# Distributed R User's Guide

## Distributed R

Software Version: 1.2.x

Document Release Date: 10/30/2015

# Contents

# New Features in Distributed R Release 1.2.x

This topic briefly describes new features introduced in Distributed R 1.2.x.

- Distributed R and HPE Vertica can now be installed on the same node or cluster. See Co-Location with HPE Vertica for more information.
- Multiple users can now run concurrent sessions of Distributed R on the same cluster. See Multiple Users with Concurrent Sessions for more information.
- The foreach() function has been updated with new deterministic functionality. See Deterministic Behavior of a foreach Function for more information.
- The HPdata package has been updated to support parallel loading of CSV and ORC files from HDFS or your local filesystem into distributed data structures. See the new functions csv2dframe and orc2dframe for more information.
  - Additionally, the object2hdfs function has been added, allowing you to write objects from Distributed R to HDFS.
- A new algorithm function, hpdegbm, has been added. It provides a distributed ensemble implementation of the R gbm package for regression and classification problems.
- A distributed sampling function that uses input data to formulate random output data has been added. See hpdsample for more information.
- A generic function for calculating permutation-based variable importance, varImportance, has been added.
- A new distributed, parallel implementation of the standard R rpart package has been introduced. See hpdrpart for more information.
- Using the new function db2dframes, you can load a data set from a table in HPE Vertica to a pair of Distributed R dframes which correspond to responses and predictors of a predictive model.

# About Distributed R

Distributed R is a high-performance scalable platform for the R language. It enables R to leverage multiple cores and multiple servers to perform Big Data Advanced Analytics. It consists of new R language constructs to easily parallelize algorithms across multiple R processes.

Distributed R simplifies large-scale analysis by extending R. Because R is a single-threaded environment, it has limited utility for Big Data analytics. Distributed R allows you to specify that parts of programs be run in multiple single-threaded R-processes. This approach results in significantly reduced execution times for Big Data analysis.

Distributed R follows R language design principles and extends existing R constructs to support distributed computing so that R developers can immediately start using Distributed R with no additional training.

You should use Distributed R when you have performance and scalability issues with R. Distributed R allows you to overcome the scalability and performance limitations of R to analyze very large data sets. Distributed R provides the ability to run an analysis on the complete data set for when you want to use all the data, and not just the sample.

Many applications need to perform advanced analytics such as machine learning, graph processing, and statistical analysis on large amounts of data. While R has many advanced analytics packages, the single-threaded nature of R limits use of these packages on Big Data. Distributed R extends R in two ways:

- **Distributed data** — Distributed R stores data across servers and manages distributed computation. If you are a data scientist, you can run your programs on very large data sets (such as terabytes) by simply adding more servers.
- **Parallel execution** — If you are a programmer, you can use Distributed R to implement code that runs in parallel. You can leverage a single multi-core machine or a cluster of machines to obtain dramatic improvement in application performance.

Distributed R provides distributed data-structures to store in-memory data across multiple machines including:

- Distributed arrays (`darray`) — Use distributed arrays whenever data contains only numeric values.
- Distributed data frames (`dframe`) — Use distributed data frames for data that contains both numeric and non-numeric data or only contains non-numeric data.
- Distributed lists (`dlist`) — Use distributed lists to store complex objects, such as R models.

You can partition these data structures by rows, columns, or blocks and also specify the size of the initial partitions.

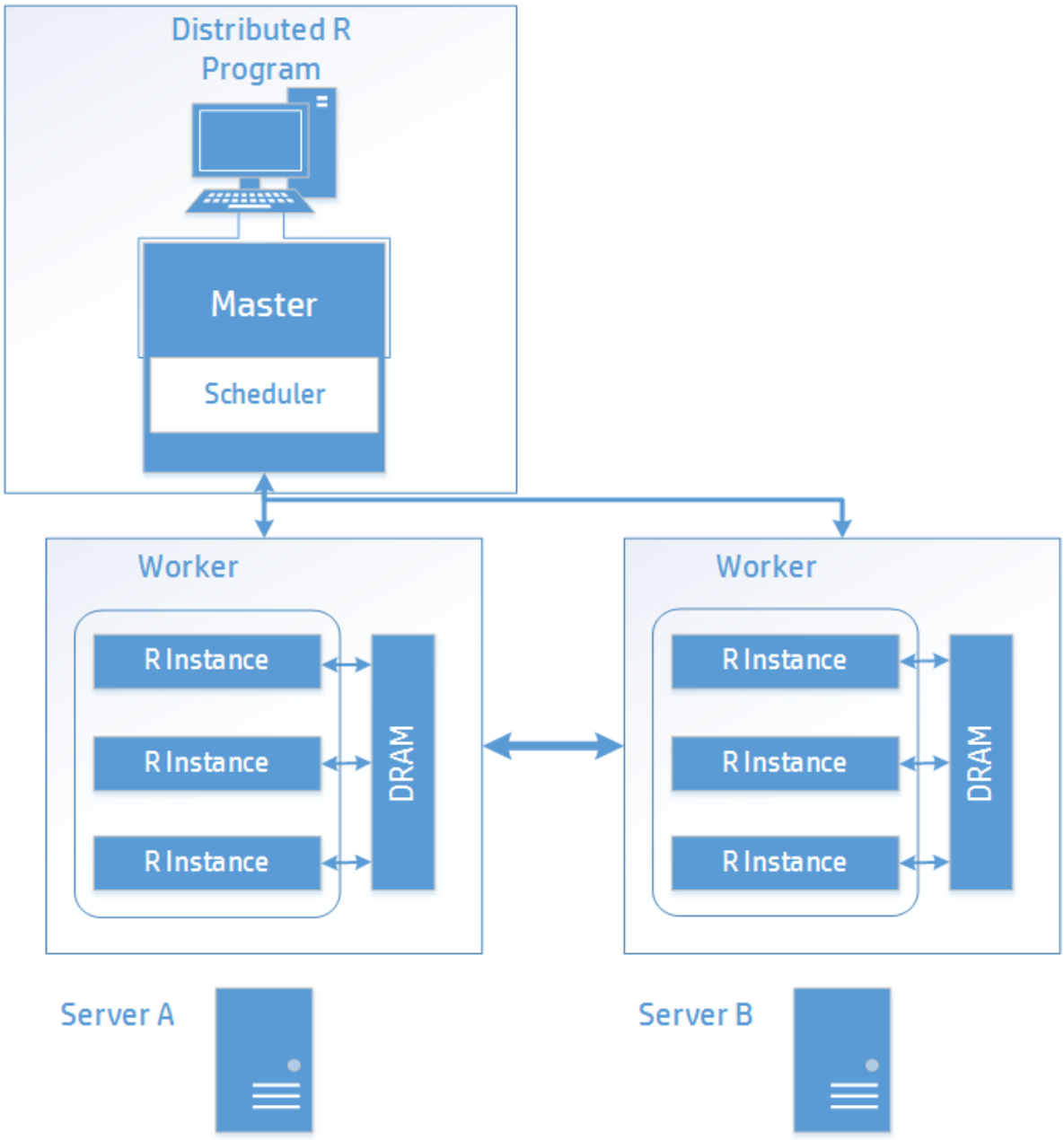
As a programmer, you can express parallel processing in Distributed R using `foreach` loops. Such loops execute a function in parallel on multiple machines. You pass parts of the distributed data to these functions. The Distributed R runtime intelligently schedules functions on remote machines to reduce data movement.

In addition to the language constructs already mentioned, Distributed R has other helper functions. For example, `distributedR_start` starts the Distributed R runtime on a cluster. You can find information about all the functions provided by the Distributed R library R by typing `?distributedR` at the R prompt. You can also access this information using the `help()` command in the R console.

Distributed R also comes packaged with several popular algorithms that have been extended to work with the new Distributed R data types and can scale and perform on large datasets. See Distributed R Package Overview.

# Distributed R Architecture

Before explaining the Distributed R programming model, it is important to understand the system architecture. Distributed R comprises a single master process and multiple workers. Logically, each worker resides on one server. The master controls each worker and can be co-located with a worker or be on a separate server. Each worker manages multiple local R instances. The figure below shows an example cluster setup with two servers. The master process runs on Server A while the workers run on servers A and B. Each worker has three R instances. You can set up the workers to use more or fewer R instances.




For programmers, the master is the R instance configure to be the master node and on which Distributed R is loaded using the `library(distributedR)` command. The master starts the program and is in charge of overall execution. Parts of the program correspond to parallel sections, such as `foreach`. These parallel processes are executed by the workers. Distributed data structures, such as `darray` and `dframe`, contain data that is stored across workers. The Distributed R API provides commands to move data between workers as well as between master and workers. You do not need to know on which worker the data resides, as the runtime hides the complexity of data movement.

# Distributed R Package Overview

Distributed R provides several algorithm packages that have been created specifically to work in the distributed computing platform of Distributed R. Distributed R packages are loaded the same as typical R package libraries using the `library()` command. Distributed R packages can be loaded with other R packages. Currently, Distributed R provided these R library packages:

- About the HPdata Package - The Distributed Data Package. It provides functions for loading data into distributed data objects from databases and files.
- About the HPdclassifier Package - The Distributed Algorithms for Classifiers Package. It contains the `hpdandomForest()` function, which is a distributed version of randomForest, and an associated predict function, `predictHPdRF()`. It also contains the `hpdrpart()`, `hpdegbm()`, and `varImportance()` functions. Note that while randomForest is popular for classification, it can also be used for regression and unsupervised learning.
- About the HPdcluster Package - The Distributed Clustering for Big Data Package. It contains the `hpdkmeans()` function, which is a distributed kmeans algorithm. It also contains `hpdapply()`, a distributed version of the apply function.
- About the HPdgraph Package - The Distributed Algorithms for Graph Analytics Package. It contains the `hpdpagerank()` function, which is a distributed pagrerank algorithm. It also contains the `hpdwhich.max()` function, which acts as a which.max function on darrays.
- About the HPdregression Package - The Distributed Regression for Big Data Package. It contains the hpdglm() function, which is a distributed glm function that performs regression analysis on distributed data types. It also contains `v.hpdglm()` and `cv.hpdglm()` which are functions used to evaluate models using split-sample and cross-validation respectively.

More details about these packages can be found in the package help files from within R. For example `help(HPdata)` or `?hpdata` after using the `library(HPdata)` command. Examples for each of the functions are available in this guide and you can access them by clicking the package name above.

# Supported Platforms

Distributed R Release 1.2.x is supported on the following software platforms:

- Red Hat Enterprise Linux 6.x and 7.0 (x86-64)
- 64-bit CentOS 6.x and 7.0 (x86-64)
- Ubuntu 12.04 LTS and 14.04 LTS (64-bit)

# HPE Vertica Version

Distributed R 1.2.x is compatible with HPE Vertica 7.2.x.

# R Language Pack for HPE Vertica Version

Distributed R 1.2.x is compatible with the 1.2.x R Language Pack for HPE Vertica.

# R Version

Distributed R 1.2.x is tested and supported on R version 3.2.2. However, Distributed R 1.2.x can run on R version 3.1.0 or greater. Distributed R 1.2.x can not run on R versions prior to 3.1.0.

# ODBC Version

unixODBC version 2.3.0 or greater is required on all nodes if you want to use RODBC, vRODBC, or the HPE Vertica Connector.

# Hardware Sizing

You can run Distributed R on commodity hardware. In total, you need enough total memory to hold all of the data you want to analyze, along with a buffer for R bookkeeping. The Distributed R Algorithm Packages section contains details on all of the HPE Vertica distributed algorithm packages as well as memory estimation formulas for each package. In general, if you have hardware with more memory, you will need fewer total nodes to do the analysis. Distributed R recommends a stand- alone cluster of HP DL380s.

You need enough servers so that your largest data set fits into the total memory of your servers, along with a buffer for bookkeeping by R. If you need your answer faster, you need to use more CPU cores.

Details on memory requirements for algorithms in the Distributed R packages are provided in this guide in each package's respective section.

# Network and Firewall Considerations

Distributed R performs best when all nodes are on the same subnet and have the same broadcast address for one or more interfaces. A cluster that has nodes on more than one subnet can experience lower performance due to the network latency associated with a multi-subnet system at high network utilization levels.

Distributed R requires SSH communications between the nodes. Distributed R also requires a configurable port range be open between the nodes. By default, Distributed R uses ports from 50000 to 50100. You can change the port range in the cluster configuration file. See Cluster Configuration for details.

You must open ports on your firewall for SSH(port 22) and the port range configured for master/worker communication (50000 - 50100 by default).

# Co-location with HPE Vertica Server

Distributed R 1.2.x is supported for co-location with HPE Vertica server. For more information, see Co-Location with HPE Vertica.

# Multiple Users with Concurrent Sessions

Distributed R supports multiple users running concurrent sessions of Distributed R on the same cluster.

In some cases, you may have users who want to run more than a single concurrent session. To do so, these users must perform the same procedure for running Distributed R that a single user must perform. For more information, see Single Node and Multi Node Operation and Programming Model.

The performance of Distributed R while concurrent sessions are running depends on many factors, such as:

- Total memory of the cluster
- Number of cores
- Type of algorithm each user is running

Be aware that concurrent sessions do not coordinate resources. For this reason, the performance of all Distributed R sessions degrades as the number of concurrent users increases.

# Installing Distributed R

A video tutorial for installing Distributed R is available on the HPE Vertica website.

The Distributed R installer installs Distributed R and configures the XML configuration files on every node in your cluster.

Before you install, see Configuring Passwordless SSH and sudo for the Installation User. This step is necessary so that the node on which you are running the installer can access the other nodes over SSH without using a password, and the user installing Distributed R has passwordless sudo privileges on all nodes in the cluster.

Next, see SELinux Configuration to verify that SELinux is disabled or set to permissive mode.

Once you have completed these pre-installation steps on each node in the cluster, you can run the Distributed R installer on your platform.

# Configuring Passwordless SSH and sudo for the Installation User

The Distributed R installer calls sudo to install Distributed R packages on each node and uses SSH to communicate between the master and worker nodes. Passwordless SSH and sudo must be enabled for the installation user during the Distributed R install. After installation is complete you can disable passwordless sudo for the installation user.

**Important:** The installation username must exist **on all nodes** in the cluster and you must give the installation user passwordless ssh and sudo access **on all nodes** in the Distributed R cluster.

**Note:** You must enable passwordless SSH and sudo even if you are doing a single-node localhost install.

## To Configure Passwordless sudo:

1. You must enable passwordless sudo on each node. Edit `/etc/sudoers` and add the following line at the end of the file (replace *username* with the appropriate username):

   ```
   username ALL=(ALL) NOPASSWD:ALL
   ```

   **Note:** You can remove the entry in `sudoers` after installation completes.

2. Verify that the installation user can run sudo without a password. For example, as the installation user, run: `sudo touch example.txt`. You should not be prompted for a password.

## To Configure Passwordless SSH:

If you are using Ubuntu, then your install may not include OpenSSH Server. Install OpenSSH server on Ubuntu with the command: `sudo apt-get install openssh-server`.

1. Create public and private SSH keys that do not have a pass phrase by using the `ssh-keygen` utility.

   - If your current `~/.ssh/id_rsa` keys do not have a pass phrase and you want to use them for passwordless SSH on all Distributed R nodes, then skip this step. Otherwise create new SSH keys for the installation user (the same username for which you granted passwordless sudo in the previous section).

- Do not create the keys as root or using sudo. You can optionally provide a file name for the key files or accept the default *id_rsa* file name.
- Do not enter text when prompted for a pass phrase. Simply press ENTER twice. For example:

```
# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
```

2. Once you have created the SSH keys, use `ssh-copy-id` to copy the public key to each node in the Distributed R cluster, including localhost. If you created the SSH keys with the default file name then simply specify the server to which you want to copy the key. Provide your password each time you copy an ID. For example:

```
ssh-copy-id localhost
user@localhosts's password:
ssh-copy-id node2.example.com
user@node2.example.com's password:
ssh-copy-id node3.example.com
user@node3.example.com's password:
```

# SELinux Configuration

Distributed R does not support SELinux except when SELinux is running in permissive mode. Before installing Distributed R, you must either disable SELinux or change it to use permissive mode.

# Red Hat and SUSE Systems

To disable SELinux:

1. Edit `/etc/selinux/config` and change setting for SELinux to disabled (`SELINUX=disabled`). This disables SELinux at boot time.
2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/etc/selinux/config` and change setting for SELINUX to permissive (`SELINUX=Permissive`).
2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

## Ubuntu and Debian Systems

To disable SELinux:

1. Edit `/selinux/config` and change setting for SELinux to disabled (`SELINUX=disabled`). This disables SELinux at boot time.
2. As root/sudo, type `setenforce 0` to disable SELinux immediately.

To change SELinux to use permissive mode:

1. Edit `/selinux/config` and change setting for SELinux to permissive (`SELINUX=Permissive`).
2. As root/sudo, type `setenforce Permissive` to switch to permissive mode immediately.

# RProfile.site

You can customize the R environment at start-up by editing the `Rprofile.site` file. The default directory for the file is *<R_HOME>*`/etc`. However, editing the file and adding any customization messages causes a compilation error during the installation of Distributed R. Hewlett Packard Enterprise recommends that you do not edit the `Rprofile.site` file prior to installation.

If you make any changes to the`Rprofile.site` file, perform either of the following actions before installing Distributed R:

- Delete the `Rprofile.site` file.
- Remove any custom messages added to the file.

# Installing or Upgrading Distributed R on Red Hat/CentOS

The Distributed R installer is packaged as a gzipped tar file. To install Distributed R on Red Hat/CentOS you must use the installer package that is specific to Red Hat/CentOS systems. The package is named `distributedR-[VERSION].RHEL6.tar.gz`, where [*VERSION*] is the version number of Distributed R. For example: `hp-distributedR-1.2.0-0.RHEL6.tar.gz`.

**Note:** The steps for Installing or Upgrading are the same. Upgrades overwrite the previous version of Distributed R.

1. Verify that you have given your installation user passwordless ssh and sudo privileges on all nodes in the Distributed R cluster. See Configuring Passwordless SSH and sudo for the Installation User.
2. Verify that SELinux is in permissive or disabled mode. See SELinux Configuration.
3. Download the installation package to the master node and unzip the package with the command:

   ```
   tar zxvf distributedR-[VERSION].RHEL6.tar.gz
   ```

4. Change directory into the newly extracted directory:

   ```
   $ cd distributedR-[VERSION]
   ```

5. Run the dependency installer located in the *helper* directory to install all required dependencies. The dependency installer takes as arguments the hostname or IP Address of each node in the Distributed R cluster. This automatically installs the dependencies on each specified node. For example:

   ```
   $ ./helpers/distributedR_install_dependencies NodeA NodeB NodeC
   ```

   The installer attempts to detect if you are behind a proxy and use the appropriate proxy settings. However, If you are behind a proxy you can add the arguments `--http-proxy` and `--https-proxy`. Run `./helpers/distributedR_install_dependencies --help` for details on usage.

6. If any R package dependencies are missing then you are prompted to install them. Answer **Y** (or press enter) to install all of the required R packages.
7. If the dependency installer is successful, then skip this step and go to the next numbered step

and run the installer. If the dependency installer fails, then complete this step.

If the dependency installer fails, then you must manually install dependencies using the following sub-steps. You must do this **on each node** in the cluster if the dependency installer fails.

a. Run the following commands to install OS dependencies. If you are behind a proxy then you must configure rpm and yum to use a proxy server.

```
$ sudo rpm -Uvh http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-
8.noarch.rpm
$ sudo yum install libcgroup-devel gcc R libxml2 libxml2-devel unixODBC unixODBC-
devel libselinux-python \
python-devel python-pip
```

b. Next, you must install python dependencies. If you are behind a proxy run the following commands to allow PIP to use a proxy server, replace *proxy:port* with your proxy information:

```
$ export http_proxy=http://proxy:port
$ export https_proxy=https://proxy:port
```

c. Use PIP to install required Python libraries.

```
$ sudo -E pip install jinja2
$ sudo -E pip install PyYaml
$ sudo -E pip install httplib2
$ sudo -E pip install paramiko
$ sudo -E pip install setuptools
```

d. Start R as sudo to install required R libraries:

```
$ sudo R
```

Run the following in R:

```
> install.packages(c("RInside", "Rcpp", "XML", "RUnit", "data.table",
"randomForest"))
```

Select a CRAN mirror from the list and allow R to install the required libraries.

Exit R:

```
> quit()
```

8. Run the installer:

   a. The Distributed R installer checks dependencies and installs and configures Distributed R on all nodes in your cluster. It first copies the required files to a `/distributedR` directory in the directory where you installed R. The installer then configures the XML configuration files in `/distributedR/conf`.

      **Note:** The installer makes the following changes to each node:

      - Changes the size of `/dev/shm` to 100% of free memory (obtained by running free().
      - Sets `shmall` and `shmmax` to 90% of total memory.
      - Sets `shmmi` to 500000.
      - Sets max open files (`/proc/sys/fs/file-max`) to 524288.

      Run the installer by specifying any options and the node(s) on which to install Distributed R:

      ```
      ./distributedR_install [OPTIONS] [NODES]
      ```

      Options:

      - `-h` or `--help`: Print help and Exit
      - `-c`or `--check-cluster-connectivity`: Check connectivity between the nodes and exit.
      - `-y` or `--yes-to-all`: Answer yes to all prompts (useful for non-interactive installs).

      Nodes: The host name or IP address of each node in the cluster. Specify all nodes regardless of which is the master. You define the master node in the next step.

      Example:

      ```
      ./distributedR_install NodeA NodeB NodeC
      ```

   b. Follow the installer's guided installation process to configure your system.

      The installer prompts you with these questions:

      i. **Which node is the master?** - Specify which node is the master node by specifying the number of the node to act as master.

      ii. **Which node(s) is/are the workers?** - Specify which nodes are workers by selecting the appropriate number or specifying a comma-separated list of workers.

iii. **Do you want to specify custom port ranges?** - You can specify the port ranges over which the master and worker nodes pass data. By default the range is ports 50000 to 50100. If you answer yes then you are prompted for the port ranges to use.

iv. **Do you want to install Vertica ODBC support?** - If yes, then the libraries required for HPE Vertica ODBC connections are installed to `/opt/hp/odbc` and the VRODBC R package is installed. However, you must still manually configure ODBC properties.

v. **Do you want to run a test to make sure the Distributed R installation is OK?** - If yes, some basic tests are run to verify the Distributed R install.

c. You must exit the terminal session in which you installed Distributed R and log back in so system changes that the installer made can take effect. Distributed R changes limits for the amount of open file handles and shared memory. These changes do not take effect in the current session or any currently open sessions. Start Distributed R in a new session.

d. The installation process automatically creates the cluster configuration file for your cluster and places it in `/distributedR/conf/cluster_conf.xml` in the directory where you installed R on the master node. See Cluster Configuration for details on modifying the configuration.

If you are using a Red Hat system, Distributed R installs in the directory where you installed R. By default, this directory is `/usr/lib/R/library`.

If you are using a CentOS system, Distributed R installs in the directory where you installed R. By default, this directory is `/usr/lib64/R/library`.

# Installing or Upgrading Distributed R on Debian/Ubuntu

The Distributed R installer is packaged as a gzipped tar file. To install Distributed R on an Ubuntu system you must use the installer package that is specific to Debian/Ubuntu systems. The package is named `distributedR-[VERSION].DEBIAN7.tar.gz`, where [*VERSION*] is the version number of Distributed R. For example: `hp-distributedR-1.2.0-0.DEBIAN7.tar.gz`.

**Note:** The steps for Installing or Upgrading are the same. Upgrades overwrite the previous version of Distributed R.

1. Verify that you have given your installation user passwordless ssh and sudo privileges on all nodes in the Distributed R cluster. See Configuring Passwordless SSH and sudo for the Installation User.
2. Verify that SELinux is in permissive or disabled mode. See SELinux Configuration.
3. Download the installation package to the master node and unzip the package with the command:

   ```
   tar zxvf distributedR-[VERSION].DEBIAN7.tar.gz
   ```

4. Change directory into the newly extracted directory:

   ```
   $ cd distributedR-[VERSION]
   ```

5. Run the dependency installer located in the *helper* directory to install all required dependencies. The dependency installer takes as arguments the hostname or IP Address of each node in the Distributed R cluster. This automatically installs the dependencies on each specified node. For example:

   ```
   $ ./helpers/distributedR_install_dependencies NodeA NodeB NodeC
   ```

   The installer attempts to detect if you are behind a proxy and use the appropriate proxy settings. However, If you are behind a proxy you can add the arguments `--http-proxy` and `--https-proxy`. Run `./helpers/distributedR_install_dependencies --help` for details on usage.

6. If any R package dependencies are missing then you are prompted to install them. Answer **Y** (or press enter) to install all of the required R packages.
7. If the dependency installer is successful then skip this step and go to the next numbered step

and run the installer. If the dependency installer fails then complete this step.

If the dependency installer fails, then you must manually install dependencies using the following sub-steps. You must do this **on each node** in the cluster if the dependency installer fails.

a. Run the following commands to install OS dependencies. If you are behind a proxy then you must configure apt-get to use a proxy server.

```
$ sudo apt-get install libcgroup-devel cgroup-bin r-base r-base-dev unixODBC-dev
libxml2 unixODBC liblzma-dev zlib1g-dev libxml2-dev
```

b. Next, you must install python dependencies. If you are behind a proxy run the following commands to allow PIP to use a proxy server, replace *proxy:port* with your proxy information:

```
$ export http_proxy=http://proxy:port
$ export https_proxy=https://proxy:port
```

c. Use PIP to install required Python libraries.

```
$ sudo -E pip install jinja2
$ sudo -E pip install PyYaml
$ sudo -E pip install httplib2
$ sudo -E pip install paramiko
$ sudo -E pip install setuptools
```

d. Start R as sudo to install required R libraries:

```
$ sudo R
```

Run the following in R:

```
> install.packages(c("RInside", "Rcpp", "XML", "RUnit", "data.table",
"randomForest"))
```

Select a CRAN mirror from the list and allow R to install the required libraries.

Exit R:

```
> quit()
```

8. Run the installer:

a. The Distributed R installer checks dependencies and installs and configures Distributed R on all nodes in your cluster. It first copies the required files to a `/distributedR` directory in the directory where you installed R. The installer then configures the XML configuration files in `/distributedR/conf`.

   **Note:** The installer makes the following changes to each node:

   - Changes the size of `/dev/shm` to 100% of free memory (obtained by running free().
   - Sets `shmall` and `shmmax` to 90% of total memory.
   - Sets `shmmi` to 500000.
   - Sets max open files (`/proc/sys/fs/file-max`) to 524288.

   Run the installer by specifying any options and the node(s) on which to install Distributed R:

   ```
   ./distributedR_install [OPTIONS] [NODES]
   ```

   Options:

   - `-h` or `--help`: Print help and Exit
   - `-c`or `--check-cluster-connectivity`: Check connectivity between the nodes and exit.
   - `-y` or `--yes-to-all`: Answer yes to all prompts (useful for non-interactive installs).

   Nodes: The host name or IP address of each node in the cluster. Specify all nodes regardless of which is the master. You define the master node in the next step.

   Example:

   ```
   ./distributedR_install NodeA NodeB NodeC
   ```

b. Follow the installer's guided installation process to configure your system.

   The installer prompts you with these questions:

   i. **Which node is the master?** - Specify which node is the master node by specifying the number of the node to act as master.

   ii. **Which node(s) is/are the workers?** - Specify which nodes are workers by selecting the appropriate number or specifying a comma-separated list of workers.

iii. **Do you want to specify custom port ranges?** - You can specify the port ranges over which the master and worker nodes pass data. By default the range is ports 50000 to 50100. If you answer yes then you are prompted for the port ranges to use.

iv. **Do you want to install Vertica ODBC support?** - If yes, then the libraries required for HPE Vertica ODBC connections are installed to `/opt/hp/odbc` and the VRODBC R package is installed. However, you must still manually configure ODBC properties.

v. **Do you want to run a test to make sure the Distributed R installation is OK?** - If yes, some basic tests are run to verify the Distributed R install.

c. You must exit the terminal session in which you installed Distributed R and log back in so system changes that the installer made can take effect. Distributed R changes limits for the amount of open file handles and shared memory. These changes do not take effect in the current session or any currently open sessions. Start Distributed R in a new session.

d. The installation process automatically creates the cluster configuration file for your cluster and places it in `/distributedR/conf/cluster_conf.xml` in the directory where you installed R on the master node. See Cluster Configuration for details on modifying the configuration.

Distributed Rinstalls in the directory where you installed R. By default, this directory is `/usr/local/lib/R/site-library`.

# Cluster Configuration

The default cluster configuration file is named `cluster_conf.xml` and defines the master and worker nodes in the cluster, the amount of executors (R sessions) for each worker, and the amount of memory to use for Distributed R.

The installation process creates the master cluster configuration file and populates it with the master/worker selections you made when you installed Distributed R. You do not need to change the configuration after installation unless you want to manually change settings or manually add or remove nodes from your cluster.

Each node has a `cluster_conf.xml` file in the directory where you installed Distributed R. However, only the cluster_conf.xml file on the master node defines the settings for the cluster.

When you run distributedR_start() from the master node, Distributed R automatically loads the cluster configuration file from the directory where you installed Distributed R. You can optionally specify a different configuration file with the `cluster_conf` argument, for example: `distributedR_start(cluster_conf='/home/dbadmin/cluster2.xml')`. By default the distributedR_start() command looks in the directory where you installed Distributed R for the configuration file. If the configuration file exists in a different directory then you must specify the full path to the file.

# Example Cluster Configuration File

```
<?xml version="1.0" ?>
<MasterConfig>
    <ServerInfo>
        <Hostname>node01</Hostname>
        <StartPortRange>50000</StartPortRange>
        <EndPortRange>50100</EndPortRange>
    </ServerInfo>
    <Workers>
        <Worker>
            <Hostname>node01</Hostname>
            <StartPortRange>50000</StartPortRange>
            <EndPortRange>50100</EndPortRange>
            <Executors>0</Executors>
            <SharedMemory>0</SharedMemory>
        </Worker>
        <Worker>
            <Hostname>node02</Hostname>
            <StartPortRange>50000</StartPortRange>
            <EndPortRange>50100</EndPortRange>
            <Executors>0</Executors>
            <SharedMemory>0</SharedMemory>
        </Worker>
        <Worker>
            <Hostname>node03</Hostname>
            <StartPortRange>50000</StartPortRange>
            <EndPortRange>50100</EndPortRange>
            <Executors>0</Executors>
            <SharedMemory>0</SharedMemory>
        </Worker>
```

```
    </Workers>
</MasterConfig>
```

# Configuration Options

- The <ServerInfo> tag specifies the local (master) node configuration.
- <Workers> tag contains the individual workers configuration and each <Worker> tag defines details on the configuration for each Worker node.
- <Hostname> specifies the hostname (or IP Address) of the respective server. Enter the master's hostname under the <ServerInfo> tag and the worker's node hostname under the <Workers> tag.
- **Port Range** is defined in <StartPortRange> and <EndPortRange> tags. These tags define the range of port numbers allowed for use by a particular node (Master or Worker). At run time, a random port number within the port range defined is opened. If the chosen port is not available, an exception is thrown. Master node port range should have a minimum of two available ports, while each Worker node port range should have a minimum of 2*(total number of worker nodes)+1 available ports at all times of Distributed R task execution.
- <Executors> defines the number of R sessions to run on the worker. <SharedMemory> defines the amount of memory to use on the worker in MB for Distributed R (total for the server, not per session). If <Executors> and <SharedMemory> options in the worker nodes is zero, then Executors is set to the number of logical CPUs on the server and SharedMemory is set to the size of the mounted shared memory region on the server.

For addition details type help(distributedR_start) at the R prompt or see the Distributed R Package Manual.

# Adding or Removing Nodes in a Distributed R Cluster

You can add to your Distributed R cluster by simply re-running the installation script and specifying additional nodes (in addition to the nodes in the original cluster). You can also remove one or more nodes from a cluster by running the uninstall utility and then re-running the install on the remaining nodes to update the cluster configuration file for the master node.

# Adding a Node to a Distributed R Cluster:

1. For any nodes you are adding to the cluster, first configure passwordless SSH from the installation node to the new node(s) and configure passwordless sudo on each new node. See Configuring Passwordless SSH and sudo for the Installation User
2. From the master node, navigate to the directory containing the Distributed R installer. Use the installation steps for your specific platform. See Installing or Upgrading Distributed R on Red

Hat/CentOS or Installing or Upgrading Distributed R on Debian/Ubuntu. When specifying nodes for the dependency installer and installer, specify all of the existing nodes as well as the new nodes. The software on the existing nodes is not changed, and the new nodes have Distributed R installed. The cluster configuration file is updated with the new nodes.

# Removing a Node From a Distributed RCluster:

1. From the master node, navigate to the directory containing the Distributed R installer. Run `distributedR_uninstall` and list the nodes to uninstall. For example, if you have a four node cluster containing Nodes A - D, and you want to remove NodeD, then run the command:

   ```
   ./distributedR_uninstall NodeD
   ```

2. Run the installer and provide the addresses for the remaining nodes in the cluster to reconfigure the cluster configuration file. For example:

   ```
   ./distributedR_install NodeA NodeB NodeC
   ```

   No software is installed, but the cluster configuration file is updated to reflect the new makeup of the cluster.

# Co-Location with HPE Vertica

You can install both Distributed R and HPE Vertica on the same node or cluster. Referred to as *co-location*, this approach allows you to move data and models between your Distributed R session and HPE Vertica database. Using co-location does not require additional hardware.

For successful co-location, you must allow both software applications adequate resources, such as memory, from the cluster. To do so, you must understand the co-location requirements and know how to correctly set the cluster configuration file.

# Related Topics

- Co-Location Requirements
- Configure Co-Location in the Cluster Configuration File

# Co-Location Requirements

If you have already installed HPE Vertica on a single or multi-node cluster, complete these requirements to co-locate Distributed R and HPE Vertica..

## *Create the distributedr Resource Pool*

Distributed R requires its own resource pool in HPE Vertica.

Create a resource pool named distributedr in a session of HPE Vertica running on the cluster you plan to co-locate with Distributed R. The following example shows what settings to use:

```
CREATE RESOURCE POOL distributedr MEMORYSIZE '50%' MAXCONCURRENCY 0 CPUAFFINITYSET '50%'
CPUAFFINITYMODE EXCLUSIVE;
```

The size of the resource pool in HPE Vertica depends on the computational needs of both Distributed R and HPE Vertica. You can estimate the needed size of the resource pool by testing the provided algorithm examples in Distributed R, and by running example queries in HPE Vertica.

For information on resource pools in HPE Vertica, see Managing Workloads in the HPE Vertica documentation.

## *Install Distributed R*

If you have not yet done so, install Distributed R. See Installing Distributed R.

## *Run the Co-Location Install Script*

1. On the node where you installed Distributed R, navigate to the directory that was created when you unzipped the Distributed R `.tar.gz` file.
2. Run the enable_Colocation installer to install the required co-location settings.

   The installer takes as arguments the hostname or IP Address of each node in the Distributed R cluster. Running the installer automatically places the co-location settings on each specified node. For example:

```
./distributedR_enable_Colocation NodeA NodeB NodeC
```

## *Install and Configure vRODBC*

To enable co-location, you must install and configure vRODBC for the connection to HPE Vertica.

# Configure Co-Location in the Cluster Configuration File

As described in Cluster Configuration, the default cluster configuration file for Distributed R is named `cluster_conf.xml`.

1. Run the co-location installer to add two additional tags to the `cluster_conf.xml` file:
   - `isColocatedWithVertica` — Default value is `FALSE`.
   - `VerticaDSN` — Default value is `NULL`.

2. Update the values of these two tags in your `cluster_conf.xml` file:
   - In the `isColocatedWithVertica` tag, replace `FALSE` with `TRUE`.
   - In the `VerticaDSN` tag, replace `NULL` with the DSN name you provided in the `odbc.ini` file when configuring vRODBC.

After you successfully complete the configuration changes, the cluster configuration file should look similar to the one in this example. The `isColocatedWithVertica` tag is set to `TRUE`, and the `VerticaDSN` tag set to a DSN is defined in the `odbc.ini` as `MyDSN`.

```
<?xml version="1.0" ?>
<MasterConfig>
    <ServerInfo>
       <Hostname>node01</Hostname>
       <StartPortRange>50000</StartPortRange>
       <EndPortRange>50100</EndPortRange>
       <isColocatedWithVertica>TRUE</isColocatedWithVertica>
       <VerticaDSN>MyDSN</VerticaDSN>
    </ServerInfo>
    <Workers>
       <Worker>
           <Hostname>node01</Hostname>
           <StartPortRange>50000</StartPortRange>
           <EndPortRange>50100</EndPortRange>
           <Executors>0</Executors>
           <SharedMemory>0</SharedMemory>
        </Worker>
        <Worker>
           <Hostname>node02</Hostname>
           <StartPortRange>50000</StartPortRange>
           <EndPortRange>50100</EndPortRange>
           <Executors>0</Executors>
           <SharedMemory>0</SharedMemory>
        </Worker>
        <Worker>
           <Hostname>node03</Hostname>
           <StartPortRange>50000</StartPortRange>
           <EndPortRange>50100</EndPortRange>
           <Executors>0</Executors>
           <SharedMemory>0</SharedMemory>
        </Worker>
    </Workers>
</MasterConfig>
```

# Uninstalling Distributed R

You can remove all the nodes from your Distributed R cluster and uninstall the Distributed R software by using the distributedR_uninstall script located in the Distributed R package you downloaded to install Distributed R. If you only want to remove one more nodes (but not all nodes) from a Distributed R cluster see Adding or Removing Nodes in a Distributed R Cluster.

## To Uninstall Distributed R:

From the master node, navigate to the directory containing the Distributed R installer. Run `distributedR_uninstall` and list all the nodes in your cluster. For example, if your cluster contained Nodes A - D, you run the command:

```
./distributedR_uninstall NodeA NodeB NodeC NodeD
```

Distributed R is uninstalled and the directory where you installed Distributed R is removed from each node in the cluster.

# Running the Additional Helper Scripts

When you install Distributed R, you unzip the `distributedR-[VERSION].RHEL6.tar.gz` file. The unzipping process creates a `distributedR-1.2.0` directory on your system. This directory contains the `helpers` directory. Within `helpers`, you find two scripts that can help you diagnose any conflicting R software versions on your system:

- `R_uninstall`
- `distributedR_list_versions`

# The R_uninstall Script

During the installation of Distributed R, the `distributedR_install_dependencies` script installs core R from a binary distribution. The `R_uninstall` script uninstalls the following:

- The binary distribution of core R installed by `distributedR_install_dependencies`
- Existing version of Distributed R
- All R packages located in `/usr/lib/R/library`

To run the script, navigate to the `distributedR-1.2.0` directory.

The script takes as arguments the host name or IP Address of each node in the Distributed R cluster. For example:

```
$ ./helpers/R_uninstall NodeA NodeB NodeC
```

The output returned resembles the following:

```
distributedR_install: INFO     Detecting OS versions ...
distributedR_install: INFO     CentOS detected. Doing CentOS install ...
distributedR_install: INFO     Running Ansible playbook: /home/dbadmin/distributedR-
1.2.0/installer/lib/../ansible/bin/ansible-playbook  -i /tmp/distributedR_nodes_2010-01-
01_01-02-03 -l dr  /home/dbadmin/distributedR-1.2.0/helpers/../installer/lib/r_
```

```
uninstall.yml  ... this may take a while!
distributedR_install: INFO     ###
distributedR_install: INFO     ### R uninstall succeeded!
distributedR_install: INFO     ###
```

For details on usage, run `./helpers/R_uninstall --help` from the `distributedR-1.2.0` directory.

# The distributedR_list_versions Script

Distributed R provides a helper script to determine the R and Distributed R software versions installed on your system, along with many packages.

To run the `distributedR_list_versions` script, navigate to the `distributedR-1.2.0` directory. The script outputs the following for each node in your cluster:

- Core R version
- Latest version of Distributed R
- R package versions

The script takes as arguments the host name or IP Address of each node in the Distributed R cluster. For example:

```
$ ./helpers/distributedR_list_versions NodeA NodeB NodeC
```

The output returned resembles the following:

```
Hostname | R_version | DistR_Version | R_package_versions
---------+-----------+---------------+--------------------
NodeA    | 3.2.2     | 1.2.0         | Rcpp: 0.12.0
         |           |               | RInside: 0.2.13
         |           |               | XML: 3.98-1.1
         |           |               | data.table: 1.9.4
         |           |               | randomForest: 4.6-10
         |           |               | RUnit: 0.4.28
---------+-----------+---------------+--------------------
NodeB    | 3.2.2     | 1.2.0         | Rcpp: 0.12.0
         |           |               | RInside: 0.2.13
         |           |               | XML: 3.98-1.1
         |           |               | data.table: 1.9.4
         |           |               | randomForest: 4.6-10
         |           |               | RUnit: 0.4.28
---------+-----------+---------------+--------------------
NodeC    | 3.2.2     | 1.2.0         | Rcpp: 0.12.0
         |           |               | RInside: 0.2.13
         |           |               | XML: 3.98-1.1
         |           |               | data.table: 1.9.4
         |           |               | randomForest: 4.6-10
         |           |               | RUnit: 0.4.28
---------+-----------+---------------+--------------------
```

For details on usage, run `./helpers/distributedR_list_versions --help` from the `distributedR-1.2.0` directory.

# Using Distributed R With RStudio

You can use RStudio and RStudio server with Distributed R. You can install RStudio or RStudio server on the master node and use Distributed R as you normally would.

You can get more details about RStudio and RStudio Server from the RStudio web site at: http://www.rstudio.com/.

**Important:** RStudio and RStudio Server are third-party products. While Distributed R has done initial testing for Distributed R with RStudio and RStudio Server, they are not officially supported for use with Distributed R.

# Configuring ODBCINI and VERTICAINI Environment Variables for RStudio

When using vRODBC to move data from an HPE Vertica database to a Distributed R cluster, you must configure the ODBCINI and VERTICAINI environment variables. The ODBCINI variable is configured in a file named odbc.ini. The VERTICAINI variable is configured in a file named vertica.ini.

If these variables are not defined or are defined incorrectly, RStudio returns an error message that states the following: "Data source name not found."

## *Determine Whether ODBCINI and VERTICAINI Are Configured*

In order to determine whether the ODBCINI and VERTICAINI environment variables are correctly configured in RStudio, enter the following commands in the RStudio terminal:

```
> Sys.getenv("ODBCINI")
> Sys.getenv("VERTICAINI")
```

If no paths to the `odbc.ini` or `vertica.ini` files are provided, then the environment variables are not correctly configured.

To determine whether VERTICAINI and ODBCINI are configured at the system level, use a text editor of your choice to view the `.bashrc` file. The `.bashrc` file is located in your home user directory. To view the file, enter the following:

```
$ vim ~/.bashrc
```

If the following lines are not in the `.bashrc` file, you must add them:

```
export ODBCINI=<Absolute filepath of odbc.ini file>
```

```
export VERTICAINI=<Absolute filepath of vertica.ini file>
```

## Configure the ODBCINI and VERTICAINI Environment Variables in .bashrc

To set the environment variables so that they are recognized by RStudio, you must first add them to the `.bashrc` file.

To do so, enter the following commands:

```
$ echo 'export ODBCINI=<Absolute filepath of odbc.ini file>' >> ~/.bashrc
$ echo 'export VERTICAINI=<Absolute filepath of vertica.ini file>' >> ~/.bashrc
```

Rather than use the `echo` command, you have the option of opening the `.bashrc` file and adding the ODBCINI and VERTICAINI environment variable information within `.bashrc` itself. To do so, enter the following lines into the `.bashrc` file, then save the file:

```
ODBCINI=<Absolute filepath of odbc.ini file>
VERTICAINI=<Absolute filepath of vertica.ini file>
```

## Configure the ODBCINI and VERTICAINI Environment Variables in RStudio

After setting the odbc.ini and vertica.ini file paths in `~/.bashrc`, add the file paths for the files in RStudio. To do so, complete either one of the following steps:

- Manually assign the environment variables in RStudio to the correct paths of the odbc.ini and vertica.ini files. In RStudio, enter the following commands:

```
> Sys.setenv(ODBCINI="<Absolute filepath of odbc.ini file>")
> Sys.setenv(VERTICAINI="<Absolute filepath of vertica.ini file>")
```

- Edit the Renviron file on your system. It is located in the `/usr/lib64/R/etc/Renviron` directory. Add the following two lines to the Renviron file, then save the file:

```
ODBCINI=${ODBCINI-'<Absolute filepath of the odbc.ini file>'}
VERTICAINI=${VERTICAINI-'<Absolute filepath of the vertica.ini file>'}
```

The ODBCINI and VERTICAINI environment variables are ported to the RStudio session.

# Single Node and Multi Node Operation

Distributed R can be run in Single Node mode (localhost mode) and Multi Node mode. By default, when you use the installer to install 2 or more nodes then Distributed R runs in Multi Node mode. In Multi Node mode you leverage the processors and memory of all nodes in the cluster. However, Single Node mode is still an effective way to use Distributed R as it leverages all the CPUs on the single server and allows for parallel R processing. If you have installed Distributed R on only one node then the only mode available is single node.

## Running Distributed R in Single Node Mode in a Distributed R Cluster

If you have installed Distributed R on multiple servers, then you can still run in single node mode. When in single node mode, the server acts as both the master and a worker. To do this:

1. On any node in the cluster, start an R session.
2. If the node is a worker node in the cluster, then you can start it in Single Node mode simply by calling:

```
> library(distributedR)
> distributedR_start()
Workers registered - 1/1.
All 1 workers are registered.
Master address:port - 127.0.0.1:50000
```

3. If the node is the master node, then you can specify the localhost configuration file that is created on every node during the installation process to start the master in single node mode. This starts Distributed R only on the local node. For example:

```
> distributedR_start(cluster_conf='cluster_conf_localhost.xml')
Workers registered - 1/1.
All 1 workers are registered.
Master address:port - 127.0.0.1:50000
```

# Programming Model

### Distributed R Data Structures

Distributed R provides you with new language extensions and a distributed runtime for R. Distributed R contains the following three groups of commands. For more details about each command, type `help(command)` from within R or type `help(distributedR)` to view an overview of all the commands.

| Command | Description |
|---|---|
| **Session Management Commands** | |
| `distributedR_start()` | Start a new Distributed R session. |
| `distributedR_status()` | Obtain master and worker information. |
| `distributedR_master_info()` | Obtain details about the master node. |
| `distributedR_shutdown()` | End session. |
| **Distributed Data Structure Commands** | |
| `darray()` | Create distributed array. |
| `dframe()` | Create distributed data frame. |
| `dlist()` | Create distributed list. |
| `getpartition()` | Fetch darray, dframe, or dlist object. |
| `npartitions()` | Obtain total number of partitions. |
| `partitionsize()` | Get the size of a partition. |
| `is.darray()` | Checks if the input is a darray. |
| `as.darray()` | Create darray object from matrix object. |
| `as.dframe()` | Create dframe object from matrix object or data.frame. |
| `factor.dframe()` | Converts categorical columns of a dframe into factors. |
| `as.factor.dframe()` | Converts categorical columns of a dframe into factors and creates a new dframe, leaving the original dframe unchanged. |
| `unfactor.dframe()` | Converts categorical columns of a dframe into their label of type character. |

| | |
|---|---|
| `levels.dframe()` | Finds the categorical columns of a dframe and returns a level array and an associated column array identifying the column for the levels. |
| `clone()` | Clone or deep copy of a darray. |
| **Parallel Execution Commands** | |
| `foreach()` | Execute function on cluster. |
| `splits()` | Pass partition to foreach loop. |
| `update()` | Make partition changes inside foreach loop globally visible. |

**Saving R Sessions** - Distributed R shuts down when you quit the R session, even if you choose to save your workspace. If you shut down Distributed R, or quit R and then save your session, you may encounter an error. Such errors occur when any of your R variables store a reference to a distributed object.

# Distributed R Data Structures

Distributed R provides three new data structures for use within R: distributed arrays, distributed data frames, and distributed lists.

**Distributed arrays** `darray` provide a shared, in-memory view of multi-dimensional data stored across multiple servers. Distributed arrays are Partitioned and Shared.

Distributed arrays can be partitioned into contiguous ranges of rows, columns, or blocks. You specify the size of the initial partitions. Distributed R workers store partitions of the distributed array in the compressed sparse column format unless the array is defined as dense.

- Use partitions to specify coarse-grained parallelism by writing functions that execute in parallel and operate on partitions. For example, partitions of a distributed array can be loaded in parallel from data stores such as HPE Vertica or from files.
- Use `getpartition` to fetch a distributed array and materialize it at the master node. For example, `getpartition(A)` reconstructs the whole array A at the master by fetching the partitions from local and remote workers.The $i^{th}$partition can be fetched by `getpartition (A,i)`.

Distributed arrays can be read-shared by multiple concurrent tasks. You simply pass the array partitions as arguments to many concurrent tasks. However, Distributed R supports only a single writer per partition.

A **distributed data frame** (`dframe`) is similar to a `darray`. The primary difference is that, unlike `darray`,distributed data frames can store non-numeric data. Even though you can use a `dframe` to store numeric only data, it is much more efficient to use `darray` in such cases. The efficiency difference is because of the underlying representation of these data structures. Numeric data takes less memory than character data. Over large data sets, there is a performance difference.

A **distributed list** (`dlist`) stores elements inside lists that are partitioned across servers. To create a distributed list, you only need to specify the number of partitions. For example,`dlist(5)` creates a distributed list with five partitions. Initially, each partition is an R list with no elements.

# Parallel Programming

If you are a programmer, you can use `foreach` loops to execute functions in parallel. You can pass data, including partitions of `darray` and `dframe`, to the functions. Array and data frame partitions can be referred to by the splits function. The splits function automatically fetches remote partitions and combines them to form a local array. For example, if `splits(A)`is an argument to a function executing on a worker, then the whole array A would be re-constructed by the runtime, from local and remote partitions, and passed to that worker. The $i^{th}$partition can be referenced by `splits (A,i).`

Functions inside `foreach` do not return data. Instead, call `update` inside the function to make distributed array or data frame changes globally visible. The Distributed R runtime starts tasks on worker nodes for parallel execution of the loop body. By default, there is a barrier at the end of the loop to ensure all tasks finish before statements after the loop are executed.

**Note:** Distributed R is bound by the same limit as R for value sizes (precision). For example, Distributed R values for the type double is limited to 2^53-1, for example 9007199254740991. Effectively, R is limited to 53 bits, and "represents to that precision a range of absolute values from about `2e-308 to 2e+308`". Enter `help(double)` in the R console for additional details.

# Programming Examples

These examples illustrate the Distributed R programming model.

## Getting Started

Follow the steps in Installing Distributed R to first install Distributed R. Load the Distributed R library, and then start the cluster by calling `distributedR_start()`.

```
> library(distributedR)
Loading required package: Rcpp
Loading required package: RInside
Loading required package: XML

> distributedR_start()
Workers registered - 3/3.

All 3 workers are registered.
Master address:port - docb01:50000
```

You can view the status of the cluster with the `distributedR_status()` command. This command returns details such as:

- Number of workers in the cluster
- Number of R instances managed by each worker
- System memory available on each worker node

```
> distributedR_status()
       Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 docb01:50001    8  11911    3214       10137          0
2 docb02:50000    8  11911    1920       10137          0
3 docb03:50000    8  11911    1924       10137          0
```

When you are done using Distributed R, you can call distributedR_shutdown() to disconnect the master from the workers and free resources.

```
> distributedR_shutdown()
Shutdown complete
```

# Creating a Distributed Array

Next, create a distributed array using `darray()`. Create a 9 × 9 dense array by specifying its size and how it is partitioned. The following example shows how to partition the array into 3 × 3 blocks and set all its elements to the value 10. Therefore, nine partitions could reside on remote nodes.

```
library(distributedR)
distributedR_start()
Workers registered - 3/3.

All 3 workers are registered.
Master address:port - docb01:50000

A <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
```

You can print the number of partitions using `npartitions()` and, by calling `getpartition()`, fetch the whole array at the master.

**Note:** For a very large array, such as one with billions of rows, Distributed R does not recommend fetching the whole array at the master. This approach does not allow you to manage huge datasets efficiently by distributing data across multiple workers.

```
npartitions(A)
[1] 9

getpartition(A)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   10   10   10   10   10   10   10   10   10
 [2,]   10   10   10   10   10   10   10   10   10
 [3,]   10   10   10   10   10   10   10   10   10
```

```
 [4,]   10   10   10   10   10   10   10   10   10
 [5,]   10   10   10   10   10   10   10   10   10
 [6,]   10   10   10   10   10   10   10   10   10
 [7,]   10   10   10   10   10   10   10   10   10
 [8,]   10   10   10   10   10   10   10   10   10
 [9,]   10   10   10   10   10   10   10   10   10
```

Typically, you partition arrays by rows or columns (for example, 1-D partitioning) instead of blocks (2-D partitioning). Because row and column partitioning is a special case of block partitioning, the above example details block partitioning. If you partition the array`A` by rows by using `blocks=c(3,9)` instead of `blocks=c(3,3)`, then each partition contains 3 rows and all the columns. For example:

```
A <- darray(dim=c(9,9), blocks=c(3,9), sparse=FALSE, data=10)
npartitions(A)
[1] 3

getpartition(A, 1)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]   10   10   10   10   10   10   10   10   10
[2,]   10   10   10   10   10   10   10   10   10
[3,]   10   10   10   10   10   10   10   10   10

getpartition(A)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   10   10   10   10   10   10   10   10   10
 [2,]   10   10   10   10   10   10   10   10   10
 [3,]   10   10   10   10   10   10   10   10   10
 [4,]   10   10   10   10   10   10   10   10   10
 [5,]   10   10   10   10   10   10   10   10   10
 [6,]   10   10   10   10   10   10   10   10   10
 [7,]   10   10   10   10   10   10   10   10   10
 [8,]   10   10   10   10   10   10   10   10   10
 [9,]   10   10   10   10   10   10   10   10   10
```

Distributed arrays can initially be declared empty. For example, it is typical to create an array and then load data into the array from a data store. The initial declaration creates a full array which is soon overwritten. By declaring an array empty, you can save memory space.

```
Aempty <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, empty=TRUE)
npartitions(Aempty)
[1] 9

getpartition(Aempty,1)
<0 x 0 matrix>
```

# Parallel Programming With foreach()

The `foreach()` loop is a flexible and powerful construct to manipulate distributed data structures. This example illustrates its use by initializing a distributed array with different values.

Create another distributed array, `B`, with the same size (9 × 9) as `A` and the same partitioning scheme. The previous example used the argument `data` to initialize all elements of `A` to 10.

However, you cannot use data to set different values to array elements. Instead, start a `foreach()` loop, pass partitions of B, and inside the loop assign values to the partition.

```
> B <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE)
> foreach(i, 1:npartitions(B),
    init <- function(b = splits(B,i), index=i) {
    b <- matrix(index, nrow=nrow(b), ncol=ncol(b))
    update(b)
})
progress: 100%
```

The syntax of `foreach()` is `foreach(iteration variable, range, function)`. In the preceding example, *i* is the iteration variable that takes values from 1 to 9. Therefore, this syntax creates nine parallel tasks that execute the functions. In the function, you pass the $i^{th}$ partition of B using `splits (B,i)` and the value of *i*. Within the function you assign a matrix to the partition. The matrix is of the same size as the partition (3 × 3) but initialized by the value of the iteration variable. Thus the $i^{th}$partition has all elements equal to i. Next, you can fetch the whole array using `getpartition()`.

```
> getpartition(B)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]    1    1    1    2    2    2    3    3    3
 [2,]    1    1    1    2    2    2    3    3    3
 [3,]    1    1    1    2    2    2    3    3    3
 [4,]    4    4    4    5    5    5    6    6    6
 [5,]    4    4    4    5    5    5    6    6    6
 [6,]    4    4    4    5    5    5    6    6    6
 [7,]    7    7    7    8    8    8    9    9    9
 [8,]    7    7    7    8    8    8    9    9    9
 [9,]    7    7    7    8    8    8    9    9    9
```

By specifying the partition index, you can fetch a particular partition, for example, the fifth partition:

```
> getpartition(B,5)
     [,1] [,2] [,3]
[1,]    5    5    5
[2,]    5    5    5
[3,]    5    5    5
```

When you use `foreach`, be aware of this function's limitations. First, only variables passed as arguments to the function (`init` in this case) are available for use within the function. For example, the array `A` or its partitions cannot be used within the function. Even the iterator variable (i) must be passed as an argument. Second, loop functions don't return any value. The only way you can make data modifications visible is to call `update()` on the partition. In addition, you can only use update on the distributed data structures `darray`, `dframe`, `dlist` arguments. For example, `update(index)` is incorrect code as `index` is not a distributed object.

# Parallel Array Addition

With the two initialized distributed arrays, you can start computations such as adding their elements. You again use a `foreach` loop to perform the parallel addition. First, create an output

array `C` of the same size and partitioning scheme. In the `foreach` loop pass the $i^{th}$ partition of all three arrays, `A`, `B`, and `C`. Within the loop you can add the corresponding partitions, put the output in c, and call update:

```
> C <- darray(dim=c(9,9), blocks=c(3,3))
> foreach(i, 1:npartitions(A),
    add <- function(a = splits(A,i), b = splits(B,i), c = splits(C,i)) {
    c <- a + b
    update(c)
})

progress: 100%

> getpartition(C)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   11   11   11   12   12   12   13   13   13
 [2,]   11   11   11   12   12   12   13   13   13
 [3,]   11   11   11   12   12   12   13   13   13
 [4,]   14   14   14   15   15   15   16   16   16
 [5,]   14   14   14   15   15   15   16   16   16
 [6,]   14   14   14   15   15   15   16   16   16
 [7,]   17   17   17   18   18   18   19   19   19
 [8,]   17   17   17   18   18   18   19   19   19
 [9,]   17   17   17   18   18   18   19   19   19
```

While you can use `foreach` to perform any parallel operation, the Distributed R package provides basic operators with factory presets that work on distributed arrays.These operators include array addition, subtraction,multiplication, and summary statistics such as max, min, mean, and sum (including their column and row versions such as `colSums`). Internally, all these operators are implemented using `foreach`. The following example illustrates the use of some of these operators:

```
> D <- A+B
progress: 100%

> getpartition(D)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
 [1,]   11   11   11   12   12   12   13   13   13
 [2,]   11   11   11   12   12   12   13   13   13
 [3,]   11   11   11   12   12   12   13   13   13
 [4,]   14   14   14   15   15   15   16   16   16
 [5,]   14   14   14   15   15   15   16   16   16
 [6,]   14   14   14   15   15   15   16   16   16
 [7,]   17   17   17   18   18   18   19   19   19
 [8,]   17   17   17   18   18   18   19   19   19
 [9,]   17   17   17   18   18   18   19   19   19

> mean(D)
[1] 15

> colSums(D)
[1] 126 126 126 135 135 135 144 144 144
```

# Creating a Distributed Data Frame

The syntax for distributed data frames, `dframe()`, is similar to distributed arrays. However, the difference is that data frames can store non-numeric values. Create a 9 × 9 data frame. To do so, specify the data frame size and how it is partitioned:

```
> dF <- dframe(dim=c(9,9), blocks=c(3,3))

> getpartition(dF)
 X1 X2 X3 X1 X2 X3 X1 X2 X3
1  0  0  0  0  0  0  0  0  0
2  0  0  0  0  0  0  0  0  0
3  0  0  0  0  0  0  0  0  0
4  0  0  0  0  0  0  0  0  0
5  0  0  0  0  0  0  0  0  0
6  0  0  0  0  0  0  0  0  0
7  0  0  0  0  0  0  0  0  0
8  0  0  0  0  0  0  0  0  0
9  0  0  0  0  0  0  0  0  0
```

The data frame dF has 9 partitions each of size 3 × 3.

You can also create an empty dframe by specifying `npartition` and not dimensions or blocks:

```
> dF <- dframe(npartitions=9)

> getpartition(dF)
data frame with 0 columns and 0 rows
```

To add data, use a `foreach()` loop:

```
> foreach(i, 1:npartitions(dF),
init <- function(df = splits(dF,i), index=i, n=3){
    p  <- matrix(index, nrow=n,ncol=n-1)
    q  <- rep("distR",n)
    df <- data.frame(p,q)
    update(df)
})
progress: 100%
```

Each partition now has a column which contains the string `distR`:

```
> getpartition(dF, 1)
  X1 X2  q
1  1  1 distR
2  1  1 distR
3  1  1 distR
```

# Creating a Distributed List

Create a distributed list by specifying the number of partitions. Initially, the list is empty.

```
> dL <- dlist(npartitions=3)

> getpartition(dL)
list()
```

Populate the list using the `foreach` loop.

```
> foreach(i, 1:npartitions(dL),
init <- function(dl=splits(dL,i), idx=i){
    dl <- list(c("distR", idx))
    update(dl)
})

progress: 100%
```

You can obtain individual partitions or the whole list:

```
> getpartition(dL,1)
[[1]]
[1] "distR" "1"

> getpartition(dL)
[[1]]
 [1] "distR" "1"

 [[2]]
 [1] "distR" "2"

 [[3]]
 [1] "distR" "3"
```

# Load and Save Data from Files

You can save or load data in parallel. This example shows how to save data to files. Use the `foreach` loop to write each partition of an array to a file:

```
> fname <- paste(getwd(),"/Data",sep="")
> foreach(i, 1:npartitions(D),
    saves <- function(d = splits(D,i), index=i, name=fname){
    write(d, paste(name,index,sep=""))
})

progress: 100%

> fname
[1] "/home/dbadmin/Data"
```

The preceding code writes each partition in a different file and the file name is appended with a number. If Distributed R is running on a single machine, all the files reside on the same machine. You can try to load one of the partitions and check its contents:

> `scan(paste(fname,4,sep=""))`

```
[1] 15 15 15 15 15 15 15 15 15
```

If partition 4 does not exist on the local worker, then an error is returned. You can look in the current working directory to find the file named Data*N* where N is the partition number and use that number in the scan/paste function.

A better method is to use a `foreach` loop to load the data in parallel into a distributed array. First, declare a new distributed array`E`, of the same size as `D`, and then load data from previously saved files. Because scan returns the values as a single vector, convert the data into a matrix for the correct size before calling `update`:

```
> E <- darray(dim=c(9,9), blocks=c(3,3))

> foreach(i, 1:npartitions(E),
    loads <- function(e = splits(E,i), index=i, name=fname) {
    fn <- paste(name,index,sep="")
    e <- matrix(scan(file=fn), nrow=nrow(e))
    update(e)
})

> getpartition(E,5)
[,1] [,2] [,3]
[1,] 15 15 15
[2,] 15 15 15
[3,] 15 15 15
```

# Loading from HPE Vertica using foreach

To manually load data from a database into a distributed array, you can use an ODBC connector such as HPE Vertica RODBC (vRODBC) or the open source RODBC connector. However, Distributed R provides the HPData package specifically for loading data from databases. The database functions are generally easier than using foreach loops. The example presented here is to illustrate the concepts of how distributed loading works.

The following example shows how to load data using a `foreach` loop that makes concurrent ODBC connections to the HPE Vertica database. Declare a 50 x 4 array in which each partition contains 5 rows. Within the loop, load each partition by querying the database for 5 rows at a time. Before you can run this example, you must install`vRODBC` and set it up to connect to the HPE Vertica database. Follow installation instructions for `vRODBC`. To use `RODBC`, just replace the occurrences of `vRODBC` with `RODBC` in the following example.

```
> X <- darray(dim=c(50, 4), blocks=c(5, 4), sparse=FALSE)
> foreach(i, 1:npartitions(X), initArrays <- function(x = splits(X,i), index=i)
> library(vRODBC)
> connect<-odbcConnect("Test")
> size <- nrow(x)
> start <- (index-1) * size
> end <- index * size
> qry <- paste("select A,B,C,D from T where id >=", start,"and id <", end,
    "order by id")
    segment<-sqlQuery(connect, qry) {
    odbcClose(connect)
    x<-cbind(segment$A, segment$B, segment$C, segment$D)
```

```
    update(x)
    })

progress: 100%
```

In this example, you must first load the ODBC package inside the loop (using library vRODBC). Because the function inside the `foreach` loop executes on the worker and packages, you must explicitly load it into the worker environment. Second, in this particular example, you use HPE Vertica internal row identifiers to select rows. For example, the first five rows in the Vertica table *T* is assigned to the first partition of array *X*. HPE Vertica has row identifiers to refer to individual rows. To fetch the first partition and display data, use `getpartition`:

```
> getpartition(X, 1)
     [,1]    [,2]    [,3]     [,4]
[1,]  5   0.903815 0.522466 0.250464
[2,]  1   0.994233 0.138644 0.139464
[3,]  3   0.117651 0.285975 0.309341
[4,]  4   0.280725 0.006694 0.684827
[5,]  6   0.331704 0.835160 0.498040
```

# Parallel Execution Using Existing R Packages

Sometimes, you can solve a problem by applying functions from existing R packages in parallel. Consider the example of finding the shortest distance from five source vertices to all other vertices in the graph. Because distance calculation from each source vertex is independent from the others, you can start five tasks to calculate them in parallel. R already has a package called `igraph` that can calculate shortest distances. The following example details how to use `igraph` in parallel to solve the preceding problem. For this example, you must install `igraph` on all machines in the Distributed R cluster.You can manually download the software from CRAN or use `install.packages(igraph)`.

First, create a sparse distributed array to store the graph. Because you don't want to partition the graph, the array has only one partition equal to the total size of the graph.

```
> G <- darray(dim=c(100,100), blocks=c(100,100), sparse=TRUE)
```

Next, use a `foreach` loop to generate a random graph and store it in the array. You must load the `igraph` library inside the loop function.

```
> foreach(i, 1:1, initGraph<-function(g=splits(G)) {
    library(igraph)
    rg<-erdos.renyi.game(nrow(g),0.1)
    g<-get.adjacency(rg, sparse=TRUE)
    update(g)
})

progress: 100%
```

Now, run parallel tasks to calculate shortest distances and store them in another array called paths. Partition the array paths such that each partition has one row and 100 columns. Therefore, each element in the array corresponds to the distance of the source vertex to a given destination vertex.

```
> paths <- darray(dim=c(5,100), blocks=c(1,100), sparse=FALSE)

> foreach(i, 1:npartitions(paths),
    calc<-function(g=splits(G), p=splits(paths,i), vertex=i) {
    library(igraph)
    p<-shortest.paths(graph.adjacency(g), vertex)
    update(p)
})

progress: 100%
```

Fetch all shortest distances from the first vertex then print the first ten values:

```
> getpartition(paths, 1)[,1:10]
[1] 0 2 2 3 2 2 3 2 1 2
```

# Examples of Parallelizing Data in Distributed R

These examples demonstrate how you can decrease the execution time of a query by using parallel programming in Distributed R.

The number of samples used in these examples is 10 million, which results in a data set size of 34.5 MB. Because this sample size is not very large, you may not notice a decrease in execution time when you run the parallel programming examples. However, the decrease in execution time becomes apparent when you use a larger data set, such as one that is several hundred megabytes in size. The examples provided only illustrate how to use parallel task execution and parallel data execution on large data sets.

# Loading and Synthesizing Data

Install and load the package named "zipcode" as follows:

```
> install.packages("zipcode")
> library(zipcode)
> data(zipcode)
```

Synthesize the data, and create a data frame named "exampleData":

```
# Limit the number of samples to 1 million.
> nSamples <- 1e6

# Create the exampleData data frame using the data from zipcode.
> exampleData <- data.frame(id = 1:nSamples, name = replicate(nSamples, paste(sample
(LETTERS, 3, replace=TRUE), collapse="")),
zipcode[sample.int(nrow(zipcode), nSamples, replace=TRUE),c("city","state","zip")],
row.names = 1:nSamples)
```

```
# Determine the size of the exampleData object. With 1 million samples, the size is
34.5MB.
> format(object.size(exampleData), units = "auto")
```

Use the data frame created,"exampleData", for all of the following examples.

# Sequential Execution

In this example, you use a sequential execution approach to determine the following values:

- Most common Name, City, State, and ZIP codes.
- Number of unique Names, Cities, States, and ZIP codes.

You can find these values without using Distributed R, using only R:

```
# Set a timer.
> system.time({

# Create a list called summarySeq. It takes as an output the data we are looking for.
     summarySeq <- list(popularName  = names(which.max(table(exampleData$name))),
                        popularCity  = names(which.max(table(exampleData$city))),
                        popularState = names(which.max(table(exampleData$state))),
                        popularZip   = names(which.max(table(exampleData$zip))),

                        numberOfUniqueNames  = length(unique(exampleData$name)),
                        numberOfUniqueCities = length(unique(exampleData$city)),
                       numberOfUniqueStates = length(unique(exampleData$state)),
                       numberOfUniqueZips   = length(unique(exampleData$zip))
                  )
})
```

View the output of summarySeq:

```
> summarySeq
$popularName
[1] "EBR"

$popularCity
[1] "Apo"

$popularState
[1] "TX"

$popularZip
[1] "04852"

$numberOfUniqueNames
[1] 17576
```

```
$numberOfUniqueCities
[1] 19108

$numberOfUniqueStates
[1] 62

$numberOfUniqueZIPs
[1] 44336
```

# Parallel Task Execution

The following example demonstrates how you can use the same data to perform the same execution shown in the preceding example. The only exception is that, this time, the tasks are performed in parallel execution.

In parallel execution, individual tasks are executed concurrently on separate executors. This allows you to spread the task execution over as many executors as there are tasks. This reduces the work load for an individual executor as well as the execution time.

```
# Load and start Distributed R.
> library(distributedR)
> distributedR_start()

# Set the timer.
> system.time({

# Set the number of tasks to be executed.

# Since we have 8 tasks (popularName, popularCity, popularState, popularZip,
numberOfUniqueNames, numberOfUniqueCities, numberOfUniqueStates, numberOfUniqueZips), we
set nTasks to 8.
     nTasks <- 8

# Create a dlist which is the size of the number of tasks to be executed (nTasks).
     dSummary <- dlist(nTasks)

# Start running parallelization with a foreach loop.
     foreach(i, 1:npartitions(dSummary), function(si = splits(dSummary, i),
data=exampleData, index=i) {
          switch(index,
              si <- list(popularName  = names(which.max(table(data$name)))),
              si <- list(popularCity  = names(which.max(table(data$city)))),
              si <- list(popularState = names(which.max(table(data$state)))),
              si <- list(popularZip   = names(which.max(table(data$zip)))),

              si <- list(numberOfUniqueNames  = length(unique(data$name))),
              si <- list(numberOfUniqueCities = length(unique(data$city))),
              si <- list(numberOfUniqueStates = length(unique(data$state))),
              si <- list(numberOfUniqueZips   = length(unique(data$zip)))
          )

# Globally push the modifications to the dlist using update().
          update(si)
     })
```

```
# Store the result in summaryPar1.
     summaryPar1 <- getpartition(dSummary)

})
```

View the output of summaryPar1:

```
> summaryPar1
$popularName
[1] "EBR"

$popularCity
[1] "Apo"

$popularState
[1] "TX"

$popularZip
[1] "04852"

$numberOfUniqueNames
[1] 17576

$numberOfUniqueCities
[1] 19108

$numberOfUniqueStates
[1] 62

$numberOfUniqueZIPs
[1] 44336
```

You can see from the result of summaryPar1 that the exact same results are obtained with parallel execution that were achieved with sequential execution.

# Parallel Data Execution

The example of parallel task execution gives you results that are superior to sequential execution. However, you need to use a second parallel approach when working with large amounts of data, such as a data set that is several hundred megabytes large. This second approach, known as parallel data execution, divides the input data into separate partitions which are then sent to the executors.

Parallel data execution is a more scalable approach. Distributed R recommends this approach for loading large amounts of data into HPE Vertica, such as a large data frame.

The parallel data execution approach also takes advantage of HPE Vertica load balancing capabilities. It does so by splitting the data into partitions and sending those partitions to particular worker nodes. By comparison, parallel task execution from the preceding example sends the entirety of exampleData to all of the worker nodes at once.

```
# Set the timer for execution.
system.time({
     DF <- as.dframe(exampleData)

# When a data frame is loaded by db2dframe, its observations must be converted first to
factors.
     factor.dframe(DF)

# In this example, dSummary is a list of the number of partitions in the exampleData data
frame.
     dSummary <- dlist(npartitions(DF))

# Create a list named si that takes in the values of all of our eight tasks.
# The worker nodes each take a portion of a task to be calculated, then these values are
aggregated later on.
# Each executor is doing the same job, just on a different partition of the data.
# In this foreach loop, rather than passing all of the exampleData at once, as we did in
the previous 2 examples, the data is partitioned among the worker nodes.
     foreach(i, 1:npartitions(dSummary), function(si=splits(dSummary, i), datai=splits
(DF, i)) {
         si <- list( nameFrequency  = table(datai$name),
                     cityFrequency  = table(datai$city),
                     stateFrequency = table(datai$state),
                     zipFrequency   = table(datai$zip),

                     localUniqueNames  = unique(datai$name),
                     localUniqueCities = unique(datai$city),
                     localUniqueStates = unique(datai$state),
                     localUniqueZips   = unique(datai$zip)
                    )
         update(si)
     })

# Intermediate holds the values of the data from all of the individual partitions.

     intermediate <- getpartition(dSummary)

# Create an empty list named summaryPar2.
     summaryPar2 <- list()

# Aggregate the data by adding the observations together to find which Name, City, State,
and ZIP occur the most frequently.
# Reduce uses a binary function to recursively combine all of the elements together in
order to aggregate the data.
     summaryPar2$popularName  <- names(which.max(Reduce('+', intermediate[which(names
(intermediate) == "nameFrequency")])))
     summaryPar2$popularCity  <- names(which.max(Reduce('+', intermediate[which(names
(intermediate) == "cityFrequency")])))
     summaryPar2$popularState <- names(which.max(Reduce('+', intermediate[which(names
(intermediate) == "stateFrequency")])))
     summaryPar2$popularZip   <- names(which.max(Reduce('+', intermediate[which(names
(intermediate) == "zipFrequency")])))

# The generic R function c combines its arguments.
     summaryPar2$numberOfUniqueNames  <- length(unique(Reduce('c', intermediate[which
(names(intermediate) == "localUniqueNames)])))
     summaryPar2$numberOfUniqueCities <- length(unique(Reduce('c', intermediate[which
(names(intermediate) == "localUniqueCities")])))
```

```
     summaryPar2$numberOfUniqueStates <- length(unique(Reduce('c', intermediate[which
(names(intermediate) == "localUniqueStates")])))
     summaryPar2$numberOfUniqueZips   <- length(unique(Reduce('c', intermediate[which
(names(intermediate) == "localUniqueZips)])))

})
```

View the output of summaryPar2:

```
> summaryPar2
$popularName
[1] "EBR"

$popularCity
[1] "Apo"

$popularState
[1] "TX"

$popularZip
[1] "04852"

$numberOfUniqueNames
[1] 17576

$numberOfUniqueCities
[1] 19108

$numberOfUniqueStates
[1] 62

$numberOfUniqueZips
[1] 44336
```

The results are the same as the first two approaches.

# Distributed R Command Reference

This section provides an overview and examples for each of the functions available in the Distributed R package,

| Command | Description |
|---|---|
| **Session Management Commands** | |
| distributedR_start() | Start a new Distributed R session. |
| distributedR_status() | Obtain master and worker information. |
| distributedR_master_info() | Obtain details about the master node. |
| distributedR_shutdown() | End session. |
| **Distributed Data Structure Commands** | |
| darray() | Create distributed array. |
| dframe() | Create distributed data frame. |
| dlist() | Create distributed list. |
| getpartition() | Fetch darray, dframe, or dlist object. |
| npartitions() | Obtain total number of partitions. |
| partitionsize() | Get the size of a partition. |
| is.darray() | Checks if the input is a darray. |
| as.darray() | Create darray object from matrix object. |
| as.dframe() | Create dframe object from matrix object or data.frame. |
| factor.dframe() | Converts categorical columns of a dframe into factors. |
| as.factor.dframe() | Converts categorical columns of a dframe into factors and creates a new dframe, leaving the original dframe unchanged. |
| unfactor.dframe() | Converts categorical columns of a dframe into their label of type character. |
| levels.dframe() | Finds the categorical columns of a dframe and returns a level array and an associated column array identifying the column for the levels. |
| clone() | Clone or deep copy of a darray. |

| Parallel Execution Commands | |
|---|---|
| `foreach()` | Execute function on cluster. |
| `splits()` | Pass partition to foreach loop. |
| `update()` | Make partition changes inside foreach loop globally visible. |

# distributedR_start

distributedR_start() opens network connections between the master and the workers and starts the Distributed R runtime on the worker nodes.

Type `help(distributedR_start)` in the R console for details on arguments or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
Loading required package: Rcpp
Loading required package: RInside
Loading required package: XML

> distributedR_start()
Workers registered - 3/3.

All 3 workers are registered.
Master address:port - node01:50000
```

# distributedR_status

`distributedR_status()` shows the status of each worker. Use with the argument `help=TRUE` to display descriptions about each column.

Type `help(distributedR_status)` in the R console for details on arguments or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
Loading required package: Rcpp
Loading required package: RInside
Loading required package: XML

> distributedR_start()
Workers registered - 3/3.

All 3 workers are registered.
```

```
Master address:port - node01:50000

> distributedR_status(help=TRUE)

distributedR_status - help
Workers: list of workers
Inst: number of R instances on a worker
SysMem: total available system memory (MB)
MemUsed: memory currently used in a worker (MB)
DarrayQuota:  memory for darrays (MB)
DarrayUsed: memory currently used by darray (MB)

       Workers Inst SysMem MemUsed DarrayQuota DarrayUsed
1 node01:50001    8  11911    3214       10137          0
2 node02:50000    8  11911    1921       10137          0
3 node03:50000    8  11911    1928       10137          0
```

# distributedR_master_info

`distributedR_master_info()` provides details about the master node, including address, port, and session ID.

Type `help(distributedR_master_info)` in the R console for additional details or see the Distributed R Package Manual.

# Example

```
> library(distributedR)
Loading required package: Rcpp
Loading required package: RInside
Loading required package: XML

> distributedR_start()
Workers registered - 3/3.

All 3 workers are registered.
Master address:port - node01:50000

> distributedR_master_info()
$address
[1] "node01"

$port
[1] 50000

$sessionID
[1] 50505
```

# distributedR_shutdown

`distributedR_master_shutdown()` stops the Distributed R runtime on the worker nodes, closes connections to them, and frees resources associated with Distributed R.

Type `help(distributedR_shutdown)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
Loading required package: Rcpp
Loading required package: RInside
Loading required package: XML

> distributedR_start()
Workers registered - 3/3.

> distributedR_shutdown()
Shutdown complete
```

# darray

```
darray (dim, blocks, sparse = FALSE, data = 0, empty=FALSE, distribution_
policy='roundrobin')
darray (npartitions, sparse = FALSE, distribution_policy='roundrobin')
```

The `darray()` function creates distributed arrays, which store in-memory, multi-dimensional data across several machines (worker nodes). Data can be partitioned into chunks of rows, columns, or blocks. Distributed arrays can store only numeric data. You can only pass scalar data types to a darray.

By default, array partitions are internally stored as dense matrices. If an array is specified sparse, partitions are stored in the compressed sparse column format. Last set of partitions may have fewer rows or columns if array size is not an integer multiple of partition size. For example, the distributed array darray(dim=c(5,5), blocks=c(2,5)) has three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

Distributed arrays can also be defined by specifying just the number of partitions, but not their sizes. This flexibility is useful when the size of an array is not known. For example, darray (npartitions=c(5,1)) is a dense array with five partitions. Each partition can contain any number of rows, though the number of columns should be same to conform to a well formed array.

Distributed arrays can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on array partitions in foreach loops. Loop body is executed at workers. Partitions can be passed as arguments using splits. Array modifications can be published globally using update.

Distributed arrays can be fetched at the master using getpartition. Number of partitions can be obtained by npartitions. Partitions are numbered from left to right, and then top to bottom (row major order). Dimension of each partition can be obtained using partitionsize.

Type `help(darray)` in the R console for additional details or see the Distributed R Package Manual.

**Note:** The `mean()` function included with darray may return `inf` (infinite) for extremely large numbers due to overflow in the R `sum()` function.

# Supported Operations

The `darray` function supports the following basic operators between a distributed array and a numeric as well as between two distributed arrays:

- Multiplication(*)
- Addition(+)
- Subtraction(-)

**Note:**For multiplication between two distributed arrays, matrix multiplication is used.

The `darray` function additionally supports the following operators when the input is a single distributed array:

- max()
- min()
- sum()
- mean()
- colSums()
- rowSums()
- colMeans()
- rowMeans()
- head()
- tail()
- ncol()
- nrow()

# Example

```
> library(distributedR)
> distributedR_start()
```

```
> da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
> cat("Number of partitions of da are ", npartitions(da),"\n")
> db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
> result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
> foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
                b = splits(db,i),
                c = splits(result,i)){
    c <- a + b
    update(c)
  })
> getpartition(result)
> distributedR_shutdown()
```

# dframe

```
dframe (dim, blocks, distribution_policy='roundrobin')
dframe (npartitions, distribution_policy='roundrobin')
```

The `dframe()` function creates distributed data frames, which store in-memory, multi-dimensional data across several machines. Data can be partitioned into chunks of rows, columns, or blocks. Unlike distributed arrays (darray), dframe can store both numeric and string data. However, dframe can be space-inefficient, and should be replaced by *darray* wherever possible.

Distributed data frame partitions are internally stored as data.frame objects. Last set of partitions may have fewer rows or columns if data frame size is not an integer multiple of partition size. For example, the distributed data frame dframe(dim=c(5,5), blocks=c(2,5)) has three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

Distributed data frames can also be defined by specifying just the number of partitions, but not their sizes. This flexibility is useful when the size of a data frame is not known ahead of time. For example, dframe(npartitions=c(5,1)) is a data frame with five partitions. Each partition can contain any number of rows, though the number of columns should be the same to conform to a well-formed data frame.

Distributed data frames can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on partitions in foreach loops. Loop body is executed at workers. Partitions can be passed as arguments using splits. Array modifications can be published globally using update.

Distributed data frames can be fetched at the master using getpartition. Number of partitions can be obtained by npartitions. Partitions are numbered from left to right, and then top to bottom (row major order). Dimension of each partition can be obtained using partitionsize.

Type `help(dframe)` in the R console for additional details or see the Distributed R Package Manual.

# Supported Operations

The `dframe` function supports the following basic operators between a distributed data frame and a numeric as well as between two distributed data frames:

- Multiplication(*)
- Subtraction(-)

The `dframe` function additionally supports the following operators when the input is a single distributed data frame:

- max()
- min()
- sum()
- mean()
- colSums()
- rowSums()
- colMeans()
- rowMeans()
- head()
- tail()
- ncol()
- nrow()

# Example

```
> library(distributedR)
> distributedR_start()
> df <- dframe(c(20,4),c(10,2))
> data_path <- system.file("extdata",package="distributedR")
> file_path <- paste(data_path,"/df_data",sep="")
##Populate distributed data frame
> foreach(i, 1:npartitions(df), function(sf=splits(df,i),ii=i,path=file_path){
  sf <- read.table(paste(path,ii,sep=""))
  update(sf)
})
> getpartition(df)
##Rename columns
```

```
> name_sample <- as.character(sample(1:4))
> dimnames(df)[[2]] <- name_sample
> getpartition(df)

##Flexible sized data frame. Five partitions, each with variable number of rows.
> dc <- dframe(npartitions=c(5,1))
> foreach(i, 1:npartitions(dc), initArrays<-function(y=splits(dc,i), index=i) {
     y<-data.frame(matrix(index, nrow=index,ncol=5))
     update(y)
})
> cat("value of 2nd partition is: \n")
> getpartition(dc,2)
> getpartition(dc)

> distributedR_shutdown()
```

# dlist

```
dlist (npartitions)
```

The `dlist()` function creates distributed lists, which stores in-memory lists across several machines. Just like R's list, dlist can store other R objects such as character, numeric and logical vectors, lists, matrices, and models. However, dlist can be space-inefficient, and should be replaced by darray when possible.

Distributed lists are internally stored as list objects. Each partition of the list can have variable number of elements in it. For example, the distributed list dlist(npartitions=5) has five partitions. Each partition is an empty list().

Distributed lists can be read-shared by multiple concurrent tasks, but modified by only a single writer per partition. Programmers express parallelism by applying functions on dlist partitions in foreach loops. Loop body is executed at workers. Partitions can be passed as arguments using splits. List modifications can be published globally using update.

Distributed lists can be fetched at the master using getpartition. Number of partitions can be obtained by npartitions. Partitions are numbered from left to right.

Type `help(dlist)` in the R console for additional details or see the Distributed R Package Manual.

# Example

```
> library(distributedR)
> distributedR_start()
> dl <- dlist(5)
##Populate distributed list
> foreach(i, 1:npartitions(dl), function(sf=splits(dl,i), idx=i){
  sf<-list(c("distR", idx))
  update(sf)
})
> getpartition(dl)
```

```
[[1]]
[1] "distR" "1"

[[2]]
[1] "distR" "2"

[[3]]
[1] "distR" "3"

[[4]]
[1] "distR" "4"

[[5]]
[1] "distR" "5"

> distributedR_shutdown()
```

# getpartition

```
getpartition (input_array, partition_index, column_index)
```

The getpartition() function fetches partition(s) from a darray, dframe or dlist.

If both partition_index and column_index are missing then the full input darray, dframe or dlist is returned.

2-D partitioning is valid only for darray and dframe. Since dlist is partitioned lengthwise, only partition_index is used to fetch a dlist partition. The column_index argument is undefined for dlist.

Partitions are numbered from left to right and then top to bottom (row-major order). Partition numbers start from 1. For row partitioning (each partition has all the columns) or column partitioning (each partition has all the rows) the column_index argument should not be used. For 2-D partitioning, both partition_index and column_index may be used.

Type `help(getpartition)` in the R console for additional details or see the Distributed R Package Manual.

# Example

```
> library(distributedR)
> distributedR_start()
##Input array of size 5X5 with 4 partitions
> da <- darray(dim=c(5,5), blocks=c(3,3), data=7)
##Return full array
> getpartition(da)
     [,1] [,2] [,3] [,4] [,5]
[1,]    7    7    7    7    7
[2,]    7    7    7    7    7
[3,]    7    7    7    7    7
[4,]    7    7    7    7    7
```

```
[5,]    7    7    7    7    7

##Return third partition (bottom-left)
> getpartition(da,3)
     [,1] [,2] [,3]
[1,]    7    7    7
[2,]    7    7    7

##Return fourth partition (bottom-right)
> getpartition(da,2,2)
     [,1] [,2]
[1,]    7    7
[2,]    7    7
```

# npartitions

```
npartitions (x)
npartitions2D (x)
```

The `npartitions()` and `npartitions2D()`functions return the number of partitions in a darray, dframe or dlist. `npartitions()` returns an integer that denotes the number of partitions. `npartitions2D()` returns a vector that denotes the number of partitions in each direction (rows/columns).

npartitions returns the total number of partitions in the distributed object. Use npartitions2D to obtain the number of partitions along each direction.

Type `help(npartitions)` in the R console for additional details or see the Distributed R Package Manual.

# Example

```
> library(distributedR)
> distributedR_start()
##Input array of size 5X5 with 4 partitions
> da <- darray(dim=c(5,5), blocks=c(3,3), data=7)
> npartitions(da)
[1] 4

> npartitions2D(da)
[1] 2 2
```

# partitionsize

```
partitionsize (x, partition_index)
partitionsize (x)
```

The `partitionsize()` function returns the dimensions of partitions in a darray, dframe or dlist. `partitionsize()` returns a matrix that denotes the number of rows and columns in the partition.

Type `help(partitionsize)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
##Input array of size 5X5 with 4 partitions
> da <- darray(dim=c(5,5), blocks=c(3,3), data=7)
> partitionsize(da,1)
     [,1] [,2]
[1,]    3    3

> partitionsize(da,2)
     [,1] [,2]
[1,]    3    2

> partitionsize(da)
     [,1] [,2]
[1,]    3    3
[2,]    3    2
[3,]    2    3
[4,]    2    2
```

# is.darray

```
is.darray(x)
```

The is.darray() function checks if the input is a darray and returns TRUE when the input is a darray and FALSE otherwise.

Type `help(is.darray)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> m <- matrix(sample(0:1, 16, replace=T), nrow=4)
> is.darray(m)
[1] FALSE

> dm <- darray(dim=c(5,5),blocks=c(1,5))
> is.darray(dm)
[1] TRUE
```

# as.darray

```
as.darray(input, blocks)
```

The as.darray() function converts an input matrix into a distributed array (darray) with dimensions equal to that of the input matrix and partitioned according to argument 'blocks'. Data may reside as partitions on remote nodes. You can load a dense matrix and convert it to a dense distributed array, or you can load a sparse matrix and convert it to a sparse distributed array.

If partition size (blocks) is missing then the input matrix is row partitioned and striped across the cluster, so that the returned distributed array has approximately as many partitions as the number of R instances in the Distributed R session. The last set of partitions may have fewer rows or columns if input matrix size is not an integer multiple of partition size. For example, the distributed array `as.darray(matrix(1,nrow=5,ncol=5), blocks=c(2,5))` has three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

To create a distributed array with just one partition, pass the dimension of the input array. For example: `as.darray(A, blocks=dim(A))`.

Type `help(as.darray)` in the R console for additional details or see the Distributed R Package Manual.

# Example

```
> library(distributedR)
> distributedR_start()
##Create 4x4 matrix
> mtx <- matrix(sample(0:1, 16, replace=T), nrow=4)
##Create distributed array spread across the cluster
> da <- as.darray(mtx)
> partitionsize(da)

     [,1] [,2]
[1,]    1    4
[2,]    1    4
[3,]    1    4
[4,]    1    4
```

# as.dframe

```
as.dframe(data.frame, blocks)
```

The as.dframe() function converts an input matrix or regular R data.frame into a distributed dframe (dframe) with dimensions equal to that of the input matrix/data.frame and partitioned according to argument 'blocks'. Data may reside as partitions on remote nodes.

If partition size (blocks) is missing then the input matrix/data.frame is row partitioned and striped across the cluster, i.e., the returned distributed frame has approximately as many partitions as the number of R instances in Distributed R session.

The last set of partitions may have fewer rows or columns if input matrix size is not an integer multiple of partition size. If 'A' is a 5x5 matrix, then 'as.dframe(A, blocks=c(2,5))' is a distributed frame with three partitions. The first two partitions have two rows each but the last partition has only one row. All three partitions have five columns.

To create a distributed frame with just one partition, pass the dimension of the input array. For example: `as.dframe(A, blocks=dim(A))`.

Type `help(as.dframe)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
# Single node Distributed R install
# Two Executors
> library(distributedR)
> distributedR_start()
> cars = mtcars
> dim(cars)
[1] 32 11
> dcars = as.dframe(cars)
# Data is split for two executors:
> partitionsize(dcars)
     [,1] [,2]
[1,]   16   11
[2,]   16   11
```

# factor.dframe

```
factor.dframe(dframe, colName, colID, trace=FALSE)
```

The factor.dframe() function converts categorical columns (character, logical, or factor) of a dframe into factors. If no columns or column IDs are provided then all supported types (character, logical and factor) in the dframe are converted. Conversions are done in place; the original dframe is modified and the function returns nothing.

This function works the same as as.factor.dframe except that it modifies the dframe in place instead of returning a modified dframe.

**Note:** This function does not support numeric values because of the potentially huge levels of arrays that could be created with large data sets.

Type `help(factor.dframe)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> DF <- dframe(c(9,3),c(3,3))
> foreach(i,1:npartitions(DF),function(dfi=splits(DF,i),idx=i){
 if(idx==1) {
     dfi[,1] <- 1:3
     dfi[,2] <- c('c1','c2','c3')
     dfi[,3] <- c('t1','t2','t3')
 } else if(idx==2) {
     dfi[,1] <- 2:4
     dfi[,2] <- c('c2','c3','c4')
     dfi[,3] <- c('t1','t2','t3')
 } else {
     dfi[,1] <- 11:13
     dfi[,2] <- c('c3','c4','c5')
     dfi[,3] <- c('t4','t5','t6')
 }
 update(dfi)
})

> str(getpartition(DF)
'data.frame':  9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: chr  "c1" "c2" "c3" "c2" ...
 $ X3: chr  "t1" "t2" "t3" "t1" ...

> factor.dframe(DF)
> str(getpartition(DF))
'data.frame':  9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: Factor w/ 5 levels "c1","c2","c3",..: 1 2 3 2 3 4 3 4 5
 $ X3: Factor w/ 6 levels "t1","t2","t3",..: 1 2 3 1 2 3 4 5 6
```

# as.factor.dframe

```
as.factor.dframe(dframe, colName, colID, trace=FALSE)
```

The as.factor.dframe() function converts categorical columns (character, logical, or factor) of a dframe into factors. If no columns or column IDs are provided then all supported types (character, logical and factor) in the dframe are converted.

This function works the same as factor.dframe except that it does not modify the dframe in place. Instead, it returns a modified dframe and leaves the original dframe unchanged.

**Note:** This function does not support numeric values because of the potentially huge levels of arrays that could be created with large data sets.

Type `help(as.factor.dframe)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> DF <- dframe(c(9,3),c(3,3))
> foreach(i,1:npartitions(DF),function(dfi=splits(DF,i),idx=i){
 if(idx==1) {
     dfi[,1] <- 1:3
     dfi[,2] <- c('c1','c2','c3')
     dfi[,3] <- c('t1','t2','t3')
 } else if(idx==2) {
     dfi[,1] <- 2:4
     dfi[,2] <- c('c2','c3','c4')
     dfi[,3] <- c('t1','t2','t3')
 } else {
     dfi[,1] <- 11:13
     dfi[,2] <- c('c3','c4','c5')
     dfi[,3] <- c('t4','t5','t6')
 }
 update(dfi)
})

> str(getpartition(DF))
'data.frame':  9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: chr  "c1" "c2" "c3" "c2" ...
 $ X3: chr  "t1" "t2" "t3" "t1" ...

> newDF = as.factor.dframe(DF)
> str(getpartition(newDF))'data.frame':        9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: Factor w/ 5 levels "c1","c2","c3",..: 1 2 3 2 3 4 3 4 5
 $ X3: Factor w/ 6 levels "t1","t2","t3",..: 1 2 3 1 2 3 4 5 6

> str(getpartition(DF))
'data.frame':  9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: chr  "c1" "c2" "c3" "c2" ...
 $ X3: chr  "t1" "t2" "t3" "t1" ...
```

# unfactor.dframe

```
unfactor.dframe(dframe, colName, colID, trace=FALSE)
```

The unfactor.dframe() function converts categorical columns (character, logical, or factor) of a dframe into their label of type character. If no columns or column IDs are provided then all supported types (character, logical and factor) in the dframe are converted. Conversions are done in place; the original dframe is modified and the function returns nothing.

Type `help(unfactor.dframe)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> DF <- dframe(c(9,3),c(3,3))

> foreach(i,1:npartitions(DF),function(dfi=splits(DF,i),idx=i){
 if(idx==1) {
     dfi[,1] <- 1:3
     dfi[,2] <- c('c1','c2','c3')
     dfi[,3] <- c('t1','t2','t3')
 } else if(idx==2) {
     dfi[,1] <- 2:4
     dfi[,2] <- c('c2','c3','c4')
     dfi[,3] <- c('t1','t2','t3')
 } else {
     dfi[,1] <- 11:13
     dfi[,2] <- c('c3','c4','c5')
     dfi[,3] <- c('t4','t5','t6')
 }
 update(dfi)
})

> str(getpartition(DF))
> factor.dframe(DF)

> str(getpartition(DF))
'data.frame':   9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: Factor w/ 5 levels "c1","c2","c3",..: 1 2 3 2 3 4 3 4 5
 $ X3: Factor w/ 6 levels "t1","t2","t3",..: 1 2 3 1 2 3 4 5 6
unfactor.dframe(DF)

> str(getpartition(DF))
'data.frame':   9 obs. of  3 variables:
 $ X1: int  1 2 3 2 3 4 11 12 13
 $ X2: chr  "c1" "c2" "c3" "c2" ...
 $ X3: chr  "t1" "t2" "t3" "t1" ...
```

# levels.dframe

```
levels.dframe(dframe, colName, colID, trace=FALSE)
```

The levels.dframe() function finds the categorical columns (character, logical, or factor) of a dframe and returns a level array and an associated column array identifying the column for the levels. If no columns or column IDs are provided then all supported types (character, logical and factor) in the dframe are considered. The dframe is not modified.

Type `help(levels.dframe)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> DF <- dframe(c(9,3),c(3,3))
> foreach(i,1:npartitions(DF),function(dfi=splits(DF,i),idx=i){
 if(idx==1) {
     dfi[,1] <- 1:3
     dfi[,2] <- c('c1','c2','c3')
     dfi[,3] <- c('t1','t2','t3')
 } else if(idx==2) {
     dfi[,1] <- 2:4
     dfi[,2] <- c('c2','c3','c4')
     dfi[,3] <- c('t1','t2','t3')
 } else {
     dfi[,1] <- 11:13
     dfi[,2] <- c('c3','c4','c5')
     dfi[,3] <- c('t4','t5','t6')
 }
 update(dfi)
})

> levels.dframe(DF)
$Levels
$Levels[[1]]
[1] "c1" "c2" "c3" "c4" "c5"

$Levels[[2]]
[1] "t1" "t2" "t3" "t4" "t5" "t6"


$columns
[1] 2 3
```

# clone

```
clone(input)
clone(input, nrow=NA, ncol=NA, data=0, sparse=NA)
```

The `clone()` function creates a copy of a darray or dframe. Depending on the input object, `clone()` returns a darray or dframe with the same dimensions, block size, and values as the input darray or dframe, unless clone is called with options.

Setting distributed data-structures such as a darray equal to another does not result in a copy. For example, after assignment da = db, the two distributed arrays da and db will refer to the same data. Operations on any of these arrays will manipulate the same single copy of data. To make a copy, a darray needs to be explicitly cloned using clone.

`clone()` can also be used to copy just the structure of a distributed object, such as the number of partitions and the partition sizes. For example, if da is a Nx10 distributed dense array, clone(da, ncol=1, data=1) will create a dense array with same number of partitions and rows as da but with

only 1 column. All elements in the resulting darray will be 1. Note that if any argument, such as ncol, is used with clone.

Type `help(clone)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> mtx <- matrix(sample(0:1, 16, replace=T), nrow=4)
> da <- as.darray(mtx)
> db <- clone(da)
progress: 100%

> all(da==db)
[1] TRUE

> dc <- clone(da, ncol=2, data=2)
> getpartition(dc)
     [,1] [,2]
[1,]    2    2
[2,]    2    2
[3,]    2    2
[4,]    2    2
```

# foreach

```
foreach(index, range, func, progress=TRUE, scheduler=0)
```

`foreach()` executes a function in parallel on worker nodes. Programmers can pass any R object as an argument to the function. darray, dframe or dlist and their partitions can be passed using splits.

The `foreach` loop or the function executed by it does not return any value. Instead, users can call update inside foreach to modify distributed arrays, data frames, or lists, and publish changes. Note that update is the only way to make the changes globally visible.

Type `help(foreach)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
> cat("Number of partitions of da are ", npartitions(da),"\n")
> db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
> result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
> foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
                b = splits(db,i),
```

```
                c = splits(result,i)){
    c <- a + b
    update(c)
  })
> getpartition(result)
> distributedR_shutdown()
```

# Deterministic Behavior of a foreach Function

If no data partitions are passed to the `foreach` function, the function always executes on each executor of each worker node. Thus, the range argument you specify must be either equal to or greater than the total number of executors on all worker nodes.

You can use this deterministic behavior, for example, to set an environment variable or load a library on all executors in your Distributed R cluster.

Run the following code to load a library on every executor on all worker nodes:

```
> library(distributedR)
> distributedR_start()
> foreach (i, 1:<total number of executors>, function() {
  library(name of package)
})
```

The `set.seed()` function cannot be passed to all of the executors. This function cannot pass because the value set by `set.seed()` is automatically deleted after the execution of the `foreach` function on an individual executor.

# Multiple Splits Inside a foreach Loop

You can pass list and vector-based arguments into the splits() function within a foreach loop. Doing so allows you to assign splits to specific executors (R sessions).

## *Syntax*

The syntax for passing list and vector-based arguments into the splits() function is:

```
splits(dobject,list)
```

```
splits(dobject,vector)
```

The dobject can be a dlist, darray, or dframe.

## *Assign Splits to a dobject*

The following example demonstrates how to assign splits to a dobject named A. In this case, A has 9 splits and a task parallelism of 3. You can divide the splits as follows:

- Splits 1, 2, and 3—Assign to executor 1
- Splits 4, 5, and 6—Assign to executor 2

- Splits 7, 8, and 9—Assign to executor 3

```
> A <- darray(dim=c(9,9), blocks=c(3,3))
> foreach(i, 1:3, function(B=splits(A,c(3*(i-1)+1, 3*(i-1)+2, 3*(i-1)+3)), index=i) {
     for (j in 1:length(B)) {
     nr <- nrow(B[[j]])
     nc <- ncol(B[[j]])
     B[[j]] <- matrix(index, nrow=nr, ncol=nc)
}
> update(B)
})
```

As an alternative, you can use the following foreach loop:

```
> foreach(i, 1:3,function(B=splits(A,2:4), index=i) {
     for (j in 1:length(B)) {
     nr <- nrow(B[[j]])
     nc <- ncol(B[[j]])
     B[[j]] <- matrix(index, nrow=nr, ncol=nc)
}
})
```

In this case, B is passed as a list on each executor. The foreach loop divides the splits as follows:

- B[[1]] in the function body has split 2 of A
- B[[2]] in the function body has split 3 of A
- B[[3]] in the function body has split 4 of A

## *Write a List to Pass to splits()*

You can also write a list and pass it to the splits() function:

```
> foreach(i, c(1,4,7),function(B=splits(A,list(i,i+1,i+2)), index=i) {
     for(j in 1:length(B)) {
     nr <- nrow(B[[j]])
     nc <- ncol(B[[j]])
     B[[j]] <- matrix(index, nrow=nr, ncol=nc)
}
> update(B)
})
```

As an alternative, you can use the following foreach loop:

```
> foreach(i, 1:3, function(B=splits(A,list(1,2,3,4)), index=i) {
     for(j in 1:length(B)) {
     nr <- nrow(B[[j]])
     nc <- ncol(B[[j]])
     B[[j]] <- matrix(index, nrow=nr, ncol=nc)
}
```

```
})
```

When writing a list and passing it to the splits() function, you can use the list function on B in the function body:

```
> foreach(i, c(1,4,7), function(B=splits(A,list(i,i+1,i+2)), index=i){

     is.list(B) #TRUE
     foo = lapply(B,is.matrix)

     # foo
     # [[1]]
     # [1] TRUE

     # [[2]]
     # [1] TRUE

     # [[3]]
     # [1] TRUE
})
```

## *Treatment of Arguments Assigned to a Split or Multiple Splits*

Suppose an argument has been assigned to a split or multiple splits. Whether the argument is considered a list-type argument by the function depends on the second argument supplied to the splits() function. This second argument can contain either multiple values or a list.

Consider a distributed array named B and a foreach loop that takes in B and another argument. The argument assigned to the splits() function is named A:

```
> B <- darray(dim=c(9,9), blocks=c(3,3))
> foreach(i, 1:3, function(A=splits(B,<second argument>))
```

The second argument's properties determine its effect on A:

| Second Argument Properties | A Is... |
|---|---|
| Contains a single numeric value or an expression that evaluates to a single scalar value (for example, *i*) | Not a list |
| Is a vector of many values | List |
| Is a list containing any number of values (even if only one) | List |

If A is a list, then the function can treat it as a list object on each executor. In this case, use the R syntax that is specific to lists when running your foreach loop.

These examples show cases where a second argument is assigned to a split or multiple splits. Both the type of the split and the type of the second argument determine whether or not the function treats A as a list-type argument.

| Assignment Style | Split Type | Is A a List? |
|---|---|---|
| A = splits(B,1) | Single, shared split | No |
| A = splits(B,i) | Single, non-shared split | No |
| A = splits(B) | Composite | No |
| A = splits(B,1:3) | Vector of shared splits | Yes |
| A = splits(B,1:1) | Single, shared split with colon<br>1:1 evaluates to 1 | No |
| A = splits(B,c(1)) | Single, shared split | No |
| A = splits(B,c(i)) | Single, non-shared split with vectorized syntax | No |
| A = splits(B,c(i, i+1)) | Multiple splits | Yes |
| A = splits(B,list(i, i+1)) | Multiple splits defined in a list style | Yes |
| A = splits(B,list(i)) | Single split defined in a list style | Yes |

# Determining Splits with Functions

When passing multiple splits inside a foreach loop, you can program which splits are sent to a particular executor. You can do so using a foreach loop with a function as an argument. In the following example, the function is named foo:

```
foreach(i,...,function(B=splits(A,foo(i,...)), index=i)
{...}
)
```

You must define the foo function as a function that can either return a list of indexes, a vector of indexes, or a scalar value.

These three examples show ways you can use the foo function.

In this list case, B becomes a list:

```
> foo <- function(index){
     ind <- list()
     ind[[1]] = 3*(index-1) + 1
     ind[[2]] = 3*(index-1) + 2
     ind[[3]] = 3*(index-1) + 3
```

```
    return(ind)
}
```

In this vector case, B becomes a list:

```
> foo <- function(index,<number of parts per executor, in this case 3>) {
    start <- (<number of parts per executor>-1)*index+1
    end <- start+<number of parts per executor>-1
    return(start:end)
}
```

In this scalar case, B does not become a list:

```
> foo <- function(index)
    ind <- (index +7) %% npartitions(A)+1
    return(ind)
}
```

## *Complete Split Determined with Function Example*

The following example uses foo as a vector function.

1. Create the foo function:

```
> foo <- function(index,<number of parts per executor, in this case 3>) {
    start <- (<number of parts per executor> - 1)*index+1
    end <- start+3-1
    return(start:end)
}
```

2. Run the foreach loop on the darray A:

```
> A <- darray(dim=c(9,9), block=c(3,3))
> foreach(i, 1:3, function(B=splits(A,foo(i,npartitions(A)), index=i) {
          for(j in 1:length(B)) {
          nr <- nrow(B[[j]])
          nc <- ncol(B[[j]])
          B[[j]] <- matrix(index, nrow=nr, ncol=nc)
    }
}
```

# splits

```
splits(x, y)
```

Pass partition(s) of darray, dframe or dlist to function in foreach.

splits can be used only as an argument to the function in a foreach loop.

If `y` is missing then the full input darray, dframe or dlist is returned.

Partitions are numbered from left to right and then top to bottom (row-major order). Partition numbers start from 1.

Type `help(splits)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
> cat("Number of partitions of da are ", npartitions(da),"\n")
> db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
> result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
> foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
                b = splits(db,i),
                c = splits(result,i)){
    c <- a + b
    update(c)
  })
> getpartition(result)
> distributedR_shutdown()
```

# update

```
update(x)
```

Globally publish modifications done to a darray, dframe or dlist inside a foreach.

`update()` can be used only inside the foreach loop function.

The `foreach` loop or the function executed by it does not return any value. Instead, users can call update to modify distributed arrays, data frames or lists and publish changes. Note that update is the only way to make side-effects globally visible.

Type `help(update)` in the R console for additional details or see the Distributed R Package Manual.

## Example

```
> library(distributedR)
> distributedR_start()
> da <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=10)
> cat("Number of partitions of da are ", npartitions(da),"\n")
> db <- darray(dim=c(9,9), blocks=c(3,3), sparse=FALSE, data=5)
> result <- darray(dim=c(9,9), blocks=c(3,3))
##Add two matrices in parallel
```

```
> foreach(i, 1:npartitions(da),
  add<-function(a = splits(da,i),
                b = splits(db,i),
                c = splits(result,i)){
    c <- a + b
    update(c)
  })
> getpartition(result)
> distributedR_shutdown()
```

# Distributed R Algorithm Packages

This section details the algorithm packages available with Distributed R. The following packages are installed when you install Distributed R:

- HPdata - The Distributed Data Packages, provides functions for loading data into distributed data objects from databases and files. For more details see: About the HPdata Package.
- HPdclassifier - The Distributed algorithms for classifiers package. It contains the `hpdrandomForest()` function, which is a distributed version of randomForest, and an associated predict function, `predictHPdRF()`. It also contains the `hpdrpart()`, `hpdegbm()`, and `varImportance()` functions. For more details see: About the HPdclassifier Package
- HPdcluster - The Distributed clustering for Big Data package. It contains the `hpdkmeans()` function, which is a distributed kmeans algorithm. It also contains `hpdapply()`, a distributed version of the apply function. For more details see: About the HPdcluster Package.
- HPdgraph - The Distributed algorithms for graph analytics package. It contains the `hpdpagerank()` function, which is a distributed pagrerank algorithm. It also contains the `hpdwhich.max()` function, which acts as a which.max function on darrays. For more details see: About the HPdgraph Package.
- HPdregression - The Distributed Regression for Big Data package. It contains the `hpdglm()` function, which is a distributed glm function that performs regression analysis on distributed data types. It also contains `v.hpdglm()` and `cv.hpdglm()` which are functions used to evaluate models using split-sample and cross-validation respectively. For more details see: About the HPdregression Package

# About the HPdregression Package

The HPdregression package is a package for distributed regression algorithms. It provides a distributed Generalized Linear Model for use in the Distributed R environment. The HPdregression package contains:

| Function | Description |
|---|---|
| `hpdglm()` | A distributed alternative for the R glm function. |
| `v.hpdglm()` | Evaluates a model built by hpdglm() using the Split-Sample-Validation method. |
| `cv.hpdglm()` | Evaluates a model built by hpdglm() using the Cross-Validation method. |

# hpdglm

A distributed alternative for the R glm() function. The R glm() function is available in the stats package and in the standard R distribution and is used for performing many forms of regression. The hpdglm() function supports linear, logistic, and Poisson regression.

## *Estimating hpdglm Memory Requirements*

You can estimate the memory requirements for hpdglm() using the following formula.

Assume the following:

- For responses (type darray): *m* = nrow(*responses*) and ncol(*responses*) = 1
- For predictors (type matrix): *m* = nrow(*predictors*) and *p* = ncol(*predictors*)

  **Note:** nrow(*responses*) and nrow(*predictors*) must always be equal for hpdglm.

- Each data element also consumes 8 bytes for the size of the element.
- The number of blocks in the darray = the number of R executors.
- *m* is considerably bigger than *p*

To approximate the total required memory (including the memory which is allocated for the input darray) use the formula:

- For Gaussian: the formula `2 * m * (7 + p) * 8` returns the total number of bytes of memory required across the cluster. However in extreme cases the formula increases to `2 * m * (9 + p) * 8`.

- For Binomial and Poisson: the formula `2 * m * (9 + p) * 8` returns the total number of bytes of memory required across the cluster. However in extreme cases the formula increases to `2 * m * (11 + p) * 8`.

**Note:** In its complete mode, hpdglm returns 6 darrays of size `m * 1`, in addition to input darrays, in the list of its outputs.

## *hpdglm() Peripheral Functions*

You can use these peripheral functions related to the output model learned by hpdglm.

- **Summary** - This function prints a summary of the learned model.
- **Coefficients** - This function prints coefficients of the learned model. The abbreviated function name is `coef`.
- **Residuals** - This function extracts model residuals in a darray. The abbreviated function name is `resid`.
- **Predict** - This function produces predicted values, obtained by evaluating the regression function on provided new data. New data can be either a darray or a normal matrix. You can get details on the function by typing `help(hpdglm.predict)` at the R prompt.

## *See Also*

For more details about the function interface and its input arguments, type `help(hpdglm)` at the R prompt or see the HPdregression Package Manual.

# v.hpdglm

Evaluates a model built by hpdglm() using the Split-Sample-Validation method. A percentage of the data is randomly selected for testing, and a new model is built using the remaining data. The result is a measurement of the prediction cost of the new model on the test data.

## *See Also*

For more details about the function interface and its input arguments, type `help(v.hpdglm)` at the R prompt or see the HPdregression Package Manual.

# cv.hpdglm

Evaluates a model built by hpdglm() using the Cross-Validation method. The data is randomly divided into K folds, and the evaluation is repeated K times. For every iteration, one fold is reserved as the test data, and the rest is used for training. Finally, the function aggregates the prediction costs of the new models on all folds.

## *See Also*

For more details about the function interface and its input arguments, type `help(cv.hpdglm)` at the R prompt or see the HPdregression Package Manual.

# HPdregression Examples

## *Linear Regression*

This linear regression example uses a small data-frame called "faithful", which is available in the standard distribution of R (package datasets). For more details in the data set run `help(faithful)` in R. This example details how to build a model to predict value of `faithful$eruptions`, given the value of `faithful$waiting`, using three partitions in each darray. The data is partitioned into three because the example system has three workers.

1. Load the required libraries and start Distributed R:

```
> library(HPdregression)
> distributedR_start()
```

2. Create darrays for the response and predictors:

```
> Y <- as.darray (data.matrix(faithful["eruptions"]))
> X <- as.darray (data.matrix(faithful["waiting"]))
```

3. Build the model:

```
> myModel <- hpdglm(responses=Y, predictors=X, family=gaussian
(link=identity),completeModel = TRUE )
```

   You can also build the model with the shorthand version of the command:

```
> myModel <- hpdglm(Y, X, completeModel = TRUE)
```

4. Print the summary:

```
> summary(myModel)


Call:
hpdglm(responses = Y, predictors = X, family = gaussian(link = identity),
    completeModel = TRUE)
```

```
Deviance Residuals:
   Min     Max
-1.299   1.193

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.874016   0.160143  -11.70   <2e-16 ***
waiting      0.075628   0.002219   34.09   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.2465251)

    Null deviance: 353.039  on 271  degrees of freedom
Residual deviance:  66.562  on 270  degrees of freedom
AIC: 395.02

Number of Fisher Scoring iterations: 2
```

5. Predict the response for a few new samples:

```
> newSamples = matrix(c(51:60),,1)
> predict(myModel, newSamples, "link")
```

The results of the predict function are displayed:

```
Calculating link prediction
          [,1]
 [1,] 1.983009
 [2,] 2.058637
 [3,] 2.134265
 [4,] 2.209893
 [5,] 2.285521
 [6,] 2.361149
 [7,] 2.436777
 [8,] 2.512405
 [9,] 2.588033
[10,] 2.663661
```

6. Run split-sample validation:

**Note:** By default, output values of type darray are dropped so the models can easily be deployed to HPE Vertica. To use the validation functions you must create a complete model (`completedModel=TRUE`).

```
> testV <- v.hpdglm (Y, X, myModel)
> testV$delta
```

```
[1] 0.2431476 0.2429429
```

7. Run cross-validation:

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta

[1] 0.2444747 0.2442814
```

## *Logistic Regression*

This logistic regression example uses a small data-frame called "mtcars", which is available in the standard distribution of R (package datasets). This example details how to build a model to predict the value of `mtcars$am` (whether the car has an automatic transmission), given the value of `mtcars$wt` and `mtcars$hp` (weight and horsepower) using three partitions in each darray.

1. Load the required libraries and start Distributed R:

   **Note:** Libraries only need to be loaded once per session. If you are running this example in the same session as the previous example then you do not need to load the libraries or start Distributed R again.

```
> library(HPdregression)
> distributedR_start()
```

2. Create a darray for response and predictors:

```
> Y <- as.darray (data.matrix(mtcars["am"]))
> X <- as.darray (data.matrix(mtcars[c("wt","hp")]))
```

3. Build the model:

```
> myModel <- hpdglm(responses=Y, predictors=X, family= binomial(logit), completeModel
= TRUE)
```

   You can also build the model with the shorthand version of the command:

```
> myModel <- hpdglm(Y, X, binomial, completeModel = TRUE)
```

4. Print the summary:

```
> summary(myModel)

Call:
hpdglm(responses = Y, predictors = X, family = binomial(logit),
    completeModel = TRUE)

Deviance Residuals:
   Min     Max
-2.254   1.345

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 18.86630    7.44356   2.535  0.01126 *
wt          -8.08348    3.06868  -2.634  0.00843 **
hp           0.03626    0.01773   2.044  0.04091 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.230  on 31  degrees of freedom
Residual deviance: 10.059  on 29  degrees of freedom
AIC: 16.059

Number of Fisher Scoring iterations: 8
```

5. Predict the response for some new samples:

```
> newSamples <- matrix(c(1:3),3,2)
> predict(myModel, newSamples, "response")
```

The results of the predict function are displayed:

```
            [,1]
[1,] 0.999979986
[2,] 0.941136088
[3,] 0.005090073
```

6. Run split-sample validation:

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta

[1] 0.06726468 0.06574563
```

7. Run cross-validation:

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta
```

```
[1] 0.04554999 0.04348628
```

## *Poisson Regression*

This example for Poisson regression uses the same data set used for the logistic regression example.

1. Load the required libraries and start Distributed R:

   **Note:** Libraries only need to be loaded once per session. If you are running this example in the same session as the previous example then you do not need to load the libraries or start Distributed R again.

   ```
   > library(HPdregression)
   > distributedR_start()
   ```

2. Create a darray for response and predictors:

   ```
   > Y <- as.darray (data.matrix(mtcars["am"]))
   > X <- as.darray (data.matrix(mtcars[c("wt","hp")]))
   ```

3. Build the model:

   ```
   > myModel <- hpdglm(Y, X, poisson, completeModel=TRUE)
   ```

4. Print the summary:

   ```
   > summary(myModel)

   Call:
   hpdglm(responses = Y, predictors = X, family = poisson, completeModel = TRUE)

   Deviance Residuals:
      Min     Max
   -1.049   1.060

   Coefficients:
               Estimate Std. Error z value Pr(>|z|)
   (Intercept)  2.287139   0.907534   2.520  0.01173 *
   wt          -1.580433   0.494010  -3.199  0.00138 **
   hp           0.008831   0.005259   1.679  0.09309 .
   ---
   ```

```
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 23.420  on 31  degrees of freedom
Residual deviance: 10.481  on 29  degrees of freedom
AIC: 42.481

Number of Fisher Scoring iterations: 6
```

5. Predict the response for some new samples:

```
> newSamples <- matrix(c(1:3),3, 2 )
> predict(myModel, newSamples, "response")
```

The results of the predict function are displayed:

```
           [,1]
[1,] 2.04528411
[2,] 0.42483043
[3,] 0.08824246
```

6. Run split-sample validation:

```
> testV <- v.hpdglm(Y, X, myModel)
> testV$delta

[1] 0.158504 0.170527
```

7. Run cross-validation:

```
> testCV <- cv.hpdglm(Y, X, myModel)
> testCV$delta

[1] 0.1478461 0.1058880
```

# About the HPdclassifier Package

HPdclassifier is a package for distributed classifier algorithms based on the Distributed R infrastructure.

The main functions of this package include:

| Function | Description |
|---|---|
| hpdRF_parallelTree() | Alias is hpdrandomForest(). Runs randomForest using a distributed approach that utilizes parallelism in creating each tree of the forest. This is an improved version of hpdRF_parallelForest(). |
| hpdRF_parallelForest() | Runs randomForest using a distributed approach that utilizes parallelism in creating sub-forests of the forest. |
| predictHPdRF() | Provides distributed predict method for applying a random forest object on a darray or a dframe. |
| confusionMatrix() | Generates a confusion matrix for a given predictive model and test data. |
| errorRate() | Calculates the total error rate and error rates of each class for a given predictive model and test data. |
| meanSquared() | Calculates the mean squared residuals for a given regression model and test data. |
| rSquared() | Calculates the R-squared (1 - mse / Var(y)) for a given regression model and test data. |
| hpdegbm() | A distributed ensemble implementation of the R gbm package for regression and classification problems |
| predict.hpdegbm() | Provides a distributed data and centralized model predict function for hpdegbm. |
| hpdrpart() | Combines standard R rpart with a simple hybrid approach to create a single tree in parallel. |
| predict.hpdrpart() | Makes a prediction based on a previously-computed hpdrpart model. |
| varImportance() | Calculates the importance of a variable, given a trained model and a set of test data. |

# hpdRF_parallelTree

A distributed function for randomForest that utilizes parallelism in creating each tree of the forest. It is an improved version of the distributed randomForest algorithm released in Distributed R 1.0. The 1.0 version of hpdrandomForest relied upon having the entire dataset at each executor. This new implementation partitions the dataset by observations and distributes a segment to each executor.

This approach reduces communication among executors and allows for more efficient processing on larger datasets.

The input data set of hpdRF_parallelTree can be ordinary R objects similar to randomForest, such as matrix and data frame. However, the size of these objects should not exceed 2 GB. For bigger datasets use darray or dframe. The darray structure does not support categorical data, but it is more memory-efficient in comparison to dframe. Therefore, Distributed R highly recommends using darray when there is no categorical data.

Note that predictors and responses must align with each other for variables xtest and ytest (have the same number of rows and the same number of blocks in the case of a dframe) because row names are not used for alignment.

**Note:** The 1.0 version of `hpdrandomForest()` is still available. It has been renamed to hpdRF_parallelForest. The function name hpdrandomForest() is re-aliased to the new version of distributed randomForest (hpdRF_parallelTree) in version 1.1 and later of Distributed R.

## Estimating hpdRF_parallelTree Memory Requirements

In general, `hpdRF_parallelTree()` requires an amount of memory equal to twice that of the input data when the number of columns (features) are small in comparison to the number of rows (samples).

When you run the predict() function on a model built using hpdRF_parallelTree(), if *completeModel* is set to TRUE, the following memory size is required of your system (in bytes):

```
nrows * ntree * 8
```

If *completeModel* is set to FALSE, this memory recommendation does not need to be taken into account.

## Data Structure Selection

Distributed R recommends the following data structures to use, depending on data requirements:

| Input Data Size | Response Type | Structure to Use |
|---|---|---|
| < 2 GB | Any | Non-distributed |
| > 2 GB | Non-categorical | darray |
| > 2 GB | Categorical | dframe |

## hpdRF_parallelTree Example

```
> iris.hpdRF_pt <- hpdRF_parallelTree(Species ~ ., data=iris)
> predicted <- predict(iris.hpdRF_pt,iris[, -5])
```

```
> confusionMatrix(iris[,5], predicted)

            predicted
observed     setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         50         0
  virginica       0          0        50
```

## *See Also*

For more detail about the function interface and its input arguments, type `help(hpdRF_parallelTree)` at the R prompt or see the HPdclassifier Package Manual.

# hpdRF_parallelForest

**Note:** In Distributed R 1.0 hpdRF_parallelForest() was named `hpdrandomForest()`. In Distributed R version 1.1 and later, hprandomForest() is re-aliased to `hpdRF_parallelTree()`. `hpdRF_parallelTree()` is an improved version of `hpdRF_parallelForest()`.

A distributed version of the R randomForest() function, which is available in the randomForest package. hpdRF_parallelForest() calls several instances of randomForest distributed across your Distributed R cluster. The master node distributes the input data among all R-executors of the Distributed R environment. Trees in different subsections of the forest are created simultaneously.

At the end of the process, all trees are combined into a single forest result. Because every R-executor should have access to all samples, the function is bounded by the physical memory of a single node of the cluster. Thus, hpdRF_parallelForest does not scale in terms of memory usage but instead speeds up randomForest because different parts of the forest are computed in parallel.

hpdRF_parallelForest supports two interfaces similar to randomForest. The input arguments for these interfaces are similar, but a few additional arguments are introduced and several are omitted. The output of hpdRF_parallelForest is compatible with output of randomForest.

The input data set of hpdRF_parallelForest can be ordinary R objects similar to randomForest, such as matrix and data frame. However, the size of these objects should not exceed 2 GB. For bigger datasets use darray or dframe. The darray structure does not support categorical data, but it is more memory-efficient in comparison to dframe. Therefore, Distributed R highly recommends using darray when there is no categorical data.

## *Estimating hpdRF_parallelForest Memory Requirements*

The hpdRF_parallelForest function internally calls several instances of the randomForest function for calculating sub-forests of the entire desired forest and then combines them. The randomForest and combine functions are not part of Distributed R; so, it is not easy to propose a formula for their memory requirements. However, you can roughly infer that the memory requirement is a function of the number of trees, the number of features, and the number of samples. You can estimate the memory requirement of running the hpdRF_parallelForest function in comparison to running a single randomForest function.

Assuming the following:

- `inputM`: memory utilization of the input data
- `rfM`: memory utilization of the randomForest function at its running time
- `nExecutor`: the input argument which determines the number of involved concurrent executions
- `nWorker`: The number of workers (Distributed R nodes) involved in execution of the hpdRF_parallelForest function

Your randomForest memory estimation is different depending on which type of input data you use. These formulas provide the total bytes in memory required across all nodes in the cluster.

- **Input Data is of type matrix or data.frame**: (ordinary R objects): In this scenario all the input data are sent to all the R-executors for concurrent calculating of the sub-forests. Note that the size of the input data cannot be bigger than 2GB, which is a limitation of ordinary R objects. There is no shared memory among the instances of a worker for ordinary R objects. Therefore, the total memory consumption can be estimated using the formula:

  ```
  (inputM + rfM) * nExecutor
  ```

- **Input data is of type darray**: This scenario can be used either for regression or unsupervised learning. The input data should be available for all the concurrently running randomForest functions; however, shared memory is available for darray data structures among the instances of the same worker. Therefore, the total memory consumption can be estimated with the formula:

  ```
  inputM * nWorker + rfM * nExecutor
  ```

- **Input data is of type dframe**: This scenario can be used either for classification or unsupervised learning. The input data should be available for all the concurrently running randomForest functions, and shared memory is not available for dframe data structure among the instances of the same worker. Therefore, the total memory consumption can be estimated with the formula:

  ```
  (inputM + rfM) * nExecutor
  ```

## Data Structure Selection

Distributed R recommends the following data structures to use, depending on data requirements:

| Input Data Size | Response Type | Structure to Use |
|---|---|---|
| < 2 GB | Any | Non-distributed |
| > 2 GB | Non-categorical | darray |
| > 2 GB | Categorical | dframe |

## See Also

For more detail about the function interface and its input arguments, type `help(hprandomForest)` at the R prompt or see the HPdclassifier Package Manual.

## hpdRF_parallelForest Examples

### Classification Example with data frame

This classification example uses a small data frame called ‘iris’, which is available in the standard distribution of R (datasets package). This example determines the Species based on the other variables in the data set.

1. Load the required libraries and start Distributed R:

   **Note:** Libraries only need to be loaded once per session.

   ```
   > library(HPdclassifier)
   > distributedR_start()
   ```

2. Run `hpdRF_parallelForest()` on the iris data set:

   ```
   > iris.rf = hpdRF_parallelForest(Species ~ ., data=iris, importance=TRUE,
   proximity=TRUE, completeModel=TRUE)
   ```

3. Display details about the model:

   ```
   > iris.rf

   Call:
    hpdRF_parallelForest(formula = Species ~ ., data = iris, importance = TRUE,
   proximity = TRUE, completeModel = TRUE)
                  Type of random forest: classification
                        Number of trees: 500
   No. of variables tried at each split: 2

           OOB estimate of  error rate: 4.67%
   Confusion matrix:
              setosa versicolor virginica class.error
   setosa         50          0         0        0.00
   versicolor      0         47         3        0.06
   virginica       0          5        45        0.10
   ```

4. Finally, because hpdRF_parallelForest creates a model that is compatible with randomForest, you can use the randomForest `predict()` function on the hpdRF_parallelForest model:

   ```
   > iris.pred = predict(iris.rf, iris)
   > table(iris$Species, as.factor(iris.pred))

               setosa versicolor virginica
     setosa         50          0         0
     versicolor      0         50         0
     virginica       0          0        50
   ```

   **Note:** The prediction in this example is not realistic because the "training" and "testing" sets are identical.

## *Example 2*

This example is similar to Example 1, but instead uses dframe as the type of input data set. Use the dframe data structure only for big datasets.

1. Load the required libraries and start Distributed R:

   **Note:** Libraries only need to be loaded once per session.

   ```
   > library(HPdclassifier)
   > distributedR_start()
   ```

2. Get Distributed R status and compute the number of instances in the cluster:

   ```
   > ds = distributedR_status()
   > nparts = sum(ds$Inst)
   ```

3. Define the number of samples, attributes per sample, and the number of splits:

   ```
   > nSamples = 100
   > nAttributes = 5
   > nSplits = 1
   ```

4. Create the darrays:

   ```
   > dax = darray(c(nSamples, nAttributes), c(round(nSamples/nSplits),nAttributes))
   > day = darray(c(nSamples,1), c(round(nSamples/nSplits),1))
   ```

5. Load the darrays:

   ```
   > foreach(i, 1:npartitions(dax), function(x=splits(dax,i),y=splits(day,i),id=i){
       x = matrix(runif(nrow(x) * ncol(x)), nrow(x), ncol(x))
       y = matrix(runif(nrow(y)), nrow(y), 1)
       update(x)
       update(y)
   })
   ```

6. Run `hpdRF_parallelForest()`:

   ```
   > myrf = hpdRF_parallelForest(dax, day)
   ```

7. Run the distributed predict function:

```
> dp = predictHPdRF(myrf, dax)
```

# hpdrpart

Creates a single tree in parallel. The model that hpdrpart() outputs is compatible with the auxiliary functions in the rpart package.

Use hpdrpart() when the training data set is too large for a single node and must be split into partitions over a cluster. During the model-building process, a single hpdrpart model builds, using all of the individual partitions. After building the hpdrpart model, you can make predictions on the model using testing data.

If your testing data is a distributed data object, then prediction occurs on each partition of the testing data in parallel. This parallel approach allows for improved execution performance over a sequential approach. If the testing data does not take the form of a distributed data object, the prediction occurs sequentially on the master node.

## *Syntax*

```
testing_data <- hpdrpart( formula, data, weights, na.action = <action>, control = <control>,
completeModel = { TRUE | FALSE }, nBins = <nBins>, do.trace = { TRUE | FALSE } )
```

## *Arguments*

The interface of hpdrpart is similar to rpart. However, hpdrpart adds additional arguments.

| Argument | Description |
|---|---|
| *formula* | [Required] A formula that describes the model to be fitted. |
| *data* | [Required] A data.frame or dframe that contains the input variables in the model. |
| *weights* | [Optional] A data.frame or dframe that contains the weights to be used. |
| *na.action* | [Optional] A function that specifies the action to take if NA values are found. |
| *control* | [Optional] Supported values are:<br>• control$minsplit<br>• control$minbucket<br>• control$maxdepth<br>• control$cp |

| Argument | Description |
|---|---|
| *completeModel* | [Optional] When set to FALSE, the output values that preserve information for each sample are discarded. This argument also discards any other variable that is calculated during the post-processing step. Using this option can reduce the size of your output model.<br><br>When set to TRUE, the output contains a complete set of values.<br><br>**Default value**: FALSE |
| *nBins* | [Optional] Determines the number of bins to use for numerical variables. The number of bins for categorical variables equals the number of categories for the variable.<br><br>**Default value**: 256 |
| *do.trace* | [Optional] When set to TRUE, gives a more verbose output as the function runs.<br><br>**Default value**: FALSE |

## See Also

Type `help(hpdrpart)` in the R console for additional details or see the HPdclassifier Manual.

## hpdrpart Examples

This classification example shows you how to determine the Species variable based on the other variables in the data set. The example uses a small data frame, called *iris*, which is available in the standard distribution of R (datasets package).

1. Load the required libraries, and start Distributed R:

   **Note:** Libraries need to be loaded only once per session.

   ```
   > library(HPdclassifier)
   > distributedR_start()
   ```

2. Run `hpdrpart()` on the iris data set:

   ```
   > iris.rpart <- hpdrpart(Species ~ ., data = iris, completeModel = TRUE)
   ```

3. Display details about the model:

```
> iris.rpart
n=0 ()

node), split, n, deviance, yval
      * denotes terminal node

1) root 0 0.66666670 1
  2) Petal.Width< 0.6035294 0 0.00000000 1 *
  3) Petal.Width>=0.6035294 0 0.50000000 2
    6) Petal.Width< 1.704706 0 0.16803840 2 *
    7) Petal.Width>=1.704706 0 0.04253308 3 *
```

4. Because hpdrpart creates a model that is compatible with rpart, you can use the rpart `predict()` function on the hpdrpart model:

```
> iris.pred <- predict(iris.rpart, iris, type = "class")
> table(iris$Species, iris.pred$predictions)

             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          5        45
```

# varImportance

Calculates the importance of a variable, given a trained model and a set of test data. A generic function, `varImportance()` calculates permutation-based variable importance on a given model.

The function permutes the observed values of a chosen variable or variables, creating a new data set in the process. As a result of the permutation, the new data set has contradictory and, essentially, meaningless values of the chosen variables added. Using a trained model, you can obtain predictions on this new, altered data, and compare it with the original predictions on the unaltered data.

Following the permutation process, the predictions for the altered data are, by design, less accurate than the predictions from the unaltered data. You can, however, determine the importance of a particular variable by comparing the two sets of data. To do so, observe how inaccurate the predictions are from the altered data, and then compare them to the unaltered data. The less accurate your predictions are, after a certain feature is altered, the more importance that feature has in the original data set.

For the most accurate results and best performance, Hewlett Packard Enterprise recommends that you use the `varImportance()` function only with models built from the following functions:

- randomForest()
- hpdRF_parallelTree()
- hpdRF_parallelForest()

- rpart()
- hpdrpart()

## Syntax

```
varImportance( model, xtest, ytest, ..., distance_metric, trace = { TRUE | FALSE }, type =
"class" )
```

## Arguments

| Arguments | Description |
|---|---|
| *model* | [Required] The trained machine learning model. |
| *xtest* | [Required] A data object containing the predictors for the model. |
| *ytest* | [Required] A data.frame or dframe containing the response variable. |
| *distance_metric* | [Optional] A function that compares the loss in accuracy between predictions.<br>**Default**:<br>• For classification models, the default is the error rate.<br>• For regression models, the default is the mean squared error. |
| *trace* | [Optional] When set to TRUE, displays the progress of the function.<br>**Default**: FALSE |
| *type* | Required only for models built with rpart() or hpdrpart(). |

## Examples

This example demonstrates how to run the `varImportance()` function on a model built using `hpdrpart()`:

```
> library(HPdclassifier)
> distributedR_start()

# Load the iris data set.
> data(iris)

# Build the hpdrpart model.
> model <- hpdrpart(Species ~ ., data = iris, completeModel = TRUE)

# Set xtest to the model predictors.
> xtest <- iris
```

```
# Remove the Species variable, because it is the variable to predict.
> xtest$Species <- NULL

# Create a data frame containing the response variables.
> ytest <- data.frame(iris$Species)

# Run varImportance on the model and two data frames.
> varImp.data.frame <- varImportance(model, xtest, ytest, type = "class")

# Run varImportance on the model and two dframes.
> varImp.dframe <- varImportance(model, as.dframe(xtest), as.dframe(ytest), type =
"class")

# View the output of varImp.data.frame.
> varImp.data.frame
             Importance
Sepal.Length    0.00000
Sepal.Width     0.00000
Petal.Length    0.00000
Petal.Width    17.16216

# View the output of varImp.dframe.
> varImp.dframe
             Importance
Sepal.Length    0.00000
Sepal.Width     0.00000
Petal.Length    0.00000
Petal.Width    16.45333
```

## *See Also*

Type `help(varImportance)` in the R console for additional details or see the HPdclassifier Manual.

# hpdegbm

A distributed ensemble implementation of the R gbm package for regression and classification problems.

Using hpdegbm() allows you to build several GBM models in parallel, each of which is trained using the gbm.fit() function. Within Distributed R, the master node distributes the input data among all of the R executors in the Distributed R environment, and GBM models are created simultaneously. Ultimately, all of these GBM models are combined to formulate an ensemble GBM model.

You can, additionally, make distributed predictions using the predict.hpdegbm() function. When you use predict.hpdegbm(), your test data must be a dframe or darray.

The result returned by the hpdegm() function is a list of GBM models which are compatible with the result of gbm.fit.

# Algorithms

There are two algorithms implemented in the hpdegbm function. Depending on the size of the training data set, hpdegbm automatically calls one of the following algorithms:

- **For small data sets** — When the training data set is small and stored as a matrix or data frame, the entire data set is loaded into each core. Each executor trains one GBM model in parallel and in a distributed manner. Because bagging is applied in gbm.fit, multiple different GBM models can result, even though the same training data is applied to every executor.
- **For large data sets** — When the training data set is large and stored as a distributed data object, such as a dframe or darray, each data partition is loaded into each executor. The *X_train* argument, containing the predictor variables, and the *Y_train* argument, containing the response variables, must have the same number of partitions and the same number of rows in each corresponding partition. The data partitions in each executor are then used to train a GBM model.

In both cases, the trained GBM models from all of the executors are combined into an ensemble on the master node to create the distributed GBM object.

# Syntax

```
finalModel <- hpdegbm( X_train = X_train, Y_train = Y_train, nExecutor = <nExecutor>,
distribution = <distribution_method>,
ntrees = <ntrees>, interaction.depth = <interaction_depth>, m.minobsinnode = <value>,
shrinkage = <value>,
bag.fraction = <value>, samplingFlag = { TRUE | FALSE }, sampleThresh = <sample_threshold>,
trace = { TRUE | FALSE }, ...  )
```

# Arguments

The interface of hpdegbm is similar to gbm.fit. However, hpdegbm uses additional arguments.

| Argument | Description |
|---|---|
| *X_train* | A data frame, dframe, darray, or matrix containing samples of predictor variables. |
| *Y_train* | A data frame, dframe, darray, or matrix containing a vector of outputs. |
| *nExecutor* | The number of GBM models in the ensemble to train in parallel.<br>If *X_train* and *Y_train* are distributed objects, the default value is the number of partitions in *X_train*.<br>If *X_train* and *Y_train* are not distributed objects, the default value is the total number of logical cores in the Distributed R cluster. |

| Argument | Description |
|---|---|
| *distribution* | Supported values are:<br>• Gaussian — Distribution for regression<br>• Bernoulli — Distribution for binary classification<br>• AdaBoost — Distribution for binary classification<br>• Multinomial — Distribution for multi-class classification |
| *n.trees* | The total number of decision trees to fit. |
| *interaction.depth* | The maximum depth of variable interactions. |
| *m.minobsinnode* | The minimum number of observations in the terminal nodes of the decision trees. |
| *shrinkage* | An argument applied to each tree in the model's expansion (learning rate). |
| *bag.fraction* | The fraction of the training set observations, which are randomly selected for the next tree in the expansion. |
| *samplingFlag* | Determines whether or not to perform distributed sampling before training ensemble models.<br>If *X_train* and *Y_train* are distributed objects, default value is TRUE.<br>If *X_train* and *Y_train* are not distributed objects, this argument is ignored. |
| *sampleThresh* | The planning factor for the sample size. Only valid when *samplingFlag* = TRUE. |
| *trace* | If set to TRUE, hpdegbm prints its progress. |

# *Examples*

The following example demonstrates how you can test the classification accuracy of a model built and boosted with hpdegbm(). In this example, you use multinomial distribution.

1. Load the required packages and start Distributed R:

```
> library(HPdclassifier)
> distributedR_start()
```

2. Separate the iris data set into training and testing data:

```
> newiris <- iris
```

```
> irisX <- newiris[which(names(newiris) != "Species")]
> irisY <- as.character(newiris$Species)

# Set a variable for the percentage of the data we want to use as training data.
> trainPerc <- 0.8

# Sample 80% of the training data.
> trainIdx <- sample(1:nrow(irisX), ceiling(trainPerc * nrow(irisX)))

# Generate the standard training and testing data.
> irisX_train <- irisX[trainIdx,]
> irisY_train <- irisY[trainIdx]
> irisX_test <- irisX[-trainIdx,]
> irisY_test <- irisY[-trainIdx]

# Generate distributed versions of the training and testing data.
> dirisX_train <- as.dframe(irisX_train)
> dirisY_train <- as.dframe(as.data.frame(irisY_train))
> dirisX_test <- as.dframe(irisX_test)
> dirisY_test <- as.dframe(as.data.frame(irisY_test))
```

3. Run hpdgebm on both the standard and the distributed testing and training data:

```
# Run hpdsample on the standard training and testing data.
> model <- hpdegbm(irisX_train, irisY_train, distribution = 'multinomial', nExecutor
= 6)

# Run hpdsample on the distributed training and testing data.
> dmodel <- hpdegbm(dirisX_train, dirisY_train, distribution = 'multinomial',
nExecutor = 6)
```

4. Run the predict function on both sets of sampled data:

```
# Run the predict function on the standard sampled data.
> pmodel <- predict(model, irisX_test)

# Run the predict function on the distributed sampled data.
> pdmodel <- predict(dmodel, irisX_test)
```

## *See Also*

Type `help(hpdegbm)` in the R console for additional details or see the HPdclassifier Manual.

# hpdsample

Performs distributed sampling of input data from a darray, dframe, or both, to formulate a random output data. The output data appears in a new dframe or darray.

You can sample input data from either one distributed object or two.

- **If you supply only a single distributed object** — The function outputs a distributed object of the same type as the input distributed object.
- **If you provide a second distributed object** — hpdsample samples both objects simultaneously and outputs a list with the following elements:
  - sdata1 — Contains the data sampled from the first distributed object
  - sdata2 — Contains the data sampled from the second distributed object

# Syntax

```
 sampling_data <- hpdsample( distdata1, distdata2, nSamplePartitions = <value>, samplingRatio
= <value>, trace = { TRUE | FALSE } )
```

# Arguments

| Argument | Description |
|---|---|
| *distdata1* | [Required] The input variables in a dframe or darray. The dframe or darray must be row-wise partitioned. |
| *distdata2* | [Optional] The input variables in a second dframe or darray. This second dframe or darray must have the same number of rows and partitions as *data1* and must also be row-wise partitioned.<br>The hpdsample function assumes that each row of *data2* corresponds to the same row of *data1*. |
| *nSamplePartitions* | [Required] The number of output partitions for the sample data. |
| *samplingRatio* | [Required] The percentage of data from *data1* and, if applicable, *data2*, to be sampled in each output partition. Must be a positive value of type float or integer. |
| *trace* | [Optional] When set to TRUE, hpdsample prints its progress. |

# Examples

The following example demonstrates how you can use hpdsample on a simulated set of data in a darray and a dframe.

1. Load the required packages and start Distributed R:

```
> library(HPdclassifier)
> distributedR_start()
```

2. Generate simulated data for a darray, distdata1, and a dframe, distdata2:

```
> distdata1 <- as.darray(as.matrix(1:1001))

> distdata2 <- as.dframe(data.frame(a = rep('c', 1001), b = 1:1001, c = runif(1001),
d = c(rep('a', 501), rep('b', 500))))
```

3. Run hpdsample and store the results in sampledData:

```
> sampledData <- hpdsample(distdata1, distdata2, nSamplePartitions = 4, samplingRatio
= 0.1)
```

sampledData, a list, contains two elements. The first element, sdata1, is a darray with four partitions. Each of the four partitions contains 1001 * 0.1 elements that are sampled from distdata1. The second element, sdata2, is a dframe with four partitions, each of which also contains 1001 * 0.1 elements.

4. Separate sampledData into two elements, sampledData1 and sampledData2:

```
> sampledData1 <- sampledData$sdata1
> sampledData2 <- sampledData$sdata2
```

## *See Also*

Type `help(hpdsample)` in the R console for additional details or see the HPdclassifier manual.

# confusionMatrix

```
confusionMatrix(observed, predicted)
```

Returns a confusion matrix based on observed and predicted values.

## *Example*

```
> library(HPdclassifier)
> distributedR_start()
> rRF <- randomForest(Species ~ ., data=iris, keep.forest=TRUE,
    xtest=iris[,-5],ytest=iris[,5])

> predicted <- predict(rRF,iris[, -5])
> confusionMatrix(iris[,5], predicted)


            predicted
observed     setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         50         0
  virginica       0          0        50
```

## *See Also*

Type `help(confusionMatrix)` in the R console for additional details or see the HPdclassifier Manual.

# errorRate

```
errorRate(observed, predicted)
```

Returns an array. The first element of the array is the error rate, which equals to the total number of correct predictions divided by the total number of predictions. The remaining elements of the array represent error rates per class. An error rate per class is the error rate for the samples with a particular category in their response.

## *Example*

```
> library(HPdclassifier)
> distributedR_start()
> rRF <- randomForest(Species ~ ., data=iris, keep.forest=TRUE,
    xtest=iris[,-5],ytest=iris[,5])

> predicted <- predict(rRF,iris[, -5])
> errorRate(iris[,5], predicted)
  err.rate     setosa versicolor  virginica
         0          0          0          0
```

## *See Also*

Type `help(errorRate)` in the R console for additional details or see the HPdclassifier Manual.

# meanSquared

```
meanSquared(observed, predicted, na.rm=FALSE)
```

Returns the mean squared of residuals.

## *Example*

```
> library(HPdclassifier)
> distributedR_start()
> testData <- na.omit(airquality)
> rRF.ozone <- randomForest(Ozone ~ ., data=airquality,
         mtry=3, na.action=na.omit,
         xtest=testData[,-1],ytest=testData[,1],keep.forest=TRUE)
```

```
> predicted <- predict(rRF.ozone,testData[,-1])
> meanSquared(testData[,1], predicted)
[1] 65.62805
```

## See Also

Type `help(meanSquared)` in the R console for additional details or see the HPdclassifier Manual.

# rSquared

```
rSquared(observed, predicted, na.rm=FALSE)
```

Returns the value of R-squared of the test set.

## Example

```
> library(HPdclassifier)
> distributedR_start()
> testData <- na.omit(airquality)
> rRF.ozone <- randomForest(Ozone ~ ., data=airquality,
         mtry=3, na.action=na.omit,
         xtest=testData[,-1],ytest=testData[,1],keep.forest=TRUE)

> predicted <- predict(rRF.ozone,testData[,-1])
> rSquared(testData[,1], predicted)
[1] 0.9407309
```

## See Also

Type `help(rSquared)` in the R console for additional details or see the HPdclassifier Manual.

# About the HPdcluster Package

HPdcluster is a package for distributed clustering, based on the k-means algorithm available in R. The package contains the distributed kmeans function `hpdkmeans()`. The HPdcluster package includes:

| Function | Description |
|---|---|
| `hpdkmeans()` | A distributed alternative for the R kmeans function. |

# hpdkmeans

A distributed version of the kmeans() function, which is available in the stats package of R. The syntax of hpdkmeans is designed to be very similar to kmeans, but the input samples must be in darray format. hpdkmeans currently supports only the Lloyd algorithm.

The kmeans algorithm result depends upon the initial cluster centers you specify because:

- The heuristic algorithm may trap in a local minimum, based on the start point.
- Based on the start point, a different number of iterations might be required for convergence.
- Even if you have the samples categorized in the same clusters, the order of the clusters and, therefore, their labels are likely to be different.

hpdkmeans and kmeans are the same in this behavior. hpdkmeans accepts only numerical values. Moreover, it can exclude the samples with invalid values; such as, NA, NaN, or Inf.

## *Estimating hpdkmeans Memory Requirements*

You can estimate the memory requirements for hpdkmeans using the following formula.

Memory requirements are based on the size of the darray being analyzed and the number of centers. Given a darray named *X*, assume the following:

- *m* = nrow(*X*)
- *p* = ncol(*X*)
- *p* = ncol(*centers*)

  **Note:** ncol(*X*) always equals ncol(*centers*) when using k-means.

- *k* = nrow(*centers*)
- Each data element also consumes 8 bytes for the size of the element.
- The number of blocks in the darray = the number of R executors.
- *m* is considerably bigger than *k* * *p*

To approximate the total required memory (including the memory which is allocated for the input darray) use the formula:

```
(p + 3) * m * 8
```

The value returned is the amount of total available memory (in bytes) required across all cluster nodes.

# *Troubleshooting hpdkmeans()*

These troubleshooting tips provide detail on several important warning or error messages that may appear after you run the function.

**Why does the warning message: “did not converge in 10 iterations” appear?**

After each iteration, hpdkmeans compares the newly calculated set of centers with the old one. When there is no update in the centers, the algorithm is considered converged. If the function reaches the maximum number of iterations before convergence, it issues this warning message. The required number of iterations for convergence and the quality of the result are very closely dependent on the set of initial centers.

When there is such a warning, look at the value of *withinss* and other returned parameters that indicate the quality of returned clusters. Decide whether the result is good enough or not. If you think that clusters are in a good shape, but that a few more iterations can make the centers more accurate, use the founded centers as the initial centers of another call to the function. Thus, the time spent on the previous run is not wasted.

**How can nstart argument help to achieve better results?**

When nstart is bigger than 1, the algorithm is executed several times, and the best result is returned. Therefore, some of the runs may not converge, which means they issue a warning, but they may not be the one which is returned. The best result is the one with highest value of withinss regardless of its number of iterations. check the value of iter to see the number of iterations related to the returned result.

**Why does the warning message "empty cluster: try a better set of initial centers" appear?**

Sometimes, especially when the initial centers are not well selected, a cluster becomes empty (contains no data points) after one or more iterations. In such situations, this warning message is issued because an empty cluster is not very meaningful. Decide if you want to execute the algorithm again. Based on the nature of the data, you may choose to repeat the algorithm with the same number of centers but different values or even use a different number of centers.

**Why does the error message “unsuccessfully tried '10' times to randomly pick district centers. Please specify centers manually” appear?**

The current implementation of hpdkmeans mimics kmeans function for selecting the initial centers. They both sample the data points based on uniform probability distribution, which means that all samples have an equal chance to be picked as a center. Although all the centers should be unique, duplicate data points can occur in the data set. To address this problem, kmeans first refines the whole data set to its distinct data points and then samples the centers. However, you cannot apply this approach efficiently to a huge data set stored in a darray. Instead, hpdkmeans simply reruns the sampling when there are is any duplication in the centers. Failure to randomly find a minimum of 10 distinct centers in a large number of rows indicates a problem with duplicate data points in the data set.

## *See Also*

The complete information about the syntax of hpdkmeans can be found in the HPdcluster Package Manual or by typing `help(hpdkmeans)` in the R console.

# hpdkmeans Example

This topic provides an example of the common usage of hpdkmeans. Refer to the HPdcluster Package Manual (or type `help(hpdkmeans)` in the R console) for complete information about the names and meanings of the arguments.

## *Clustering Example*

The clustering example uses a small data-frame called 'iris', which is available in the standard distribution of R (datasets package). The data-frame is loaded into a darray because hpdkmeans requires a darray as input.

1. Load the required libraries and start Distributed R.

   **Note:** Libraries only need to be loaded once per session.

   ```
   > library(HPdcluster)
   > distributedR_start()
   ```

2. Get Distributed R status and compute the number of instances in the cluster:

   ```
   > ds = distributedR_status()
   > nparts = sum(ds$Inst)
   ```

3. Load the iris data into a darray. Only the numeric values are loaded. The fifth column in the data-frame, Species, is not included.

   ```
   > X = as.darray(data.matrix(iris[c("Sepal.Length", "Sepal.Width", "Petal.Length",
   "Petal.Width")]))
   ```

4. Run `hpdkmeans()` on the darray, finding three clusters:

   ```
   > dkm = hpdkmeans(X,centers=3)
   ```

5. Compare the clusters to the Species labels in the original iris data frame by using the `hpdapply()` function. You must use the getpartition() function because the data returned from `hpdapply()` is a darray:

```
> table(iris$Species, getpartition(hpdapply(X, dkm$centers)))

             1  2  3
  setosa     50  0  0
  versicolor  0 47  3
  virginica   0 14 36
```

You can also use the completeModel=TRUE argument to populate the cluster component of the model. However, the cluster component is also a darray which requires you to use `getpartition()`. By default, the cluster component is not populated to reduce the model size.

```
> dkm = hpdkmeans(X,centers=3, completeModel=TRUE)
> table(iris$Species, getpartition(dkm$cluster))

             1  2  3
  setosa     50  0  0
  versicolor  0 47  3
  virginica   0 14 36
```

# About the HPdgraph Package

The HPdgraph package provides distributed algorithms for graph analytics in Distributed R. This package includes two main functions:

| Function | Description |
|---|---|
| `hpdpagerank()` | A distributed function for computing pagerank vector of a graph. |
| `hpdwhich.max()` | Finds and returns the index of the maximum value stored in a darray. |

# hpdpagerank

A distributed implementation of the pagerank algorithm for directed graphs. The Power Method is used to calculate the pagerank vector of a graph. Self-loops are allowed, but multiple edges are not considered. The input graph should be in the adjacency matrix format, stored in a darray, and column-wise partitioned.

## *Estimating hpdpagerank Memory Requirements*

hpdpagerank() has three inputs of type darray:

- **dgraph** (required), can be a dense or sparse.
- **personalized** (optional), should always be dense.
- **weights** (optional), can be dense or sparse.

Assume that:

- `n = |V|` (the number of vertices in the graph)
- `m = |E|` (the number of edges in the graph)

You can estimate the memory required for the input and output arrays using the following tables:

**Input darrays**

| Type | Dimensions | Size Approximation (in bytes) |
|---|---|---|
| dgraph | `n * n` | `16*m` for sparse and `8*n^2` for dense |
| personalized | `1 * n` | `8*n` |
| weights | `n * n` | `16*m` for sparse and `8*n^2` for dense |

**Output darray**

| Type | Dimensions | Size Approximation (in bytes) |
|---|---|---|
| pr | `1 * n` | `8*n` |

Considering the size of the input and output darrays, the minimum memory requirement for running hdpagerank can be approximated by the following formulas. In these formulas:

- *k* is the number of workers used for running the algorithm
- *w* and *p* are 1 or 0 for existence or non-existence of 'weights' and 'personalized':
  - *w* = 1 when there is a weighted darray; otherwise *w* = 0
  - *p* = 1 when there is a personalized darray; otherwise *p* = 0
- *e* = 1 when the na_action="exclude"; otherwise *e* = 0 (default)

Memory requirement in bytes when dgraph and weight are sparse darrays:

```
16 * m + 16 * m * w + 8 * n * p + 16 * m * e + 32 * n * k + 16 * n
```

Memory requirement in bytes when dgraph and weight are dense arrays. The following case of using dense arrays is unusual for big graphs. Additionally, note that `db2dgraph()` and `file2dgraph()` never provide a dense darray:

```
8 * n^2 + 8 * n^2 * w + 8 * n * p + 8 * n^2 * e + 32 * n * k + 16 * n
```

When dgraph and weight are dense darrays, and you store their elements as integers, then their memory consumption almost halves. Therefore, the memory requirement for dense arrays can be decreased to:

```
4 * n^2 + 4 * n^2 * w + 8 * n * p + 4 * n^2 * e + 32 * n * k + 16 * n
```

The calculated numbers are approximations of the total required memory (among all the nodes) for running the algorithm excluding the memory consumed by the platform. It is always safer to consider the largest use case with some additional overhead. Moreover, the needed memory per node when dgraph is sparse totally depends on the distribution of the edges among different partitions of the darray. When the distribution is very skewed, a single node may go out-of-memory while the total memory on cluster nodes is enough; that is, the memory utilization might become very imbalanced.

## *Memory Estimation Example*

Assume that a dgraph is a sparse darray representing a graph with `10^9` vertices and `10^10` edges where the distribution of edges among the vertices is almost uniform. There is no weight or personalized darrays, and 4 workers are used for running the algorithm. Also consider that na_action="pass" (default).

- n = 10^8
- m = 10^9
- k = 4
- w = 0
- p = 0
- e = 0

Using the first estimation formula (using sparse arrays), total memory required (in bytes) is:

```
16 * 10^9 + 16 * 10^9 * 0 + 8 * 10^8 * 0 + 16 * 10^9 * 0 + 32 * 10^8 * 4 + 16 * 10^8 =
(160 + 128 + 16) * 10^8
```

Which equals approximately 30.4 GB of memory. Only with the assumption that we have a uniform distribution of the edges (which is not always the case), we can divide this number by 4 (the number of workers) to calculate the memory requirement by node. After adding a safety margin, you can conclude that about 8 GB free memory per node is enough for this example.

## *hdpagerank Example*

As an example, use the real-world graph of the LiveJournal social network, which can be downloaded from http://snap.stanford.edu/data/soc-LiveJournal1.html. This graph has 4,847,571 vertices and 68,993,773 edges.

Before running the example, use gunzip to uncompress the downloaded file as a text file. Assume that the text file is stored in the path "/graph/soc-LiveJournal1.txt" and that you want to run hpdpagerank on two nodes with 12 cores each. Therefore, you need to split the input graph to 24 parts as shown in the following figure. The file "cluster2.xml" is used to configure the cluster specification. More information about this configuration file can be found in the Distributed R documentation. Also, more information about "splitGraphFile" and "file2graph" functions can be found in the HPdata Package Manual.

1. Download the graph file to your Distributed R master node and unzip it. Note that the unzipped file is roughly 1GB in size.

```
$ wget http://snap.stanford.edu/data/soc-LiveJournal1.txt.gz
$ gunzip soc-LiveJournal1.txt.gz
```

2. In R, load the required libraries, and then start Distributed R:

**Note:** Libraries only need to be loaded once per session.

```
> library(HPdgraph)
> library(HPdata)
> distributedR_start()
```

3. Using `splitGraphFile()` from the HPdata package, split the file amongst the nodes in the cluster. This example assumes the graph file is in `/home/dbadmin/graph` on the master node and that ALL nodes (including the master) have a directory named `/home/dbadmin/graph/temp` :

```
> ret <- splitGraphFile(inputFile="/home/dbadmin/graph/soc-LiveJournal1.txt",
outputPath="/home/dbadmin/graph/temp", npartitions=24)
```

Wait as the graph file is split and sent to the nodes in the cluster:

```
Sending the files to node01
```

```
Sending the files to node02
Sending the files to node03
```

4. Load the distributed files into a distributed graph using the `file2dgraph()` function from the HPdata package:

```
> dg <- file2dgraph(ret$pathPrefix, ret$nVertices, ret$verticesInSplit,
ret$isWeighted)
Opening files:
 /home/dbadmin/graph/temp/soc-LiveJournal1.txt0
...until...
/home/dbadmin/graph/temp/soc-LiveJournal1.txt23
progress: 100%
```

5. Calculate dimensions, edges, pagerank, and highest pagerank value:

```
> dim(dg$X) # dimensions of the adjacency matrix
[1] 4847571 4847571


> sum(dg$X) # number of edges
[1] 68993773

> pr <-hpdpagerank(dg$X)

> dim(pr) # the dimensions of the pagerank vector
[1]       1 4847571

> hpdwhich.max(pr) # the vertex with highest pagerank value
[1] 8738
```

## *See Also*

More details about the interface of the function and its input arguments can be found in the HPdgraph Package Manual or by typing `help(hpdpagerank)` in the R console.

# hpdwhich.max

A distributed version of the which.max function for a 1D-array that has darray as its input argument. As shown in the example below, you can use this function for finding the vertex with highest rank in the pagerank vector, which is returned by `hpdpagerank()` function.

```
> dim(dg$X) # dimensions of the adjacency matrix
[1] 4847571 4847571

> sum(dg$X) # number of edges
[1] 68993773

> pr <- hpdpagerank(dg$X)
```

```
> dim(pr) # the dimensions of the pagerank vector
[1]       1 4847571

> hpdwhich.max(pr) # the vertex with highest pagerank value
[1] 8738
```

## *See Also*

More details about the interface of the function and its input arguments can be found in the HPdgraph Package Manual or by typing `help(hpdwhich.max)` in the R console.

# About the HPdata Package

The HPdata package contains data-related functions for the Distributed R environment. You can use these functions for data loading, data preparation, and so on.

**Note:** If any HPE Vertica nodes go down while data is being loaded by these packages, then the data load fails and Distributed R returns an error.

| Function | Description |
|---|---|
| `db2darray()` | Loads a set of samples stored in a table to a darray. |
| `db2darrays()` | Loads a set of samples stored in a table to two darrays. |
| `db2matrix()` | Loads a set of samples stored in a table to a matrix. |
| `db2dframe()` | Loads a set of samples stored in a table to a dframe. |
| `db2dframes()` | Loads a set of samples stored in a table to two dframes. |
| `db2dgraph()` | Loads an adjacency matrix to a darray from an edgelist stored in a database. |
| `splitGraphFile()` | Splits an edgelist file of a graph and distributes the results among the active nodes of a cluster system. |
| `file2dgraph()` | Loads an adjacency matrix to a darray from an edgelist stored in a set of files. |
| `csv2dframe()` | Loads a CSV file into a dframe. |
| `orc2dframe()` | Loads an ORC file into a dframe. |
| `object2hdfs()` | Writes an object from Distributed R to HDFS. |

# Configuring the HDFS Configuration File

To write to or read from HDFS using Distributed R, you must first configure the HDFS configuration file.

The `dataconnector` package includes the HDFS configuration file. This file determines the ports, username, and host when loading from or writing files to HDFS. Within the directory where you installed the `dataconnector` package, the file's default location is */conf/hdfs.json*.

The following sample shows a typical HDFS configuration file:

```
{
    "webhdfsPort": 50070,
    "hdfsPort": 9000,
    "hdfsHost": "127.0.0.1",
```

```
    "hdfsUser": "dbadmin"
}
```

## *Parameters*

The following table describes the parameters that you can configure in the HDFS configuration file:

| Parameters | Description |
|---|---|
| webhdfsPort | The HTTP port where the HDFS NameNode web UI listens. Must be of type integer. |
| hdfsPort | The HDFS NameNode port. Must be of type integer. |
| hdfsHost | The HDFS NameNode host. Must be of type string. |
| hdfsUser | The HDFS user. Must be of type string. |

# db2darray

Loads a set of samples stored in a table to a darray.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into Distributed R using this function.

This example loads data from a single table in an HPE Vertica database using the Distributed R HPE Vertica Connector. If you are loading the data using vRODBC, then the table must also contain a column of integer numbers called rowid. The values of this column must start from 0 and be continuous (there is no missed rowid number in the table). The npartitions argument specifies the desired number of partitions in the darray. When not specified, the default value for npartitions is the number of active R executors in the Distributed R environment.

You can use the optional npartitions argument to specify the number of splits (partitions) in the distributed array. The number of partitions returned is close to the specified number, but may not be exactly the same. HPdata determines the final number of partitions by:

1. Invoking rowsInBlock <- ceiling(nobs/npartitions), where nobs is the number of observations (number of rows) in the table and npartitions is the specified number of splits.

2. Using the value of rowsInBlock As to build the distributed data structure.

In this case, the final number of partitions is ceiling (nobs/rowsInBlock), which might differ from npartitions.

**Important:** A warning message displays the selected number of splits when it differs from the specified number.

You can omit columns you don't want to load by using the `except` argument. For example, say that you have a table, "mortgage", with the following columns: *rowid*, *def*, *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*. The DSN used is "VerticaDSN". To load all of the columns except for *agespline2*, run the following command:

```
> loadedSamples <- db2darray ("mortgage", "VerticaDSN", except=list("agespline2"))
```

# *Examples*

This example shows how to load a set of samples stored in a table to a darray. The data is stored in a table named mortgage and the configuration for the database connection is *VerticaDSN*. Use six columns to load the samples to a darray: *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*.

1. Create a table for mortgage details:

   ```
   => CREATE TABLE mortgage (rowid INT, def INT, mltvspline1 FLOAT, mltvspline2 FLOAT,
   agespline1 FLOAT, agespline2 FLOAT, hpichgspline FLOAT, ficospline FLOAT);
   ```

2. Add some sample data:

   ```
   INSERT INTO mortgage VALUES(0, 1, 0.760777, 0.006632, 0.948052, 0.906403, 0.058021,
   0.960328);
   INSERT INTO mortgage VALUES(1, 0, 0.135741, 0.205449, 0.516031, 0.013455, 0.827438,
   0.659125);
   INSERT INTO mortgage VALUES(2, 0, 0.021796, 0.138996, 0.862165, 0.034211, 0.150524,
   0.345917);
   INSERT INTO mortgage VALUES(3, 1, 0.271257, 0.543280, 0.940978, 0.891880, 0.993050,
   0.000160);
   INSERT INTO mortgage VALUES(4, 1, 0.986207, 0.053896, 0.119611, 0.646744, 0.819753,
   0.663289);
   ```

   The values in the database table look like this:

| rowid | def | mltvspline1 | mltvspline2 | agespline1 | agespline2 | hpichgspline | ficospline |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.760777 | 0.006632 | 0.948052 | 0.906403 | 0.058021 | 0.960328 |
| 1 | 0 | 0.135741 | 0.205449 | 0.516031 | 0.013455 | 0.827438 | 0.659125 |
| 2 | 0 | 0.021796 | 0.138996 | 0.862165 | 0.034211 | 0.150524 | 0.345917 |
| 3 | 1 | 0.271257 | 0.543280 | 0.940978 | 0.891880 | 0.993050 | 0.000160 |
| 4 | 1 | 0.986207 | 0.053896 | 0.119611 | 0.646744 | 0.819753 | 0.663289 |

Use the following command to load the data from this table into Distributed R:

```
> loadedSamples <- db2darray ("mortgage", "VerticaDSN", list("mltvspline1",
```

```
"mltvspline2",
"agespline1", "agespline2", "hpichgspline", "ficospline"))
```

The darray for samples becomes available as *loadedSamples*.

```
Loading required package: vRODBC
Loading total 5 rows from mortgage from Vertica with approximate partition of 1 rows
progress: 100%
```

You can view the contents of the darray with the `getpartition()` function:

```
> getpartition(loadedSamples)

     mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
[1,]    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
[2,]    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
[3,]    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
[4,]    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
[5,]    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
```

## *See Also*

Details on the function are available in the HPdata Package Manual or by typing `?db2darray()` at the R prompt.

# db2darrays

Loads a data set from a table in HPE Vertica to a pair of Distributed R darrays. The two darrays correspond to responses and predictors of a predictive model. The samples stored in the database (including responses and predictors) must be stored in a single table. If you are not using the Distributed R HPE Vertica Connector, then the table must contain a column called rowid. The column rowid must start from 0 and be continuous across all values in the column. If you have a table with five rows, the rowid would contain the values 0, 1, 2, 3, 4, as the following example shows. Similar to db2darray, an optional input argument, npartitions, allows you to specify the number of splits.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into Distributed R using this function.

HPdata determines the final number of partitions by:

1. Invoking rowsInBlock <- ceiling(nobs/npartitions), where nobs is the number of observations (number of rows) in the table and npartitions is the specified number of splits.

2. Using the value of rowsInBlock As to build the distributed data structure.

In this case, ceiling(nobs/rowsInBlock) determines the number of rows in each partition; therefore, the number of partitions in the result might differ from the specified one.

**Important:** A warning message displays the selected number of splits when it differs from the specified number.

You can omit columns you don't want to load by using the `except` argument. For example, say that you have a table, "mortgage", with the following columns: *rowid*, *def*, *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*. The DSN used is "VerticaDSN". To load all of the columns except for *agespline2*, run the following command:

```
> loadedSamples <- db2darrays ("mortgage", "VerticaDSN", list("def"), except=list
("agespline2"))
```

# *Examples*

This example shows how to load a set of samples stored in a table to a pair of darrays which correspond to response and predictors of a predictive model. The data is stored in a table named mortgage, and the name of the response column is *def*. The names of the predictive columns are *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*.

The values in the database table look like this:

| rowid | def | mltvspline1 | mltvspline2 | agespline1 | agespline2 | hpichgspline | ficospline |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.760777 | 0.006632 | 0.948052 | 0.906403 | 0.058021 | 0.960328 |
| 1 | 0 | 0.135741 | 0.205449 | 0.516031 | 0.013455 | 0.827438 | 0.659125 |
| 2 | 0 | 0.021796 | 0.138996 | 0.862165 | 0.034211 | 0.150524 | 0.345917 |
| 3 | 1 | 0.271257 | 0.543280 | 0.940978 | 0.891880 | 0.993050 | 0.000160 |
| 4 | 1 | 0.986207 | 0.053896 | 0.119611 | 0.646744 | 0.819753 | 0.663289 |

Use the following command to load all the samples stored in the table to a pair of darrays with two partitions:

```
> loadedData <- db2darrays ("mortgage", "VerticaDSN", list("def"),list("mltvspline1",
 "mltvspline2", "agespline1", "agespline2", "hpichgspline", "ficospline"))
```

The darrays for the data become available as *loadedData*.

```
Loading required package: vRODBC
Loading total 5 rows from mortgage from Vertica with approximate partition of 1 rows
progress: 100%
```

The darrays for response and predictors are `loadedData$Y` and `loadedData$X`, respectively. The number of blocks and their size are exactly the same in both darrays. In fact, each row in Y corresponds to the response of a row in X with exactly the same row position. Moreover, blocks are row-wise partitioned.

```
> names(loadedData)
[1] "Y" "X"

> getpartition(loadedData$Y)

     def
[1,]   1
[2,]   1
[3,]   0
[4,]   1
[5,]   0

> getpartition(loadedData$X)

     mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
[1,]    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
[2,]    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
[3,]    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
[4,]    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
[5,]    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
```

**Note:** The rows of X and Y with the same vertical position correspond to the values of a single row of the database table. Nevertheless, the order of the samples in the output darrays is not necessarily the same as their order in the table.

## *See Also*

For more details about db2arrays and its input arguments, see the HPdata Package Manual or type `?db2darrays()` at the R prompt.

# db2dframe

Loads a set of samples stored in a table to a dframe.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into Distributed R using this function.

Data must be stored in a single table. If you are not using the Distributed R HPE Vertica Connector then the table must contain a column called rowid. rowid must start at 0 and be contiguous.

Similar to db2darray, there is an optional input argument called npartitions to specify the number of splits you want.

HPdata determines the final number of partitions by:

1. Invoking rowsInBlock <- ceiling(nobs/npartitions), where nobs is the number of observations (number of rows) in the table and npartitions is the specified number of splits.

2. Using the value of rowsInBlock As to build the distributed data structure.

In this case, ceiling(nobs/rowsInBlock) determines the number of rows in each partition; therefore, the number of partitions in the result might differ from the specified one.

**Important:** A warning message displays the selected number of splits when it differs from the specified number.

You can omit columns you don't want to load by using the `except` argument. For example, say that you have a table, "mortgage", with the following columns: *rowid*, *def*, *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*. The DSN used is "VerticaDSN". To load all of the columns except for *agespline2*, run the following command:

```
> loadedSamples <- db2dframe ("mortgage", "VerticaDSN", except=list("agespline2"))
```

## *Examples*

This example shows how to load a set of samples stored in a table to a dframe. The db2dframe function stores data in a dframe data structure instead of a darray. Consequently, it can store also categorical data. Let's consider a table in a database similar to the previous examples, however, this table contains the strings "*Yes*" and "*No*" for the value of its *def* column.

1. Create a table for mortgage details:

```
=> CREATE TABLE mortgage (rowid INT, def INT, mltvspline1 FLOAT, mltvspline2 FLOAT,
agespline1 FLOAT, agespline2 FLOAT, hpichgspline FLOAT, ficospline FLOAT);
```

2. Add some sample data:

```
INSERT INTO mortgage VALUES(0, 'Yes', 0.760777, 0.006632, 0.948052, 0.906403,
0.058021, 0.960328);
INSERT INTO mortgage VALUES(1, 'No', 0.135741, 0.205449, 0.516031, 0.013455,
0.827438, 0.659125);
INSERT INTO mortgage VALUES(2, 'No', 0.021796, 0.138996, 0.862165, 0.034211,
0.150524, 0.345917);
INSERT INTO mortgage VALUES(3, 'Yes', 0.271257, 0.543280, 0.940978, 0.891880,
0.993050, 0.000160);
INSERT INTO mortgage VALUES(4, 'Yes', 0.986207, 0.053896, 0.119611, 0.646744,
0.819753, 0.663289);
```

The table in the database looks like this:

| rowid | def | mltvspline1 | mltvspline2 | agespline1 | agespline2 | hpichgspline | ficospline |
|---|---|---|---|---|---|---|---|
| 0 | Yes | 0.760777 | 0.006632 | 0.948052 | 0.906403 | 0.058021 | 0.960328 |
| 1 | No | 0.135741 | 0.205449 | 0.516031 | 0.013455 | 0.827438 | 0.659125 |
| 2 | No | 0.021796 | 0.138996 | 0.862165 | 0.034211 | 0.150524 | 0.345917 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | Yes | 0.271257 | 0.543280 | 0.940978 | 0.891880 | 0.993050 | 0.000160 |
| 4 | Yes | 0.986207 | 0.053896 | 0.119611 | 0.646744 | 0.819753 | 0.663289 |

Load the data into the *loadedSamples* dframe with the command:

```
> loadedSamples <- db2dframe ("mortgage", "VerticaDSN", list("def", "mltvspline1",
"mltvspline2", "agespline1", "agespline2", "hpichgspline", "ficospline"))

Loading total 5 rows from mortgage from Vertica with approximate partition of 1 rows
progress: 100%
```

You can view the data in the dframe with the `getpartition()`function:

```
> getpartition(loadedSamples)

    def   mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
1   Yes    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
2   Yes    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
3   No     0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
4   No     0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
5   Yes    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
```

## *See Also*

For more details about `db2dframe()` and its input arguments, see the HPdata Package Manual or type `help(db2dframe)` at the R prompt.

# db2dframes

Loads a data set from a table in HPE Vertica to a pair of Distributed R dframes which correspond to responses and predictors of a predictive model. The samples stored in the database (including responses and predictors) must reside in a single table. If you are not using the Distributed R HPE Vertica Connector, then the table must contain a column called rowid. The column rowid must start from 0 and be continuous across all values in the column. For example, if you have a table with five rows, the rowid would contain the values 0, 1, 2, 3, 4. Similar to db2frame, an optional input argument, npartitions, allows you to specify the number of splits.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into Distributed R using this function.

HPdata determines the final number of partitions through a two-step process:

1. Invoking the command `rowsInBlock <- ceiling(nobs/npartitions)`, where nobs is the number of observations (number of rows) in the table and npartitions is the specified number of

splits.

2. Using the value of rowsInBlock As to build the distributed data structure.

In this case, ceiling(nobs/rowsInBlock) determines the number of rows in each partition. Therefore, the number of partitions in the result might differ from the numbered specified.

**Important:** A warning message displays the selected number of splits when it differs from the specified number.

If you have columns you do not want to load, you can omit thems by using the `except` argument. For example, suppose you have a table, "mortgage", with the following columns: *rowid*, *def*, *mltvspline1*, *mltvspline2*, *agespline1*, *agespline2*, *hpichgspline*, and *ficospline*. The DSN used is "VerticaDSN". To load all of the columns except for *agespline2*, run the following command:

```
> loadedSamples <- db2dframes ("mortgage", "VerticaDSN", list("def"), except=list
("agespline2"))
```

# *Examples*

This example shows how you can load a set of samples store in a table to a pair of dframes. This pair of dframes corresponds to responses and predictors of a predictive model. The db2dframes function stores data in a dframe data structure instead of a darray. Consequently, the function can store categorical data. Suppose you have a table in a database similar to the ones in the previous examples. However, this table differs because it contains the strings "*Yes*" and "*No*" for the value of its *def* column.

This database contains the following table:

| rowid | def | mltvspline1 | mltvspline2 | agespline1 | agespline2 | hpichgspline | ficospline |
|---|---|---|---|---|---|---|---|
| 0 | Yes | 0.760777 | 0.006632 | 0.948052 | 0.906403 | 0.058021 | 0.960328 |
| 1 | No | 0.135741 | 0.205449 | 0.516031 | 0.013455 | 0.827438 | 0.659125 |
| 2 | No | 0.021796 | 0.138996 | 0.862165 | 0.034211 | 0.150524 | 0.345917 |
| 3 | Yes | 0.271257 | 0.543280 | 0.940978 | 0.891880 | 0.993050 | 0.000160 |
| 4 | Yes | 0.986207 | 0.053896 | 0.119611 | 0.646744 | 0.819753 | 0.663289 |

Use the following command to load all the samples stored in the table to a pair of dframes with two partitions:

```
> loadedSamples <- db2dframes ("mortgage", "VerticaDSN", list("def"), list("mltvspline1",
"mltvspline2", "agespline1", "agespline2", "hpichgspline", "ficospline"))
```

The dframes for the data become available as *loadedSamples*.

```
Loading total 5 rows from mortgage from Vertica with approximate partition of 1 rows
progress: 100%
```

The dframes for response and predictors are `loadedSamples$Y` and `loadedData$X`, respectively. The number of blocks and their size are exactly the same in both dframes. In fact, each row in Y corresponds to the response of a row in X with exactly the same row position. Moreover, blocks are row-wise partitioned.

```
> names(loadedSamples)
[1] "Y" "X"

> getpartition(loadedSamples$Y)

  def
1  No
2 Yes
3 Yes
4  No
5 Yes

> getpartition(loadedSamples$X)

  mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
1    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
2    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
3    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
4    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
5    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
```

**Note:** The rows of X and Y with the same vertical position correspond to the values of a single row of the database table. However, the order of the samples in the output dframes is not necessarily the same as their order in the table.

## *See Also*

For more details about `db2dframes()` and its input arguments, see the HPdata Package Manual or type `help(db2dframes)` at the R prompt.

# db2dgraph

Loads an adjacency matrix from a table of edge-lists stored in a database to a pair of sparse darrays. Because the returned darray is column-wise partitioned by default, you can use this function most efficiently when there is projection (index) on the 'to' column of the table. Similar to db2darray, an optional input argument, npartitions, allows you to specify the number of splits (partitions) in the darray. Therefore, the number of splits in the result might differ from the requested number.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into

Distributed R using this function.

The data being loaded must be stored in a single table and the ID of vertices in the table must start at zero. The maximum ID number is used to find the number of vertices in the graph. This function assumes that all missed ID numbers between the first ID (0) and the maximum ID vertices do not have any connected edge.

You can partition the returned darrays by rows, rather than columns, by adding the `row_wise = TRUE` argument:

```
> dg <- db2dgraph (tableName="wgraph", dsn="VerticaDSN", from="x", to="y", weight="w",
row_wise=TRUE)
```

## *Example: Load an Adjacency Matrix to a darray from an Edgelist Stored in a Database*

Assume that there is a weighted graph stored in the *wgraph* table of the database like below.

| x | y | w |
|---|---|---|
| 8 | 7 | 2.5 |
| 2 | 6 | 3.1 |
| 3 | 5 | 1.2 |
| 4 | 7 | 2.2 |

```
> dg <- db2dgraph (tableName="wgraph", dsn="VerticaDSN", from="x", to="y", weight="w")

progress: 100%

> getpartition(dg$X)
[1] . . . . . . . . .
[2] . . . . . . . . .
[3] . . . . . . 1 . .
[4] . . . . . 1 . . .
[5] . . . . . . . 1 .
[6] . . . . . . . . .
[7] . . . . . . . . .
[8] . . . . . . . . .
[9] . . . . . . . 1 .

> getpartition(dg$W)
[1] . . . . . .   .   .   .
[2] . . . . . .   .   .   .
[3] . . . . . .   3.1 .   .
[4] . . . . . 1.2 .   .   .
[5] . . . . . .   .   2.2 .
[6] . . . . . .   .   .   .
[7] . . . . . .   .   .   .
[8] . . . . . .   .   .   .
```

```
[9] . . . . . .   .   2.5 .
```

## See Also

For more details about `db2dgraph()` and its input arguments, see the HPdata Package Manual or type `help(db2dgraph)` at the R prompt.

# db2matrix

Loads a set of unlabeled samples stored in a table to a normal R matrix. The data must be stored in a single table. All the rows of the table are read and sampled.

**Important:** You can use this function only to load regular user tables from HPE Vertica. System tables and other non-user tables such, as data collector tables, cannot be loaded into Distributed R using this function.

## Example: Load a Set of Unlabeled Samples Stored in a Table to a Matrix

Assume that two samples are stored in a table called mortgage, and the table has six columns corresponding to different features of the samples. Each row of this table is read as a sample.

The values in the database table look like this:

| rowid | def | mltvspline1 | mltvspline2 | agespline1 | agespline2 | hpichgspline | ficospline |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0.760777 | 0.006632 | 0.948052 | 0.906403 | 0.058021 | 0.960328 |
| 1 | 0 | 0.135741 | 0.205449 | 0.516031 | 0.013455 | 0.827438 | 0.659125 |
| 2 | 0 | 0.021796 | 0.138996 | 0.862165 | 0.034211 | 0.150524 | 0.345917 |
| 3 | 1 | 0.271257 | 0.543280 | 0.940978 | 0.891880 | 0.993050 | 0.000160 |
| 4 | 1 | 0.986207 | 0.053896 | 0.119611 | 0.646744 | 0.819753 | 0.663289 |

```
> mortgage <- db2matrix("mortgage", "VerticaDSN", features=list("def", "mltvspline1",
 "mltvspline2", "agespline1", "agespline2", "hpichgspline", "ficospline"))

> print(mortgage)
  def mltvspline1 mltvspline2 agespline1 agespline2 hpichgspline ficospline
1   0    0.021796    0.138996   0.862165   0.034211     0.150524   0.345917
2   1    0.271257    0.543280   0.940978   0.891880     0.993050   0.000160
3   1    0.986207    0.053896   0.119611   0.646744     0.819753   0.663289
4   1    0.760777    0.006632   0.948052   0.906403     0.058021   0.960328
5   0    0.135741    0.205449   0.516031   0.013455     0.827438   0.659125
```

## *See Also*

Details on the function are available in the HPdata Package Manual or by typing `?db2matrix()` at the R prompt.

# splitGraphFile

Splits an edge list file of a graph, and distributes the results among the active nodes of a cluster system. You can use the result of this function to create a darray for adjacency matrix using the file2dgraph function. The input edgelist must be zero-based (the id of the first edge is zero). This function calculates the total number of vertices according to the maximum vertex id found in the input file. For any missed vertex ID, such as a vertex with no connected edge, a dummy vertex is substituted.

The function first creates a temporary folder at the same path of the input file for generating split files, and then sends the files to the nodes of the clusters where on each an active worker is available in the Distributed R environment. To call this function, you must have read and write access to both the input path and the output path.

The number of files returned will be close to the number specified number, but probably not exactly the same.

1. Calculate the number of vertices in each split file as verticesInSplit <- ceiling (nVertices/npartitions) where nVertices is the number of vertices in the input graph.
2. Based on this result, determine the value of nFiles <- ceiling(nVertices/verticesInSplit). This value might differ from the input npartitions.
3. Index the files from 0 to (nFiles – 1).

You can partition the output by rows, rather than columns, by adding the `row_wise = TRUE` argument. Otherwise, the returned output is partitioned column-wise by default. The following example demonstrates how to add the `row_wise = TRUE` argument:

```
> ret <- splitGraphFile (inputFile="/home/dbadmin/GraphSource/small.txt",
outputPath="/graphSplits", npartitions=2, isNFS=TRUE, row_wise=TRUE)
```

## *Example: Split an Edgelist file and Distribute Results Among the Active Nodes on a Cluster*

Assume that a file, small.txt, contains an edge list of a small directed graph and is located in the local address: `/home/dbadmin/graphSource`. Also assume that you want to split this file into 2 partitions and store the resulted files in an NFS directory with this path: "/graphSplits". The names of the split files created are small.txt0 and small.txt1. The content of these files is displayed below.

a) small.txt

```
# The original
# small graph
0 1
1 2
2 6
2 7
3 4
3 6
5 3
5 4
6 7
7 1
```

b) small.txt0

```
0 1
1 2
5 3
7 1
```

c) small.txt1

```
2 6
2 7
3 4
3 6
5 4
6 7
```

```
> ret <- splitGraphFile (inputFile= “/home/dbadmin/GraphSource/small.txt”,
outputPath= ”/graphSplits”, npartitions=2, isNFS=TRUE )

> ret

$pathPrefix
[1] ”/graphSplits/small.txt”

$nVertices
[1] 8

$verticesInSplit
[1] 4

$nFiles
[1] 2

$isWeighted
[1] FALSE
```

## *See Also*

Details on the function are available in the HPdata Package Manual or by typing `?splitGraphFile()` at the R prompt.

# file2dgraph

Loads an adjacency matrix from an edge list that is split into a set of files. The function assumes that the edge list is properly split among the nFiles. Split files should correspond to a column-wise partitioned adjacency matrix. Each file is used to load one partition of such a matrix. The files should be available from the same path from all the nodes. Therefore, they must be located either on NFS or on the same path in different nodes of the cluster. Set the *isWeighted* input argument according to the graph stored in the files.

You can partition the output by rows, rather than columns, by adding the `row_wise = TRUE` argument. Otherwise, the returned output is partitioned column-wise by default. The following example demonstrates how to add the `row_wise = TRUE` argument:

```
> dg <- file2dgraph (pathPrefix=ret$pathPrefix, nVertices=ret$nVertices,
verticesInSplit =ret$verticesInSplit, isWeighted=ret$isWeighted, row_wise=TRUE)
```

## *Example: Load an Adjacency Matrix to a darray from an Edgelist Stored in a File Set*

You can use the output of the splitGraphFile function as the input arguments for the file2dgraph function. Dimensions of the resulted darray equal the number of vertices. Also, the sum of the darray elements equals the number of edges because each edge has an element of value 1.

```
> dg <- file2dgraph (pathPrefix=ret$pathPrefix, nVertices=ret$nVertices,
verticesInSplit =ret$verticesInSplit, isWeighted=ret$isWeighted)

Openning files:
/graphSplits/small.txt0
... until ...
/graphSplits/small.txt1
Progress: 100%

> getpartition(dg$X)
8 * 8 sparse Matrix of class “dgCMatrix”

[1,] . 1 . . . . . .
[2 ,] . . 1 . . . . .
[3 ,] . . . . . . 1 1
[4 ,] . . . . 1 . 1 .
[5 ,] . . . . . . . .
[6 ,] . . . 1 1 . . .
[7 ,] . . . . . . . 1
[8 ,] . 1 . . . . . .

> dg$W
NULL

> dim(dg$X)
[1] 8 8

> sum(dg$X)
```

```
[1] 10
```

## See Also

Details on the function are available in the HPdata Package Manual or by typing `?file2dgraph()` at the R prompt.

# csv2dframe

Loads a CSV file into a dframe.

Before using this function, you must edit the HDFS configuration file as described in Configuring the HDFS Configuration File.

- **When loading a single CSV file —** The file splits by lines and loads in parallel. As it loads, the file splits into a number of individual chunks equal to the number of executors.
- **When loading multiple CSV files —** The files load in parallel into a distributed data structure. The data splits among the executors into as many partitions as there are files.

## Syntax

```
CSV2dframe (url = 'file_location'
[, schema = '<col0_name>:<col0_type>,<col1_name>:<col1_type>...<colN_name>:<colN_type>' ]
[, filetype = <filetype> ]
[, delimiter = <delimiter> ]
[, hdfsconfiguration = '<hdfsconfiguration>' ]
[, commentcharacter = <commentcharacter> ]
)
```

## Arguments

| Arguments | Description |
|---|---|
| *url* | [Required] Location of the file or files to be loaded.<br><br>The `csv2dframe` function supports the use of wildcard (*) characters in place of file names. |

| Arguments | Description |
|---|---|
| *schema* | [Optional] Format of the columns in the CSV file. Supported types are:<br>• int64<br>• integer<br>• logical<br>• double<br>• character<br>The syntax for the argument is `'<colN_name>:<colN_type>'`. The label for the column can be either:<br>• The name of the column in the CSV file<br>• The index number of the column (index origin = 0)<br>**Example:**<br>To load the first column in a CSV file, with type int64, enter the following: `schema='0:int64'`. |
| *filetype* | [Optional] The file extension automatically determines the fileType argument. You can use this argument when the file you are loading does not have an extension or if you want to override the extension. |
| *delimiter* | [Optional] Column separator value for the CSV file.<br>**Default Value:** ',' |
| *hdfsconfigurationfile* | [Optional] The configuration file used to load data from or write data to HDFS.<br>**Default Value:** paste(system.file (package='ddc'),'/conf/hdfs.json',sep='') |
| *commentcharacter* | [Optional] Character used to determine where comments begin. |

# *Examples*

The following example demonstrates how you can load a CSV file from HDFS:

```
> library(HPdata)
> distributedR_start()
> mydframe <- csv2dframe(url='hdfs:///tmp/file.csv', schema='0:int64,1:string',
delimiter=',')
```

The following example demonstrates how you can load a CSV file from the local file system:

```
> library(HPdata)
> distributedR_start()
> mydframe <- csv2dframe(url='file:///tmp/file.csv', schema='age:int,salary:int',
delimiter=',')
```

The csv2dframe function also supports wildcards (*) when selecting CSV files. When using wildcards, all CSV files must have the same schema and delimiter. For example, you can use a wildcard to specify that the query return all files in the `/tmp` directory with the .csv file extension:

```
> library(HPdata)
> distributedR_start()
> mydframe <- csv2dframe(url='hdfs:///tmp/*.csv', schema='0:int64,1:string')
```

## *See Also*

For further details, see the HPdata Package Manual, or type `?csv2dframe()` at the R prompt.

# orc2dframe

Loads an ORC file into a dframe.

Before using this function, you must edit the HDFS configuration file as described in Configuring the HDFS Configuration File.

- **When loading a single ORC file —** The file splits by stripes and loads in parallel. As it loads, the file splits up into a number of individual chunks equal to the number of executors.
- **When loading multiple ORC files —** The files load in parallel into a distributed data structure. The data splits into partitions among the executors into as many partitions as the total number of ORC stripes.

## *Syntax*

```
ORC2dframe (url = 'file_location'
[, filetype = <file extension> ]
[, selectedstripes = <stripes to include> ]
[, hdfsconfiguration = '<hdfsconfiguration>' ]
[, commentcharacter = <commentcharacter> ]
)
```

## *Arguments*

| Arguments | Description |
|---|---|
| *url* | [Required] Location of the file or files to be loaded.<br><br>The `orc2dframe` function supports the use of wildcard (*) characters in place of file names. |

| Arguments | Description |
|---|---|
| *filetype* | [Optional] The file extension automatically determines the fileType argument. You can use this argument when the file you are loading does not have an extension or if you want to override the extension. |
| *selectedstripes* | [Optional] Determines which ORC stripes to include.<br>**Default Value:** All |
| *hdfsconfigurationfile* | [Optional] The configuration file used to load data from or write data to HDFS.<br>**Default Value:** paste(system.file (package='ddc'),'/conf/hdfs.json',sep='') |
| *commentcharacter* | [Optional] Character used to determine where comments begin. |

## *Examples*

The following example demonstrates how you can load an ORC file from HDFS:

```
> library(HPdata)
> distributedR_start()
> mydframe <- orcdframe(url='hdfs:///tmp/file.orc', selectedStripes='0,1,2')
```

The following example demonstrates how you can load an ORC file from the local file system:

```
> library(HPdata)
> distributedR_start()
> mydframe <- orcdframe(url='file:///tmp/file.orc')
```

The orc2dframe function also supports wildcards (*) when selecting ORC files. For example, you can use a wildcard to retrieve all files in the `/tmp` directory with the extension .orc:

```
> library(HPdata)
> distributedR_start()
> mydframe <- orc2dframe(url='hdfs:///tmp/*.orc')
```

## *See Also*

For details, see the HPdata Package Manual, or type `?orc2dframe()` at the R prompt.

# object2hdfs

Writes an object from Distributed R to HDFS.

To use this function, you must first complete the steps in Configuring the HDFS Configuration File.

## Syntax

```
object2hdfs ( object, url,
[ overwrite = { 0 | 1 } ]
[ ... ]
)
```

## Arguments

| Arguments | Description |
|---|---|
| *object* | [Required] The distributed object written to HDFS. If the object is a string, then the object writes to a text file. Otherwise, the object is serialized to a raw vector and writes to a binary file. |
| *url* | [Required] The name and location of the output file. |
| *overwrite* | [Optional] Parameter:<br>• When set to 1 — If the file already exists, overwrite it.<br>• When set to 0 — If the file already exists, the object does not write to HDFS and Distributed R returns an error.<br>**Default Value:** 0 |

## Examples

The following example demonstrates how you can write a distributed object to HDFS:

```
> library(dataconnector)
> distributedR_start()
> object2hdfs(mymodel, 'hdfs:///file.out', overwrite=1)
```

## See Also

Details on the function are available in the HPdata Package Manual or by typing `?object2hdfs()` at the R prompt.

# Open Source Software Acknowledgments

This document contains the licenses and notices for open source software used in Distributed R. If you have any questions or wish to receive a copy of the source code to which you are entitled under the applicable free/open source license(s) (such as the GNU Lesser/General Public License), please contact HPE.

HPE makes no representations or warranties regarding any third party software. All third-party software is provided or recommended by HPE on an AS IS basis.



## R

R version 3.2.2 (2015-08-14) -- "Fire Safety"

Copyright (C) 2015 The R Foundation for Statistical Computing

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program

is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL

DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

# Protobuf

Copyright 2008 Google Inc. All rights reserved.

Version 2.4.1

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# Rcpp

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers

Version 0.12.0

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such

as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at

nocharge under subsection 6d.A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material

you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the

benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later

version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

# Rinside

Copyright © Dirk Eddelbuettel and Romain Francois, with contributions by Douglas Bates and John Chambers

Version 0.2.13

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program

is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL

DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The complete source code of the rinside package can be found at

# data.table

Copyright (c) M Dowle, T Short, S Lianoglou, A Srinivasan with contributions from R Saporta, E Antonyan

Version 1.9.4

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit

operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you

offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at nocharge under subsection 6d.A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and

adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES

SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

# randomForest

Copyright (c) Fortran original by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener

Version 4.6.10

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at nocharge under subsection 6d.A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified

or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES

SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

# R XML

Copright (c) Duncan Temple Lang (duncan@r-project.org)

Version 3.98.1-1

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# RODBC

Copyright (c) Brian Ripley [aut, cre] 2014, Michael Lapsley [aut] (1999 to Oct 2002)

Version 1.3-11

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared

libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at nocharge under subsection 6d.A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional

permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not

permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

# Boost

Copyright Beman Dawes, Daniel Frey, David Abrahams, 2003-2004.

Copyright Rene Rivera 2004-2005.

Boost Software License - Version 1.38 - February 8th, 2009

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# atomicio.cpp

Copyright (c) 2006 Damien Miller. All rights reserved.

Copyright (c) 2005 Anil Madhavapeddy. All rights reserved.

Copyright (c) 1995,1999 Theo de Raadt. All rights reserved.

https://code.google.com/p/openbsd-netcat/source/browse/trunk/atomicio.c?spec=svn2&r=2

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING

NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# ZeroMQ

ØMQ is copyright (c) Copyright (c) 2007-2014 iMatix Corporation and Contributors

http://zeromq.org/

This package is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This package is distributed in the hope that it is useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this package; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

# pip

Copyright (c) 2008-2014 The pip developers (see AUTHORS.txt file)

https://github.com/pypa/pip

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# testthat

Copyright (c) 2013-2014, Hadley Wickham; RStudio

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense,

and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# memoise

Copyright (c) 2013-2014, Hadley Wickham; RStudio

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

# digest

digest version 0.6.8 (2014-12-31)

Copyright © Dirk Eddelbuettel, with contributions by Antoine Lucas, Jarek Tuszynski, Henrik Bengtsson, Simon Urbanek, Mario Frasca, Bryan Lewis, Murray Stokely, Hannes Muehleisen, Duncan Murdoch, Jim Hester and Wush Wu.

The GNU General Public License (GPL) Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this

License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices.

Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

# crayon



# GBM

(and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major

components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you".

"Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for he work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices"

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at

nocharge under subsection 6d.A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material

you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms.

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the

benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later

version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS