP_STOP` script (to be developed by user) to stop the application.

    This script is called by the Serviceguard master control script to perform the following:

    - On package startup, it starts the application service and keeps monitoring it.
    - On package halt, it halts the monitoring process and stops the application.

      This script returns “success” status to the Serviceguard master control script if it is successful in performing the specified action; otherwise, it returns a non-zero value. You must not make any changes to this script for integration.

You must implement the application start script (TKIT_APP_START) and stop script (TKIT_APP_STOP). The guidelines for developing these modules are:

- *APPLICATION START SCRIPT (TKIT_APP_START)*

  This script must be developed to perform the following:

  - Validate the application start-up parameters
  - Start the application with the specified configuration options
  - Verify whether the start-up was successful, and return status to the Toolkit module script;

    Success or 0 if the application started successfully, and Failure or 1 if the start-up failed.
- *APPLICATION STOP SCRIPT (TKIT_APP_STOP)*

  This script must be developed to perform the following:

  - Shutdown application services
  - Return the status code to the Toolkit;

    Success or 0 if the application was brought down gracefully, and Failure or 1 if the application halt failed.

    For sample Toolkit scripts see, **MySQL Toolkit scripts** on page 17 .

Using the test tool

The test tool tests a given package for CHO. It continuously fails over the given package between the nodes, and logs any errors reported.

To use the test tool, untar the S`G-toolbox-testtool.tar` in to Toolkit configuration directory. The tool consists of two scripts named `choapp.conf` and `choapp.sh`. Configure the `choapp.conf` file as below:

Variable Name	Description
CL_NAME	The variable takes in the cluster name on which the test is being run as the value. For example, CL_NAME=TbxCluster
PKG_DIR	The variable takes in the absolute path of the Toolkit configuration directory. PKG_DIR=/usr/local/cmcluster/pkg1
PKG_NAME	The variable takes in the Package name on which the test is to be run. For example, PKG_NAME=pkg1
APP	The variable takes in the name of the application that is being tested. For Toolbox, the value must always be **`toolbox`**. For example, APP=toolbox
CHO_SLEEP	The variable indicates the duration until which the package is to be run on any node before its failover. The time specified must be in minutes. For example, CHO_SLEEP=2
CHO_DURATION	The variable indicates the duration up to which the test must be run. The time is specified in minutes. Normally the package is tested for 48 hours, that is, 2880 minutes of CHO. CHO_DURATION=2880

After the configuration is complete, run the `choapp.sh` script to begin the test:

```
#nohup <toolkit configuration directory>/choapp.sh &
```

The output is logged in the `cho <package name>.log` file in the toolkit configuration directory.

Related documentation

For related information, see the latest versions of the following documents:

- Managing Serviceguard manual at **http://www.hpe.com/info/hpux-serviceguard-docs** or **http://www.hpe.com/info/linux-serviceguard-docs**.
- High Availability Solutions section of the Hewlett Packard Enterprise website at **http://www.hpe.com/info/serviceguard**.
- Modular package support in Serviceguard for Linux Toolkits manual at **http://www.hpe.com/info/linux-serviceguard-docs**.
- Modular package support in Serviceguard for Linux and ECMT Toolkits manual at **http://www.hpe.com/info/linux-serviceguard-docs**.

Support and other resources

Accessing Hewlett Packard Enterprise Support

- For live assistance, go to the Contact Hewlett Packard Enterprise Worldwide website:

  www.hpe.com/assistance
- To access documentation and support services, go to the Hewlett Packard Enterprise Support Center website:

  www.hpe.com/support/hpesc

Information to collect

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

Accessing updates

- Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.
- To download product updates, go to either of the following:
  - Hewlett Packard Enterprise Support Center **Get connected with updates** page:

    www.hpe.com/support/e-updates
  - Software Depot website:

    www.hpe.com/support/softwaredepot
- To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center **More Information on Access to Support Materials** page:

  www.hpe.com/support/AccessToSupportMaterials

IMPORTANT:

Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HP Passport set up with relevant entitlements.

Websites

Website	Link
Hewlett Packard Enterprise Information Library	**www.hpe.com/info/enterprise/docs**
Hewlett Packard Enterprise Support Center	**www.hpe.com/support/hpesc**
Contact Hewlett Packard Enterprise Worldwide	**www.hpe.com/assistance**
Subscription Service/Support Alerts	**www.hpe.com/support/e-updates**
Software Depot	**www.hpe.com/support/softwaredepot**
Customer Self Repair	**www.hpe.com/support/selfrepair**
Insight Remote Support	**www.hpe.com/info/insightremotesupport/docs**
Serviceguard Solutions for HP-UX	**www.hpe.com/info/hpux-serviceguard-docs**
Single Point of Connectivity Knowledge (SPOCK) Storage compatibility matrix	**www.hpe.com/storage/spock**
Storage white papers and analyst reports	**www.hpe.com/storage/whitepapers**

Customer self repair

Hewlett Packard Enterprise customer self repair (CSR) programs allow you to repair your product. If a CSR part needs to be replaced, it will be shipped directly to you so that you can install it at your convenience. Some parts do not qualify for CSR. Your Hewlett Packard Enterprise authorized service provider will determine whether a repair can be accomplished by CSR.

For more information about CSR, contact your local service provider or go to the CSR website:

www.hpe.com/support/selfrepair

Remote support

Remote support is available with supported devices as part of your warranty or contractual support agreement. It provides intelligent event diagnosis, and automatic, secure submission of hardware event notifications to Hewlett Packard Enterprise, which will initiate a fast and accurate resolution based on your product’s service level. Hewlett Packard Enterprise strongly recommends that you register your device for remote support.

For more information and device support details, go to the following website:

www.hpe.com/info/insightremotesupport/docs

Documentation feedback

Hewlett Packard Enterprise is committed to providing documentation that meets your needs. To help us improve the documentation, send any errors, suggestions, or comments to Documentation Feedback (**docsfeedback@hpe.com**). When submitting your feedback, include the document title, part number, edition, and publication date located on the front cover of the document. For online help content, include the product name, product version, help edition, and publication date located on the legal notices page.

MySQL Toolkit scripts

MySQL Toolkit is used to bring MySQL Database Server into Serviceguard environment.

Package Configuration File (pkg.conf):

```
# **********************************************************************
# ****** HIGH AVAILABILITY PACKAGE CONFIGURATION FILE (template) *******
# **********************************************************************
# ******* Note: This file MUST be edited before it can be used. ********
# * For complete details about package parameters and how to set them, *
# * consult the Serviceguard manual.
# **********************************************************************
#
# "package_name" is the name that is used to identify the package.
#
# Package names must be unique within a cluster.
#
# Legal values for package_name:
# Any string that starts and ends with an alphanumeric character, and
# contains only alphanumeric characters, dot(.), dash(-), or underscore(_)
# in between.
# Maximum length is 39 characters
# package_name mysql_pkg
# "package_description" specifies the application that the package runs.
#
#
# package_description specifies the application that the package
# runs. This is a descriptive parameter that can be set to any value you
# choose, up to a maximum of 80 characters. Default is "Serviceguard
# Package".
#
# Example:
# If the package is running the sendmail application, then
# package_description can be set to:
#
# package_description "Sendmail application"
#
# If the package is running the Apache and Tomcat applications, then
# package_description can be set to:
#
# package_description "Apache and Tomcat applications"
#
#
# Legal values for package_description:  package_description "Serviceguard
Package"
# "module_name" specifies the package module from which
# this package was created. Do not change the module_name.
#
# "module_version" indicates the version of the module included in the
# package configuration file. Do not change "module_version".
#
# Legal values for module_name: <Any String>
# Legal values for module_version: (value >= 0)
```

```
module_name                        sg/basic
module_version                 1
module_name                        tkit/toolbox/toolbox
module_version                    1
module_name                        sg/service
module_version                    1
module_name                        sg/all
module_version                    2
module_name                        sg/failover
module_version                    1
module_name                        sg/priority
module_version                    1
module_name                        sg/dependency
module_version                    1
module_name                        sg/weight
module_version                    1
module_name                        sg/monitor_subnet
module_version                    1
module_name                        sg/package_ip
module_version                    1
module_name                        sg/volume_group
module_version                    1
module_name                        sg/filesystem
module_version                    1
module_name                        sg/pev
module_version                    1
module_name                        sg/external_pre
module_version                    1
module_name                        sg/external
module_version                    1
module_name                        sg/acp
module_version                    1
module_name                        sg/pr_cntl
module_version                    2

# "package_type" is the type of package.
#
# The package_type attribute specifies the behavior for this
# package. Legal values and their meanings are:
#
# failover package
                                   runs on one node at a time and if a failure
#                                    occurs it can switch to an alternate
node.
#
# multi_node package
                                   runs on multiple nodes at the same time and
#                                    can be independently started and halted
on
#                                    individual nodes. Failures of package
components such
#                                    as services, EMS resources or subnets,
will cause
#                                    the package to be halted only on the node
on which the
#                                    failure occurred. Relocatable IP
addresses cannot be
```

```
#                                     assigned to "multi_node" packages.
#
# system_multi_node
#                                     package runs on all cluster nodes at the
same time.
#                                     It cannot be started and halted on
individual nodes.
#                                     Both "node_fail_fast_enabled" and
"auto_run"
#                                     must be set to "yes" for this type of
package. All
#                                     "services" must have
"service_fail_fast_enabled" set
#                                     to "yes". system_multi_node packages are
only
#                                     supported for use by applications
provided by
#                                     Hewlett-Packard.
#
#
# Since "multi_node" and "system_multi_node" packages can run on more
# than one node at a time and do not failover in the event of a
# package failure, the following parameters cannot be
# specified when configuring packages of these types:
#
# failover_policy
# failback_policy
#
# Since an IP address cannot be assigned to more than one node at a
# time, relocatable IP addresses cannot be assigned to "multi_node"
# packages. If volume groups are used in a "multi_node" package,
# they must be activated in a shared mode, leaving the application
# responsible for data integrity.
#
# Shared access requires a shared volume manager.
#
# The default value for "package_type" is "failover".
#
# Legal values for package_type: failover, multi_node, system_multi_node.
package_type                             failover
# "node_name" specified which nodes this package can run on.
#
# Enter the names of the nodes configured to run this package, repeat
# this line for each cluster member node configured to run this package.
#
# NOTE: The order in which the nodes are specified here determines the
# order of priority when Serviceguard is deciding where to run the
# package.
#
# Example : node_name first_priority_node
# node_name second_priority_node
#
# If all nodes in the cluster can run the package, and order is not
# important, specify "node_name *".
#
# Example : node_name *
#
```

```
# Legal values for node_name:
# "*", or any node name in the cluster.
# "node name" is any string that starts and ends with an alphanumeric
# character, and contains only alphanumeric characters, dot(.), dash(-),
# or underscore(_) in between.
# Maximum name length is 39 characters.
# node_name *
# "auto_run" defines whether the package is to be started when the
# cluster is started, and if it will fail over automatically.
#
# Possible values are "yes" and "no".
# The default for "auto_run" is "yes", meaning that the package will be
# automatically started when the cluster is started, and that, in the
# event of a failure the package will be started on an adoptive node.
# If "auto_run is "no", the package is not started when the cluster
# is started, and must be started with the cmrunpkg command.
#
# "auto_run" replaces "pkg_switching_enabled".
#
# Legal values for auto_run: yes, no. auto_run yes
# "node_fail_fast_enabled" will cause the node to fail if the package fails.
#
# Possible values are "yes" and "no". The default for
# "node_fail_fast_enabled" is "no". In the event of failure, if
# "node_fail_fast_enabled" is set to "yes", Serviceguard will halt the
# node on which the package is running. All system multi-node packages
# must have "node_fail_fast_enabled" set to "yes".
#
#
# Legal values for node_fail_fast_enabled: yes, no. node_fail_fast_enabled no
# "run_script_timeout" is the number of seconds allowed for the package to
start.
# "halt_script_timeout" is the number of seconds allowed for the package to
halt.
#
# If the start or halt function has not completed in the specified
# number of seconds, the function will be terminated. The default is
# "no_timeout". Adjust the timeouts as necessary to permit full
# execution of each function.
#
# Note: The "halt_script_timeout" should be greater than the sum of
# all "service_halt_timeout" values specified for all services.
#
# Legal values for run_script_timeout: no_timeout, (value > 0).
run_script_timeout no_timeout
# Legal values for halt_script_timeout: no_timeout, (value > 0).
halt_script_timeout no_timeout
# "successor_halt_timeout" limits the amount of time
# Serviceguard waits for packages that depend on this package
# ("successor packages") to halt, before running the halt script of this
# package.
#
# Permissible values are 0 - 4294 (specifying the maximum
# number of seconds Serviceguard will wait).
# The default value is "no_timeout", which means Serviceguard
# will wait as long as it takes for the successor packages to halt.
# A timeout of 0 indicates that this package will halt without
# waiting for successors packages to halt

# Example:
# successor_halt_timeout             no_timeout
# successor_halt_timeout             60
#
# Legal values for successor_halt_timeout: no_timeout, ( (value >= 0) && (value
<= 4294) ).
successor_halt_timeout no_timeout
# "script_log_file" is the full path name for the package control script log
# file. The maximum length of the path name is MAXPATHLEN characters long.
#
# If this parameter is not set, script output is sent to
# $SGRUN/log/<package_name>.log.
#
#
# Legal values for script_log_file:<Any String>

script_log_file                     /usr/local/cmcluster/mysql/log

# "operation_sequence" defines the order in which the individual script
# programs will be executed in the package start action. The package halt
action
# will be executed in the reverse order.
#
# This attribute or list must not be modified. It is not supported if modified.
#
# Legal values for operation_sequence: <Any String>

operation_sequence                                        $SGCONF/scripts/sg/
pr_cntl.sh
operation_sequence                                        $SGCONF/scripts/sg/
external_pre.sh
operation_sequence                                        $SGCONF/scripts/sg/
volume_group.sh
operation_sequence                                        $SGCONF/scripts/sg/
filesystem.sh
operation_sequence                                        $SGCONF/scripts/sg/
package_ip.sh
operation_sequence                                        $SGCONF/scripts/tkit/
toolbox/tkit_module.sh
operation_sequence                                        $SGCONF/scripts/sg/
external.sh
operation_sequence                                        $SGCONF/scripts/sg/
service.sh

# "log_level" controls the amount of information printed
# during validation and package startup or shutdown time.
#
# "log_level" controls the amount of information printed to stdout when
# the package configuration is validated, and to the script_log_file
# when the package starts up and shuts down. Legal values are 0 through
# 5, where 0 is the least amount of logging and 5 is the most. log_level
# 5 includes all information from level 0 to 5. The default value is 0.
#
#
# Level 0 : user visible informative messages
# Level 1 : slightly more detail user visible informative messages
# Level 2 : messages provide logic flow
# Level 3 : messages provide detailed data structure information
# Level 4 : debugging information that provides detailed data
# Level 5 : function call flow
#
# Legal values for log_level: ( (value >= 0) && (value <= 5) ).
#log_level
# "failover_policy" is the policy to be applied when package fails.
#
# This policy will be used to select a node whenever the package needs
# to be started or restarted. The default policy is "configured_node".
# This policy means Serviceguard will select nodes in priority order
# from the list of "node_name" entries.
#
# An alternative policy is "site_preferred". This policy means
# that when selecting nodes from the list of "node_name" entries,
# Serviceguard will give priority to nodes that belong to the site the
# package last ran on, over those that belong to a different site.
#
# Another policy is "min_package_node". This policy means
# Serviceguard will select from the list of "node_name" entries the
# node, which is running fewest packages when this package needs to
# start.
#
#
# Legal values for failover_policy: configured_node, min_package_node,
site_preferred. failover_policy configured_node
# "failback_policy" is the action to take when a package is not running
# on its primary node.
#
# This policy will be used to determine what action to take when a
# package is not running on its primary node and its primary node is
# capable of running the package. The default policy is "manual". The
# "manual" policy means no attempt will be made to move the package back
# to its primary node when it is running on an adoptive node.
#
# The alternative policy is "automatic". This policy means Serviceguard
# will attempt to move the package back to its primary node as soon as
# the primary node is capable of running the package.
#
#
# Legal values for failback_policy: manual, automatic. failback_policy manual
# The "priority" parameter specifies the priority of the package.
#
# This is an optional parameter. Valid values are a number between
# 1 and 3000 or no_priority. Default is no_priority.
# A smaller number indicates higher priority. A package with a
# numerical priority has higher priority than a package with no_priority.
#
# If a number is specified, it must be unique in the cluster.
# To help assign unique priorities, HP recommends you use
# priorities in increments of 10. This will allow you
# to add new packages without having to reassign priorities.
#
# Multi-node and System multi-node package cannot be assigned a priority.
#
```

```
# This parameter is used only when a weight has been defined for a package,
# a package depends on other packages,
# or other packages depend on this package, but can be specified even
# when no weights or dependencies have yet been configured.
# If priority is not configured, the package is assigned the default
# priority value, no_priority.
#
# Serviceguard gives preference to running the higher priority package.
# This means that, if necessary, Serviceguard will halt a package (or
# halt and restart on anther node) in order to run a higher priority
# package. The reason may be:
# * the node's capacity would otherwise be exceeded
# * there is a direct or indirect dependency between the lower and
# higher priority packages.
#
# For example, suppose package pkg1 depends on package pkg2 to
# be up on the same node, both have package switching enabled
# and both are currently up on node node1. If pkg1 needs to
# fail over to node2, it will also need pkg2 to move to node2.
# If pkg1 has higher priority than pkg2, it can force pkg2 to
# move to node2. Otherwise, pkg1 cannot fail over because pkg2 is
# running on node1.
# The following is are examples of package priorities and failover results:
#
# pkg1 priority pkg2 priority results
# 10 20 pkg1 is higher; fails over
# 20 10 pkg1 is lower; will not fail over
# any number no_priority pkg1 is higher; fails over
# no_priority no_priority equal priority; will not fail over
# no_priority any number pkg1 is lower; will not fail over
#
# Legal values for priority: no_priority, ( (value >= 1) && (value <= 3000) ).

priority                     no_priority

# ***********************APP TOOLKIT PARAMETERS*****************
#
# This directory holds the toolkit configuration file.To put toolkit into
# maintenance mode, create toolbox.debug under this directory.
#
# Legal values for tkit/toolbox/toolbox/TKIT_CONF_DIR:<Any String>
```

tkit/toolbox/toolbox/TKIT_CONF_DIR /usr/local/cmcluster/mysql

```
# Maintenance flag is used to bring this toolkit into maintenance mode.If set
to
# "yes" then this will enable maintenance feature in the toolkit. App Toolkit
will
# look out for a file "toolbox.debug" in the TKIT_CONF_DIR directory where all
the
# application toolkit files reside.If the file exists, monitoring is paused.
#
# Application can be brought down for maintenance and package would not be
failed
# over to the adoptive node eventhough application instance has been brought
down
# for maintenance.
#
```

```
# After the maintenance work, it is the user's responsibility
# to make sure that Application is brought up properly.You should delete the
file
# "toolbox.debug" in the package directory. This would enable toolkit to
restart
# monitoring the application.
#
# Note: if Maintenance flag is set to "no" then the above feature would not be
# available.
#
# Legal values for tkit/toolbox/toolbox/TKIT_MAINTENANCE_FLAG: yes, no.

tkit/toolbox/toolbox/TKIT_MAINTENANCE_FLAG                 yes
#
# The interval (in seconds) between checking if application is are up and
running.
# The default setting is 5 seconds.
#
#
# Legal values for tkit/toolbox/toolbox/TKIT_MONITOR_INTERVAL: <Any String>

tkit/toolbox/toolbox/TKIT_MONITOR_INTERVAL            5
# Legal values for tkit/toolbox/toolbox/TKIT_MONITORING_TYPE: <Any String>

tkit/toolbox/toolbox/TKIT_MONITORING_TYPE             pid_file

# Legal values for tkit/toolbox/toolbox/TKIT_MONITORING_ELEMENT: <Any String>

tkit/toolbox/toolbox/TKIT_MONITORING_ELEMENT         /var/run/mysqld/mysqld.pid
# Legal values for tkit/toolbox/toolbox/TKIT_APP_START: <Any String>

tkit/toolbox/toolbox/TKIT_APP_START                       /usr/local/cmcluster/
mysql/start_app.sh

# Legal values for tkit/toolbox/toolbox/TKIT_APP_STOP: <Any String>

tkit/toolbox/toolbox/TKIT_APP_STOP                          /usr/local/cmcluster/
mysql/stop_app.sh


# Legal values for tkit/toolbox/toolbox/APP_SERVER_PROC: <Any String>

tkit/toolbox/toolbox/APP_SERVER_PROC                 mysqld
# Services: A service is an long lived (daemon) executable which
# Serviceguard will monitor while the package is up.
#
# "service_name", "service_cmd", "service_restart", "service_fail_fast_enabled"
# and "service_halt_timeout" specify a service for this package.
#
# "service_cmd" is the command line to be executed to start the service.
#
# The value for "service_restart" can be "unlimited", "none" or any
# positive integer value. If the value is "unlimited" the service will be
# restarted an infinite number of times. If the value is "none", the
# service will not be restarted. If the value is a positive integer,
# the service will be restarted the specified number of times
# before failing. If "service_restart" is not specified, the
# default will be "none".
#
# The value for "service_fail_fast_enabled" can be either "yes" or "no".
# The default is "no". If "service_fail_fast_enabled" is set to "yes",
# and the service fails, Serviceguard will halt the node on which the
# service is running.
#
# "service_halt_timeout" is a number of seconds.
# This timeout is used to determine the length of time
# Serviceguard will wait for the service to halt before a SIGKILL signal
# is sent to force the termination of the service. In the event of a
# service halt, Serviceguard will first send a SIGTERM signal to
# terminate the service. If the service does not halt, Serviceguard will
# wait for the specified "service_halt_timeout", then send
# the SIGKILL signal to force the service to terminate.
# This timeout value should be large enough to allow all cleanup
# processes associated with the service to complete. If the
# "service_halt_timeout" is not specified, a zero timeout will be
# assumed, meaning the cluster software will not wait at all # before sending
the SIGKILL signal to halt the service.
#
# Example:
# service_name                                        service_1a
# service_cmd                                          "/usr/bin/X11/xclock -
display 192.10.25.54:0"
# service_restart                                   none
# service_fail_fast_enabled             no
# service_halt_timeout                         300
#
# service_name                                        service_1b
# service_cmd                                         "/usr/bin/X11/xload -
display 192.10.25.54:0"
# service_restart                                  2
# service_fail_fast_enabled             no
# service_halt_timeout                         300
#
# service_name service_1c
# service_cmd "/usr/sbin/ping node_a"
# service_restart unlimited
# service_fail_fast_enabled no
# service_halt_timeout 300
#
# Note: Default shell is /usr/bin/sh.
#
# Legal values for service_name:
# Any string that starts and ends with an alphanumeric character, and
# contains only alphanumeric characters, dot(.), dash(-), or underscore(_)
# in between.
# Maximum string length is 39 characters.
#
# Legal values for service_cmd:                                    <Any String>
# Legal values for service_restart:                              none,
unlimited, (value > 0).
# Legal values for service_fail_fast_enabled:     yes, no.
# Legal values for service_halt_timeout:                (value >= 0).

service_name
```

```
         app.monitor
service_cmd
            "/usr/local/cmcluster/conf/scripts/tkit/toolbox/tkit_module.sh
toolbox_monitor"
service_restart
        none
service_fail_fast_enabled                                                no
service_halt_timeout
 300

# The package dependency parameters are "dependency_name",
# "dependency_condition" and "dependency_location".
#
# Dependencies are used to describe the relationship between two packages.
# To define a dependency, "dependency_name" and "dependency_condition"
# are required and "dependency_location is optional.
#
# "dependency_name" must be a unique identifier for the dependency.
#
# "dependency_condition" describes what must be true for
# the dependency to be satisfied.
#
# The syntax is: <package name> = <package status>
#
# The valid values for <package status> are "up" or "down".
#
# "up" means that this package requires the package identified
# by "package_name" to be up (status reported by cmviewcl is "up").
#
# If "up" is specified, the dependency rules are as follows:
#
# * A multi-node package can depend only on another multi-
# node or system multi-node package.
#
# * A failover package whose failover_policy is
# min_package_node can depend only on a multi-node or
# system multi-node package.
#
# * A failover package whose failover_policy is
# configured_node can depend on a multi-node or system
# multi-node package, or another failover package whose
# failover_policy is configured_node.
#
# "down" means that this package requires the package
# identified by "package name" to be down (status reported by
# cmviewcl is "down"). This is known as an exclusion dependency.
#
# This means that only one of these packages can be running at
# any given time.
#
# If the "down" value is specified, the exclusion dependency must be
# mutual; that is, if pkgA depends on pkgB to be down, pkgB must
# also depend on pkgA to be down.
#
# This means that in order to create an exclusion dependency
# between two packages, you must apply both packages to the
# cluster configuration at the same time.
```

```
#
# An exclusion dependency is allowed only between failover
# packages with configured_node as failover policy, and at least one
# of the packages must specify a priority.
#
# "dependency_location"
# This describes where the condition must be satisfied.
#
# This parameter is optional. If it is not specified, the default
# value "same_node" will be used.
#
# The possible values for this attribute depend on the
# dependency condition.
#
# If an "up" dependency is specified, the possible values
# are "same_node", "any_node", and "different_node".
#
# "same_node" means the dependency must be satisifed on
# the same node.
#
# "any_node" means the dependency can be satisified on
# any node in the cluster.
#
# "different_node" means the dependency must be satisfied
# on a node other than the dependent package's node.
#
# If a "down" dependency is specified, the possible values
# are "same_node" and "all_nodes".
#
# "same_node" means the package depended on must be down on
# the same node.
#
# "all_nodes" means the package depended on must be down on
# all nodes in the cluster.
#
# NOTE:
# Within a package, you cannot specifiy more than one dependency on the
# same package. For example, pkg1 cannot have one same_node and one
# any_node dependency on pkg2.
#
# When a package requires that another package be up and the
# dependency_location is any_node or different_node, the priority of the
# the package depended on must be higher or equal to the dependent
# package and its dependents. For example, if pkg1 has a same_node
# dependency on pkg2 and pkg2 has an any_node dependency on pkg3,
# the priority of pkg3 must be higher or equal to the priority of
# pkg1 and pkg2.
#
# In a CFS cluster, the dependencies among the mount point, disk group,
# and system multi-node packages are automatically created by the commands
# that construct those packages.
#
# Example 1 : To specify a "same_node" dependency between pkg1 and pkg2:

# pkg1's ascii configuration file:
#
# dependency_name                                         pkg2_dep
```

```
#
dependency_condition pkg2 =                 up
#
dependency_location                                 same_node
#
# Example 2 : To specify a "same_node" exclusion dependency between
#         pkg1 and pkg2:
#
#     pkg1's ascii configuration file:
#
#         dependency_name                                 pkg2_dep
#         dependency_condition pkg2 =     down
#         dependency_location                     same_node
#
#         pkg2's ascii configuration file:
#
#         dependency_name                                 pkg1_dep
#         dependency_condition pkg1 =     down
#         dependency_location                     same_node
#
#
# Note that pkg1 and pkg2 must be applied at the same time.
#
# Legal values for dependency_name:
# Any string that starts and ends with an alphanumeric character, and
# contains only alphanumeric characters, dot(.), dash(-), or underscore(_)
# in the middle.
# Maximum string length is 39 characters.
#
# Legal values for dependency_condition: <Any String>
# Legal values for dependency_location: same_node, any_node, different_node,
all_nodes.
#dependency_name
#dependency_condition
#dependency_location

# The package weight parameters are the "weight_name" and "weight_value".
#
#
# These optional attributes provide additional data which the
# Serviceguard package manager uses when selecting a node on which to
# place the package. As with all attribute names, they are case
# insensitive.
#
# A package can use this mechanism to define up to four arbitrary
# weight names with corresponding values that are meant to represent
# the runtime resource consumption of the package. In the cluster
# configuration file, you configure capacity limits for the named
# weights on the cluster nodes. During package placement,
# the package manager will ensure the total value of any given named
# weight does not exceed the capacity limit configured for the node.
#
# The "weight_name" is string of up to 39 characters.
# The "weight_value" specifies a value for the named weight that
# precedes it. This is an unsigned floating point value between 0 and
# 1000000 with at most three digits after the decimal point.
#
```

```
# If "weight_name" is specified, "weight_value" must also be specified
# and "weight_name" must come first. To specifiy more than one weight,
# repeat this process.
#
# You can define weights either individually within each package
# configuration file, or by means of a default value in the
# cluster configuration file that applies to all configured
# packages (except system multi-node packages). If a particular
# weight name is defined in both the cluster and package
# configuration files, the value specified in the package
# configuration file takes precedence. This allows you to set
# an overall default, but to override it for a particular package.
#
# For example, if you specify WEIGHT_NAME "memory" with WEIGHT_DEFAULT
# 1000 in the cluster configuration file, and you do not specify a weight
# value for "memory" in the package configuration file for pkgA, pkgA's
# "memory" weight will be 1000. If you define a weight value of 2000 for
# "memory" in the configuration file for pkgA, pkgA's "memory" weight
# will be 2000.
#
# If no WEIGHT_NAME/WEIGHT_DEFAULT value is specified in the cluster
# configuration file for a given CAPACITY, and weight_name and weight_value
# are not specified in this package configuration file for that CAPACITY,
# then the weight_value for this package is set to zero or one depending
# on the capacity name. If the capacity name is the reserved capacity
# "package_limit", the weight_value for this package is set to one;
# otherwise, the weight_value is set to zero.
# For example, if you specify CAPACITY "memory" and do not specify
# a WEIGHT_DEFAULT for "memory" in the cluster configuration file,
# and do not specify weight "memory" in the package configuration
# file for pkgA, then pkgA's "memory" weight will be zero.
#
# Note that cmapplyconf will fail if you define a weight in the
# package configuration file and no node in the cluster configuration
# file specifies a capacity of the same name.
#
# Weight can be assigned only to multi-node packages, and failover packages
# with configured_node as the failover_policy and manual as failback policy.
#
# For more information on how to configure default weights and
# node capacities, see the cmquerycl man page, the cluster configuration
# template file, and the Managing Serviceguard manual.
#
# Example :
# weight_name package_limit
# weight_value 10
#
# This overrides the default value of 1 and sets the weight for this
# package to 10
#
# Legal values for weight_name:
# Any string that starts and ends with an alphanumeric character, and
# contains only alphanumeric characters, dot(.), dash(-), or underscore(_)
# in the middle.
# Maximum string length is 39 characters.
#
# Legal values for weight_value:
```

```
# Any unsigned floating point string. Only 3 digits after the decimal point
# are significant. Maximum string length is 11 characters.
#
#    weight_name
#    weight_value
# "monitored_subnet" specifies the addresses of subnets that are to be
monitored for this package.
#
# Enter the network subnet name that is to be monitored for this package.
# Repeat this line as necessary for additional subnets. If any of
# the subnets defined goes down, the package will be switched to another
# node that is configured for this package and has all the defined subnets
# available.
#
# "monitored_subnet" replaces "subnet".
#
# The subnet names can be IPv4 or IPv6, or a mix of both.
#
# Example :
# monitored_subnet 192.10.25.0
# (netmask=255.255.255.0)
# monitored_subnet 2001::/64
# (netmask=ffff:ffff:ffff:ffff::)
# monitored_subnet 2001::
# (netmask=ffff:ffff:ffff:ffff::)
# Legal values for monitored_subnet: <Any String>
# "monitored_subnet_access" defines how the monitored_subnet is
# configured in the cluster.
#
#
# monitored_subnet_access defines whether access to a monitored_subnet
# is configured on all of the nodes that can run this package, or only
# some. Possible values are "partial" and "full". "partial" means that
# the monitored_subnet is expected to be configured on one or more of
# the nodes this package can run on, but not all. "full" means that the
# monitored_subnet is expected to be configured on all the nodes that
# this package can run on. "full" is the default. (Specifying "full" is
# equivalent to not specifying the monitored_subnet_access at all.)
#
# The monitored_subnet_access is defined per monitored_subnet entry.
#
# Example :
# monitored_subnet 192.10.25.0
# monitored_subnet_access partial
# 192.10.25.0 is available on one
#
# or more nodes of the cluster,
#
# but not all.
#
# monitored_subnet 192.10.26.0
# no monitored_subnet_access entry,
#
# hence this subnet is available
#
# on all nodes of the cluster.
# monitored_subnet 2001::/64
# monitored_subnet_access full
# 2001::/64 is available on all
#
# nodes of the cluster.
#
# Legal values for monitored_subnet_access: partial, full.
    monitored_subnet 192.168.100.0 monitored_subnet_access full
# IP subnets and addresses
#
# "ip_subnet" and "ip_address" specify subnets and
# IP addresses used by this package.
# "ip_subnet_node" specify the nodes on which the subnet is available.
#
# Enter the network subnet name that is to be used by this package,
# along with all the relocatable IP addresses on this subnet to be used
# by this package. Repeat these lines as necessary for additional
# subnets and relocatable IP addresses. The subnets and relocatable
# addresses can be IPv4 or IPv6, or a mix of both.
#
# For each subnet line, enter the name of the nodes on which this subnet
# is available. This is an optional parameter. Default is set to all
# nodes in the cluster.
#
# For example, if this package uses a subnet 192.10.25.0 and 2 IP
# addresses 192.10.25.12 and 192.10.25.13 enter:
# ip_subnet 192.10.25.0
#
# (netmask=255.255.255.0)
# ip_address 192.10.25.12
# ip_address 192.10.25.13
#
# (No ip_subnet_node entry means that subnet 192.10.25.0 is available on
# all nodes of the cluster).
#
# Hint: Run "netstat -i -e" or "ifconfig" to see the configured IP addresses
# and their netmask.
#
# However if only two nodes out of a four node cluster would have this
# subnet available, enter:
# ip_subnet 192.10.25.0
# ip_subnet_node nodeA
# ip_subnet_node nodeB
#
# (netmask=255.255.255.0)
# ip_address 192.10.25.12
# ip_address 192.10.25.13
#
#
# For example, if this package uses two IPv6 addresses 2001::1/64 and
2001::2/64,
# the address prefix identifies the subnet as 2001::/64, which is an available
# subnet.
# Enter:
# ip_subnet 2001::/64
#
# (netmask=ffff:ffff:ffff:ffff::)
# ip_address 2001::1
# ip_address 2001::2
#
# Alternatively the IPv6 IP/Subnet pair can be specified without the prefix
# for the IPv6 subnet.
# Enter:
# ip_subnet 2001::
#
# (netmask=ffff:ffff:ffff:ffff::)
# ip_address 2001::1
# ip_address 2001::2
#
# Hint: In this case, run "netstat -i" and look at the address prefixes
# to find the available IPv6 subnets.
#
# Note that "ip_address" specifies a relocatable IP address, which will
# be added and removed when the package starts and halts.
#
# "ip_subnet" replaces the legacy package control script parameter "subnet".
# "ip_address" replaces the legacy package control script parameter "ip".
#
# Legal values for ip_subnet: <Any String>
# Legal values for ip_subnet_node: /^[0-9A-Za-z][0-9A-Za-z_.\-]*[0-9A-Za-z]$/, /
^[0-9A-Za-z]$/.
# Legal values for ip_address: <Any String>

ip_subnet 192.168.100.0
ip_address 192.168.100.10

# Volume Group Activation
#
# "vgchange_cmd" is the method of activation for LVM volume groups.
#
# Specify the method of activation for volume groups.
# Leave the default ("vgchange_cmd "vgchange -a y") if you want volume
# groups activated in default mode.
#
# "vgchange_cmd" replaces the legacy package control script parameter
"vgchange".
#
# Legal values for vgchange_cmd: <Any String>

vgchange_cmd "vgchange -a y"

# Volume Groups
#
# "vg" is used to specify which volume groups are used by this package.
#
# Specify the name of each volume group.
#


# For example, if this package uses your volume groups vg01 and vg02, enter:
# vg vg01
# vg vg02
#
# The volume groups must not be set if the underlying file system is GFS.
#
```

```
# The volume group activation method is defined above. The filesystems
# associated with these volume groups are specified below.
#
# Legal values for vg: /^[0-9A-Za-z\/][0-9A-Za-z_.\/\-]*[0-9A-Za-z]$/, /^[0-9A-
Za-z]$/. vg mysql_vg
# "concurrent_fsck_operations" specifies the number of concurrent fsck
# operations.
#
# FILESYSTEMS
#
# Filesystems are defined by entries specifying the logical volume, the
# mount point, the options for mount, umount and fsck, and the type of the file
system.
# Each filesystem will be fsck'd prior to being mounted. The filesystems
# will be mounted in the order specified during package startup and will
# be unmounted in reverse order during package shutdown. Ensure that
# volume groups referenced by the logical volume definitions below are
# included in volume group definitions ("vg") above.
#
# concurrent_fsck_operations
#
# Specify the number of concurrent fsck operations to allow during
# package startup.
# Setting this value to an appropriate number may improve performance if
# a large number of file systems need to be checked. The default is 1.
# This attribute is ignored, if the underlying file system is Red Hat
# GFS, since fsck operations will not be performed in case of GFS.
#
# Legal values for concurrent_fsck_operations: (value > 0).
concurrent_fsck_operations 1
# "concurrent_mount_and_umount_operations" specifies the number of concurrent
# mounts and umounts to allow during package startup and shutdown..
#
# Setting this value to an appropriate number may improve the performance
# if the package needs to mount and un-mount a large number of file systems.
# The default is 1.
#
# Legal values for concurrent_mount_and_umount_operations: (value > 0).
concurrent_mount_and_umount_operations 1
# "fs_mount_retry_count" specifies the number of mount retries.
#
# The default is 0. During startup, if a mount point is busy
# and "fs_mount_retry_count" is 0, package startup will fail and
# the script will exit with 1. If a mount point is busy and
# "fs_mount_retry_count" is greater than 0, the script will attempt
# to kill the user process responsible for the busy mount point
# ("fuser -ku") and then mount the file system. It will do this
# the number of times specified in "fs_mount_retry_count".
# If the mount still fails after this number of attempts, the script
# will exit with 1.
#
# fs_mount_retry_count must be set to zero (default), if the underlying
# file system is of type Red Hat GFS.
#
# Legal values for fs_mount_retry_count: (value >= 0). fs_mount_retry_count 0
# "fs_umount_retry_count" specifies the number of unmount retries for each
filesystem during package shutdown.
```

```
#
# The default is set to 1. During package halt time, if a mount point
# is busy the script will attempt to kill the user responsible for the
# busy mount point and then umount the file system. It will attempt to
# kill user and retry umount, for the number of times specified in
# "fs_umount_retry_count". If the umount still fails after this number
# of attempts, the script will exit with 1.
#
#
# NOTE: The script will execute "fuser -ku" to free up busy mount point by
# default.
#
# Specify the number of unmount retries for each filesystem during package
# shutdown. The default is set to 1.
#
# fs_umount_retry_count must be set to 1 (default), if the underlying
# file system is of type Red Hat GFS.
#
# "fs_umount_retry_count" replaces the legacy package control script
# parameter "fs_umount_count".
#
# Legal values for fs_umount_retry_count: (value > 0). fs_umount_retry_count 1
# "fs_name", "fs_directory", "fs_mount_opt", "fs_umount_opt", "fs_fsck_opt",
and
# "fs_type" specify the filesystems which are used by this package.
#
# The "fs_type" parameter lets you specify the type of filesystem to be
# mounted. The only supported file systems are 'ext2', 'ext3', 'reiserfs'
# and 'gfs'.
#
# NOTE: Mixing of 'gfs' with non-gfs filesystems in the same package
# is not permitted. A single package can define either a 'gfs' filesystem
# or a non-gfs filesystem but not both.
#
# The following section applies if the underlying file system is 'ext2',
# 'ext3' or 'reiserfs'.
#
# The filesystems are defined as entries specifying the logical
# volume, the mount point, the file system type, the mount,
# umount and fsck options.
#
# Each filesystem will be fsck'd prior to being mounted.
# The filesystems will be mounted in the order specified during package
# startup and will be unmounted in reverse order during package
# shutdown. Ensure that volume groups referenced by the logical volume
# definitions below are included in volume group definitions.
#
# For example, if this package uses the following:
# logical volume: /dev/vg01/lvol1 /dev/vg01/lvol2
# mount point: /pkg1a /pkg1b
# filesystem type: ext2 reiserfs
# mount options: read/write read/write
#
# Then the following would be entered:
#
# fs_name                 /dev/vg01/lvol1
# fs_directory     /pkg01a
# fs_type                 "ext2"
# fs_mount_opt     "-o rw"
# fs_umount_opt ""
# fs_fsck_opt         ""
#
# fs_name                 /dev/vg01/lvol2
# fs_directory     /pkg02a
# fs_type                 "reiserfs"
# fs_mount_opt     "-o rw"
# fs_umount_opt ""
# fs_fsck_opt         ""
#
# The following section applies if the underlying file system is 'gfs'
#
# The filesystems are defined as entries specifying the physical pool
# device file, the mount point, the file system type and mount options.
# A check is performed to see if the partition is already been mounted
# or not. If the partition is not mounted then it will be mounted.
#
# Specify the filesystems which are used by this package. Uncomment
# fs_name""; fs_directory""; fs_type""; fs_mount_opt"" and fill in
# the name of your first pool, filesystem, type and mount,
# options for the file system.
#
# For example, if this package uses the following:
# GFS6.0 uses pool for logical volume management whereas GFS6.1 uses LVM2.
# Their device name formats differ and an example for each is shown below.
# Use the appropriate one.
# Pool                          : /dev/pool/pool1 (GFS 6.0) OR
# LVM2                          : /dev/mapper/vgX-lvY (GFS6.1)
# mount point             : /pkg1a
# filesystem type : gfs
# mount options          : read/write
#
# Then the following would be entered:
# fs_name /dev/pool/pool1; (GFS6.0) OR
# fs_name /dev/mapper/vgX-lvY; (GFS6.1)
# fs_directory /pkg1a; fs_type "gfs";
# fs_mount_opt "-o rw";
#
# "fs_name" replaces the legacy package control script parameter "lv".
#
# "fs_directory" replaces the legacy package control script parameter "fs".
#
# Legal values for fs_name: /^[^"|]+$/.
# Legal values for fs_directory: /^[^"|]+$/.
# Legal values for fs_type: <Any String>
# Legal values for fs_mount_opt: <Any String>
# Legal values for fs_umount_opt: <Any String>
# Legal values for fs_fsck_opt: <Any String>
fs_name                 /dev/mysql_vg/lvol0
fs_directory     /mysql_mnt
fs_type                 "ext2"
fs_mount_opt     "-o rw"
fs_umount_opt     ""
fs_fsck_opt        ""
# "pev_<name> defines an environment variable for the package.
```

```
#
# package environment variable
#
# You can define one or more variables which will be set as environment
# variables for scripts identified by external script and external pre
# script to use.
#
# The name is a string with the prefix "pev_", that contains only
# alphanumeric characters and underscores(_), for example
# pev_monitoring_interval.
# The value is a string that can contain any character except the pipe (|).
# For example, to define a variable pev_monitoring_interval with
# a value of 30, enter:
#
# pev_monitoring_interval 30
#
# The maximum length for the name (including the prefix) is MAXPATHLEN
# characters long.
#
# The maximum length for the value is MAXPATHLEN characters long.
#
# The prefixed parameter name is case insensitive. The prefix defined
# as "pev_" can be either upper case or lowercase as well. However,
# when used in the control script context, the name is set to
# all upper case characters as an environment variable.
#
# Serviceguard checks only the syntax of package environment
# variables. The values can be validated by means of "validate" section
# in the external script. The section will be called automatically by
# "cmcheckconf -P" and "cmapplyconf -P".
#
# Legal values for pev_<name>: /^[^|]+$/.

#pev_

# "external_pre_script" specifies an additional script to be run during package
# start and halt time.
#
# external_pre_script <program path name>
#
# The specified "external_pre_script" will be executed before
# any volume groups and disk groups are activated during package
# validation and start time. It will be executed after any volume groups
# and disk groups are deactivated during package halt time.
#
# When the program is run the first argument "$1" will be set to "start"
# when starting, "stop" when halting and "validate" when doing a cmapplyconf
# or cmcheckconf.
#
# If more than one "external_pre_script" is specified, the one specified first
# in this file will be run first during package start time, and last
# during package halt time.
#
# NOTE: All "external_pre_script" programs will be run with the first
# argument "$1" set to "validate" when the package is applied to the
# configuration.
#
```

```
# The specified "external_pre_script" will be executed before
# any volume groups and disk groups are activated during package
# validation and start time. It will be executed after any volume groups
# and disk groups are deactivated during package halt time.
#
# When the program is run the first argument "$1" will be set to "start"
# when starting, "stop" when halting and "validate" when doing a cmapplyconf
# or cmcheckconf.
#
# If more than one "external_pre_script" is specified, the one specified first
# in this file will be run first during package start time, and last
# during package halt time.
#
# NOTE: All "external_pre_script" programs will be run with the first
# argument "$1" set to "validate" when the package is applied to the
# configuration.
#
# Examples:
#
# external_pre_script /etc/cmcluster/packages/pkg1a/task0
# external_pre_script /etc/cmcluster/packages/pkg1a/task1
# external_pre_script /etc/cmcluster/packages/pkg1a/task2
#
# The following is the sequence of standard steps (for a package without
# external_pre scripts):
#
# Package starting | Package halting
#                                   |
# STEP SG_SCRIPT_ACTION | STEP SG_SCRIPT_ACTION
#                                                      |
# START | START
# VOLUME_GROUPS start | SERVICES stop
# FILESYSTEMS start | IP_ADDRESSES stop
# IP_ADDRESSES start | FILESYSTEMS stop
# SERVICES start | VOLUME_GROUPS stop
# END | END
#
# When a package includes external_pre scripts, the steps are executed
# in the following order (for both validation and execution):
#
# START
# /etc/cmcluster/packages/pkg1a/task0 start
# /etc/cmcluster/packages/pkg1a/task1 start
# /etc/cmcluster/packages/pkg1a/task2 start
# VOLUME_GROUPS
# FILESYSTEMS
# IP_ADDRESSES
# SERVICES
# END
#
# When the package halts, the steps are executed in the following order:
#
# START
# SERVICES
# IP_ADDRESSES
# FILESYSTEMS
# VOLUME_GROUPS
# /etc/cmcluster/packages/pkg1a/task2 stop
# /etc/cmcluster/packages/pkg1a/task1 stop
# /etc/cmcluster/packages/pkg1a/task0 stop
# END
#
# A script template is available at $SGCONF/examples/external_script.template
#
# Legal values for external_pre_script: <Any String>
#external_pre_script
# "external_script" specify additional programs to be run during package
# starts and halt time.
#
# external_script <program path name>
#
# The "external_script" replaces the customer_defined_run_cmds() and
# customer_defined_halt_cmds() function in the legacy package control script.
#
# The specified "external_script" program will be executed after IP
# addresses are assigned but before services are started during package
# validation and start time. It will be executed after services are
# halted but before removing IP addresses during package halt time.
#
# When the program is run the first argument "$1" will be set to "start"
# when starting, "stop" when halting and "validate" when doing a cmapplyconf
# or cmcheckconf.
#
# If more than one "external_script" is specified, the one specified first
# in this file will be run first during package start time, and last
# during package halt time.
#
#
# NOTE: All "external_script" programs will be run with the first argment "$1"
# set to "validate" when the package is applied to the configuration.
#
# Examples:
#
# external_script /etc/cmcluster/packages/pkg1a/task0
# external_script /etc/cmcluster/packages/pkg1a/task1
# external_script /etc/cmcluster/packages/pkg1a/task2
#
# The following is the sequence of standard steps (for a package without
# external scripts):
#
# Package starting                  | Package halting
                                        |
# STEP                                 SG_SCRIPT_ACTION
                                        |
STEP                                     SG_SCRIPT_ACTION
#
                                        |
# START                                 | START
# VOLUME_GROUPS             start | DEFERRED_RESOURCES stop
# FILESYSTEMS                 start | SERVICES stop
# IP_ADDRESSES             start | IP_ADDRESSES stop
# SERVICES                     start | FILESYSTEMS stop
# DEFERRED_RESOURCES     start | VOLUME_GROUPS stop
# END | END
```

```
#

# When a package includes external scripts, the steps are executed in
# the following order (for both validation and execution):
#
# START
# VOLUME_GROUPS
# FILESYSTEMS
# IP_ADDRESSES
# /etc/cmcluster/packages/pkg1a/task0 start
# /etc/cmcluster/packages/pkg1a/task1 start
# /etc/cmcluster/packages/pkg1a/task2 start
# SERVICES
# DEFERRED_RESOURCES
# END
#
# When the package halts, the steps are executed in the following order:
#
# START
# DEFERRED_RESOURCES
# SERVICES
# /etc/cmcluster/packages/pkg1a/task2 stop
# /etc/cmcluster/packages/pkg1a/task1 stop
# /etc/cmcluster/packages/pkg1a/task0 stop
# IP_ADDRESSES
# FILESYSTEMS
# VOLUME_GROUPS
# END
#
# A script template is available at $SGCONF/examples/external_script.template
#
# Legal values for external_script: <Any String>
#external_script
# Access Control Policy Parameters.
#
# "user_name", "user_host" and "user_role" specify who can administer
# this package.
#
# Three entries set the access control policy for the package: the
# first line must be "user_name", the second "user_host", and the third
"user_role".
# Enter a value after each.
#
# 1. "user_name" can either be "any_user", or a maximum of
# 8 login names from the /etc/passwd file on user host.
# 2. "user_host" is where the user can issue Serviceguard commands.
# Choose one of these three values: "any_serviceguard_node",
# or (any) "cluster_member_node", or a specific node. For node,
# use the name portion of the official hostname supplied by the
# domain name server, not the IP addresses or fully qualified name.
# 3. "user_role" must be "package_admin". This role grants permission
# to "monitor", plus for administrative commands for the package.
#
# These policies do not affect root users. Access Policies defined in
# this file must not conflict with policies defined in the cluster
# configuration file.
#
```

```
# Example: to configure a role for user john from node noir to
# administer the package, enter:
# user_name john
# user_host noir
# user_role package_admin
#
# Legal values for user_name:
# A string of tokens each of which starts with alphanumeric character and
contains
# only alphanumeric and underscore(_) characters. The tokens must be separated
by a space
# or a tab character.
# Maximum length of each user_name is 39 character.
#
# Legal values for user_host:
# Any string that starts and ends with an alphanumeric character, and
# contains only alphanumeric characters, dot(.), dash(-), or underscore(_)
# in the middle.
# Maximum length is 39 character.
#
# Legal values for user_role: package_admin.
#user_name
#user_host
#user_role
# Physical Volumes
#
# "pv" is used to specify physical volumes used by this package.
# This should only be used by HP specified applications
# Do not modify this section otherwise.
#
# Specify the name of each physical volume.
# The name must be an absolute path starting with "/".
#
# For example, if this package uses the physical volumes /dev/sda1 and /dev/
sdb1, enter:
# pv /dev/sda1
# pv /dev/sdb1
#
# Legal values for pv: /^\/[0-9A-Za-z][0-9A-Za-z_.\/\-]*[0-9A-Za-z]$/.
#pv
```

Toolkit Module Script (tkit_module.sh):

This script is used by the Serviceguard master control script to start and stop packages that are created using the Toolbox. You can modify the monitor function, if required.

```
###########################
# Source utility functions.
###########################

. /etc/cmcluster.conf
if [[ -z $SG_UTILS ]]; then
             SG_UTILS=$SGCONF/scripts/mscripts/utils.sh
fi

if [[ -f ${SG_UTILS} ]]; then
        . ${SG_UTILS} if (( $? != 0 ))
        then
        echo "ERROR: Unable to source package utility functions file: $
{SG_UTILS}"
        exit 1
fi
else
echo "ERROR: Unable to find package utility functions file: ${SG_UTILS}"
exit 1
fi

################################################################### #
#
Get the environment for this package through utility function
# sg_source_pkg_env().
#
##################################################################
# export PATH=$PATH:$SGSBIN
sg_source_pkg_env $*
#
#################################################################
#
# VALIDATE FUNCTION
#
#################################################################

function validate_function
{
            sg_log 0 "App Toolkit validation"

            for i in `set | grep "^APP_TKIT_"`
            do
                name=`echo $i | awk -F"=" '{print $1}'`

value=`echo $i | awk -F"=" '{for(i=2;i<=NF;i++)printf "%s ",$i;print ""}'`

case $name in
            APP_TKIT_MAINTENANCE_FLAG)
                 validate_yes_no $name $value
                 ;;
            *)
                validate_not_empty $name $value
                ;;
                esac
    done
}

##################################################################
#
#
VALIDATE YES NO FUNCTION
#
#################################################################

function validate_yes_no
{
        if [[ $# < 2 ]]; then
                    sg_log 0 "Usage : check_yes_or_no VAR VALUE";
                    sg_log 0 "ERROR: $1 is not defined";
                    check_return 1 5
fi

if [ -z "$2" ]; then
            arg="invalid"
else
            arg=`echo $2 | tr "[:upper:]" "[:lower:]" | tr -d "[:space:]"`

fi

case         $arg in
yes)
              return 0;
             ;;
no)
              return 0;
             ;;
*)
            sg_log 0 "Invalid option $1=$2.\n$1 can be set either as $1=\"yes\"
or
$1=\"no\"."
            check_return 1 5
            ;;
    esac
}

##################################################################
#
# VALIDATE NOT EMPTY FUNCTION
#
#################################################################

function validate_not_empty
{
    if [[ $# < 2 ]]; then
            sg_log 0
            "$2 value"
            sg_log 0 "Usage : notempty VAR VALUE"
            sg_log 0 "ERROR: $1 is not defined"
            check_return 1 5
    else
            if [ -z $2 ]; then
                        sg_log 0 "ERROR: $1 is not defined"
                        check_return 1 5
            fi
    fi
}

###############################################
# The function which takes care of returning the exit status to SG.
###############################################

function check_return
{
            if (( $1 != 0 ))
            then
            case $2 in
                1) sg_log 0 "ERROR: Function app_start_server: Failed to start"
                    stop_app_server
                    exit 1
                        ;;
                2) sg_log 0 "ERROR: Function app_stop_server: Failed to stop"
                    sg_log 0 "WARNING: Make sure to kill all lingering
processes of this package"
                    exit_code=0
                    ;;
                3) sg_log 0 "ERROR: Could not Start the Monitor Function"
                    exit 1
                    ;;
                4) sg_log 0 "ERROR: The toolkit configuration file (haapp.conf)
does not exist"
                    exit 1
                    ;;
                5) sg_log 0 "ERROR: Validation Failed. Improper parameters in
Toolkit Configuration file"
                    exit 1
                    ;;
                *) sg_log 0 "ERROR: Failed, unknown error"
                    ;;
            esac
            fi
}

##############################################################
#
# START FUNCTION
#
#############################################################

function start_function
{

            . $APP_TKIT_APP_START
            exit_code=$?
}

##############################################################
#
# STOP FUNCTION
#
#############################################################

function stop_function
{

            . $APP_TKIT_APP_STOP
            exit_code=$?
}

##############################################################
#
# MONITOR FUNCTION
#
```

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

function monitor_function
{
local maintenance=0;

echo "$(date '+%b %e %T') - Node \"$(hostname)\": Starting the monitoring
process"

while :; do

# Maintenance code. This script will check whether the debug file
# exists and only then Package gets into maintenance mode. Monitoring
# is stopped completely at this moment.

if [ -z "$MAINTENANCE_FILE" ] || [ ! -f "$MAINTENANCE_FILE" ]; then
        if [ $maintenance -eq 1 ]; then
              sg_log 0 "Starting monitoring again after maintenance"
              maintenance=0
        fi
# Invoke monitor function
            if the maintenance file does not exist if [ ! -f
"$MAINTENANCE_FILE" ]
            then
                    case $APP_TKIT_MONITORING_TYPE in

                    pid_file | PID_FILE)
                                 pid_file $APP_TKIT_MONITORING_ELEMENT
                                 ;;
                    process_monitor | PROCESS_MONITOR)
                              process_monitor $APP_TKIT_MONITORING_ELEMENT
                              ;;
                    port_monitor | PORT_MONITOR)
                          port_monitor $APP_TKIT_MONITOR_ELEMENT
                          ;;
              *)
                         echo "Invalid APP_TKIT_MONITORING_TYPE :
$APP_TKIT_MONITORING_TYPE option specified !"
                             exit 1
                             ;;
                         esac
                    fi
                         exit_status=$?
                    if [[ $exit_status != 0 ]]
                        then
                                sg_log 0 "Exiting Toolkit Monitoring because
one of the key process has failed."
                                echo "Exiting Toolkit Monitoring because one of
the key process has failed."
                                #exit 1
                                        return 1
                    fi

# TOOLKIT IN MAINTENANCE MODE
else
        if [ $maintenance -eq 0 ]; then
                sg_log 0 "Toolkit pausing monitoring and entering maintenance
```

```
mode"
                maintenance=1
        fi
fi
sleep ${APP_TKIT_MONITOR_INTERVAL}
done

}

##############################################################
#
# PID FILE
#
#############################################################

function pid_file
{
PID_LIST=$*

if [ -z "$APP_APP_SERVER_PROC" ]; then
            sg_log 0 "ERROR: APP_APP_SERVER_PROC is not defined.";
            sg_log 0 "ERROR: Validation Failed. Improper parameters in Toolkit
Configuration file"
            return 1;
fi

for i in ${PID_LIST[@]}
do
            id=`cat $i 2> /dev/null`
            exit_code=$?
            if [[ $exit_code != 0 ]]; then
                    sg_log 0 "ERROR: $i file not found. Halting the monitor
script."
                    return 1;
            fi
if [[ $ENVIRONMENT = "HP-UX" ]]; then
        p_name=`ps -p $id | awk '{ print $4}' | grep "^$APP_APP_SERVER_PROC$"`
        if [ -z "$p_name" ]; then
                    sg_log 0 "ERROR: The process $id is not running. Halting
the monitor script."
                    return 1;
        else
                    continue;
        fi
else
        grep $APP_APP_SERVER_PROC /proc/$id/stat >/dev/null
        if [ $? -ne 0 ]; then
            sg_log 0 "ERROR: The process $id is not running. Halting the
monitor script."
            return 1;
        else
        continue;
fi

fi
done
return 0;
```

```
}

#############################################################
#
# MONITORING_PROCESS
#
#############################################################

function process_monitor
{

for k in ${APP_TKIT_MONITORING_ELEMENT[@]}
    do
        result=`UNIX95=yes ps -eo args | grep -i $k | grep -v grep`
        if [ -z "$result" ] ; then
            $ECHO "ERROR: The process $k is not running. Halting the monitor
script."
            return 1;
        else
            continue;
        fi
    done
    return 0;
}

############################################################
#
# MONITORING_PORT
#
############################################################

function port_monitor
{

for k in ${APP_TKIT_MONITORING_ELEMENT[@]}
        do
            if [[ $ENVIRONMENT = "HP-UX" ]]; then
            running=`netstat -an | grep "\.${k}[ ]*\*\.\*[ ]*LISTEN$"`
            if [ -z "$running" ] ; then
                    echo "ERROR: The Application is not running. Port NOT
Listening. Halting the monitor script."
                    return 1;
                else
                    continue;
                fi
            else
                running=`netstat -lnp | grep "\:${k}[ ]*\:[0-9\.]*\:[0-9\.]*\:
\*[ ]*LISTEN"`
                if [ -z "$running" ] ; then
                        echo "ERROR: The Application is not running. Port NOT
Listening. Halting the monitor script."
                        return 1;
                else
                        continue;
                fi
            fi
    done
    return 0;
}

##################################
#
#Start of Routines
#
#################################

typeset -i exit_code=0

export APP_TKIT_DIR=$APP_TKIT_CONF_DIR
MAINTENANCE_FILE="$APP_TKIT_DIR/toolbox.debug"

ENVIRONMENT=`uname`

case $1 in
        validate)
                    validate_function
                    ;;
        start)
                    start_function
                    ;;
        stop)
                    stop_function
                    ;;
        toolbox_monitor)
                  monitor_function
                  ;;
        *)
                    sg_log 0 "Usage: ${0} [ start | stop | toolbox_monitor |
validate ]"
                    ;;
esac
exit $exit_code
```

Application Start Script, (start_app.sh):

This script must be written by the user to start the application that is being packaged using the Toolbox. The sample script given here is used to start MySQL server.

```
#!/bin/sh
#
###############################################
# This script starts up the application
#
CONFIGURATION_FILE_PATH="/mysql/my.cnf"
PID_FILE="/var/run/mysqld/mysqld. pid"
STARTUP_RETRIES=10 hostname==`
hostname`
###############################################

echo "$(date '+%b %e %T') - Node \"$(hostname)\": Starting Application
daemons." options=" --
pid-file=$PID_FILE "

if [ ! -z $CONFIGURATION_FILE_PATH ] && [ -f $CONFIGURATION_FILE_PATH ];
```

```
        then options="--defaults-file=$CONFIGURATION_FILE_PATH $options"
elif [ ! -z $ DATA_DIRECTORY ] && [ -d $ DATA_DIRECTORY ];
    then
    options="--defaults-file=$ DATA_DI RECTORY/my.cnf $options"
else
    echo "$(date '+%b %e %T') - Node \"$(hostname)\": ERROR :
Application Start up failed: Application configuration file not specified."
    exit 1
fi

safe_mysqld $options &
ret_value=$?
if [[ $ret_value != 0 ]];
    then
    echo "ERROR: Application Start up failed " exit 1
fi
# Check if the pid file is ready.
counter=0
while :;
do
    if [ -f $PID_FILE ]; then
        break
    else
        if [ $counter -ge $STARTUP_RETRIES ]; then
            echo "$(date '+%b %e %T') - ERROR: Application failed to create PID
file $PID_FILE" exit 1
        else
            sleep 1
            let counter="$counter+1"
        fi
    fi
done
```

Application Stop Script (stop_app.sh):

This script must be written by the user to stop the application that is being packaged using the Toolbox. The following application stop script is written to stop MySQL server.

```
#!/bin/sh
#
###############################################
# This script halts the Application.
# CONFIGURATION_FILE_PATH="/mysql/my.cnf"
PID_FILE="/var/run/mysqld/mysqld. pid"
STARTUP_RETRIES=10
hostname=` hostname`
###############################################

echo "$(date '+%b %e %T') - Node \"$(hostname)\": Stopping Application daemons"
if [ -f $PID_FILE ];
then
        read pid < $PID_FILE
else
        echo "$(date '+%b %e %T') - ERROR: The $PID_FILE does not exist."
        return 1
fi
if [ -n $pid ];
```

```
then
    kill -s SIGTERM $pid
    return_value=$?
    if [[ $return_value != 0 ]];
        then
            echo "ERROR: stop_app failed "
            return 1
    fi
else
    echo "$(date '+%b %e %T') - ERROR: stop_app failed: The $PID_FILE was
corrupted."
    return 1
fi
```