VERITAS™

VERITAS Replication Exec™ version 3.1 for Windows

srTool Reference Guide

N163538

December 2004

Disclaimer

The information contained in this publication is subject to change without notice. VERITAS Software Corporation makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. VERITAS Software Corporation shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this manual.

Copyright

VERITAS Software Corporation
1600 Plymouth St.
Mountain View, CA 94043
Phone 650–335–8000
Fax 650–335–8050
www.veritas.com

Preface

Document Release Notice

This revision (3.1) of the *srTool Reference Guide* addresses minimal changes to the srTool utility. These changes are as follows.

- The product of which srTool is a feature has changed names. *VERITAS Storage Replicator (VSR)* is now *VERITAS Replication Exec (VRE).*
- An additional six error messages have been added to the Client Interface Messages (see ***"High-Level Client Interface Messages"*** on page 206***).***

Additional features have been added to *VERITAS Replication Exec* version 3.1 as follows.

- *Backup Exec SmartLink* is a command line utility that adds replication job monitoring and alerting capability to *VERITAS Backup Exec* version 10.0. This feature is documented in the *VERITAS Replication Exec* version 3.1, *Backup Exec SmartLink Reference Guide*.
- *VERITAS Replication Exec* version 3.1 now supports clustering of the Job Agent and Replication Management Server (RMS) using *VERITAS Cluster Server* and *Microsoft Cluster Server*. These features are documented in the *VERITAS Replication Exec* version 3.1, *Clustering Reference Guide*.
- Refer also to the *VERITAS Replication Exec* version 3.1 Readme file for additional changes to the srTool utility and *Replication Exec*.

What You'll Find in this Guide

***Chapter 1. "Introduction"* on page 1**

Provides an overview of srTool and a summary of changes in srTool for *VERITAS Replication Exec*.

***Chapter 2. "Getting Started Using srTool"* on page 11**

Demonstrates the power of srTool to automate the control and configuration of replication Jobs through a series of practical examples.

***Chapter 3. "Language Reference"* on page 23**

Provides a description of the command language of srTool, including the srTool command, output redirection, basic command syntax, data types, constants (literals), variables, expressions, flow control, functions, macros and embedded commands, object reference, and object specifications.

***Chapter 4. "srTool Command Reference"* on page 73**

Provides a detailed description of srTool's commands, including syntax, aliases, required and optional parameters, and examples. The command reference is presented in alphabetical order.

***Chapter 5. "srTool Object Reference"* on page 135**

Describes in detail each of srTool's objects, including their discovery, properties, and examples for adding, changing or deleting them. The objects reference is presented in alphabetical order.

Appendix A. "Messages and Troubleshooting" on page 187

Provides a description of the various errors and messages of srTool, as well as troubleshooting solutions.

Getting Help

VERITAS offers you a variety of support options.

Accessing the VERITAS Support Web Site

The VERITAS Support Web site allows you to:

- contact the VERITAS Support staff and post questions to them
- get the latest patches, upgrades, and utilities
- view the *Replication Exec* Frequently Asked Questions (FAQ) page
- search the knowledge base for answers to technical support questions
- receive automatic notice of product updates
- find out about *Replication Exec* training
- read current white papers related to *Replication Exec*

The address for the VERITAS Support Web site is:

`http://support.veritas.com`

Replication Exec Documentation Set

The *srTool Reference Guide* is part of the *VERITAS Replication Exec* (VRE 3.1) documentation set, which consists of the following references.

Document Title	Description
Replication Exec Administrator's Guide (vreadmin_en.pdf)	The *Replication Exec* Administrator's Guide in Adobe Acrobat format.
Replication Exec Help files (vreadmin_en.chm)	Accessible as standard Windows help files from the *Replication Exec* software.
srTool Reference Guide (srtool_en.pdf)	The srTool Reference Guide in Adobe Acrobat format.
srTool Help files (srtool_en.chm)	Accessible with any html browser and located on the VRE CD-ROM.

Conventions

The following conventions apply in this manual.

Syntax Conventions

The syntax conventions used in this document are as follows.

Convention	Description
{required}	◆ Anything in curly brackets *must* be specificed.
[optional]	◆ Anything in square brackets is *optional*.
x | y	◆ A vertical bar indicates a choice of syntax elements; for example "x" *or* "y".
...	◆ Elipses indicate the preceding syntax element can be repeated any number of times.

Typographical Conventions

The typographical conventions used in this document are as follows.

Convention	Description
Keyword	◆ Used to depict srTool keywords (reserved words) and operators.
Placeholder	◆ Used for for placeholder text, book titles, new terms, or emphasis. Replace placeholder text with your specific text. For example: Replace *filename* with the name of your file. ◆ Used for expressions, variables and parameters, except **Keywords** and ***Commands***.
Command	◆ Used to show commands specific to the command line interface: The ***stop*** command stops all functions at this point.
User Input	◆ Used to show data to be input by the user or examples: **create 4 jobs**
Path Name	◆ Used to show path names or path name variables: Move to the following file: C:\Program Files\VRE\widget.exe

Tips, Notes, and Cautions

Tips, notes, and cautions are used to emphasize information. The following samples describe when each is used.

Tip Used for nice-to-know information, such as a shortcut.

Note Used for important information that you should know, but that should not cause any damage to your data or your system if you choose to ignore it.

Caution Used for information that will prevent a problem. Ignore a caution at your own risk.

Contents

Introduction 1

srTool is a powerful complement to *VERITAS Replication Exec* (VRE 3.1) that allows users to automate the control and configuration of a replication system.

srTool:

- provides a command-line interface for detailed scripting of replication tasks
- extends capabilities beyond the limits of the VRE graphical user interface
- greatly simplifies otherwise tedious object-by-object tasks
- extends diagnostic functionality through monitoring of replication activity

Upgrading from Prior Versions

If you are running scripts generated under prior versions of srTool be sure to thoughly review the "Summary of Changes to srTool for VRE 3.1" on page 2.

Note Changes to VRE 3.1 may cause srTool scripts that worked under previous releases to either stop working or work unpredictably.

Current Documentation

While this reference guide describes the command language and operation of srTool, this product may undergo periodic or last-minute changes. For the most current documentation for srTool, see the `Readme` file and online help accompanying the srTool software.

For information on the functionality and operation of VRE 3.1, see the *VERITAS Replication Exec (VRE 3.1) Administrator's Guide*.

User Proficiency

This reference guide assumes the user is proficient in using *Replication Exec*, Windows Command Interpreter, and network administration.

Summary of Changes to srTool for VRE 3.1

VERITAS Replication Exec (VRE 3.1) is an important milestone in the technical evolution of VERITAS' file replicator for Windows-based servers. Its command-line utility, srTool, has also undergone substantial changes. Some essential improvements to VRE 3.1 necessitated the change of various aspects of the command language and constructs used in the srTool utility.

Caution *Please review this Reference thoroughly.* Some of the changes to srTool will cause srTool scripts that worked under previous releases to either stop working or work unpredictably.

This summary provides an overview of changes made in VRE 3.1, and provides users the opportunity to determine whether their existing scripts will need to be altered to function properly with VRE 3.1.

Following are the significant changes in srTool's command language for VRE 3.1.

Data Types

- The **boolean**, **enum** and **path** data types have been eliminated. The functionality of the **boolean** and **enum** types has been replaced by the **uint32** type. Both keywords still exist and are now aliases for **uint32**.
- The data type formerly called **count** is now called **uint32**. The keyword **count** still exists, and is an alias for **uint32**.
- The data type formerly called **count64** is now called **uint64**. The keyword **count64** still exists, and is an alias for **uint64**.
- A new data type has been introduced called **blob** or **byteArray,** which is used for storing an ordered string of bytes of arbitrary length.
- srTool is now much more effective at converting one data type into another.

See “Output Redirection” on page 30 for additional information.

Constants (Literals)

- In VRE 3.1, string literals can only be specified with quotation marks. Unquoted alphanumeric names that aren't keywords or property names are assumed to be the names of shell variables, which can store data values of any permissible type (**string**, **TimeStamp**, **timespan**, **guid**, **uint64**, **float**, and so on).

Prior versions of srTool allowed string literals to be specified undelimited, which made it difficult, if not impossible, for srTool to distinguish between a variable and a string literal. This is why srTool relied on the use of a crude macro replacement when using the value of a variable.

Note Scripts that use unquoted string literals will no longer work and must be changed. Anything in quotes is case sensitive by default unless this option is changed by the user.

Old Syntax	New Syntax
`list every job whose description startsWith A`	`list every job whose description startsWith "A"`
`list job Foo`	`list job "Foo"`
`add job with name = Foo, type = OneToOne`	`add job with name = "Foo", type = OneToOne`
`set jobName = "The Job"` `list job %jobName`	`jobName = "The Job"` `list first job whose name eq jobName`

- Character escape capability has been added to string literals.

  It was difficult in prior versions of srTool to compose a string that contained double and single quotation marks along with tab, newline and certain other characters.

New Syntax
`"One\tTwo\t\"Buckle My Shoe\"\tThree\tFour\n"`
`'Mr. Smith\'s Vacation'`
`"\\\\MACHINE\\C$\\WINNT\\SYSTEM32"`

See “Constants (Literals)” on page 38 for additional information.

Variables

- Variables now have scoping rules. By default, the scope of a newly created variable terminates when the execution context in which it was created terminates.
- Variables now have access attributes. By default, a newly created variable is always writeable. Certain predefined shell variables are read-only and their values cannot be changed.

See “Variables” on page 41 for additional information.

Expressions

- Expressions are now accepted in commands anywhere constant literals were previously accepted.

See “Expressions” on page 50 for additional information.

Object Specifications

- Object specifications are now *grouped* or *indexed*.
- Indexed object specifications are now zero-based instead of one-based. That is, **job 0** refers to the *first* job, and **job 1** refers to the *second* job.

Old Syntax	New Syntax
list job 1	list job 0 *...or...* list first job

Note srTool Scripts that use *indexed object specifications* will no longer work as expected and will have to be changed.

- Indexed object specifications can now include any number of index ranges. Thus, you can refer to **logEntries 0 thru 199, 300 thru 399**.
- Grouped object specifications can now restrict both the size and relative position of the group, for example, **first 20**, **last 100**, **any 5**, **every**, **all**, **middle**, and so on As in prior versions, they can incorporate a filter by using a **whose** clause.

  See “Object Specifications” on page 67 for additional information.
- Specifying ordinal values for property identification is no longer supported. Only property names may be specified. See “Property Specifications” on page 55.

Execution Control Structures

Changes made to the execution control structures consist of four new constructs added to the ***loop*** command. The ***loop*** command now supports the following additional constructs:

- loop a specificed number of times
- loop through a range

- loop through a list
- loop over objects resulting from compound object specifications

See "`loop` command" on page 108.

Functions

Functions are a new feature to VRE 3.1. Functions are used in expressions, include several built-in functions, and provide for user-defined functions. See "Functions" on page 57 for additional information.

Macros and Embedded Commands

The macros and embedded commands are essentially unchanged from prior release of srTool. However, the reliability and capabilities of macros and embedded commands have been enhanced. See "Macros and Embedded Commands" on page 63.

Commands

VRE 3.1 introduces several new commands, changes several commands (some substantially), and eliminates a few commands that are no longer needed. See "srTool Command Reference" for in-depth command descriptions and functionality.

New Commands

There are two new commands in this version of srTool.

New Commands	Description	See...
Configure	Allows you to control certain tunable parameters of the client interface	page 83
Spawn	Executes scripts or commands in the background	page 124

Changed Commands

The changed commands in srTool are indicated in the following table.

Changed Commands	Description	See...
Call	The maximum nesting level of called command shells has been increased to 32 from 12.	page 78
Enable, ***Disable***	The ***Enable*** and ***Disable*** commands now only apply to server objects.	page 95 and page 89
IsSelected	The ***IsSelected*** command is now the ***Check*** command.	page 81
List, Get	The functionality of the ***get*** command has been merged into and is now an alias of the ***list*** command. Functionality of the ***list*** command includes: ◆ restricting the displayed properties to a specific set and display order ◆ displaying the property data in a non-tabular format using optional user-defined field and record delimiters ◆ displaying property data of type string, dateTime or timespan in quotes The command also retains its original functionality, including: ◆ displaying all properties by default, or all but a specific set of properties ◆ displaying property data unquoted in a neat columnized table ◆ optionally displaying column headings The syntax rules regarding use of the '**of**' keyword between the property list and the object specification have been relaxed.	page 106

Objects

srTool's hierarchy of objects has changed. Significant changes to srTool's objects are as follows.

- Some objects previously available at the root level can now only be found at their appropriate place in the object hierarchy. These include ReplicationPairs, PathRules, SelectionRules, and DestinationRules.

- Abbreviations and plural forms of some object names have been added.

Prior Object Hierarchy

Following is the prior (older) Object Hierarchy. See "srTool Object Hierarchy" on page 64 for the current Object Hierarchy.

Note *Please review these hierarchies carefully as the changes may warrant significant changes in scripts written under prior verisons of srTool.*

Server
- Volume
  - Folder
    - Folder
    - File
  - LogEntry

FileReplicationJob
- ReplicationPair
  - LogEntry
- PathRule
  - SelectionRule
  - DestinationRule
- LogEntry
- SourceServer
- TargetServer

Alert

Credential

ReplicationPair

PathRule

SelectionRule

DestinationRule

See also "srTool Object Hierarchy" on page 64.

Changed Objects

Note VRE 3.1 incorporates many changed objects. Some objects have entirely new properties, some have lost properties, and some have property name or data type changes. Please review the individual objects' definitions and functionalities to determine any changes that may be required to srTool scripts written under prior versions.

New Objects

The following new objects have been added to srTool for VRE 3.1.

New Objects	Description	See...
License	Licenses are objects that represent actual licenses that are installed on a server. See "License Objects" on page 149.	page 149
ObjectKind	An ObjectKind is a meta object that describes srTool objects. See "ObjectKind Objects" on page 153.	page 153
Property	A Property is a meta object that describes the properties of objects. See "Property Objects" on page 156.	page 156
rms	An RMS is a Replication Management Server that is designated to manage replication jobs on the network. The RMS contains a database that stores information about Jobs, Servers and Alerts. See "RMS Objects" on page 161.	page 161
script	Script objects inform the replication system what special program to run when some special event occurs on either the Source or Target, such as when synchronization has been achieved. See "Script Objects" on page 164.	page 164
subfile	SubFile objects are files that represent any file on a server. See "SourceServer Objects" on page 171.	page 174
subfolder	SubFolders are directories inside volumes on any given server. See "SubFolder Objects" on page 176.	page 176
subitem	SubItems are folder or file objects, which allow the user to obtain both in a single query. See "SubItem Objects" on page 178.	page 178

srTool Help

srTool's *help* command relies on the Windows Explorer shell application (`explorer.exe`) to open srTool's online documentation. The online documentation is stored in a compiled help file (`srTool.chm`) on the local host machine and is opened using Microsoft's HTML Help application (`hh.exe`).

If you start srTool from a service, such as the "at" or "Scheduled Tasks" service, such that it cannot interact with the Windows desktop, you cannot use the ***help*** command.

Note srTool help features are available from the VRE 3.1 Administrative Console or by using the ***help*** command. See "`help` command" on page 104.

Launching the srTool Utility

srTool may be launched in several ways:

- by double-clicking the executable file (*SRTOOL.EXE*) from the directory:

  ```
  <drive>:\Program Files\VERITAS\Replication Exec\
  ```

- by typing in the full path name and including file name in the command prompt window:

  ```
  <drive>:\Program Files\VERITAS\Replication Exec\srtool.exe
  ```

- directly from the *Information Desk* of the VRE 3.1 Administration Console.

Getting Started Using srTool 2

This section is intended to demonstrate the power of srTool to automate the control and configuration of replication Jobs through a series of practical examples.

Note Before proceeding with srTool, the user should be familiar with the basic VRE 3.1 operations. Refer to the *VERITAS Replication Exec (VRE 3.1) Administrator's Guide* for a detailed description of storage replication and VRE 3.1's features.

The following srTool techniques are described in this section.

Creating and Modifying Replication Jobs

Monitoring, Querying and Reporting Job Status

Creating and Modifying Replication Jobs

Example 1. Create a Standard (One-to-One) Job

Creating a Job in srTool entails creating all of the subpieces of a Job. The minimum items that must be specified to create a Job are Job name, Job type, pair, path rule, and selection rule. There are many other properties that can also be specified. If the optional properties are not specified when creating a Job, the Job uses the default properties.

The goal of this example is to replicate all of the Excel files of the `c:\documents` folder of the machine “Accounting” to the `d:\backup` folder of the target machine "BackupServer". The creator of this Job must also establish access to the Target machine before creating the Job pair.

1. Run srTool by launching it from the **Start** menu or VRE 3.1 Administration Console, or by typing the following in the command prompt window:

   ```
   <drive>:\Program Files\VERITAS\Replication Exec\srtool.exe
   ```

2. Create a Job with the name **`Accounting Data`** and a Job type of one-to-one (replicates data from just one server to another server). Enter the following command to create this Job:

   ```
   create job with name="Accounting Data", type=OneToOne
   ```

3. The Job is now created. The next step is to add the user’s credentials to the Target server, by entering the following:

   ```
   add credential with serverName = "BackupServer", userName =
   "Joe", domain = "Administrative", password = Encrypt ("Joe’s
   password")
   ```

4. The next step is to add a pair to this Job. A pair defines the source and destination for the data. Enter the following command to create the pair:

   ```
   add pair to job "Accounting Data" with
   sourceserver="Accounting", targetserver="BackupServer"
   ```

5. You now have a Job and a pair that specifies which machine the data is coming from and where the data is going. The next step is to create a path rule, which tells the Job exactly which directory to replicate. Enter the following command to create a rule for this Job:

   ```
   add rule to job "Accounting Data" with
   sourceserver="Accounting", path="C:\\documents"
   ```

6. Specify which files in the `c:\documents` directory you want to replicate. In this case, you want to backup all of the Excel files, which have a file extention of `.XLS`. The files that will be replicated are selected by creating a selection rule. Enter the following command:

   `add selrule to first rule of job "Accounting Data" with namespec="*.xls"`

7. The Job is now ready to run and all data will go to the default replication target directory specified during installation. However, our goal was to replicate these files to the `d:\backup` folder of the machine "BackupServer". To do this, run the following command:

   `add destrule to first rule of job "Accounting Data" with targetserver="BackupServer", path="d:\\backup"`

   Note All paths that contain backslashes must use two backslashes, for example `c:\\marketing\\sales\\invoices`. Also note the use of quotes for character strings that were entered. The Job created in this example will inherit the default schedule, which is open to replication at all times. This Job will begin replicating data as soon as the Job is created.

8. Schedule the Job. For most Job schedules, it may be easier to use the Administration Console for selecting when the Job should run. In this case, you want the Job to run at the default schedule, and no schedule changes are required.

9. Start the Job by entering the following:

   `start job "Accounting Data"`

Summary

- **`create job with name="Accounting Data", type=OneToOne`**
  Creates a one-to-one Job with name "Accounting Data".
- **`add credential with serverName = "BackupServer", userName = "Joe", domain = "Administrative", password = Encrypt ("Joe's password")`**
  This adds Joe's credentials to access the Target server.
- **`add pair to job "Accounting Data" with sourceserver="Accounting", targetserver="BackupServer"`**
  Adds a pair to the Job with source and target machines.
- **`add rule to job "Accounting Data" with sourceserver="Accounting", path="C:\\documents"`**
  Adds rule stating what data is to be replicated.

- **add selrule to first rule of job "Accounting Data" with namespec="*.xls"**
  Add selection rule that specifies what files to replicate (all Excel files in this case).
- **add destrule to first rule of job "Accounting Data" with targetserver="BackupServer", path="d:\\backup"**
  Defines where the data is to go.
- **start job "Accounting Data"**
  Starts running the Job immediately.

Example 2. Create a Centralization (Many-to-One) Job

Creating a Centralization (many-to-one) Job is very similar to creating a simple (Standard) Job. The minimum items that must be specified to create a Job include Job name, Job type, pairs, path rules, and selection rules. There are many other properties that can also be specified. If the optional properties are not specified when the Job is created, it uses the defaults.

The goal of this example is to replicate all of the MS Excel files of the `c:\documents` folders of 125 Source machines to the `d:\backup` folder of a single Target machine. The Source servers from which you are going to replicate are named "Accounting001" through "Accounting100", and "Bookkeeping001" through "Bookeeping025". The Target server is named "BackupAccounting". Further, you wish to schedule the Job to run each weekday (Monday through Friday) from 2100 to 2200 hours.

1. Run srTool by launching it from the **Start** menu or VRE 3.1 Administration Console, or by typing the following in the command prompt window:

   ```
   <drive>:\Program Files\VERITAS\Replication Exec\srtool.exe
   ```

2. Create a Job with the name **Accounting Data** and a Job type of many-to-one (replicates data from many Source servers to a single Target server). Enter the following command to create this Job. Because you will be making many changes to the Job, the enabled property is set to false ("0") to prevent it from starting before it has finished being configured.

   ```
   create job with name="Accounting Data", type=ManyToOne,
   enabled=0
   ```

3. Because of the large number of Source servers (125), it is much easier to define the replication pairs, path rules, selection rules and destination rules with a loop command. This procedure iterates over the list of servers one at a time with these properties. To do this, enter the following:

   ```
   loop over all servers whose (name startsWith "Accounting" or
   "Bookkeeping") and isAvailable
   ```

```
    add pair to job "Accounting Data" with
    sourceServer=propName, targetServer="Backup Accounting"

    theRule = `add pathRule to job "Accounting Data" with
    sourceServer=propName, path="C:\\documents"`

    add selRule to %theRule with nameSpec="*.xls"

    add destRule to %theRule with targetServer="Backup
    Accounting", path="D:\\backup"

end loop
```

4. You have now created all of the Job properties for the Job **`Accounting Data`**. In order to confirm that all of the Source servers are available for the Job, the **`isAvailable`** server property was also used in the loop.

5. Schedule the Job. In this example, you wish to schedule the Job to start each weekday (Monday through Friday) from 2100 to 2200 hours. The **Schedule** property is a byteArray data type, and a bit mask that specifies the Job's schedule. Each bit indicates "eligible" (1) or "ineligible" (0) to run. Each bit represents a 30-minute time span. The entire blob must be exactly 336 bits long, thus representing a 7-day (week-long) schedule. To specify the desired Job schedule, enter the following:

```
set enable=true, schedule = "0b
000000000000 000000000000 000000000000 000001100000
000000000000 000000000000 000000000000 000001100000
000000000000 000000000000 000000000000 000001100000
000000000000 000000000000 000000000000 000001100000
000000000000 000000000000 000000000000 000001100000
000000000000 000000000000 000000000000 000000000000
000000000000 000000000000 000000000000 000000000000"
for job "Accounting Data"
```

Note In the above entry, each line of bits represents the 48 30-minute time spans beginning with Monday. In the actual command entry, this string should contain *no* blanks spaces or line breaks.

The Job is now fully configured and will automatically start and run at the scheduled times.

Summary

- **`create job with name="Accounting Data", type=ManyToOne, enabled=0`**
  Creates a many-to-one Job with name "Accounting Data".

Note Because you will be making many changes to the Job, the enabled property is set to false ("0") to prevent it from starting before it has finished being configured.

- **loop over all servers whose (name startsWith "Accounting" or "Bookkeeping") and isAvailable**
  Initiates the loop command to identify the available servers whose names begin with "Accounting" or Bookkeeping". The property **isAvailable** indicates the RSA service is started and the license is valid.
  - **add pair to job "Accounting Data" with sourceServer=propName, targetServer="Backup Accounting"**
    Creates all replication pairs for the Source and Target servers within the loop. The server name is represented by the variable **propName**.
  - **theRule = 'add pathRule to job "Accounting Data" with sourceServer=propName, path="C:\\documents"'**
    Creates the path rule for the Job and defines the Source directory for each source server. The variable **theRule** receives a reference to the newly created pathRule.
  - **add selRule to %theRule with nameSpec="*.xls"**
    Creates the selection rule that specifies the data (all .xls files) to be replicated from the pathRule's directory on the source server.
  - **add destRule %theRule with targetServer="Backup Accounting", path="D:\\backup"**
    Creates the destination rule that specifies the destination directory for the data being replicated for the current pathRule.
- **end loop**
  Ends the current Job properties definition loop.

Example 3. Modify the Centralization (Many-to-One) Job

You are now going to modify the Centralization Job in Example 2 to replicate every Excel file from every volume of every server, as follows.

1. The first step to modifying the previously created Job is to delete all of the path rules from the existing Job. Enter the following:

   ```
   delete all pathRules of job "Accounting Data"
   ```

2. Now you will create another loop to redefine the path rules, as follows:

   ```
   loop over all servers whose (name startsWith "Accounting" or
   "Bookkeeping") and isAvailable
   ```

```
        currentserverName = propName

        loop over all volumes of first server whose name EQ propName

            add pathRule to job "Accounting Data" with sourceServer=
            currentServerName, path = propFullPath

        end loop over all volumes

    end loop over all servers
```

3. The path rules have now been assigned to each pair. The next step is to add the selection rules and destination rules for each path rule. Enter the following:

```
add selRule to all pathRules of job "Accounting Data" with
nameSpec = "*.xls"

add destRule to all pathRules of job "Accounting Data" with
targetServer="Backup Accounting", path = "D:\\Backup"
```

Summary

- **`loop over all servers whose (name startsWith "Accounting" or "Bookkeeping") and isAvailable`**
  Initiates the server loop. Only those servers that are available and have names that start with "Accounting" or "Bookkeeping" will be considered.
- **`currentserverName=propName`**
  This copies the "Name" property of the current server being considered into the temporary variable "currentServerName". This is done so that the second loop will also define a "propName" variable (since volumes have names). The server's name is needed to create the pathRule in the inner loop.
- **`loop over all volumes of first server whose name EQ propName`**
  Initiates the volume loop. A pathRule is added for each volume on this server.
- **`add pathRule to job "Accounting Data" with sourceServer= currentServerName, path=propFullPath`**
  Creates the pathRule for the current volume. The pathRule property refers to the volume's root directory.
- **`end loop over all volumes`**
  Terminates the volume loop, ensuring that there is one pathRule for each volume server.
- **`end loop over all servers`**
  Terminates the server loop, ensuring that at least one pathRule exists for each source server.

- **add selRule to all pathRules of job "Accounting Data" with nameSpec = "*.xls"**
  Creates the selection rule that specifies the data (all .xls files) to be replicated from the pathRule's directory on the source server.
- **add destRule to all pathRules of job "Accounting Data" with targetServer="Backup Accounting", path = "D:\\Backup"**
  Creates the destination rule that specifies the destination directory (D:\Backup) for the data being replicated for the current pathRule.

Example 4. Create a Publication (One-to-Many) Job

Creating a Publication (one-to-many) Job is similar to creating a Centralization (many-to-one) Job. The minimum items that must be specified to create a Job include Job name, Job type, pairs, path rules, and selection rules. There are many other properties that can also be specified. If the optional properties are not specified when creating a Job, the Job uses the default properties.

The goal of this example is to replicate all of the files from the server "Captain" to the c:\inetpub folder of all Target machines, using the Job name **Newsletter 025**. This Job will also be scheduled to run every day between 10:00 and 11:30 in the morning.

1. Run srTool by launching it from the **Start** menu or VRE 3.1 Administration Console, or by typing the following in the command prompt window:

   ```
   <drive>:\Program Files\VERITAS\Replication Exec\srtool.exe
   ```

2. Create a Job with the name **Newsletter 025** of type one-to-many (replicates data from a single Source server to many Target servers). Enter the following command to create this Job:

   ```
   create job with name="Newsletter 025", type=OneToMany
   ```

3. As in the prior example, you are going to use a loop command to define the Job properties. In this loop, you specify the Job name and add all available pairs for each server that is online and available. To do this, enter the following:

   ```
   loop over every server whose isOnline=1 and isAvailable=1

      add pair to first job whose name="Newsletter 025" with
      targetserver=propName, sourceServer="Captain"

   end
   ```

4. In the next step, you add the path rule for the Job

   ```
   add pathrule to job "Newsletter 025" with
   sourceserver="Captain", path="c:\\inetpub"
   ```

5. You are now ready to add the selection rule. . .

```
add selrule to first rule of job "Newsletter 025" with
namespec="*.*"
```

6. You are now ready to add the destination rule. . .

```
add destrule to first rule of job "Newsletter 025" with
targetServer=propName, path="c:\\inetpub"
```

7. In the final step you are going to schedule the Job. To do this you are going to use the **MakeString** property, as follows.

```
day = MakeString ("0", 18) + "111" + Makestring ("0", 27)
week = "0b" + MakeString (day, 7)
set schedule of every job to week
```

Summary

- **`create job with name="Newsletter 025", type=OneToMany`**
  Creates a one-to-many Job with name "Newsletter 025".
- **`loop over every server whose isOnline=1 and isAvailable=1`**

  `add pair to first job whose name="Newsletter 025" with targetserver=propName, sourceServer="Captain"`

  `end`
  Adds all pairs to the Job for all online and available Target machines with the "Captain" Source server.
- **`add pathrule to job "Newsletter 025" with sourceserver="Captain", path="c:\\inetpub"`**
  Adds a rule stating the path from the Source data to the Target directory.
- **`add selrule to every rule of every job whose name startswith "Newsletter 025" with namespec="*.*"`**
  Adds selection rule that specifies what files to replicate (all files in this case).
- **`day = MakeString ("0", 18) + "111" + ("0", 27)`**
  `week = "0b" + MakeString (day, 7)`
  `set schedule of every job to week`

  Defines when the replication Job will start. In this case we created a string of 18 zeros, followed by three ones and an additional 27 zeros, which represents the 48 half periods in each day with the period from 10:00 until 11:30 AM being toggled "on". This string is then reproduced seven (7) times for each day of the week, and enabled with the set schedule commands.

Monitoring, Querying and Reporting Job Status

Example 5. Monitor Jobs

1. This example shows the 10 most recent alerts, and identifies the objects that generated them:

   ```
   select TimeStamp, (name of first objectKind whose ordinalValue
   EQ assocObjType) + ' " ' + assocObjName + ' " ', description
   from first 10 alerts sortedBy descending timeStamp
   ```

2. The following example lists all alerts less than four hours old that are either warning or error messages, but not information alerts.

   ```
   List severity, timeStamp, OrigServerName, MessageText of all
   alerts whose (TimeStamp GT (now() - "4 hours" as Timespan)) and
   severity NE Inform
   ```

3. The following example ....

   ```
   List Name, JobState, LastStarted of all jobs
   ```

   This gives output like the following:

   ```
   Accounting_01 Completed 2/10/2004 17:23:21.902
   Accounting_05 Running 2/10/2004 17:23:21.902
   ```

4. To continuously monitor all jobs that start with "Accounting", enter the following. Every time the status of one of those jobs changes, it will be output to the command prompt window.

   ```
   Mon all jobs whose name startsWith "Accounting"
   ```

5. This example starts monitoring all job and server activities.

   ```
   mon all jobs, all servers
   ```

6. This example monitors all pairs that use the target server "LOGAN01", and only for those jobs whose names begin with "Boston".

   ```
   mon all pairs whose name endsWith ":LOGAN01" of every job
   whose name startsWith "Boston"
   ```

7. Any of these examples will show what is currently being monitored.

   ```
   monitor; monitor list; monitor show
   ```

8. This example immediately stops all monitoring completely.

```
mon stop all
```

9. This example pauses the 15th monitor that was started, then immediately resumes it, as well as 2, 3, 4 and 8 (if currently paused).

```
monitor pause 15; monitor resume 8, 15, 2 thru 4
```

Example 6. Query for Server Storage Information

Following is an example of querying all server volumes to report the server's storage state. The first step is to define a function to convert bytes into gigabytes:

```
function Gigabytes (inBytes)
returnValue = inBytes / 1024.0 / 1024.0 / 1024.0
end function
```

Now use the function to get the storage information about server "Chicago":

```
select serverName, Name, FileSystem, Gigabytes (capacity),
Gigabytes (BytesFree) from all volumes of server "Chicago"
```

```
Chigago C: NTFS 33.8819 8.40384
```

```
Chicago E: NTFS 34.1806 25.07370
```

Another Example:

```
get name, domain, address, isAvailable, OSVersion,
OSWindowsSubType of all servers
```

Here is the output:

```
Accounting_15 Chicago 10.51.24.240 true Microsoft Windows NT
5.00.2195 Service Pack 2 Professional
```

```
Marketing_021 Seattle 10.51.24.120 true Microsoft Windows NT
5.00.2195 Service Pack 2 Professional
```

```
Marketing_024 Seattle 10.51.28.204 true Microsoft Windows NT
5.00.2195 Service Pack 4 Professional
```


Example 7. Groom/Read/Summarize Logs and Alerts

1. The following example loops over all jobs that are not in the expected *Running* or *Completed* state. Log entries corresponding to these jobs are returned. The output will contain information that may help troubleshoot the cause of a Job not running or being completed.

   ```
   loop over all jobs whose JobState NE Running and JobState NE
   Completed

   echo -x "..... Job" + propName + "......."

   get TimeStamp, MessageText of all logEntries of first job whose
   ID EQ propID

   end loop
   ```

2. Use the ***list*** or ***select*** command to see the LogEntries for a particular Job:

   ```
   get TimeStamp, MessageText of all logEntries of job "Foo"
   ```

3. To determine the number of log entries for a given server:

   ```
   count all logEntries of server "Foo"
   ```

4. Use the ***delete*** command to remove LogEntries. This example deletes all log entries for the Job "Foo" that are more than three days old:

   ```
   use job "Foo" of first RMS
   delete every LogEntry whose TimeStamp LT (now () - '3 days' as
   timespan)
   ```

Language Reference 3

This section discusses the basic language of srTool.

Starting srTool

srTool is launched from the Windows NT command shell (`cmd.exe`) using the **`srTool`** command. Like most command-line utilities, the **`srTool`** command line can contain parameters to control how it starts up and operates.

Command Line Syntax

[*pathSpec*] **`srTool`**[**`.exe`**] [*parameters...*]

As with any other program invoked from the Windows NT command shell, you can optionally specify the fully qualified path name of the **`srTool`** executable itself. By default, **`srTool`** is installed in the following directory:

```
<drive>:\Program Files\VERITAS\Replication Exec\
```

Note srTool, by default, starts with a high-level command shell that uses a default compound object specification of **first rms**.

Required Parameters

There are no required parameters.

Optional Parameters

The optional parameters can be entered on the command line in any order except for the **-command (-cmd)** option.

The '/' character can also be used in place of the '-' character to designate the option keyword.

Parameter	Description
-v **-verbose**	Causes the **verbose** srTool built-in shell variable to be initially set *true*.
-help	Causes srTool to bring up the first page of its online help documentation.
-h **-hl** **-hlsob**	Causes srTool to start up with a shell that uses the High-Level Client Interface. (By default, srTool starts up with a shell that uses the High-Level Client Interface.) Note that this option is mutually exclusive with **-nointerface.**
-n **-no** **-nointerface**	Causes srTool to start up with a shell that does not use any Client Interface, High-Level or Low-Level. Note that such a shell cannot interact in any way with the *Replication Exec* system.
-cmd **-command**	Instructs srTool to treat the rest of the Windows command line as srTool commands that are to be immediately executed upon startup.
-stay	Used in conjunction with the **-command** option (see above), this option prevents srTool from exiting after executing the commands on the Windows command line. Note that this option must be used in combination with the **-command** option.
-nofirst	If the new shell uses the High-Level Client Interface, this option specifies that the shell should ***not*** set a default compound object specification of "**first rms**" as it normally does. It is illegal to use this option for a startup shell that uses no Client Interface.
-u {*userName*} **-user** **-userid** **-username**	Specifies a user name to be used in authenticating to the RMS server. If this option is used, the **-password** and **-domain** options must also be specified. Note: the user name must be in double quotes, for example **-user "Administrator"**

Parameter	Description
-p {*encryptedPassword*} **-pw** **-pass** **-password**	Specifies the password to be used in authenticating to the RMS server. If this option is used, the **-user** and **-domain** options must also be specified. Caution: *The use of clear text passwords is not advisable and poses a security risk*.
-d {*domainName*} **-dom** **-domain**	Specifies the name of the domain to be used in authenticating to the RMS server. This option is required if the user must authenticate with the RMS with credentials that are different from the logged on user. When used, the **-user** and **-password** options must also be specified. Note: the domain name must be in double quotes, such as **-domain ("MyDomain")**. If the machine is not part of a domain, the switch is still required, such as **-domain "".**

See also:

"Object Properties" on page 66

Basic Command Syntax

The srTool command language is quite simple. Most commands are of the following form:

command [*compoundObjectSpec*] [*outputRedirection*] [; ...]

...where...

command is a verb (for example, ***list*** or ***quit***)

compoundObjectSpec is an optional compound object specification that refers to one or more objects.

outputRedirection optionally instructs the srTool command shell to redirect the text output of the command to a file on the local host computer system.

Any number of srTool commands may be entered on a single command line, as long as they are separated by a semicolon.

srTool reads commands from the standard input stream, which by default is the keyboard. srTool prompts for command input with the **SRT301A** message and will wait indefinitely for a command to be entered. The ***quit*** command, when received by the original starting command shell, will terminate srTool and return control to the host operating system.

Note Anything not a literal or constant can be specified in mixed case, that is is treated case insensitively by srTool.

Command Summary

Object Discovery, Creation, Deletion and Changing Commands

Command	Description
add	Creates one or more objects
count	Counts objects
delete	Deletes one or more objects, and can delete functions and global variables
list	Displays property data obtained from one or more objects
select	Displays the results of expressions using property data from one or more objects
set	Changes property values of one or more objects; also sets values of shell variables

Server-Specific Commands

Command	Description
enable	Enables one or more servers
disable	Disables one or more servers

Job-Specific Commands

Command	Description
cancel	Immediately cancels execution of one or more jobs
start	Starts one or more jobs running
stop	Gracefully stops one or more jobs
check	Shows which files will be replicated

SelectionRule (selRule)-Specific Commands

Command	Description
demote	Demotes one or more selection rules
promote	Promotes one or more selection rules

Flow Control Commands

Command	Description
break	Exits the nearest 'Loop'
call	Executes commands from a file in a new command shell
continue	Jumps to the 'End' of the nearest 'Loop' block
else	Ends an 'If' or 'ElseIf' block and starts an 'Else' block
elseIf	Ends an 'If' or 'ElseIf' block and starts an 'ElseIf' block
end	Terminates a 'Loop', 'If', 'ElseIf' or 'Else' block
exec	Executes commands from a file in the current command shell
global	Creates, declares or deletes global variables
if	Starts a conditional block of commands
loop	Begins a block of commands that may be repeatedly executed
spawn	Executes commands in a new command shell in the background
wait	Waits for a given amount of time or until an expression becomes true

Other Commands

Command	Description
comment	Doesn't do anything; used for documenting script files
configure	Shows or modifies configuration parameters of client-side software components.
dump	Produces a file that can be used to restore or duplicate a replication system.

Command	Description
echo	Echoes parameters to the output stream
function	Defines new or redefines existing functions
help	Shows documentation information about srTool
monitor	Monitors replication system object creation, deletion and modification.
quit	Quits the srTool command shell
set	Displays, defines, changes or removes shell variables
shell	Executes a command on the host operating system
shift	Shifts parameters in called scripts
show	Shows available commands, objects, data types, properties, and so on
use	Sets a default object specification for the shell
xml	Displays objects or expressions in XML format

Output Redirection

The standard or diagnostic output from any srTool command can be redirected to a file.

Output Type	Description
Standard	The ordinary and expected information emitted by a command.
Diagnostic	The unexpected information emitted by a command, such as error and warning messages.

Redirection is specified by terminating the command with a redirection directive. If there are multiple commands on the command line, the redirection directive must terminate the command being redirected, but immediately precede any delimiting semicolon that separates it from the adjacent command.

Syntax:

[1 | **2]>[>]***stringLiteral*

...where the *stringLiteral* must contain a valid path to a file.

If '**1**' is specified, or if neither '**1**' nor '**2**' is specified, the command's *standard* output is redirected to the file. If '**2**' is specified, the command's *diagnostic* output is redirected to the file. If the file does not yet exist, it is created.

If the second '**>**' is specified, the output information is appended to the file. The file will be created if it does not yet exist.

Notes:

- Redirection only applies to a single command. To apply it to several commands, the commands should be placed in a single file to be ***call***ed or ***exec***'d using redirection on the ***call*** or ***exec*** command.
- Only one redirection directive may be specified for a single command. Thus, it is not possible to concurrently redirect both standard and diagnostic output.
- Since srTool commands do not themselves accept input from the keyboard or standard input, there is no provision inside srTool for redirecting srTool command input from a file. However, this can be accomplished externally through the Windows NT command shell.
- There is no support for UNIX-style "pipes" in srTool, nor is there support inside srTool for directly "piping" the output of an srTool command to the input of an (external) Windows NT command. This can be accomplished externally through the Windows NT command shell.

- Be careful with Windows NT path delimiters inside the path string. Backslash ('\') characters in string literals are used to "escape" certain special characters. See “Constants (Literals)” on page 38 for information on specifying string constants.
- Spaces cannot separate any of the redirection syntax elements.

Example:

This example dumps the current job list to a file named "Current Jobs.txt" on JSmith's desktop, overwriting any file by that name if it already exists.

```
list all jobs >"C:\\Documents and
Settings\\JSmith\\Desktop\\Current Jobs.txt"
```

This example appends all alerts to a file named "alerts.log" on the root of the C drive.

```
list all alerts >>"C:\\alerts.log"
```


Data Types

srTool supports a variety of data types used in object properties and shell variables.

Use the ***show types*** command to discover what data types are available.

Data Type	Purpose	Minimum Value	Maximum Value
blob (byteArray)	Represents an arbitrary bit sequence.	<empty>	Limited only by available system memory.
dateTime	Represents exact moments in time with 1 millisecond resolution.	<never>	Approximately Mon Jan 18 19:14:04 2038
float	Represents double-precision floating-point numeric values, with 15 digit precision.	-1.7e+308	1.7e+308
int32	Represents signed 32-bit integer values.	-2,147,483,648	+2,147,483,648
int64	Represents signed 64-bit integer values.	-9,223,372,036,854,775,808	+9,223,372,036,854,775,808
string	Represents text information, including file or directory paths.	"" (empty string)	Limited only by available system memory.
timeSpan	Represents spans of time, with 1 millisecond resolution.	0.000 seconds	2,147,483,647 days, 23 hours, 59 minutes, 59.999 seconds
uint32	Represents unsigned 32-bit integer values.	0	4,294,967,296
uint64	Represents unsigned 64-bit integer values.	0	18,446,744,073,709,551,616
uniqueID	Represents globally unique identifiers.	{00000000-0000-0000-0000-000000000000}	{FFFFFFFFF-FFFF-FFFF-FFFF-FFFFFFFFFFFF}

Converting Between Different Data Types

srTool attempts to convert data to other types as needed (or as requested with the ***as*** operator). The following chart describes how each data type can convert to any of the other types.

	to **string**	to **uniqueID**	to **dateTime**	to **timeSpan**	to **uint32**	to **float**	to **uint64**	to **int64**	to **int32**	to **blob**
from **string**		Yes[1]	Yes[2]	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes[5]
from **uniqueID**	Yes		No	No	No	No	No	No	No	Yes
from **dateTime**	Yes	No		Yes	Yes[3]	No	Yes	Yes	Yes[3]	Yes
from **timeSpan**	Yes	No	No		Yes[3]	No	Yes[3]	Yes	Yes[3]	Yes
from **uint32**	Yes	No	Yes	Yes		Yes	Yes[3]	Yes[3]	Yes	Yes
from **float**	Yes	No	Yes[3]	Yes[3]	Yes[3]		Yes[3]	Yes[3]	Yes[3]	Yes
from **uint64**	Yes	No	Yes	Yes	Yes[3]	Yes[3]		Yes	Yes[3]	Yes
from **int64**	Yes	No	Yes[3]	Yes	Yes[3]	Yes	Yes[3]		Yes[3]	Yes
from **int32**	Yes	No	Yes[3]	Yes	Yes[3]	Yes	Yes[3]	Yes		Yes
from **blob**	Yes	Yes[3]	Yes[3]	Yes[3]	Yes[3]	Yes[3]	Yes[3]	Yes[3]	Yes[3]	

1. The string must contain a valid uniqueID constant, otherwise it will not convert. See "Constants (Literals)" on page 38 for syntax.

2. The string must contain a valid date specification. See "Converting a String into a DateTime" on page 35.

3. The conversion will take place without error, but some data loss—possibly significant—will occur.

4. The string must contain a valid timespan specification. See "Converting a String into a TimeSpan" on page 36.

5. The string must contain a "bit string" or "hex string". See "Converting a String into a Blob" on page 34.

Converting a String into a Blob

A blob is specified inside a string by a "bit string" or a "hex string".

Bit-String Method

The string starts with the character sequence '0b' (or '0B') and is followed by any number of '0' or '1' characters. Any other characters found in the sequence fails the conversion. If the sequence fails to end on a byte boundary, it will be zero-padded such that it does.

```
0b0101010101110100101110101010000101010100101011010001000000000000
```

Hex-String Method

The other method is by encoding a "hex-string" in which the string starts with the character sequence '0x' (or '0X') and following it by at least 33 hexadecimal digits ('0'...'9' plus 'a'...'f' or 'A'...'F'). Any other characters found in the sequence fail the conversion. If the sequence fails to end on a byte boundary, it will be zero-padded such that it does.

```
0x0123456789ABCDEFfedcba9876543210AaBbCcDdEeFfEeDdCcBbAa0011223344
```

Converting a String into a UniqueID

A globally unique identifier is specified by enclosing the following exact character sequence in curly braces ('{' and '}'):

- 8 hexadecimal digits
- 1 hyphen
- 4 hexadecimal digits
- 1 hyphen
- 4 hexadecimal digits
- 1 hyphen
- 4 hexadecimal digits
- 1 hyphen
- 12 hexadecimal digits

Example:

```
{A32Fba1E-D2D7-4583-850A-1FA58CbB9eB0}
```

Converting a String into a DateTime

A string value can be converted into a **dateTime** value provided the string meets the following syntax rules:

datePart | *timePart* | *{datePart timePart}*

where...

datePart = *monNumber* **'/'** *dyNumber* [**'/'** *yrNumber*]

timePart = *hrNumber* [**':'** *minNumber* [**':'** *secNumber*]] [*meridianSpec*]

and...

Variable	Description
monNumber	An unsigned decimal integer literal whose value is between 1 and 12
dyNumber	An unsigned decimal integer literal whose value is between 1 and 31
yrNumber	An unsigned decimal integer literal whose value is between 0 and 2038
hrNumber	An unsigned decimal integer literal whose value is between 0 and 23
minNumber *secNumber*	An unsigned decimal integer literal whose value is between 0 and 59
meridianSpec	An (optional) 'AM' or 'PM' in upper or lower case

Examples:

```
"4/2/2002 4:03 PM"
"01/01/01 22:02:03:222"
```

This example specifies April 15th of this year:

```
"4/15"
```

This example specifies 5 P.M. today:

```
"5 PM"
```


Converting a String into a TimeSpan

A string value can be converted into a **timeSpan** value, provided the string meets the following syntax rules:

{*shorthandSpec*} | {*longhandSpec*}

where...

shorthandSpec

number '**:**' *number* ['**:**' *number* [...]]

There can be a maximum of 4 numbers with 3 intervening colons. From the least significant (rightmost) number to the most significant (leftmost) number, the values represent, respectively, seconds, minutes, hours and days. Fractional decimal values can be used for any of the numeric values.

Examples:

```
"0:3" as timeSpan
"2.5 : 13.5 : 98.7 : 1023.33" as timespan
"4:5:3:22" as timespan
```

longhandSpec

{*number timeUnit*} [...]

One or more numeric values get associated with a specific time unit. The resulting timeSpan is the sum of all specified time spans. Any numeric values can have fractional values. No time unit can be specified more than once.

Examples:

```
"3 seconds" as timeSpan
"2.5 days 13.5 hours 98.7 minutes 1023.33 seconds" as timespan
"4d5h3m22s" as timespan
```

number

digit [...] ['**.**' *digit* [...]]

A number, as expressed in either of the above specs, is an unsigned decimal value that is either integer or floating point. Floating point values cannot have exponents. There can be no intervening whitespace characters between digits or the decimal point.

Examples:

```
00000000000321321
987654321.123456789
```

timeUnit

A timeUnit, as expressed in any of the above specs, is any of these keyword values:

```
'millenia' | 'milleniums' | 'millenium' | 'mills' | 'mill' |
'centuries' | 'century' | 'cents' | 'cent' |
'decades' | 'decade' | 'decs' | 'dec' |
'years' | 'year' | 'yrs' | 'yr' | 'y'
'months' | 'month' | 'mons' | 'mon' | 'mos' | 'mo' |
'weeks' | 'week' | 'wks' | 'wk' | 'w' |
'days' | 'day' | 'dys' | 'dy' | 'd' |
'hours' | 'hour' | 'hrs' | 'hr' | 'h' |
'minutes' | 'minute' | 'mins' | 'min' | 'mns' | 'mn' | 'm' |
'seconds' | 'second' | 'secs' | 'sec' | 's' |
'milliseconds' | 'millisecond' | 'msecs' | 'msec' | 'ms'
```

See also:

"Constants (Literals)" on page 38

Constants (Literals)

A constant is a literal specification of a data value in an srTool command line.

There are only six data types that can be specified as constants in the srTool command line, as follows.

string

A string constant is specified by enclosing any text in single or double quote marks.

{" | ' } [*character*] [...] {" | ' }

```
"This is Bob's string"

'This is also a "big" string'
```

Note If a single quote is used to start the string, a single quote must terminate the string; and, if a double quote is used to start the string, a double quote must terminate it.

Note srTool does not support the wrapping of string constants onto subsequent command lines.

Character escaping is supported through the use of the backslash ('\') character, although the use of quote marks different from those delimiting the string itself can be used in lieu of escapes for quote marks. The characters following the backslash are as follows.

Escape Sequence	Represents
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\"	Double quotation mark
****	Backslash

If the character that follows the backslash is not represented in the table above, both the backslash and the subsequent character are placed in the resulting string.

blob (byteArray)

A blob is specified as a "hex string" or a "bit string":

Bit String:

'0b' or **'0B'**

followed by any number of '0' or '1' characters

```
0b10001010110101011010101111010101000010100101011010101101
01111010101010100000101010101
```

- If the specified bits do not end on a byte boundary, the resulting byte array is zero-padded at the end such that it will end on a byte boundary.

Hex String:

'0x' or **'0X'**

followed by at least 17 hexadecimal digits ('0'..'9' plus 'a'..'f' or 'A'..'F')

```
0x0123456789AbCdEfFeDcBa98765432100123456789AbCdEfFeDcBa9876
543210
```

Notes:

- If the specified hex digits do not end on a byte boundary, a "zero nibble" is automatically appended such that the data will end on a byte boundary.
- If 16 hex digits or less are specified, the value of the literal is a **uint64** value. If 8 hex digits or less are specified, the value of the literal is a **uint32** value.

uniqueID

A globally unique identifier is specified by enclosing the following exact character sequence in curly braces:

- 8 hexadecimal digits
- 1 hyphen
- 4 hexadecimal digits
- 1 hyphen
- 4 hexadecimal digits
- 1 hyphen

- 4 hexadecimal digits
- 1 hyphen
- 12 hexadecimal digits

Example:

```
{A32FBA1E-D2D7-4583-850A-1FA58CBB9EB0}
```

uint32

Any unsigned decimal digit sequence or the characters "0x" or "0X" followed by a sequence of 8 or fewer hexadecimal digits is treated as a **uint32**, an unsigned 32-bit integer value.

```
3212
```

```
0x21Abc9
```

uint64

Any unsigned decimal digit sequence larger than a uint32 or the characters "0x" or "0X" followed by a sequence of 9 to 16 hexadecimal digits is treated as a **uint64**, which is an unsigned 64-bit integer value.

```
32127458398
```

```
0x21Abc9A015B
```

float

A floating point number is specified as an unsigned decimal digit sequence followed by a decimal point, then optionally followed by another digit sequence representing the fractional portion and an optional exponent.

```
1.2
```

```
12345.678e+5
```

'as' operator

To generate a data type other than the above six types in places where expressions are allowed, use the **as** operator to convert it to the desired type. For example:

```
53274583 as timespan
```

```
0x20 as integer
```

Variables

A variable is a named data value that can be used in expressions.

Syntax

letter [*letter* | *digit*] [...]

Like all identifier names in srTool, a variable name must start with an alphabetic (letter) character, and optionally be followed by any number of letters or digits. There is no limit to the number of characters that may comprise an identifier name.

Note You may choose any name that you wish for your variables, with the exception of property names, which are reserved. It is strongly recommended that you not use the names of existing objects, operators, data types, or other keywords.

Some variables are predefined by srTool, and of those, some are read-only. That is, you cannot change their values in a ***set*** command.

Examples:

```
myVariableName

AnExtremelyLongVariableName1234567
```

Execution Contexts

Scoping Rules

A variable exists in the execution context in which it was defined. Execution contexts are maintained for ***begin, if/else, loop***, and ***function*** commands, plus the command shell itself. For example, if you define a variable inside of a loop construct, the variable will cease to exist once the loop has exited. Likewise, a variable defined inside of a user-defined function will go out of scope once the function has finished executing. Or, if a file of srTool commands was executed by a ***call*** command, any variables defined therein will be out of scope when the called file returns.

Global Variables

The exceptions to the above scoping rules are global variables, which are created using the ***global*** command. To prevent confusion, do not choose the same names for global and local variables. Global variables are accessible from any execution context or command shell.

Variable Discovery

To discover what variables are currently defined, use the ***set*** command with no parameters or use the ***show Contexts*** command. The ***show global*** command can also be used to display just the global variables.

Creating Variables

- To create a local variable, define it using the ***set*** command.
- To create a global variable, use the ***global*** command.

Note You cannot use the name of any object properties when naming a variable.

Examples:

```
set x = 32
global y = " "
```

Changing Variables

To change the value of a variable, use the ***set*** command. The type of data that a variable can store is not fixed and can be changed any time, provided its access is not read-only.

Example:

```
set x = 2.0
```

Deleting Variables

To delete a variable, use the ***set*** command (with nothing past the "=" sign). Any future references to that variable will produce an error. To specifically delete a global variable, use either the ***delete global*** command or the ***global*** command with no expression following the "=" sign.

Example:

```
set x =
delete global y
global z =
```

Built-in Variables

The following variables are defined automatically by srTool's command shell, and are local in scope and valid for that shell until it has exited.

Variable	Description
commandResult	This variable contains a string whose value is set to the result code of the last command that was executed by the command shell. If the last command was successful, it will be set to "RXRESULT_Success". Default value: "RXRESULT_Success" Data Type: string Access: ReadOnly
continueOnError	This variable, if it evaluates to logical **true**, causes the shell to continue executing queued commands, even if a command fails. If its value is **false**, the shell will stop executing queued commands when one fails. Default value: true Data Type: uint32 Access: ReadWrite
debugLevel	This variable acts as a "volume control" that controls the volume of diagnostic output of the command shell to the “HLSOBLog...” log files, which may be useful on occasion when diagnosing problems with srTool itself. The "off" setting is zero; 1 is "low", 2 is "medium" and 3 is "high" (or "maximum"). See also the **config** command. Default value: 0 Data Type: uint32 Access: ReadWrite
echoCommands	This variable, if **true**, echoes the tokenized command, along with a time stamp, to the standard diagnostic stream prior to its execution. If **false**, no such command echo takes place. Setting this **true** greatly assists in the debugging of srTool command scripts. Default value: false Data Type: uint32 with value of 0 or 1. Access: ReadWrite

Variable	Description
fieldDelimiter	This variable determines the text that will appear between adjacent fields in listings caused by the ***list*** or ***select*** commands. The default value is a comma followed by a space. Often, it's helpful to use a Tab character to separate property values. To do this, simply execute this command: fieldDelimiter = ToChar (9) Default value: **","** Data Type: string Access: ReadWrite
inexactShorthand	This variable specifies how name matching is done when using shorthand object specifications that are done by name. If **true**, name matching is done inexactly using the "contains" operator (although still case-sensitively). If **false**, name matching is done exactly. Default value: false Data Type: boolean Access: ReadWrite
nestingLevel	This contains the nesting level of the command shell. When srTool is started, the default shell's nesting level is 1. All subsequent shells invoked by the ***call*** command will have nesting levels one higher than the parent shell that invoked it. Shells that are created by the **spawn** command will always have a nesting level of one. Default value: (see description) Data Type: count Access: ReadOnly
promptString	This variable determines the prompt that the command shell emits prior to awaiting command input from the standard input stream. The default prompt is the **SRT301A** message followed by a newline. To restore the default prompt, set it to an empty string or delete it. Default value: (see description) Data Type: string Access: ReadWrite

Variable	Description
recordDelimiter	This variable dictates the text that will appear between adjacent rows in listings caused by the ***list*** or ***select*** commands. The default value is a newline sequence. To restore this default, simply execute this command: **recordDelimiter = ToChar (13) + ToChar (10)** Default value: newline Data Type: string Access: ReadWrite
remoteFiltering	This variable, if **true**, enables the use of server-side filtering (if the server supports it). If **false**, all filtering is done on the client side, which as a general rule is much less efficient. Default value: true Data Type: uint32 Access: ReadWrite
remoteSorting	This variable, if **true**, enables the use of server-side sorting (if the server supports it). If **false**, all sorting is done on the client side, which as a general rule is much less efficient. Default value: true Data Type: uint32 Access: ReadWrite
shellID	This variable contains a human-readable string that contains an identifying name for the command shell, which is guaranteed to be unique within a single execution instance of srTool. It is possible to decipher the lineage of a command shell from this name. For example, the name "4.7.9" had to be a shell that was invoked using the ***call*** command from another shell that was invoked using ***call*** from a shell that was ***spawn***ed from the root shell. Default value: “1” Data Type: string Access: ReadWrite

Variable	Description
sobType	This variable contains a human-readable string that identifies the Client Interface that's in use by the command shell. It can be one of two values: "HLSOB" (High Level Client Interface) or " " (empty, which means, no Client Interface). Default value: (see description) Data Type: string Access: ReadOnly
verbose	This variable, if **true**, causes srTool to report greater details about what transpires. If **false**, srTool is much more brief in its reporting. Default value: false Data Type: uint32 Access: ReadWrite
whenShellStarted	This variable contains the date and time when the command shell was started in local time to the machine that is hosting srTool. Default value: (see description) Data Type: dateTime Access: ReadOnly

Global Variables

srTool automatically defines many built-in global variables, all of which are read-only and valid for the high-level command shell.

Variable Name	Value	Data Type
NullID	{00000000-0000-0000-0000-000000000000}	uniqueID
False	0	uint32
True	1	uint32
Pi	3.14159	float
JobState Property Values: These predefined global constants can be used when comparing the *JobState* property of job objects.		
• **Canceled**	2	uint32
• **CanceledWithErrors**	3	uint32

Variable Name	Value	Data Type
• **Canceling**	16	uint32
• **Completed**	6	uint32
• **CompletedWithErrors**	7	uint32
• **Completing**	18	uint32
• **Expired**	4	uint32
• **ExpiredWithErrors**	5	uint32
• **Expiring**	17	uint32
• **NeverRun**	1	uint32
• **Paused**	13	uint32
• **Pausing**	12	uint32
• **Rallying**	14	uint32
• **Resuming**	14	uint32
• **Running**	9	uint32
• **RunningWithErrors**	10	uint32
• **Starting**	8	uint32
JobType Property Values: These predefined global constants can be used when comparing the *JobType* property of job objects.		
• **ManyToMany**	3	uint32
• **ManyToOne**	1	uint32
• **OneToMany**	2	uint32
• **OneToOne**	0	uint32
TargetReplicaType Property Values: These predefined global constants can be used when comparing the *TargetReplicaType* property of job objects.		
• **Merge**	2	uint32

Variable Name	Value	Data Type
◆ **Pure**	0	uint32
◆ **Qualified**	1	uint32
◆ **UpdateOnly**	3	uint32
ClusterType Property Values: These predefined global constants can be used when comparing the *ClusterType* property of job objects.		
◆ **MSCS**	2	uint32
◆ **VCS**	1	uint32
MappingMethod Property Values: These predefined global constants can be used when comparing the *MappingMethod* property of job objects.		
◆ **PrependNone**	3	uint32
◆ **PrependSourceImmediateParent**	2	uint32
◆ **PrependSourceRootDirPath**	1	uint32
◆ **PrependSourceServerPath**	0	uint32

See also:

"Expressions" on page 50
"Terms" on page 51
"Functions" on page 57
"`call` command" on page 78
"`exec` command" on page 97
"`global` command" on page 103
"`set` command" on page 118
"`show` command" on page 122
"`spawn` command" on page 124

Parameter Variables

These variables are defined automatically by srTool in the process of invoking an srTool command script using the ***call***, ***exec*** or ***spawn*** command. They are local in scope and have read-write access.

Variable Name	Description	Data Type
paramCount	This contains the number of parameters passed to the script, including the name of the script file itself.	integer
param0	This variable contains the first parameter passed to the script. It contains the name of the script file being executed.	string
param1, param2, ...	These variables contain any other parameter data that was passed to the script.	(any)

Expressions

An expression is a single term or two or more terms that are separated by certain lower-precedence binary operators.

Syntax

term [*binaryOperator term*] [...]

The valid binary operators that can separate terms are as follows, shown in decreasing precedence order.

Binary Operators	Description
* or **multiplyBy**	Multiply
/ or **dividedBy**	Divide left-neighboring term by right-neighboring one
mod	The remainder after dividing left-neighboring term by right-neighboring one
startsWith	• True if the left string starts with the right string
contains	• True if the left string contains the right string
endsWith	• True if the left string ends with the right string
+ or **plus**	Adds or concatenates the left and right term
– or **minus**	Subtract the right term from the left term
Comparisons**:**	
eq, **is**, **=**, **equal** or **equalTo**	• Equality: results in True if left and right terms are equal
ne, **notEqual**, or **notEqualTo**	• Inequality: results in True if left and right terms are unequal
lt, **<**, or **lessThan**	• Less than: results in True if left term is less than right term
ge or **greaterThanOrEqual**	• Greater than or equal: results in True if left term is greater than or equal to right term
gt or **greaterThan**	• Greater than: results in True if left term is greater than right term
le or **lessThan**	• Less than: results in True if right term is greater than left term

Binary Operators	Description
& or **and**	Boolean ‘and’ operation. Results in True if both left and right terms are True
| or **or**	Boolean 'or' operation. Results in True if either left or right terms are True.

Examples:

```
"Foo"

5 + 4

10 plus 5.0 minus "3"
```

See also:

“Terms” on page 51

Terms

A term is a factor or two or more factors that are separated by certain moderate-precedence binary operators. A term can optionally be preceded by a unary operator.

Syntax:

[*unaryoperator*] *factor* [[**^**] *factor*] [...]

The valid operators allowed to separate or precede factors are as follows.

Operators	Operator Type	Description
^ or raisedTo	Binary (separate)	Raise left-neighboring factor to the power determined by the right-neighboring factor
! or **not**	Unary (precede)	Performs a boolean negation of the factor
-, minus, negate	Unary (precede)	Performs an arithmetic negation of the factor

Examples:

```
5
-5
5^3
- 5^3
```

See also:

"Factors" on page 52

Factors

A factor can be a constant, a local or global variable, the value of an object's property, the result of a function call, or another expression (provided it is enclosed in parentheses).

Note Expressions cannot be nested more than 64 levels deep.

Syntax:

constant

| *variableName*

| *propertySpec*

| *functionName* ([*expression* [,...]])

| (*expression*)

[**as** *dataType*]

Examples:

```
5
name of first RMS
pi
now ()
(5.4 GT 10.0)
23.445e-6 as string
```

See also:

"`function` command" on page 100
"Object Properties" on page 66
"Output Redirection" on page 30
"Property Specifications" on page 55
"`set` command" on page 118
"SortedBy Clause" on page 71
"srTool Object Hierarchy" on page 64
"Variables" on page 41

Operators

There are two classes of operators in srTool: unary and binary.

- Unary operators precede an operand (a term) in an expression.
- Binary operators conjoin two operands, either in a term, a simple expression or expression. Only the ***as*** operator can be used in factors in expressions.

Of the binary operators, six are further sub-classed as comparison operators.

Using the ***show operators*** command yields a complete list of the available operators and their classification:

Operator	Classification
not	unary
negate	unary
and	binary
or	binary
xor	binary
plus	binary
minus	binary
multiplyBy	binary
dividedBy	binary
mod	binary
raisedTo	binary
startsWith	binary
contains	binary
endsWith	binary
eq	binary, comparison
ne	binary, comparison
lt	binary, comparison
le	binary, comparison

Operator	Classification
gt	binary, comparison
ge	binary, comparison
as	binary

See also:

"Expressions" on page 50
"Terms" on page 51

Property Specifications

A property specification uniquely identifies a property of a specific kind of object, or a special value that is computed from a specific property of multiple objects.

Syntax:

[*specialValue*] *propertyName* [**of** *compoundObjectSpec*]

The *specialValue* , if present, must be one of the following keyword values:

Value	Description
minimum	Results in the minimum value of the specified property
average	Results in the average value of the specified property
median	Results in the median value of the specified property
maximum	Results in the maximum value of the specified property
total	Results in the sum of the specified property

- If the *specialValue* is present, the "**of** *compoundObjectSpec*" must also be present.
- If the *compoundObjectSpec* (object specification) is present, and the *specialValue* is not present, the object specification must resolve to a single object, or else the evaluation of the expression in which the property specification appears will fail.
- If the *compoundObjectSpec* is not present, the property is assumed to come from the object under consideration while executing a query.

Notes:

Because these property specifications can appear inside comparison expressions of an object specification's "whose clause" (inside of a larger compound object specification), this can lead to an ambiguity in the object specification.

For example, suppose you want the name of every available server. The novice srTool user could innocently code the command without the use of a default object specification:

```
get name of every server whose isAvailable of first RMS
```

This command will not work. srTool gets confused when parsing the "whose" clause, thinking that "isAvailable of first RMS" is a property specification, which is not the intent in this example. When srTool tries to get the data, it cannot because it is looking for server objects at the root level, which do not exist (servers come from RMS objects). To avoid the ambiguity, use parenthesis:

```
get name of every server whose (isAvailable) of first RMS
```

...or add a comparison:

```
get name of every server whose isAvailable is true of first RMS
```

The ***use*** command will also avoid this ambiguity:

```
use first RMS; get name of every server whose isAvailable
```

Example:

To get the average number of pairs for Jobs that have more than one pair:

```
echo -x average pairCount of all jobs whose pairCount GT 1
```

To get the highest alert count of any job:

```
echo -x maximum alertCount of all jobs
```

See also:

"srTool Object Reference" on page 135
"Object Properties" on page 66
"`use` command" on page 129

Functions

Functions are named entities that can accept any amount of parameter data, process that data, and return a single data value. srTool has a number of built-in functions, and provides users the ability to create additional functions.

Discovering Functions

Use the ***show functions*** command to see the available functions.

Built-in Functions

The following table shows the functions that are built into srTool.

Function Name	Description
Encrypt (*inString*)	Performs a one-way encryption of the string, typically used to encrypt a clear text password while creating credentials. Example: **`echo -x encrypt("My Name")`** Will return something like: **`0x313E4A42301FE78B2B76E74F0993E53749C8F6450107AB2F6761F75C01629792`**
Errorsof *(inCommands)*	Returns the error message(s) that results from executing the text contents of inCommands in a separate srTool command shell. If the command(s) succeeded, the resulting string will be "RXRESULT_Success". For a description and catalog of these messages, see Appendix A, Errors & Messages. Example: **`echo -x ErrorsOf ("Foo")`** **`RXRESULT_Syntax`**
FileExists *(inFilePath)*	Returns "true" if the file identified by *inFilePath* exists, or "false" if it does not. Example: **`echo -x FileExists("c:\\winnt\\system32\\calc.exe")`** This will return 1 if true, 0 if false.
Find *(inNeedle, inHaystack)*	Returns the character position of *inNeedle* in *inHaystack*. If not found, returns -1. Example: **`echo -x find ("g", "abcdefghijklmnop")`** **`6`**

Function Name	Description
If *(inCondition, trueResult, falseResult)*	Returns *trueResult* if *inCondition* evaluates to true (non-zero), or *falseResult* if false. Example: **`echo -x If( 1 = 1, 4, 5)`** This will return 4 because 1 = 1 is true. **`echo -x If( 3 = 1, 4, 5)`** This will return 5 because 3 = 1 is false
Getproperty *(inObjectID, inProperty)*	Returns the property data value for the given object, given its globally unique identifier, and the name or ordinal value of the desired property. Example: **`echo -x GetProperty (id of first job, "Name")`**
Indent (*inString*)	Replaces all newline character sequences in the given string with a new character sequence consisting of a newline and a tab character. Examples: **`set s = "This\nis\na\ntest"`** **`echo -x s`** **`This`** **`is`** **`a`** **`test`** **`echo -x indent(s)`** **`  This`** **`  is`** **`  a`** **`  test`**
Left *(inText, inNumChars)*	Returns one or more of the left-most characters of *inText*. The number of characters to be returned is determined by *inNumChars*. Example: **`echo -x Left ("This is a test", 7)`** Outputs: **`This is`**
MakeString *(inTextToRepeat, inRepetitions)*	Returns a string that contains the text of *inTextToRepeat* repeated the desired number of times. Example: **`echo -x MakeString("Hello World ", 4)`** Returns the following: **`Hello World Hello World Hello World Hello World`**

Function Name	Description

Max *(inValue1, inValue2, ...)*	Returns the value of the largest of all given arguments.
	Example:
	`echo -x Max (5, 10, 30)`
	`30`

Min *(inValue1, inValue2, ...)*	Returns the value of the smallest of all given arguments.
	Example:
	`echo -x Min (5, 10, 30)`
	`5`

NewID *()*	Returns a new globally unique identifier.
	Example:
	`echo -x NewID ()`
	`{1B1330E6-06C0-4665-9037-AF5A131B1D73}`

Now *()*	Returns a *dateTime* value representing the local time of the host machine on which srTool is running.
	Example:
	`echo -x now ()`
	`2/5/2004 21:31:46.230`

OutputOf *(inCommands)*	Returns the standard output that results from executing the commands stored in *inCommands* in a separate srTool command shell.
	Example:
	`a = OutputOf ("verbose = true; get name, jobstate of first 3 jobs of first rms"); echo -x a`
	`3 objects:`
	`Name JobState`
	`----------- --------`
	`Untitled 90 Canceled`
	`Untitled 35 Canceled`
	`Untitled 56 Canceled`

Random *()*	Returns a pseudo-random *int32* value between -32767 and +32767.
	Example:
	`echo -x random ()`
	`2341`

Function Name	Description
ReadFile *(inFilePath)*	Returns a string that contains the contents of the specified file. Example: **`set a = ReadFile("c:\\data.txt")`** **`echo -x a`** **`This is a test`** Provides the contents of this file. Note: Character escaping syntax is observed and may give unexpected results if the user is not aware of this.
Replace *(inText, inSearchText, inReplacementText [, inCount])*	Replaces every occurrence of *inSearchText* in *inText* with *inReplacementText*, unless *inCount* is specified, in which case the number of replacements made is capped by that value. Example: **`echo -x replace("abcdeabcdeabcde", "a", "Z", 2)`** **`ZbcdeZbcdeabcde`**
Right *(inText, inNumChars)*	Returns one or more of the right-most characters of *inText*. The number of characters to be returned is determined by *inNumChars*. Example: **`echo -x Right("Hello World!", 8)`** **`o World!`**
SizeOf *(inValue)*	Returns the size of *inValue*, in bytes, or in the case of text data, in characters. Example: **`loop for a in "Hi", 12, 0.1, NewID (), Now ()`** **`  echo -x SizeOf (a)`** **`end`** **`2`** **`4`** **`8`** **`16`** **`8`**
SubString *(inText, inStartingPos, inDesiredLength)*	Returns a sub-string from *inText*, starting in the character position determined by the value of *inStartingPos*, whose length, in characters, is determined by the value of *inDesiredLength*. Example: **`echo -x SubString("one two three four five", 5, 4)`** **`wo t`**

Function Name	Description

ToAscii *(inCharacter)*	Returns a numeric value that represents the character code of the first character of *inCharacter*.
	Example:
	`echo -x ToAscii("a")`
	`111`

ToBitString *(inValue)*	Returns a string that contains the bit-string encoding of *inValue*.
	Example:
	`echo -x ToBitString ("TEST")`
	`0b01010100000000000100010100000000010100110000000`

ToChar *(inNumber)*	Returns a string that contains the character whose code is *inNumber*.
	Example:
	`echo -x ToChar(112)`
	`p`

ToHexString *(inValue)*	Returns a string that contains the hex-string encoding of *inValue*.
	Example:
	`echo -x ToHexString(15)`
	`0x0000000F`

ToLower *(inText)*	Returns *inText* folded to lower case.
	Example:
	`echo -x ToLower("This is a Test")`
	`this is a test`

ToUpper *(inText)*	Returns *inText* folded to upper case.
	Example:
	`echo -x ToUpper("This is a Test")`
	`THIS IS A TEST`

Function Name	Description

TypeOf *(inValue)*	Returns the name of the argument's data type as a mixed-case string.
	Example:
	`loop for a in 0x123456789ABCDEF0123,"Hi", 12, 0.2, NullID`
	`echo -x TypeOf(a)`
	`end`
	`byteArray`
	`string`
	`uint32`
	`float`
	`uniqueID`

WriteFile *(inFilePath, dataToWrite)*	The **WriteFile** function writes the given data into the specified file. When writing anything but "byteArray" (or "blob") data, **WriteFile** always writes its data as Unicode strings. For "byteArray" data, **WriteFile** always writes raw binary data.
	For example, to perform a binary copy of the file "foo.mp3" to "bar.mp3" in the current directory:
	`echo -x WriteFile ("bar.mp3", ReadFile ("foo.mp3", ByteArray))`
	To copy the text file "foo.txt" to "bar.txt":
	`echo -x WriteFile ("bar.txt", ReadFile ("foo.txt"))`
	Another example:
	`echo -x WriteFile("c:\\test.txt", "This is a test")`
	`!"type c:\\test.txt"`
	Returns the following:
	`T h i s  i s  a  t e s t`

Creating Functions

To create your own functions, use the ***function*** command. See "`function` command" on page 100.

Deleting Functions

To delete user-created functions, use the ***delete*** command. See "`delete` command" on page 87.

Note You cannot delete any of the built-in functions.

Macros and Embedded Commands

Macros

srTool command shell variables can be used to make "macro" replacements in the srTool command line.

A macro replacement is only performed in the context of a single command, and as such, must itself not contain any command delimiters (semicolon characters). Whatever ultimately gets executed must remain a single command.

Example:

This example displays only the read-only properties of all replication jobs.

```
propList = Replace (OutputOf ('get name of all properties whose
access EQ ReadOnly of ObjectKind "FileReplicationJob"'), "\n",
",") - ","

get %propList of all jobs
```

Embedded Commands

By enclosing srTool commands in back-quote characters, the output of the command(s) can be used as arguments passed to other srTool commands.

The characters that replace the embedded command in the currently executing command must not contain any command delimiters (semicolon characters). Whatever gets executed must remain a single command.

The embedded command runs in its own separate command shell of the same type as its parent (that is, a high-level shell will use another high-level shell). The embedded command's shell inherits most of the parent shell's variables, as well as its default object specification.

Example:

This example lists all properties of all replication jobs *except for the read-only ones*.

```
list -omit `get name of all properties whose access EQ ReadOnly
of objectkind "FileReplicationJob"` of all Jobs
```


Objects

srTool's objects represent entities in the replication system that may be interrogated and directed to perform certain operations. There are many objects in the *Replication Exec* system, such as servers, jobs, alerts, and so on.

The ***show objects*** command shows the complete set of replication system objects in a hierarchical fashion.

srTool Object Hierarchy

Following is the current object hierarchy of srTool.

RMS
- Server
  - LogEntry
  - License
  - Volume
    - Folder
      - Folder
      - File
      - SubItem
    - File
    - SubItem
- FileReplicationJob
  - SourceServer
    - LogEntry
    - License
    - Volume
      - Folder
        - Folder
        - File
        - SubItem
      - File
      - SubItem
  - TargetServer
    - LogEntry
    - License
    - Volume
      - Folder
        - Folder

File
SubItem
File
SubItem
ReplicationPair (pair)
LogEntry
Script
PathRule
SelectionRule (selRule)
DestinationRule (destRule)
LogEntry
Alert
Credential
ObjectKind
Property

Note Names in parenthesis (above) indicate aliases of the objects.

Objects contain information in the form of Properties. For example, the name of a replication job is a property of the job object. Some objects can also contain other objects. The list above shows the containment hierarchy.

The ***list*** **all objectKinds** command displays the complete set of replication system objects in a table.

Root-Level Objects

Root-level objects are situated at the top of the object ownership hierarchy, and they vary, depending on which Client Interface is used by the srTool command shell. The default srTool command shell uses the "High-Level Client Interface".

The High-Level Interface supplies three root-level objects: *RMS, Credential* and *ObjectKind*.

Discovering Objects

To discover what object instances exist of a particular kind, you can ***List*** them and their properties, ***Count*** them or ***Select*** expressions using their properties:

```
list all servers

count every server

get name, address of all servers

select "\\\\" + name + "\\C$" from every server
```


Creating New Objects

To add a new object, use the ***add*** (or ***create***) command:

```
add job with type = OneToOne
```

Deleting Objects

To delete an existing object, use the ***delete*** command:

```
delete every pair whose name startsWith "A" of job "Milan"
```

See also:

"add command" on page 73
"count command" on page 86
"delete command" on page 87
"list command" on page 106
"select command" on page 117
"show command" on page 122
"Objects" on page 64

Object Properties

There are many object properties in the *Replication Exec* system. Use the ***list*** **every property of every objectKind** command to display them all.

Access Types

Each property has an "access" attribute that determines whether or not it can be changed in srTool.

Access	Description
Constant	The property value never changes during the life of the object. An example of this is the **ID** property of an object.
ReadOnly	The property value cannot be directly changed using the ***set*** command, but is useful for reporting on the status of some aspect of the replication system. For example, the *JobState* property of a **job** object can tell you if the job is running or not.
Mutable	The property value can be changed freely. For example, the **Description** property of a job can be set to any desired text.

Querying Property Values

To inspect all of the properties of an object, use the ***list*** command without specifying any specific properties. For example:

```
list any server
```

To show only those properties of specific interest, use the ***list*** command, and specify those properties in the order they should be viewed. For example:

```
get name, jobstate of every job
```

Modifying Property Values

To change a mutable property's value, use the ***set*** command. For example:

```
set nameSpec of first selRule of first Rule of job "Roma" to
"*.DOC"
```

Discovering Properties

To find the names of all the properties for a specific object type, use the ***list*** command. For example, to show the name of all properties of ReplicationPair objects:

```
get name of all properties of objectKind "ReplicationPair"
```

See also:

"srTool Object Reference" on page 135.

Object Specifications

Object specifications allow you to specify the objects in the replication system you wish to apply to the action inferred by the command verb.

Object specifications only refer to one or more objects of a single type. This means, for example, that you can't refer to servers and jobs within the same object specification.

To refer to objects that belong to other objects, *compound object specifications* must be used, which are built from the simple object specifications.

Syntax:

GroupingSpec ObjectKind [*WhoseClause*] [*SortedByClause*]

...or...

ObjectKind IndexingSpec [*WhoseClause*] [*SortedByClause*]

...or...

ObjectKind {*stringLiteral* | *uniqueIDLiteral*} [, ...]

- *ObjectKind* refers to the kind of object of interest, for example **jobs**, **servers**, and so on. It can be specified in singular or plural form.
- The last form is a kind of shorthand for conveniently referring to a few objects by name or ID. For example:

```
jobs "Roma", {4EE727B0-AB6F-11d4-A007-00C04F3F7867},
"Brussels"
```

Note For shorthand, names can be matched exactly or inexactly, depending on the value of the built-in shell variable *inexactShorthand*.

- It is illegal to use a *GroupingSpec* and an *IndexingSpec* in the same object specification.

Note If the **SortedBy** clause is not specified, the object ordering is indiscriminant.

- When resolving object specifications into a resulting ordered list of objects, *WhoseClause* filtering is performed first (if any), followed by sorting (if any), followed by grouping or indexing (if any).

See also:

"Compound Object Specifications" on page 72
"Grouping Specifications" on page 68
"Indexing Specifications" on page 70
"Objects" on page 64
"SortedBy Clause" on page 71
"srTool Object Hierarchy" on page 64
"Whose Clause" on page 71

Grouping Specifications

A grouping specification allows you to specify one or more objects relative to other objects in a group.

Syntax:

{**all** | **every**}

...or...

{**first** | **last** | **middle** | **any**} [*expression*]

In the latter form, the expression specifies the number of objects being requested. If the expression is omitted, it is assumed that only a single object is requested.

Keyword	Meaning
all, **every**	Results in all existing objects.
first	Results in any number of objects chosen from the top of the object list.
last	Results in any number of objects chosen from the bottom of the object list.
middle	Results in any number of objects chosen from around the middle of the object list.
any	Results in any number of randomly-selected objects from the object list.

Examples:

```
all jobs

first 5 selectionRules

any pair
```

See also:

"Expressions" on page 50

Indexing Specifications

An indexing specification allows you to explicitly specify one or more objects by index position or index range.

Syntax:

expression [{~ | **to** | **thru** | **through**} *expression*] [, ...]

An index value of zero refers to the first (or top-most) object in the list.

Using any of the **~**, **to**, **thru** or **through** keywords between two expressions indicates a range of objects.

A range that has identical border index values (for example, '5 thru 5') reduces to a single index, and therefore, a single object.

A reversed range (that is, a larger index value on the left side of the **thru** keyword than the value on the right side of it) is treated as if it were not reversed. (Thus, '6 thru 2' is treated as '2 thru 6'.)

It is not an error to specify index positions for which there are no corresponding objects.

You may freely overlap index ranges and repeat index values or ranges without incurring performance penalties or errors. srTool reduces the indexing specification to its smallest possible form before communicating the request to the server. For example, '100 thru 300, 200 thru 400, 300 thru 500' is automatically reduced to '100 thru 500'.

Example:

```
servers 6, 5 ~ 7, 5 thru 6, 9 to 11 of first context
```

This specification yields servers 5, 6, 7, 9, 10 and 11.

See also:

"Expressions" on page 50

SortedBy Clause

The **sortedBy** clause is used to specify the order of the objects that result from an object specification.

Syntax:

sortedBy { [*sortDirection*] *propertyName* } [, ...]

...where *sortDirection* must follow this syntax:

ascending | **descending** | **-** | **+**

Specifying **ascending** (or **+)** causes objects with property values of lesser magnitude to appear earlier in the resulting sequence. Specifying **descending** (or **-**) causes objects with property values of greater magnitude to appear earlier.

If the sort direction is not specified, the ordering for the property is assumed to be **ascending**.

Examples:

```
sortedBy ascending name, descending created

sortedBy TimeStamp, messageText
```

Note If the SortedBy clause is not specified, the object ordering is indiscriminant.

See also:

"srTool Object Reference" on page 135
"Object Properties" on page 66

Whose Clause

The **whose** clause is used to refer to a subset of objects that match some user-defined criteria.

Syntax:

whose *expression*

For each object that causes the *expression* to result in a logically "true" value (or non-empty or non-zero or non-null), that object will be included in the resulting object set.

Example:

```
whose name endsWith "xls" or runStage is Dynamic
```

See also:

"Expressions" on page 50
"Object Properties" on page 66

Compound Object Specifications

Based on the object hierarchy, a number of objects belong to other objects. For example, a job object can own a pair object. To specify an object that belongs to another, you must use a compound object specification.

Because these specifications can often grow lengthy, the srTool command shell can utilize an *implicit* or default compound object specification for all object specs that are specified on the srTool command line. The ***use*** command is used to inquire about or change the implied parent object specification.

Syntax:

ObjectSpec [**of** *ObjectSpec*] [...]

A compound object specification is one or more simple object specifications separated by the "**of**" keyword.

Example:

```
first rule of job "Roma"
```

This compound object specification refers to...

the first path rule object that is owned by...

the (first) job object whose name is "Roma" that belongs to ...

the first RMS object (implicitly).

See also:

"srTool Object Hierarchy" on page 64
"`use` command" on page 129

srTool Command Reference 4

This section provides a detailed description of each srTool command, including their description, syntax, aliases, required and optional parameters and examples, as applicable. The commands are presented in alphabetical order.

add command

The ***add*** command is used to create and initialize new objects in the replication system.

Syntax:

add [*objectCount*] *objectKind*

[**to** *compoundObjectSpec*]

[{**with** | **set** | **setting**} *propertyAssignmentList*]

Aliases:

create, new, make

Required Parameters:

Parameter	Description
objectKind	This specifies the kind of object(s) that will be created.

Optional Parameters:

Parameter	Description
objectCount	An expression that must result in an unsigned integer value that specifies the number of objects to create. If the *objectCount* parameter is omitted, only one object will be created.
to *compoundObjectSpec*	Specifies the parent object(s), if any, that will contain the new object(s).
{with | set | setting} *propertyAssignmentList*	*propertyAssignmentList* is ... *propertyAssignment* [, ...] where *propertyAssignment* is ... *propertyName = expression* This allows the user to specify the initial values of one or more properties for the new object(s).

Note Some properties must be specified when creating certain objects; others cannot be specified; and some are optional. See “Creating Properties” on page 157.

Examples:

To create four new, one-to-one replication jobs:

```
create 4 jobs with type = OneToOne
```

To create a one-to-many job named "My Job":

```
add job with type=OneToMany, name="My Job"
```

To add the new replication pair “SRC:TARG” to the job named “Foo”:

```
add pair to job "Foo" with sourceServer = "SRC",
targetServer = "TARG"
```

See also:

"Expressions" on page 50
"Object Specifications" on page 67
"Objects" on page 64

begin command

The ***begin*** command starts a new block of commands that has its own local variable (execution) context. The block is terminated with a corresponding ***end*** command.

Syntax:

begin

Aliases:

none

Required Parameters:

none

Optional Parameters:

none

Examples:

This example demonstrates the scope of a variable named **x**.

```
x =;            ## Undefine x
? x;            ## x? Never heard of it!
begin;          ## Start new execution context
     x = 3.4;   ## Define new local variable x
     ? x;       ## x? Sure, I've heard of it!
end;            ## x now goes out of scope
? x;            ## x? Never heard of it!
```

See also:

"Execution Contexts" on page 41
"end command" on page 96

break command

The ***break*** command terminates the nearest ***loop***, resuming command execution at the first command past the ***loop***'s end.

Syntax:

break

Aliases:

none

Required Parameters:

none

Optional Parameters:

none

Examples:

In this example, on average, there will usually be about ten 'x' characters written to the standard output stream.

```
counter = 15
if counter LT 10
    comment -- This will be skipped
else
    echo %counter is greaterThan or EQ to 10
end if
```

See also:

"loop command" on page 108

call command

The ***call*** command is used to execute a set of srTool commands that are stored in a text file in the temporary execution context of a new srTool command shell. When the commands in the file have finished executing, the temporary shell is destroyed, and execution of commands from within the calling shell is resumed.

Note Scripts that are ***call***ed can be nested to a maximum level that typically cannot exceed 64.

Parameter data is passed to script files in a set of two or more variables, all of which have names that start with "param". See “Variables” on page 41 for more information.

The new command shell inherits most of the calling shell’s default object specification and most of its variables.

Syntax:

call [**-hlsob** | **-nointerface**] *filePathString* [*constParameter* [...]]

In this form, the parameters are passed verbatim to the called subroutine.

call [**-hlsob** | **-nointerface**] *filePathString* [(*expression* [,...])]

In this form, each parameter is assumed to be an expression that is evaluated with each result being passed to the subroutine.

Aliases:

@

Required Parameters:

Parameter	Description
filePathString	This string constant must contain a valid path to a file that contains the commands to be executed. Note: If the file name is "**con**" or "**tty**", commands will be read from the standard input stream.

Optional Parameters:

Parameter	Description
-n **-no** **-nointerface**	This switch specifies that the command interpreter to be used by the ***call***ed srTool script utilizes no client interface. The ***call***ed script would not be able to inquire about or control any aspect of the replication system. Note: This option is mutually exclusive to the **-h[l[sob]]** option.
-h **-hl** **-hlsob**	This switch specifies that the command interpreter to be used by the ***call***ed srTool script utilizes the high-level client interface. Note: This option is mutually exclusive to the **-n[o[interface]]** option.
constParameter	This specifies the constant parameter data to be passed to the called srTool script. There is no limit to the number of constants that can be passed to the script. Each parameter must be separated from the next by at least one space. The parameter data is passed to the script file in a set of two or more variables, all of which have names that start with "param". See “Variables” on page 41.
expression	This expression determines the parameter data that is to be passed by value (never by reference) to the called srTool script. There is no limit to the number of parameters that can be passed to the script. Each parameter must be separated from the next by a comma. The parameter data is passed to the script file in a set of two or more variables, all of which have names that start with "**param**". See “Variables” on page 41.

Examples:

In the following example, the srTool commands that are stored in the file *ThrottleDown.txt* are executed in the context of a new srTool command shell. The script will inherit all of the calling shell's variables, its default object specification, and will have these additional variables available to it:

paramCount will contain the value 2.
param0 will contain the 'string' value "ThrottleDown.txt".
param1 will contain the 'count64' value 50.

```
call "ThrottleDown.txt" 50
```


See also:

"Variables" on page 41
"Expressions" on page 50
"exec command" on page 97
"spawn command" on page 124

cancel command

The ***cancel*** command immediately stops execution of one or more jobs. This differs substantially from the ***stop*** command, which allows the job to finish synchronizing its targets, and have its targets play out their incoming change journals. It is acceptable to ask a job that is not running to cancel, and to ask a job that is in the process of canceling to cancel.

To manually start the job running again, use the ***start*** command.

Syntax:

cancel *compoundObjectSpec* **[-target]**

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	A compound object specification that results in one or more job objects

Optional Parameters:

Parameter	Description
-target	Commands the target server(s) used in the specified job(s) to cancel, which is useful when the source server(s) are not functioning. The default is to command the source server(s) of the job(s) to cancel.

Example:

In this example, jobs that were running with errors would be immediately stopped.

```
cancel every job whose jobState is runningWithErrors
```

See also:

"Compound Object Specifications" on page 72
"`start` command" on page 127
"`stop` command" on page 128

check command

The ***check*** command tells whether or not a set of files or folders will be replicated in one or more replication jobs. The command emits a list of full (absolute) path specifications of files or folders (or both), each preceded by a '+' or a '-' to the standard output stream. The '+' indicates the file (or folder) will be replicated; the '-' indicates it will not.

Syntax:

check [**-noExcl**[**udes**]] *jobObjectSpec* **for** *fileOrFolderObjectSpec* [, ...]

Aliases:

none

Required Parameters:

Parameter	Description
jobObjectSpec	A compound object specification that must result in one or more job objects. These are the jobs against whose rules (pathRules and selectionRules) will be used to test each file or folder.
fileOrFolderObjectSpec	A compound object specification that must result in one or more files or folders (or subfiles, subfolders, items, or subItems), that will each be tested for inclusion in the replication job(s).

Optional Parameters:

Parameter	Description
-noExcludes **-noExcl**	An optional keyword parameter that, if specified, will suppress the listing of files or folders that are excluded from replication (that is, only *included* files will be reported).

Examples:

In this example, a complete report will be generated that tells whether or not every file and folder on the "C" drive of the server named "Milan" will be replicated by the "Italy" job.

```
check job "Italy" for all subItems of vol "C:" of server
"Milan"
```

This example is identical to the previous example, except that it will only show replicated files that end in ".DOC" that are on every volume of that same server.

```
check -noexcl job "Italy" for every subFile whose name
endsWith ".DOC" of every volume of server "Milan"
```

See also:

"Compound Object Specifications" on page 72
"FileReplicationJob Objects" on page 142
"File Objects" on page 140
"Folder Objects" on page 146
"SourceServer Objects" on page 171
"SubFolder Objects" on page 176

comment command

The ***comment*** command is used only to adorn or document an srTool command script, and is completely ignored.

Syntax:

comment [*anything* [...]]

Aliases:

remark, ***rem***, ***#***

Required Parameters:

none

Optional Parameters:

Anything other than a semicolon.

Examples:

```
Comment -- This is a very simple comment.

#######################################
## This is a banner-style comment. ##
#######################################

##
## This is a banner-style comment.
##

rem This is a remark-style comment.
```

configure command

The ***configure*** command gets or sets configuration settings for various client-side components of the replication system.

Syntax:

configure [*componentName* [*configParamList*]]

Aliases:

config

Required Parameters:

none

Optional Parameters:

Parameter	Description
componentName	An identifier that is the name of the software component (for example, driver) of interest. If omitted, or if the keyword **all** is used, the command will apply to all available components.
configParamList	A comma-delimited list of one or more configuration parameter names that is optionally followed by an assignment clause. The list's syntax is as follows: {*parameterName* [= *expression*]} [, ...] A parameter name followed by an equal sign (=) must be followed by an expression, the result of which will be assigned to the named parameter in each chosen component. A parameter name that is not followed by an equal sign will display the value of that named parameter as obtained from the component. Parameter names not recognized by the software component(s) will be reported with a warning.

Examples:

This example displays the names of all available client-side components.

```
configure
```

This example displays all available configuration information for all available client-side components.

```
config all
```

This example turns up the debug logging level for the job cache component to the maximum level.

```
configure job debugLevel = 3
```

This example turns off all debug logging for all client-side components. (Note that the srTool command shell has its own debug logging level, which is controlled by the shell variable *debugLevel*.)

```
config all debugLevel = 0
```

See also:

"Expressions" on page 50

continue command

The ***continue*** command suspends execution of all intervening commands until the ***end*** of the nearest ***loop*** is encountered, at which point command execution is restored to the state when the ***continue*** command was encountered.

Syntax:

continue

Aliases:

none

Required Parameters:

none

Optional Parameters:

none

Example:

In this example, on average, there will be about ten 'x' characters written to the standard output stream.

```
loop 20 times
   if (random() mod 2) EQ 0
      continue
```


```
      end if
      echo -n x
   end loop
```

See also:

"loop command" on page 108

count command

The ***count*** command emits a count of the number of objects that were specified to the standard output stream.

Syntax:

count *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	Specifies the object(s) to be counted.

Optional Parameters:

none

Examples:

To display the number of jobs that have no alerts:

```
count every job whose AlertCount is 0
```

See also:

"Compound Object Specifications" on page 72

delete command

The ***delete*** command deletes any number of objects of the same kind. It is also used to delete functions or global variables.

Caution srTool does not provide a warning prior to deleting anything with this command!

Syntax:

delete *compoundObjectSpec*

This form of the command is for deleting objects.

delete {**function** | **global**} {*functionName* | *variableName*} [, ...]

This form of the command is for deleting functions or global variables.

Aliases:

del

Required Parameters

Parameter	Description
compoundObjectSpec	Specifies the object(s) to be deleted.
functionName [, ...]	An identifier that specifies the name of each function to be deleted.
variableName [, ...]	An identifier that specifies the name of each global variable to be deleted.

Optional Parameters:

none

Examples:

To delete every replication pair that is in the job named "My Job":

```
delete every pair of job "My Job"
```

To delete the user-defined functions named "NumberOfRunningJobs" and "BadPairs":

```
delete function NumberOfRunningJobs, BadPairs
```

To delete the global variables named "gPrimaryServerName" and "TheRMS":

```
del global gPrimaryServerName, TheRMS
```

See also:

"Compound Object Specifications" on page 72
"`function` command" on page 100
"`global` command" on page 103

demote command

The ***demote*** command demotes one or more selection rules, such that they are considered after other selection rules while qualifying files for replication. To demote a selection rule to the bottom, it may be necessary to use this command several times, if there are many selection rules that are inferior to it.

To promote a selection rule, use the ***promote*** command.

Syntax:

demote *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	Specifies one or more selection rule objects

Optional Parameters:

none

Example:

This example demotes all selection rules that specify "*.DOC" for all path rules of all jobs.

```
demote all selRules whose nameSpec is "*.DOC" of all rules of
all jobs
```

See also:

"Compound Object Specifications" on page 72
"promote command" on page 114
"SelectionRule Objects" on page 166

disable command

The ***disable*** command disables one or more servers. A disabled server cannot participate in replication.

Syntax:

disable *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	Specifies one or more server objects.

Optional Parameters:

none

Examples:

In this example, servers that had the name "ADMIN East" and "ADMIN West" would be disabled.

```
disable every server whose name startsWith "ADMIN"
```

See also:

"Compound Object Specifications" on page 72
"enable command" on page 95
"Server Objects" on page 168

dump command

The ***dump*** command recursively emits all known property information about all known objects to the standard output stream as a series of ***add*** and/or ***set*** commands. This provides a way for administrators to reconstruct or duplicate a replication system with a minimum of manual intervention.

Syntax:

dump [*compoundObjectSpec*] [**-omit** *objectKind* [, ...]] [**-a**[**ll**]]

Required Parameters:

none

In the default case, this command dumps all objects — except for **servers, volumes, folders**, **files**, **alerts**, **logEntries, objectKinds**, and **properties** — to the standard output stream. To include these omitted objects in the dump, use the **-all** option (see below).

Optional Parameters:

Parameter	Description
compoundObjectSpec	This option restricts the dump to incorporate just those objects that are specified, and any objects that belong to them (and so on, recursively), subject to the default or explicit exclusions (that is, **servers, volumes, folders**, **files**, **alerts**, **logEntries**, **objectKinds**, and **properties** unless **-all** is specified).
-omit *objectKind* [, ...]	This option allows you to specify which kinds of contained objects will be excluded from the dump. By default, only **servers, volumes, folders**, **files**, **alerts**, **logEntries**, **objectKinds**, and **properties** are excluded.
-a[**ll**]	This option includes **servers, volumes, folders**, **files**, **alerts**, **logEntries**, **objectKinds**, and **properties** in the dump unless specifically excluded with the **-omit** option.

Examples:

This example dumps all objects (except **servers, volumes, folders**, **files**, **alerts**, **logEntries**, **objectKinds**, and **properties**), to the standard output stream.

```
dump
```

This example dumps only the job named "Foo" and all of its sub-objects (except **licenses**, **volumes**, **servers**, **folders**, **files**, **alerts**, **logEntries**, **objectKinds**, and **properties**) to the file "fooRestore.txt".

```
dump job "foo" -omit license >"fooRestore.txt"
```

See also:

"Compound Object Specifications" on page 72
"srTool Object Reference" on page 135

echo command

The ***echo*** command emits whatever is on the command line to the standard output stream.

Syntax:

echo [[**-n**] [*anything* [...]] [**-x**] [*expression*[, ...]]]

Aliases:

none

Required Parameters:

n/a

Optional Parameters:

Parameter	Description
-n	This option prevents **echo** from emitting a newline sequence at the end of the line written to the output stream.
anything	Any srTool token except for the semicolon. The tokens are passed through uninterpreted and unchanged to the standard output stream with one difference: all whitespace (if any) between each successive token is compressed into a single space, or if there was no space between them, is expanded to a single space.
-x	This option causes **echo** to interpret the rest of its command line parameters as a comma-delimited list of expressions.
expression	An expression that is evaluated, then converted to a string and sent to the output stream.

Examples:

```
echo Jane's wagon isn't broken.
   Jane s wagon isn t broken .
echo "Jane's wagon isn't broken."
   Jane's wagon isn't broken.
```

```
echo -x (355.0 / 113.0) as string + "***", now () + 5432 as
timespan
```

```
3.14159***Thu Jan 17 11:07:51 2002
```

See also:

"Expressions" on page 50

else command

The ***else*** command terminates a conditionally executed block of commands, and begins a final one that must itself be terminated by an ***end*** command.

Syntax:

else

Aliases:

none

Required Parameters:

none

Optional Parameters:

none

Examples:

In this example, the ***echo*** command will be executed.

```
counter = 15
if counter LT 10
     comment -- This will be skipped
else
     echo Counter is greater than or equal to 10
end if
```

See also:

"elseif command" on page 94
"if command" on page 105

elseif command

The ***elseif*** command terminates a conditionally-executed block of commands, and begins another block that must be terminated by another ***elseif*** command, or by an ***else*** or ***end*** command.

Syntax:

elseif *expression*

Aliases:

none

Required Parameters:

Parameter	Description
expression	The expression to be evaluated. If the result of the evaluation is not "0" (zero) or not empty, the commands immediately following the ***elseif*** command will be executed.

Optional Parameters:

none

Examples:

In this example, the ***echo*** command will be executed.

```
counter = 5
if counter GT 10
   comment -- This will be skipped
elseif counter LT 6
   echo Counter is less than 6
end if
```

See also:

"else command" on page 93
"end command" on page 96
"if command" on page 105

enable command

The ***enable*** command enables one or more servers. An enabled server can participate in replication.

Syntax:

enable *compoundObjectSpec*

Aliases:

none

Required Parameter

Parameter	Description
compoundObjectSpec	Specifies one or more server objects.

Optional Parameter:

none

Examples:

In this example, servers that had the name "ADMIN East" and "ADMIN West" would be enabled.

```
enable every server whose name startsWith "ADMIN"
```

See also:

"Compound Object Specifications" on page 72
"disable command" on page 89
"Server Objects" on page 168

end command

The ***end*** command terminates an execution context, such as a ***loop***, an ***if/else/elseif*** construct, or a function definition.

Any non-global variables that were defined in the context that is terminated by this command are deleted once execution proceeds past this command.

Syntax

end [*anything* [...]]

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
anything	Any srTool token other than a semicolon. It is recommended that script authors utilize these tokens to describe which block is being terminated by the ***end*** command.

Examples:

Notice the wise practice of putting '**if**' in the ***end*** command, to document that the ***end*** command terminates the **if**.

```
counter = 5
if counter LT 10
   echo Counter is less than 10
end if
```

Notice the wise practice of putting ***loop*** in the ***end*** command to document that the ***end*** command terminates the ***loop***.

```
loop 20 times
   add job with type = OnetoOne
end loop
```

See also:

"`begin` command" on page 76
"`else` command" on page 93
"`elseif` command" on page 94
"`if` command" on page 105
"`loop` command" on page 108
"Execution Contexts" on page 41

exec command

The ***exec*** command is used to execute a set of commands that are stored in a text file in the execution context of the current srTool command shell. When the commands in the file have finished executing, the shell resumes reading subsequent commands from the original input stream.

Parameter data is passed to script files in a set of two or more variables, all of which have names that start with "param". See "Variables" on page 41 for more information.

Syntax:

exec *filePathString* [*constParameter* [...]]

In this form, the parameters are passed verbatim to the called script file.

exec *filePathString* [(*expression* [,...])]

In this form, each parameter is assumed to be an expression that is evaluated with each result being passed to the script file.

Aliases:

none

Required Parameters:

Parameter	Description
filePathString	This string constant must contain a valid path to a file that contains the commands to be executed. Note: If the file name is "**con**" or "**tty**", commands will be read from the standard input stream.

Optional Parameters:

Parameter	Description
constParameter	This specifies the constant parameter data to be passed to the called srTool script. There is no limit to the number of constants that can be passed to the script. Each parameter must be separated from the next by at least one space. The parameter data is passed to the script file in a set of two or more variables, all of which have names that start with "param". See “Variables” on page 41.
expression	This expression determines the parameter data that is to be passed by value (never by reference) to the called srTool script. There is no limit to the number of parameters that can be passed to the script. Each parameter must be separated from the next by a comma. The parameter data is passed to the script file in a set of two or more variables, all of which have names that start with "**param**". See “Variables” on page 41.

Examples:

This example executes the commands that are stored in the file `restore.txt` in the context of the current command shell. The script will have these additional variables available:

paramCount will contain the value 1.
param0 will contain the 'string' value "restore.txt".

```
exec "restore.txt"
```

See also:

“Variables” on page 41
“Expressions” on page 50
“`call` command” on page 78
“`spawn` command” on page 124

flush command

The ***flush*** command is used to flush the client software's internal caches.

Syntax:

flush [*objectKind* [,...]]

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
objectKind [,...]	One or more objectKinds, separated by commas. If no parameter is specified, all caches are flushed.

Examples:

flush

This example flushes all of the client's internal caches.

function command

The ***function*** command starts a new block of commands that will be used to define a new function or redefine an existing function.

All functions return a result through the local variable *returnValue*. If no value is assigned to this variable, the function's result will be zero (an unsigned integer).

Syntax:

function *functionName* ([*argumentVariableName* [, ...]])

Aliases:

none

Required Parameters:

Parameter	Description
functionName	This must be a valid function name. It identifies the function that will be called when it is used in an expression. Using the same name as an existing function will redefine that function.

Optional Parameters:

Parameter	Description
argumentVariableName	This must be a valid variable name. It identifies the parameter value passed in to the function by the caller. All parameters are passed in by value and not by reference. There is no limit to the number of parameters that a function may employ.

Examples:

This example defines the function "max", which returns the largest of its two arguments.

```
function max (a, b)
   if a GT b
      returnValue = a
   else
      returnValue = b
   end if
end function max
```

This example defines the function rand, which returns a pseudo-random number that is evenly distributed between the specified *minValue* and *maxValue*.

```
function rand (minValue, maxValue)
   returnValue = (random () * (maxValue - minValue) /
0x7FFF) as integer + minValue
end function rand
```

Here's how the function could be called:

```
echo -x rand (3 * 10 + 4, "2000 widgets" as integer)
```

The echo command in this case would emit a pseudo-random number that is between 34 and 2000.

Built-in Functions

srTool has several built-in functions. See “Functions” on page 57.

See also:

“Expressions” on page 50
“Variables” on page 41

global command

The ***global*** command is used to display, declare or delete shell variables that are global in scope.

Syntax:

global [*identifierName* [,...] = [*expression*]]

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
identifierName	A valid srTool variable name. See "Variables" on page 3, for information about names and the rules associated with the naming of variables.
expression	If this parameter is omitted, the specified variables will be deleted. Otherwise, the specified variables will be assigned the value that results from the expression.

Examples:

This example emits a set of ***global*** commands to the standard output stream, one for each known global variable. This is useful for displaying the current values of all global variables.

```
global
```

This example assigns zero to both *myFlag* and *currentPosition* global variables. If the variables do not yet exist, they are created.

```
global myFlag, currentPosition = 0
```

This example deletes the global variable *myGlobal*.

```
global myGlobal =
```


See also:

"Expressions" on page 50
"Variables" on page 41
"set command" on page 118

help command

The ***help*** command provides information about nearly anything in srTool and its current environment. It can evaluate expressions, as well as provide information about a command, a data type, a property, a predefined property value, an object kind, expression operators, and so on

Help can also be obtained for any command by entering the command verb followed by the '*?*' character.

Syntax:

help [*topic*]

Aliases:

?, h

Required Parameters:

none

Optional Parameters:

Parameter	Description
topic	If omitted, the command brings up a description of srTool with links that lead the user through the online documentation. If specified, this parameter can contain nearly anything, including, but not limited to... ◆ a specific help topic (for example, variables, objects) ◆ an object kind (for example, **server**) ◆ a command verb (for example, ***wait***, ***list***) ◆ a built-in or user-defined shell variable, local or global in scope ◆ the name of a property (for example, **name**, **id**) ◆ the name of a data type (for example, **integer**, **dateTime**) ◆ an operator (for example, **+**, **GT**) ◆ the name of a built-in or user-defined function ◆ an expression enclosed in parenthesis (for example, `(now () GT WhenShellStarted)`)

Examples:

This example provides information about the ***call*** command.

```
help call
```

This example also provides information about the ***call*** command.

```
call ?
```

The amount of time elapsed since the job named "Blue" last started.

```
? (now () - lastStarted of job "Blue")
```

See also:

"`show` command" on page 122

if command

The ***if*** command begins a conditionally-executed block of commands, which is bounded by an ***else***, ***elseif*** or ***end*** command.

Syntax:

if *expression*

Aliases:

none

Required Parameters:

Parameter	Description
expression	The expression to be evaluated. If the result of the evaluation is not "0" (zero) or not empty, the commands immediately following the ***if*** command will be executed.

Optional Parameters:

none

Examples:

In this example, the ***echo*** command will be executed.

```
counter = 5
if counter LT 10
   echo Counter is less than 10
end if
```

See also:

"else command" on page 93
"elseif command" on page 94
"end command" on page 96

list command

The ***list*** command emits property data from one or more objects to the standard output stream.

Syntax:

list [**-noTable**] [[**-omit**] *propertyList* [**of**]] [*compoundObjectSpec*]

Aliases:

`ls, get`

Required Parameters:

none

Optional Parameters:

Parameter	Description
propertyList	A comma-delimited list of property names that identify which properties are to be shown, or if the **-omit** option is used, not shown in the listing (see below). By default, all properties of the resulting objects are displayed.
compoundObjectSpec	A compound object specification that specifies which objects are to be listed.
-noTable	This option causes the data to be displayed ragged, without aligned columns, and any property data of type **string**, **timespan** or **dateTime** will be enclosed in quotation marks. The **-noTable** option will use the shell's **fieldDelimiter** variable as the field delimiter that separates each property in the output, as well as the shell's **recordDelimiter** variable to separate each object displayed in the output.
-omit	The **-omit** option assumes that all properties are to be displayed except for those listed after the **omit** keyword.

By default, the resulting output is organized into a table, with property data appearing in columns, in the order that was specified. (A default order is used if no properties were specified.) If the shell's **verbose** variable is set to true (or any non-zero or non-null value), an object count will precede the table, and column headings will appear at the top of the displayed table. Note that if the resulting table is too wide to fit in your console window, each line will automatically wrap to the next, and will cause the table to appear garbled.

Examples:

This example lists every property except the "schedule" property of the first job found.

```
list -omit schedule first job
```

This example lists all jobs in tabular format, displaying all properties of each job.

```
list every job
```

This example lists just the two properties "name" and "OSVersion" for every server.

```
get name, OSVersion of every server
```

This example puts all pair information from the job named "Foo" into a tab-delimited text file, and then opens the file in Microsoft Excel.

```
savedFieldDelimiter = fieldDelimiter
fieldDelimiter = "\t"
savedRecordDelimiter = recordDelimiter
recordDelimiter = "\n"
get -noTable all pairs of job "Foo" >"FileToImport.txt"
fieldDelimiter = savedFieldDelimiter
recordDelimiter = savedRecordDelimiter
xl='"C:\\Program Files\\Microsoft Office\\Office10'
!"start " + xl + "\\excel\" FileToImport.txt"
```

See also:

"Compound Object Specifications" on page 72
"Object Properties" on page 66

loop command

The ***loop*** command begins a block of commands that may be repeatedly executed, depending upon the form of the command that is used.

Syntax:

loop [**forever**

| **while** *expression*

| *expression* **times**

| **for** *variableList* = *startExpression* **to** *endExpression* [{**step** | **by**} *incrExpression*]

| **for** *variableList* **in** *expressionList*]

| **over** *compoundObjectSpec*]

... where *variableList* is a comma-delimited list of variable identifier names that will be defined and assigned values as the loop executes; and

... where *expressionList* is a comma-delimited list of expressions that each will be successively evaluated and whose result is assigned to the loop variable(s) as the loop executes.

Note Any variables specified in the *variableList* must *not* be global in scope, nor match the names of any properties, nor be read-only. It is highly recommended that they not be defined in any other execution context.

Aliases:

repeat

Required Parameters:

none

Optional Parameters:

See Command Operation, below.

Command Operation:

Commands that are repeated are those that immediately follow the ***loop*** command, up to, but not including, the ***loop*** command's corresponding ***end*** command.

The ***loop*** command has five distinct forms:

- In the ***loop*** **forever** form of the command, or when the ***loop*** command appears without any other parameters (the default condition), results in an "infinite loop" or a loop that will not terminate without one of the following occurring:
  - a ***break*** command is executed;
  - the srTool process is terminated;
  - in the case of the loop executing in a ***spawn***ed task, the task is killed via srTool ***quit***ting or an explicit ***spawn -kill***;
  - a Control-C interrupt occurs.
- In the ***loop*** **while** form, the *expression* is evaluated once each time prior to re-entering the body of the ***loop***. If the *expression* results in a logically true (or non-zero or non-empty) value, the body of the ***loop*** is executed again.

- In the ***loop n* times** form of the command, the expression is evaluated only once prior to entering the body of the ***loop***. Even if the expression contains a variable whose value is changed inside the ***loop***, the number of times the ***loop*** body will execute is determined only once at the beginning.
- In the ***loop* for...to...step** form of the command, the *startExpression*, *endExpression* and *incrExpression* (if given) are evaluated once prior to entering the body of the ***loop***, and the *startExpression's* result is assigned into each of the ***loop*** variables. It is irrelevant if any of those expressions contain a variable whose value changes inside the ***loop***, because the expressions are evaluated only once at the beginning.
- In the ***loop* for...in** form of the command, each expression in the *expressionList* is evaluated once prior to entering the body of the ***loop***. Each time around the ***loop***, each of the ***loop*** variables is assigned the result of the next expression in the *expressionList*, in order.
- In the ***loop* over** form of the command, every property of each object that results from the query implied in the compound object specification is assigned to a variable whose name is the concatenation of "prop" and the name of the property. See the ***loop* over** example below.

Note Because the comma is used as a delimiter for index values in object specifications and in expression lists in this command, you should avoid using commas in expression lists because of the ambiguities that may result. For example, the following script will fail:

```
loop for x in name of job 0, ID of job 0
   ? x
end loop
```

The problem is that the first comma encountered in the ***loop*** statement is in an indexing specification, which is a list of either one expression or two separated by a "thru" keyword. The indexing specification accepts "ID of job 0" as another index value. However, its result, a GUID, will not convert to a 64-bit unsigned value, which is what the indexing specification is looking for.

The solution is to either avoid the use of indexing specification or to use parentheses to surround the expressions, as shown in the following examples.

```
loop for x in name of first job, ID of first job
   ? x
end loop

loop for x in (name of job 0), (ID of job 0)
   ? x
end loop
```

Examples:

In this example, the console window will indefinitely fill with 'x's. It can only be stopped by terminating the srTool process or interrupting it (via Control-C).

```
loop
   echo -n x
end
```

This example prints the 10 values of the "counter" variable in the standard output stream.

```
loop for counter = 0 to 9
   echo -x "counter=" + (counter as string)
end loop
```

In this example, the variable **str** finishes by being a string that contains 1,048,576 'x' characters.

```
str = "x"
loop 20 times
   str = str + str
end loop
```

In this example, the local variable **x** changes type and value each time through the loop.

```
loop for x in 1,pi,now(),now()-whenShellStarted, name of
first job,ID of job 0
   ? x
end loop
```

In this example, 15 pairs are added to a centralization job named "TestJob". Note that the properties of each of the 15 servers are treated as local variables inside the loop.

```
targ = name of server "HP"
jobID = id of job "TestJob"
loop over last 15 servers whose name startsWith "DELL"
   add pair to job %jobID with sourceServer = propName,
      targetServer = targ, throttle = If (PropOSMajorVersion
      EQ 4, 50, 100)
end loop
```

In this example, the last server in the server list is the target, and all other servers are sources for all jobs. Note that the properties of each server are referred to by a local variable whose name is "prop" followed by the property name.

```
loop over first 'count all servers of first RMS' - 1 servers
   add pair to all jobs whose type EQ ManyToOne with
sourceServer = propName,
   targetServer = name of last server
end loop
```

See also:

"continue command" on page 85
"begin command" on page 76
"break command" on page 77

monitor command

The ***monitor*** command is used to monitor object activity in the replication system. Creations and deletions of objects, and object changes are displayed in the srTool console window.

Syntax:

monitor {*compoundObjectSpec* [, ...] | {**list** | **show**} | {**stop** | **pause** | **resume**} {*indexingSpec* | **all** }

Notes:

- It is possible to actively monitor the same kinds of objects more than once, in which case the console window will receive duplicate messages for single add/change/delete events. srTool will detect identical object specifications, and prevent the creation of a duplicate; however, it is easy to compose different object specifications whose result sets match.
- Each monitored object specification (query) will be associated with an unsigned whole number that is unique to that query. That number is used to refer to the query being monitored in order to stop, pause or resume its monitoring.

Aliases:

mon

Required Parameters:

none

Optional Parameters:

Parameter	Description
compoundObjectSpec	Determines which objects will be monitored.
indexingSpec	Provides a list of one or more "monitor numbers" or ranges of "monitor numbers".
list	Displays what is being monitored and indicates the state of each query (whether paused or not, and, if paused, a count of the number of saved messages).
stop *indexingSpec*	Stops the specified monitor(s), permanently removing it from the active list. Any saved messages associated with paused monitor queries are lost.
pause *indexingSpec*	Pauses the specified monitor(s). Any messages associated with it (them) are saved for later playback using `resume`.
resume *indexingSpec*	Resumes the specified monitor(s). Any saved messages associated with it (them) are emitted to the shell's output stream.

Examples:

This example starts monitoring all job and server activities.

```
mon all jobs, all servers
```

This example monitors all pairs that use the target server "LOGAN01", and only for those jobs whose names begin with "Boston".

```
mon all pairs whose name endsWith ":LOGAN01" of every job
whose name startsWith "Boston"
```

This example immediately stops all monitoring completely.

```
mon stop all
```

Each of these examples will show what is currently being monitored.

```
monitor; monitor list; monitor show
```

This example pauses monitoring of the object(s) associated with the number 15, then immediately resumes monitoring them, as well as any paused monitors with the monitor values of 2, 3, 4 and 8.

```
monitor pause 15; monitor resume 8, 15, 2 thru 4
```


See also:

"Compound Object Specifications" on page 72
"Indexing Specifications" on page 70

pause command

Reserved for future use.

Syntax:

pause *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	The *compoundObjectSpec* must refer to one or more jobs.

Optional Parameters:

none

promote command

The ***promote*** command promotes one or more selection rules, such that they are considered before other selection rules while qualifying files for replication. To promote a selection rule to the top, it may be necessary to use this command several times if there are many selection rules that are superior to it.

To demote a selection rule, use the ***demote*** command.

Syntax:

promote *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	Specifies one or more selectionRule objects.

Optional Parameters:

none

Example:

This example promotes all selection rules that specify `"*.DOC"` for all path rules of all jobs.

```
promote all selRules whose nameSpec is "*.DOC" of all rules
of all jobs
```

See also:

"Compound Object Specifications" on page 72
"demote command" on page 88
"SelectionRule Objects" on page 166

quit command

The ***quit*** command terminates the currently running command shell. If the shell's nesting level is 1 and the shell was not created using the *spawn* command, srTool will terminate and return control to the host operating system.

Syntax:

quit

Aliases:

q, ***exit***

Required Parameters:

none

Optional Parameters:

none

Examples:

q

This terminates srTool and returns control to the operating system (assuming the command shell's nesting level is 1).

See also:

nestingLevel in "Built-in Variables" on page 43

resume command

Reserved for future use.

Syntax:

resume *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	The *compoundObjectSpec* must refer to one or more jobs.

Optional Parameters:

none

select command

The ***select*** command writes the result of one or more expressions, each computed from the values of one or more properties of each object that results from the object specification to the output stream.

Syntax:

select {*expression* [, ...]} [**from** *compoundObjectSpec*]

For each object that results from the object specification, each *expression* is evaluated in the context of that object, and the value that results from each *expression* is written to the standard output stream, each separated by the value of the shell's **fieldDelimiter** variable.

Successive objects are separated in the output stream by the value of the shell's *recordDelimiter* variable.

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
compoundObjectSpec	The object specification specifies the objects whose property data is to be used when evaluating the expressions or comparisons. It also determines the order of the resulting objects.
expression	Each expression that is specified determines what information is to be displayed for each resulting object. If more than one expression is specified, their results are separated in the output stream by the value of the shell's **fieldDelimiter** variable.

Examples:

This example lists the name of each running job and how long ago each was started.

```
select name, now () - lastStarted from all jobs whose
jobState EQ Running
```


This example displays the name of each running job and how long before the most recently started running job each was started. The jobs that were started closest to the most recently started one are listed first.

```
select name, (lastStarted of first job whose jobState EQ
Running sortedBy descending lastStarted) - lastStarted from
all jobs whose jobState EQ Running sortedBy lastStarted
```

See also:

"Expressions" on page 50
"Compound Object Specifications" on page 72

set command

The ***set*** command is used to change the value of a mutable property of one or more objects, or to add, change or delete shell variables (local or global).

Syntax:

set

This form displays all currently defined shell variables.

set *propertyName* [, ...] **of** *compoundObjectSpec* {**to** | =} *expression*

This form sets the values of any mutable properties of one or more objects.

set {*propertyName* = *expression*} [, ...] **for** *compoundObjectSpec*

This form sets the values of any mutable properties of one or more objects.

[***set***] *variableName* [, ...] = [*expression*]

This form adds, changes or deletes a shell variable. If the expression is omitted, the variable will be deleted.

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
compoundObjectSpec	Specifies which object(s) to modify.
expression	An expression whose result is the value stored in the object property or variable.
propertyName	The name of the mutable property of the existing object(s) to be changed.
variableName	The name of a mutable variable that will receive the expression results.

Examples:

This example emits the commands that would define the currently defined global and local shell variables to the standard output stream. Read-only variables are placed in comment commands. This is useful for displaying the current values of all variables, and also for saving and restoring their values (by redirecting the output into a file, then later executing the file using the ***exec*** command).

```
set
```

This example sets the *Description* property of a randomly selected job to the string value "For testing only".

```
set description of any job to "For testing only"
```

This example assigns the result of the expression 355.0 divided by 113.0 to all three of the variables **piEstimation**, **foo** and **bar**. (Note that the ***set*** command verb is not required to assign shell variables.)

```
piEstimation, foo, bar = 355.0 / 113.0
```

This example deletes the local variables **piEstimation** and **foo**.

```
set piEstimation, foo =
```

See also:

"Expressions" on page 50
"Compound Object Specifications" on page 72
"Object Properties" on page 66
"Variables" on page 41

shell command

The ***shell*** command escapes the srTool command shell and executes an external command using the temporary context of the native operating system's command interpreter.

Note The standard output and diagnostic streams of srTool and the host shell are completely independent of each other.

Syntax:

shell *expression*

Aliases:

!

Required Parameters:

Parameter	Description
expression	The result of the expression (which is expected to be a string), specifies the native host operating system command, including all parameters, to be executed.

Optional Parameters:

none

Examples:

This Microsoft Windows example displays the contents of the directory "winnt" on the local machine's "C" drive.

```
shell "dir C:\winnt\"
```

This Microsoft Windows example appends only the directories inside of the "Windows" directory on the local machine's "C" drive to a file in the current directory named "a.out". Note the use of a pipe and an append-style output redirection, all of which are passed verbatim to the host command shell using a single-quoted character string in srTool.

```
! 'dir C:\\windows\\ | find "<DIR>" >>a.out'
```

shift command

The ***shift*** command shifts data values stored in higher-numbered parameter variables into lower-numbered ones, overwriting their previous contents. Typically, this is used in a *Loop* to interrogate parameters one-by-one that were passed into scripts that were ***call***ed (or ***exec***uted or ***spawn***ed).

Syntax:

shift [*countExpression*]

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
countExpression	This is an expression that should result in an unsigned integer value. This value determines the number of positions that the parameter variables will be shifted down by. If this parameter is not specified a default value of 1 is used.

Examples:

In this example, all higher-numbered parameter variables are shifted into the next lower-numbered ones.

```
shift
```

See also:

"call command" on page 78
"Variables" on page 41
"Expressions" on page 50

show command

The ***show*** command displays information about what commands, properties, object kinds, predefined values, expression operators and data types are available in srTool.

Syntax:

show [**commands**

| **drivers**

| **function[s]** [*functionName*]

| **globals**

| **objects**

| **operators**

| **properties** [**of** *objectKind*]

| **types**

| **values** [**of** *propertyName*]

| **variables**

| **verbs**

| **versions]**

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
commands	Displays all available commands in alphabetical order: **show commands**
drivers	Displays the names of all installed drivers that srTool can configure using the ***configure*** command. **show drivers**
function[s] [*functionName*]	Displays all currently defined functions, or only the specified one. **show functions** **show function Foo**
globals	Displays all currently active global functions. **show globals**
objects	Displays the complete object containment hierarchy for the replication system. **show objects**
operators	Displays the available operators that can be used in srTool expressions, both unary and binary. **show operators**
properties	Displays a comprehensive list of all available properties and the object kinds that incorporate them. **show properties**
properties [**of** *objectKind*]	Displays just those properties associated with **server** objects. **show properties of server**
types	Displays the available data types that are known to srTool. **show types**
values [**of** *propertyName*]	Displays all available pre-defined values that are known to srTool. **show values of jobState**
variables	Displays all non-global variables for each of the currently active execution contexts. **show variables**

Parameter	Description
verbs	Displays all available commands in alphabetical order. **show commands** **show verbs**
versions	Displays the versions of the currently installed replication software. **show version**

Examples:

See Optional Parameters, above.

spawn command

The ***spawn*** command is used to execute a single command or a set of commands that are stored in a text file in the execution context of a new command shell that runs concurrently with the calling shell. When the commands in the file have finished executing, the shell is automatically destroyed.

There is a preset limit on the maximum number of concurrently running command shells.

Note Unless output redirection is specified on the commands inside the script file, all standard and diagnostic output from commands get sent to the 'NULL' device, and cannot be captured or recovered.

Parameter data is passed to script files in a set of two or more variables, all of which have names that start with "param". See “Variables” on page 41 for more information.

The new command shell inherits most of the calling shell’s variables, as well as its default object specification.

Syntax:

spawn [**-hlsob** | **-nointerface**] *filePathString* [*constParameter* [...]]

... or ...

spawn [**-hlsob** | **-nointerface**] *filePathString* [(*comparison* [,...])]

This form is used to spawn a script using expressions to generate the parameter data.

... or ...

spawn **[-hlsob | -nointerface]** {**-list | -kill [all** | *taskNum* **] | -c** *command* **| -x** *expression* }

This form is used to execute one srTool command (using **-c**) or several commands (using **-x**) as a background task, or to list background tasks or kill them.

Aliases:

&

Required Parameters:

The following required parameters are mutually exclusive (that is, they may *not* be used in combination with one another).

Parameter	Description
filePathString	This string constant must contain a valid path to a file that contains the srTool commands to be executed. A file name of "**con**" or "**tty**", is not allowed, unlike *call* or *exec*.
-c *command*	This specifies the one command to be executed as a background task.
-x *expression*	This specifies one or more commands to be executed as a background task. The expression must result in a single text value that contains the srTool command line to execute.

Optional Parameters:

Parameter	Description
-n **-no** **-nointerface**	This specifies that the command interpreter to be used by the ***spawn***ed script utilizes no client interface. The ***spawn***ed script would not be able to inquire about, or control any aspect of the replication system. Note: This option is mutually exclusive to the **-h[l[sob]]** option.
-h **-hl** **-hlsob**	This switch specifies that the command interpreter to be used by the ***spawn***ed srTool script utilizes the high-level client interface. Note: This option is mutually exclusive to the **-n[o[interface]]** option.
constParameter	This specifies the constant parameter data to be passed to the spawned srTool script. There is no limit to the number of constants that can be passed to the script.
expression	This expression determines the positional parameter data that is to be passed by value to the **spawn**ed srTool script. There is no limit to the number of parameters that can be passed to the script.
-list	Displays a list of tasks that are currently running or are completed.
-kill [all] [all | *taskNum*]	Terminates the currently running task identified by the given task number, or all running tasks if **all** was specified. *TaskNum* must be an unsigned decimal constant.

Examples:

These two examples execute the one command "**start all jobs of first rms**" as a background task:

```
&-c start all jobs of first rms
&-x "start all jobs of first rms"
```

This example lists the currently running tasks to the standard output stream, one per line:

```
spawn -list
```

This example terminates the task whose number is 7 without waiting for it to complete:

```
spawn -kill 7
```

This example terminates all running background tasks:

```
spawn -kill all
```

The following example executes the srTool commands that are stored in the file "**MyScript.txt**" in the context of a new srTool command shell that will run concurrently with the calling shell. The script will inherit all of the calling shell's variables. In addition, the following variables will be defined for it:

paramCount will contain the numeric value 1.
param0 will contain the string value "MyScript.txt".
param1 will contain the numeric value 8.
param2 will contain the string value "35".
param3 will contain the name of the first RMS object.

```
spawn "MyScript.txt" (3 + 5, "3" + "5", name of first RMS)
```

See also:

"Expressions" on page 50
"Variables" on page 41
"call command" on page 78
"exec command" on page 97

start command

The ***start*** command starts execution of one or more jobs. To manually stop a running job, use the ***stop*** or ***cancel*** command. It is not an error to start a job that is already starting or running.

Syntax:

start *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	The *compoundObjectSpec* must refer to one or more jobs.

Optional Parameters:

none

Example:

In this example, only jobs that were stopped would be started.

```
start every job whose jobState is stopped
```

See also:

"Compound Object Specifications" on page 72
"cancel command" on page 80
"stop command" on page 128

stop command

The ***stop*** command stops execution of one or more jobs after it finishes synchronizing its targets, and the targets play out any pending incoming changes. This differs substantially from the ***cancel*** command, which immediately terminates synchronization. It is not an error to stop a job that is already stopped.

To manually start the job running again, use the ***start*** command.

Syntax:

stop *compoundObjectSpec*

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	The *compoundObjectSpec* must refer to one or more jobs.

Optional Parameters:

none

Example:

In this example, jobs that were running without any errors would eventually be stopped.

```
stop every job whose jobState is running
```

See also:

"Compound Object Specifications" on page 72
"`cancel` command" on page 80
"`start` command" on page 127

use command

The ***use*** command allows you to specify a default compound object specification for srTool to use in evaluating compound object specifications used in subsequent commands. This can save significant amounts of typing.

Note srTool's command shell will ignore the default object specification if you specify a compound object specification that terminates in a root-level object.

Syntax:

use [**none** | *compoundObjectSpec*]

Aliases:

none

Required Parameters:

none

Optional Parameters:

Parameter	Description
none	Specifying this keyword causes the current srTool command shell to discard its current default object specification. Specifying object specifications in subsequent commands would require them to be completely specified from a root-level object.
compoundObjectSpec	An *absolute* object specification (that is, one that terminates in a root-level object) which will be appended to all compound object specifications used in subsequent commands.

Note This command can also help you avoid certain ambiguities in the syntax for object specifications. See“Operators” on page 54, for more information.

If no parameters are specified, the command will display the command shell’s current default compound object specification.

Examples:

This example sets srTool's default object specification to Job **"Foo" of** RMS **"Master"**, which will be appended to any object specification used in subsequent commands.

```
use job "Foo" of rms "Master"
```

Thus, instead of having to type

```
list all pairs of job "Foo" of RMS "Master"
```

you would only have to type

```
list all pairs
```

to have the same effect.

This example eliminates srTool's default object specification, requiring object specifications used in subsequent commands to be completely specified from a root-level object:

```
use none
```

This example displays srTool's current default object specification to the standard output stream:

use

See also:

"Object Specifications" on page 67
"Compound Object Specifications" on page 72
"Operators" on page 54

wait command

The ***wait*** command efficiently suspends subsequent command execution until some criteria is satisfied, or until a specified amount of time has passed.

Syntax:

wait {**until** *expression* [**for** *timeExpression*]} | *timeExpression*

Aliases:

none

Required Parameters:

Parameter	Description
until *expression*	An arbitrary expression that determines when the command will complete. When the expression becomes logically **true**, waiting will cease, and command execution will continue. If this is not a specified parameter, the *timeExpression* (below) must be used.
timeExpression	An arbitrary expression that can convert to a *timeSpan*, which determines the amount of time the command will wait. If this is not specified, the **until** parameter (above) must be used.

Optional Parameters:

Parameter	Description
for [*timeExpression*]	If no '**for**' clause follows the '**until**' *expression*, the command will wait for the condition to become logically **true** for 7 days (the default value). otherwise the command will wait for the specified amount of time.

Examples:

This example delays subsequent command execution by 3 seconds:

```
wait "3 seconds"
```

This example will wait for the default period of 7 full days:

```
wait until 1 GT 2
```

This example waits until the job named "Foo" is no longer running, or 6-1/2 hours, whichever occurs first:

```
wait until jobState of job "Foo" NE running and jobState of
job "Foo" NE runningWithErrors for "6.5 hours"
```

See also:

"Expressions" on page 50

xml command

The ***xml*** command emits XML data from one or more objects or an expression to the standard output stream.

Syntax:

xml {**[-src]** **[-raw]** {*compoundObjectSpec*}} | {**-x** *expression*}

Aliases:

none

Required Parameters:

Parameter	Description
compoundObjectSpec	This specifies the object(s) to be listed in XML format.
-x *expression*	The **-x** option indicates that an expression will be specified instead of an object specification.

Optional Parameters:

Parameter	Description
-src	The **-src** option only displays the XML equivalent of the given object specification *or* expression*, instead of the* objects *or data* resulting from its evaluation.
-raw	The **-raw** option specifies that unformatted XML is to be used (that is, without any indenting or line breaks).

Examples:

This example dumps all properties of three randomly chosen jobs in XML format.

```
xml any 3 jobs
```

See also:

"Expressions" on page 50
"Compound Object Specifications" on page 72

srTool Object Reference 5

This section provides a detailed description of each srTool object, including their properties. Examples are also included for discovering, adding, changing and deleting the objects.

Alert Objects

An alert is a timely message of some importance, that is, a problem or condition, posted by the replication system for evaluation by the end user. Alerts are obtained from the RMS object. Alerts are generated internally by the replication system and cannot be created by srTool or the Console.

Aliases:

Alerts

Alert Properties

Most alert properties are constant, and cannot be modified using the ***set*** command.

Property Name	Data Type	Access	Description
AssocObjID	uniqueID	Constant	The globally unique identifier of the object that is associated with the alert.
AssocObjName	string	Constant	The name of the object that is associated with the alert.
AssocObjType	uint32	Constant	The kind of object associated with the alert. This corresponds to the **OrdinalValue** property of the *objectKind* meta object. See "ObjectKind Objects" on page 153.

Property Name	Data Type	Access	Description
Description	string	Constant	The alert's description.
GroupCode	uint32	Constant	The group code of the alert.
HasBeenDeleted	uint32	ReadOnly	True if the alert has been marked for deletion, false if not.
HasBeenRead	uint32	Mutable	True if the alert has been marked as read, false if not.
ID	uniqueID	Constant	The alert's message text.
MessageText	string	Constant	The human-readable text of the alert.
OrigServerID	uniqueID	Constant	The globally unique identifier of the server that created the alert.
OrigServerName	string	Constant	The name of the server that created the alert.
Severity	uint32	Constant	The severity code of the alert.
TextID	uint32	Constant	The unique identier of the alert's text message.
TimeStamp	dateTime	Constant	The time and date the alert was created.

Discovering Alerts

Use the ***list*** or ***selects*** command to inquire about alerts in the replication environment:

```
list all alerts
```

This example shows the 10 most recent alerts, and identifies the objects that generated them:

```
select TimeStamp, (name of first objectKind whose ordinalValue
EQ assocObjType) + ` " ’ + assocObjName + ` " ’, description
from first 10 alerts sortedBy descending timeStamp
```

Changing Alerts

Use the ***set*** command to change any writeable property on a particular alert:

```
set hasBeenRead of first 50 alerts to true
```

Deleting Alerts

Use the ***delete*** command to remove alerts from the RMS. This example deletes all alerts that are more than three days old:

```
delete all alerts whose TimeStamp LT (now () - "3 days" as
timespan)
```

Credential Objects

A credential is used internally by the VRE 3.1 console and srTool to connect to servers that do not belong to the workgroup or domain of the VRE 3.1 administrator user. Credentials are root-level objects.

Note Credentials are not validated until used.

Aliases:

Credentials

Credential Properties

All credential properties are constant, and cannot be modified with the ***set*** command.

Property Name	Data Type	Required to Create	Description
Domain	string	Yes	The name of the domain, if any, the user belongs to. Note: If there is no domain, the domain option is also required using the following format: **-domain= ""**
Password	byteArray	Yes	The encrypted password for this credential.
ServerName	string	Yes	The name of the computer this credential applies to. If empty, it will apply to all computers.
UserName	string	Yes	The name that identifies the user to the domain and/or computer.

Discovering Credentials

Use the ***list*** or ***select*** command to inquire about credentials:

```
list all credentials
```

Creating Credentials

Use the ***add*** command to create a new credential or to replace an existing credential for a given domain and server:

```
add credential with serverName = "Rome", Domain = "My Domain",
userName = "Joe", password = Encrypt ("Joe's password")
```

Changing Credentials

Once created the individual properties of a credential cannot be changed. However an existing credential with a given serverName can be replaced with a different userName, password, and domain.

Deleting Credentials

Use the ***delete*** command to delete credentials:

```
delete credentials 2 thru 4
```

DestinationRule Objects

A DestinationRule is an object that identifies a directory root and its target server that will receive the replicated files and folders. DestRules belong to PathRule objects.

Aliases:

DestinationRules, DestRule, DestRules

DestinationRule Properties

Some destinationRule properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Required to Create	Access	Description
ID	uniqueID	No	Constant	The globally unique identifier of the DestinationRule.
Name	string	No	ReadOnly	A string containing the synthesized name of the DestinationRule.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the PathRule object that owns this DestinationRule.
ControllerID	uniqueID	No	Constant	The globally unique identifier of the job object that controls this DestinationRule.
TargetServerID	uniqueID	No	Mutable	The globally unique identifier of the target server.
TargetServer	string	Yes	Mutable	The name of the target server.
Path	string	Yes	Mutable	The absolute, volume-rooted path that identifies where replicated data will go.

Discovering DestinationRules

Use the ***list*** or ***select*** command to inquire about destinationRules in a particular PathRule:

```
get name of every destinationRule of every rule of job "Foo"
```

Creating DestinationRules

Use the ***add*** command to create DestinationRules for a particular PathRule:

```
use first pathRule of first job of first rms
add destrule with targetServer = "Test1", path = "C:\\replica"
```

Note that in order to create a DestinationRule, you must identify the target server and the path on that server where replicated files will go.

Changing DestinationRules

Use the ***set*** command to change any writeable property on a particular DestinationRule:

```
use first pathRule whose Path EQ "C:\\data" of first job of
first rms
set path of every destinationRule to "C:\\replica"
```

Deleting DestinationRules

Use the ***delete*** command to remove DestinationRules from a particular PathRule:

```
use all pathRules of first job of first rms
delete all destRules whose name startsWith "\\\\TestServer1"
```

File Objects

File objects represent files kept on any volume or a server. File objects can be obtained from folder or volume objects.

Aliases:

Files

File Properties

All file properties are constant and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the file was last accessed.
Created	dateTime	The date and time that the file was created.
Depth	uint32	The depth (valence) of the file in the hierarchy.
FullPath	string	The full path to the file.
IsArchive	uint32	True if archive bit is set.
IsCompressed	uint32	True if the file is compressed.

Property Name	Data Type	Description
IsContainer	uint32	True if the file is a container (folder). This property is always false.
IsEncrypted	uint32	True if the file is encrypted.
IsHidden	uint32	True if the file is hidden.
IsOffline	uint32	True if the file is offline.
IsReadOnly	uint32	True if the file is read-only.
IsReparsePoint	uint32	True if the file is a reparse point.
IsSparseFile	uint32	True if the file is a sparse file.
IsSystem	uint32	True if the file is a system file.
IsTemporary	uint32	True if the file is temporary.
Modified	dateTime	The date and time that the file was last modified.
Name	string	The name of the file, including its extension, if any.
ServerName	string	The name of the server on which the file can be found.
Size	uint64	The size of the file, in bytes.

Discovering Files

Use the ***list*** or ***select*** command to inquire about files in a particular volume or folder:

```
get fullPath of every file of volume C: of server "Foo"
```

Creating Files

Files cannot be created using srTool.

Changing Files

Files cannot be changed using srTool.

Deleting Files

Files cannot be deleted using srTool.

FileReplicationJob Objects

A FileReplicationJob is a named object that has a schedule that can replicate data between source and target server machines. Jobs contain all the information necessary to facilitate replication for a given set of source and target servers and the files and folders to be replicated. This job information is kept in a database on the Replication Management Server (RMS). Thus, job objects "belong" to an RMS.

Jobs contain three other kinds of objects: ReplicationPairs, PathRules and LogEntries.

Aliases:

FileReplicationJobs, Job, Jobs

Job Properties

Many job properties are constant or read-only, and cannot be modified using the ***set*** command. However, there are several that can be changed:

Property Name	Data Type	Required to Create	Access	Description
AlertCount	uint32	No	ReadOnly	The number of alerts posted for the job.
AlertWhenConsistent	uint32	No	Mutable	Controls when an information alert will be generated when a Target of a pair reaches the consistent state at the end of synchronization. The default value is "false".
ClusterID	string	No	ReadOnly	For private use by VCS or MSCS.
ClusterName	string	No	ReadOnly	For private use by VCS or MSCS.
ClusterType	uint32	No	ReadOnly	For private use by VCS or MSCS.

Property Name	Data Type	Required to Create	Access	Description
CurrentExecutingOperation	uint32	No	ReadOnly	For internal use only.
Description	string	No	Mutable	A string containing the job's description.
Enabled	uint32	No	ReadOnly	True if the job is currently enabled; false if not. (Enabled means "available to run at its next scheduled time".)
ID	uniqueID	No	Constant	The globally unique identifier of the job.
IsClusterOwned	uint32	No	ReadOnly	For private use by VCS or MSCS.
IsSyncedWithJCD	uint32	No	ReadOnly	For internal use only.
JCDServerID	uniqueID	No	ReadOnly	The unique identifier of the job control delegate server of the job.
JCDServerName	string	No	ReadOnly	The name of the job control delegate server for the job.
JobState	uint32	No	ReadOnly	A value that indicates the current state of the job: NeverRun, Canceled, CanceledWithErrors, Expired, ExpiredWithErrors, Completed, CompletedWithErrors, Starting, Running, RunningWithErrors, Pausing, Paused, Resuming, Rallying, Canceling, Expiring, Completing.
LastAddedOperationRequest	uint32	No	ReadOnly	For internal use only.
LastStarted	dateTime	No	ReadOnly	The date and time when the job was last started.
MappingMethod	uint32	No	Mutable	A value that specifies the mapping method to use during replication for the job: PrependSourceServerPath, PrependSourceRootDirPath, PrependSourceImmediateParent or PrependNone.

Property Name	Data Type	Required to Create	Access	Description
Name	string	No	Mutable	A string containing the name of the job. When creating new jobs, you do not need to provide a name; the replication system will invent a unique one for you. If you do specify one, it must be unique among all other jobs.
NextPendingOperationRequest	uint32	No	ReadOnly	For internal use only.
NoChgsOnTarget	uint32	No	Mutable	True if the job is to enforce read-only targets when the job is running, or false if not.
NoDynamicJournal	uint32	No	Mutable	True if changes that occur to the sources during synchronization are not saved and later replayed on the target.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the object that owns this job.
PairCount	uint32	No	ReadOnly	The number of replication pair objects that belong to the job.
PendingUpdateCount	uint32	No	ReadOnly	For internal use only.
Prescan	uint32	No	Mutable	True if the job is to prescan the source files to be able to accurately estimate when initial synchronization will complete; false if not.
RealTime	uint32	No	Mutable	True if the job is to enter dynamic mode after initial synchronization; false if not.
Schedule	byteArray	No	Mutable	A bit mask that specifies the job's schedule in which each bit indicates "eligible" (1) or "ineligible" (0) to run. Each bit represents a 30-minute time span. The entire blob must be exactly 336 bits long, thus representing a 7-day (week-long) schedule.

Property Name	Data Type	Required to Create	Access	Description
ScheduledStopsCancel	uint32	No	Mutable	If true, jobs cancel abruptly instead of gracefully when the job's schedule window closes.
SyncReportFilenames	uint32	No	Mutable	Controls if the synchronization report of the respective pairs of a job will contain the names of synchronized files. The default value is "false".
TargetReplicaType	uint32	No	ReadOnly	A value that specifies the target replica type for the job: Pure, Qualified, Merge or UpdateOnly.
Type	uint32	Yes	Constant	A value that specifies the job topology OneToOne, OneToMany or ManyToOne.

Discovering Jobs

Use the ***list*** or ***select*** command to inquire about jobs on a particular RMS:

```
list all jobs
get name, jobState of every job
```

Creating Jobs

Use the ***add*** command to create jobs on a particular RMS:

```
add job with name = "Test", type = OneToMany
```

Note In order to create a job, you must specify a value for its 'type' property, which can be OneToOne, OneToMany, or ManyToOne.

Changing Jobs

Use the ***set*** command to change any writeable job property:

```
set name of job "Foo" to "Bar"
```


Deleting Jobs

Use the ***delete*** command to remove jobs from a particular RMS:

```
delete every job whose realTime is true
```

Controlling Jobs

Use the ***start***, ***stop*** or ***cancel*** commands to start, stop or cancel jobs:

```
start every job whose jobState NE running AND jobState NE
runningWithErrors
```

Folder Objects

Folders are directories located inside volumes on any given server. Folders can be obtained from other folder or volume objects. A folder may contain other folders or files.

Aliases:

Folders

Folder Properties

All folder properties are constant, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the folder was last accessed.
Created	dateTime	The date and time that the folder was created.
Depth	uint32	The depth (valence) of the folder in the hierarchy.
FullPath	string	The full path to the folder.
HasContainers	uint32	True if the folders contain any folders.
HasFiles	uint32	True if the folders contain any files.
IsArchive	uint32	True if the archive bit is set.

Property Name	Data Type	Description
IsCompressed	uint32	True if the folder is compressed.
IsContainer	uint32	True if the folder is a container (folder). This property is always true.
IsEncrypted	uint32	True if the folder is encrypted.
IsHidden	uint32	True if the folder is hidden.
IsOffline	uint32	True if the folder is offline.
IsSystem	uint32	True if the folder is a system folder.
IsVolume	uint32	True if the folder is a volume. This property is always set to false.
Modified	dateTime	The date and time that the folder was last modified.
Name	string	The name of the folder, including its extension, if any.
ServerName	string	The name of the server on which the folder can be found.

Discovering Folders

Use the ***list*** or ***select*** command to inquire about folders:

```
list all folders of volume "C:" of server "FOO"
```

Creating Folders

Folders cannot be created using srTool.

Changing Folders

Folders cannot be changed using srTool.

Deleting Folders

Folders cannot be deleted using srTool.

Item Objects

Items are folder or file objects, which allow the user to obtain either one in a single query. Items come from folder or volume objects.

Aliases:

Items

Item Properties

All item properties are constant and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the item was last accessed.
Created	dateTime	The date and time that the item was created.
Depth	uint32	The depth (valence) of the item in the hierarchy.
FullPath	string	The full path to the item.
HasContainers	uint32	True if the item contains any folders.
HasFiles	uint32	True if the item contains any files.
IsArchive	uint32	True if the archive bit is set.
IsCompressed	uint32	True if the item is compressed.
IsContainer	uint32	True if the item is a container (folder)., false if it is a file.
IsEncrypted	uint32	True fi the item is encrypted.
IsHidden	uint32	True if the item is hidden.
IsOffline	uint32	True if the item is offline.
IsReadOnly	uint32	True if the item is read only.
IsReparsePoint	uint32	True if the item is a reparse point.
IsSparseFile	uint32	True if the item is a sparse file.

Property Name	Data Type	Description
IsSystem	uint32	True if the item is a system file.
IsTemporary	uint32	True if the item is temporary.
IsVolume	uint32	True if the object is a volume. This property is always set to false.
Modified	dateTime	The date and time that the item was last modified.
Name	string	The name of the item, including its extensions, if any.
ServerName	string	The name of the server on which the item can be found.

Discovering Items

Use the ***list*** or ***select*** command to inquire about items:

```
get fullPath, isContainer of all items of volume "C:" of server
"Foo"
```

Creating Items

Items cannot be created in srTool.

Changing Items

Items cannot be changed in srTool.

Deleting Items

Items cannot be deleted in srTool.

License Objects

Licenses are objects that represent actual licenses that are installed on a server, and as such, they come from server objects.

Aliases:

Licenses

License Properties

All License properties are constant, and cannot be modified with the ***set*** command.

Property Name	Data Type	Required To Create	Description
ControllerID	uniqueID	No	The globally unique ID of the server that controls this object.
ID	uniqueID	No	The globally unique ID of this object.
IsBase	uint32	No	True if base license
IsBEOption	uint32	No	True if Backup Exec option
IsClusterOption	uint32	No	True if cluster option
IsDemo	uint32	No	True if Demo license
IsExpired	uint32	No	True if license has expired
IsNBUOption	uint32	No	True if NetBackup option
IsNFR	uint32	No	True if not-for-resale license
IsPermanent	uint32	No	True if permanent license
IsSiteLicense	uint32	No	True if site license
LicenseEndDate	dateTime	No	The license\licenses expiration date and time
LicenseKeyString	string	Yes	The license key, or the serial number when creating a license
LicenseProductID	int32	No	The product ID for this license
LicenseProductName	string	No	The product name for this license
LicenseTimeLeft	timeSpan	No	The amount of time the license has left before it expires

Property Name	Data Type	Required To Create	Description
OwnerID	uniqueID	No	The globally unique ID of the server that owns this object.

Discovering Licenses

Use the ***list*** or ***select*** command to view a server's currently installed licenses:

```
list all licenses of server "Milan"
```

Creating Licenses

When a license is purchased, use the serial number you received as follows:

```
add license to server "Athens" with licenseKeyString =
"12345678910"
```

Changing Licenses

Once created, licenses cannot be changed using srTool.

Deleting Licenses

Use the ***delete*** command to remove licenses from servers:

```
delete all licenses whose (IsExpired) of all servers
```

LogEntry Objects

LogEntries are the individual entries in logs that are maintained by the replication system. LogEntries come from server, job, or pair objects.

Aliases:

LogEntries

LogEntry Properties

All logEntry properties are constant, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
ControllerID	uniqueID	The globally unique identifier of the object (server, job, or pair) that posted the log entry.
GroupCode	uint32	The group code of the log entry.
ID	uniqueID	The globally unique identifier of the log entry.
LogBlob	byteArray	For internal use only.
MessageText	string	The complete text of the log entry.
OwnerID	uniqueID	The globally unique identifier of the object that owns the log entry.
SequenceNumber	uint64	The internal sequence number of the log entry.
StatusCode	uint32	The status code of the log entry.
TextID	uint32	The unique identifier of the text message for the log entry.
TimeStamp	dateTime	The date and time the log entry was posted.

Discovering LogEntries

Use the ***list*** or ***select*** command to see the LogEntries for a particular Job:

```
get TimeStamp, MessageText of all logEntries of job "Foo"
```

To determine the number of log entries for a given server:

```
count all logEntries of server "Foo"
```

Creating LogEntries

LogEntries cannot be created in srTool.

Changing Log Entries

LogEntries cannot be changed in srTool.

Deleting LogEntries

Use the ***delete*** command to remove LogEntries. This example deletes all log entries for the Job "Foo" that are more than three days old:

```
use job "Foo" of first RMS
delete every LogEntry whose TimeStamp LT (now () - '3 days' as
timespan)
```

ObjectKind Objects

This object is a root-level meta-object that describes other objects. ObjectKinds may contain a number of property objects.

Aliases:

ObjectKinds

ObjectKind Properties

All ObjectKind properties are constants, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Name	string	The name of the object kind.
Description	string	A string that contains a brief description of the object kind.
OrdinalValue	uint32	The internal ordinal value of the object kind.

Discovering ObjectKinds

Use the ***list*** or ***select*** command to discover ObjectKind objects:

```
list all objectKinds whose name startsWith "S"
```


Creating ObjectKinds

You cannot create ObjectKind objects using srTool.

Changing ObjectKinds

You cannot change ObjectKind objects using srTool.

Deleting ObjectKinds

You cannot delete ObjectKind objects using srTool.

PathRule Objects

A PathRule is an object that identifies a directory root and its source server that contains the files and folders to be replicated to the target server. PathRules belong to an owning job object.

PathRules may contain a number of SelectionRule or DestinationRule objects, or both.

Aliases:

PathRules, Rule, Rules

PathRule Properties

Most PathRule properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Required to Create	Access	Description
ControllerID	uniqueID	No	Constant	The globally unique identifier of the job object that controls this path rule.
ID	uniqueID	No	Constant	The globally unique identifier of the path rule.

Property Name	Data Type	Required to Create	Access	Description
Name	string	No	ReadOnly	A string containing the synthesiazed name of the path rule.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the job object that owns this path rule.
SourceServer	string	Yes	ReadOnly	The name of the path rule's source server.
SourceServerID	uniqueid	No	ReadOnly	The globally unique identifier of the path rule's source server.
Path	path	Yes	ReadOnly	The absolute, volume-rooted path of the directory to be replicated with their path rule.

Discovering PathRules

Use the ***list*** or ***select*** command to inquire about PathRules in a particular job:

```
list all rules of job "Foo"
```

Creating PathRules

Use the ***add*** command to create PathRules for a particular job:

```
use first job of first RMS
add pathRule with sourceServer = "Test1", path = "C:\\Foo"
```

Note In order to create a pathRule, you must specify values for its 'sourceServer' and 'path' properties.

Changing PathRules

Use the ***set*** command to change any writeable property on a particular PathRule:

```
use first job of first RMS
set throttle of every pair to 50
```


Deleting PathRules

Use the ***delete*** command to remove pathRules from a particular job:

```
del every pathRule whose path endsWith "\\Foo"
```

Property Objects

A property object describes the properties of srTool objects.

Aliases:

Properties

Property Objects Properties

All Property properties are constant and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Access	uint32	The access of the property (Constant, Read-Only or Mutable)
AutoStale	uint32	True if the property automatically goes stale after a certain amount of time, or false if not
DataSource	uint32	The data source of the property (Synthesized, FromRMSDB, FromRSA, FromDiscovery or FromStatsService)
DataType	uint32	The intrinsic data type of the property
DefaultValue	string	The property's default value
Description	string	The property's description
InitiallyStale	uint32	True if the property is initially stale, or false if not
Name	string	The name of the property
OrdinalValue	uint32	The ordinal value the property

Property Name	Data Type	Description
RequiredToCreate	uint32	The property's object creation requirements (**CannotBeSpecified**, **CanBeSpecified** or **MustBeSpecified**)

Discovering Properties

Use the ***list*** or ***select*** command to discover properties:

```
list -omit description of all properties of ObjectKind "RMS"
```

Creating Properties

Properties cannot be created in srTool.

Changing Properties

Properties cannot be changed in srTool.

Deleting Properties

Properties cannot be deleted in srTool.

ReplicationPair Objects

A ReplicationPair is an object that identifies a source and target server that will participate in a replication job. Pairs belong to Job objects. Pairs can contain a number of Script objects.

Aliases:

Pair, Pairs, ReplicationPairs

Pair Properties

Most pair properties are constant or read-only, and cannot be modified using the ***set*** command. Many properties of pairs exist only when the pair is actively replicating data.

Property Name	Data Type	Required to Create	Access	Description
ControllerID	uniqueID		Constant	FromRMSDB
CurrentFileName*	string	No	Constant	The name of the current file being replicated in sync mode. FromStatsService
CurrentFileSize*	uint64	No	Constant	The size of the current file being replicated in sync mode. FromStatsService
DataIsConsistentOnTarget	uint32		Constant	FromStatsService
ID	uniqueID	No	Constant	The globally unique identifier of the pair. FromRMSDB
IsPairDisabled	uint32	No	Mutable	If TRUE, the pair is disabled and cannot synchronize. FromRMSDB
LastKnownRmsPairJobInstance	uint32	No	ReadOnly	For internal use only. FromRMSDB
Name	string	No	ReadOnly	A string containing the synthesized name of the pair.
NetMaxKbitsPerSecond	uint32	No	Mutable	Specifies the maximum data throughput allowed for this pair.
NoRally	uint32	No	Mutable	If TRUE, prevents the pair from rallying.
NoRallyAutoReset	uint32	No	Mutable	If TRUE, allows the RMS to reset the "NoRally" property.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the job object that owns this pair.

Property Name	Data Type	Required to Create	Access	Description
ResyncPctComplete*	uint32	No	Constant	A value that indicates the current percentage completion of synchronization for the pair.
Resyncs*	uint32	No	Constant	A value that indicates the current number of resyncs done by the pair.
RunStage*	uint32	No	Constant	A value that indicates the current run stage of the pair: NotRunning, PreScan, Synchronization, Dynamic, ReSynchronization and NoConnection.
RunState	uint32	No	ReadOnly	A value that indicates the current run state of the pair: Starting, Running, Canceling, Canceled, Expiring, Expired, Aborting, Aborted, Resuming, Pausing, Paused, Completed, CompletedWithErrors, Disconnected, Crashed, Restarting, Completing, Hung or Force32Bits.
ScannedObjectTally*	uint64	No	Constant	A value that indicates the current tally of the number of files or folders that have been replicated during synchronization mode for the pair.
SourceServer	string	Yes	Constant	The name of the pair's source server.
SourceServerID	uniqueID	No	Constant	A value that identifies the source server of the pair.
TargetMappingPrefix	string	No	Mutable	A string that contains the target mapping prefix to use for the pair.
TargetServer	string	Yes	Constant	The name of the pair's target server.
TargetServerID	uniqueID	No	Constant	A value that identifies the target server of the pair.
Throttle	uint32	No	Mutable	An unsigned integer value between 0 and 100 that indicates the throttling to use for this pair, specified as a percentage.

Property Name	Data Type	Required to Create	Access	Description
TimeTilSyncDone*	dateTime	No	Constant	A timespan value that estimates how long it will take to complete synchronization for the pair.
TotalBytesSent*	uint64	No	Constant	An unsigned integer value that indicates the current total number of bytes sent from the source server to the target server for this replication pair.
TransferRate*	uint32	No	Constant	A value that indicates the current data transfer rate between the source and the target for this replication pair.
WhenStageStarted*	dateTime	No	Constant	A dateTime value that indicates when the current pair stage (see RunStage property) started for this replication pair.

* These properties are statistics that are only available when the pair is actively replicating data.

Discovering Pairs

Use the ***list*** command to inquire about pairs in a particular job:

```
get name of all pairs of job "Foo"
```

Creating Pairs

Use the ***add*** command to create pairs for a particular job:

```
use first job of first rms
add pair with sourceServer = "Test1", targetServer = "Test2"
```

Note In order to create a pair, you must specify values for its **SourceServer** and **TargetServer** properties.

Changing Pairs

Use the ***set*** command to change any writeable property on a particular pair:

```
set throttle of every pair of every job to 50
```

Deleting Pairs

Use the ***delete*** command to remove pairs from a particular job:

```
delete all pairs whose name startsWith "Test1:" of first job
```

RMS Objects

An RMS is a Replication Management Server that is designated to manage replication jobs on the network. The RMS contains a database that stores information about Jobs, Servers and Alerts.

RMS objects are root-level objects when srTool uses the high-level client interface.

An RMS can only be created during the VRE 3.1 installation process. srTool cannot be used to create an RMS object. Similarly, srTool cannot be used to delete an RMS. This can only be done by removing VRE 3.1 from the RMS machine.

Aliases:

none

RMS Properties

Most properties of RMS objects are either constant or read-only, and thus, cannot be modified using the ***set*** command.

Property Name	Data Type	Access	Description
Address	string	Constant	The RMS's IP address.
BuildVersionString	string	ReadOnly	The build version of the RMS server software running on the RMS.
Domain	string	ReadOnly	The name of the domain the RMS belongs to.
FeaturePackVersion	uint32	ReadOnly	The version number of the server's feature pack.
ID	uniqueID	Constant	The RMS's globally unique identifier.

Property Name	Data Type	Access	Description
IsAvailable	uint32	ReadOnly	True if the RMS is available (that is, accessible to the network).
MaintenancePackVersion	uint32	ReadOnly	The maintenance pack version number of the software running on this RMS.
MajorBuildNumber	uint32	ReadOnly	The major build number of the software running on this RMS.
MajorProductVersion	uint32	ReadOnly	The major product version number of the software running on this RMS.
MinorBuildNumber	uint32	ReadOnly	The minor build number of the software running on this RMS.
MinorProductVersion	uint32	ReadOnly	The minor product version number of the software running on this RMS.
Modified	dateTime	ReadOnly	The date and time when the RMS property data was last modified.
Name	string	Constant	The name of the RMS.
OSBuildNumber	uint32	ReadOnly	The build number of the operating system software running on the RMS.
OSClass	uint32	ReadOnly	The class code of the operating system software running on the RMS.
OSMajorVersion	uint32	ReadOnly	The major version number of the operating system software running on the RMS.
OSMinorVersion	uint32	ReadOnly	The minor version number of the operating system software running on the RMS.
OSRevisionNumber	uint32	ReadOnly	The revision number of the operating system software running on the RMS.
OSServicePackMajor	uint32	ReadOnly	The service pack number of the operating system software running on the RMS.
OSServicePackMinor	uint32	ReadOnly	The service pack number of the operating system software running on the RMS.

Property Name	Data Type	Access	Description
OSWindowsSubType	uint32	ReadOnly	The windows sub-type code of the operating system software running on the RMS.
PatchVersion	uint32	ReadOnly	The patch version number of the software running on the RMS.
SpecialBuildString	string	ReadOnly	The special build string of the software running on the RMS.
RMSGatewayAddress	string	ReadOnly	The name or IP address of the RMS gateway for use in remote "push" deployment.
TimeToKeepLogItems	timeSpan	Mutable	The maximum amount of time to retain log entries on the RMS before automatically purging them.

Discovering RMSs

Normally, replication environments will have a single RMS. Use the ***list*** command to find out about it:

```
list first rms
```

Creating RMSs

RMS objects cannot be created using srTool.

Changing RMSs

Use the ***set*** command to change the value of any mutable properties of the RMS:

```
set TimeToKeepLogItems of first RMS to "1 week" as timeSpan
```

Deleting RMSs

RMS objects cannot be deleted using srTool.

Script Objects

Script objects designate special programs to run when a certain event occurs on either the Source or Target, such as synchronization having been achieved. Scripts belong to pairs.

Aliases:

Scripts

Script Properties

Many script properties are mutable, and can be modified using the ***set*** command.

Property Name	Data Type	Required to Create	Access	Description
CommandLine	string	Yes	Mutable	A string that contains the command line that is to execute when the triggering event occurs. This property must contain a string that contains at least one character and at most 4,096 characters. Its content should be a well-formed command line that begins with a path specification that leads to the `.exe` program file to be executed, followed by any number of command line parameters required by that program to properly execute.
ControllerID	uniqueID	No	Constant	The globally unique identifier of the job that controls this object.
ID	uniqueID	No	Constant	The globally unique identifier of the script.
IsRunAsynch	uint32	No	Mutable	True if the CommandLine is to execute asynchonously (that is, concurrently); false, if not. The default value is false.
IsSource	uint32	Yes	Mutable	True if the CommandLine is to execute on the source server; false if on the target.

Property Name	Data Type	Required to Create	Access	Description
Name	string	No	ReadOnly	A string containing the synthesized name of the script.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the pair object that owns this script.
Timeout	timeSpan	No	Mutable	Specifies the amount of time to wait for the command to finish before proceeding. The default value is zero time.
TriggeringEvent	uniqueID	Yes	Mutable	The globally unique identifier of the triggering event, some of which are predefined as global variables: JobStart, JobStop, PairDataConsistent, PairDataInconsistent. See "Variables" on page 41.

Discovering Scripts

Use the ***list*** or ***select*** command to inquire about scripts in a particular ReplicationPair:

```
get name of all scripts of every pair of job "Foo"
```

Creating Scripts

Use the ***add*** command to create scripts for a particular ReplicationPair:

```
use first pair of first job of first rms
add script with triggeringEvent = PairDataConsistent,
commandLine = "startBackup", isSource = false, timeout = "1
hour" as timespan, isRunAsynch = false
```

Note In order to create a script, you must specify values for the IsSource and CommandLine properties.

Changing Scripts

Use the ***set*** command to change any writeable property on a particular script :

```
use first pair of first job of first rms
set commandLine of every script to "startbackup.bat"
```

Deleting Scripts

Use the ***delete*** command to remove scripts from a particular ReplicationPair:

```
delete every script of all pairs of job "Foo"
```

SelectionRule Objects

SelectionRule objects determine the kinds of files that will be replicated for a particular PathRule. SelectionRules belong to PathRules.

Aliases:

SelectionRules, SelRule, SelRules

SelectionRule Properties

Most selectionRule properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Required To Create	Access	Description
ControllerID	uniqueID	No	Constant	The globally unique identifier of the job the selection rule is controlled by.
ID	uniqueID	No	Constant	The globally unique identifier of the selection rule.
IsExclude	uint32	No	Mutable	True if the selection rule excludes matching files or folders from replication; false if it includes them.

Property Name	Data Type	Required To Create	Access	Description
IsRecursive	uint32	No	Mutable	True if the selection rule applies to its path rule's subdirectories; false if it applies only to the items inside its path rule's directory.
Name	string	No	ReadOnly	The name of the selection rule.
NameSpec	string	Yes	Mutable	The name specification, perhaps a wild card, that determines the files or folders that will be selected.
OwnerID	uniqueID	No	Constant	The globally unique identifier of the path rule that owns the selection rule.
SortOrder	uint32	No	ReadOnly	For internal use only.

Discovering SelectionRules

Use the ***list*** or ***select*** command to inquire about SelectionRules in a particular job:

```
get name of all SelRules of all rules of job "Foo"
```

Creating SelectionRules

Use the ***add*** command to create SelectionRules:

```
use first job of first rms
add SelRule with nameSpec = "*.DOC" to first PathRule
```

Note In order to create a selection rule, you must specify a value for the 'nameSpec' property.

Changing SelectionRules

Use the ***set*** command to change any mutable property on a particular SelectionRule :

```
use first job of first rms
set nameSpec = "*.log" for all SelRules whose nameSpec EQ
"*.exe" of all rules
```


Deleting SelectionRules

Use the ***delete*** command to remove selectionRules from a particular PathRule:

```
delete every selRule whose nameSpec EQ "*.DOC" of all pathRules
of job "Foo"
```

Server Objects

A server is a machine that can participate in replication as either a source or a target. Servers are tracked by the RMS database. Server objects contain logEntry and volume objects. Servers can only be obtained from an RMS.

Note A server can only be created during the VRE 3.1 installation or deployment process. srTool cannot be used to create one. A server can only be deleted by first removing the machine from the network or removing VRE 3.1 from it, then using the delete command in srTool (running on a different machine) to force the replication system to "forget" about that server.

Aliases:

Servers

Server Properties

Most server properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Access	Description
Address	string	Constant	The server's IP address.
BuildVersionString	string	ReadOnly	The build version of the RSA server software running on the server.
DefaultTargetPath	string	Mutable	The default target path for replica files copied to the server
Domain	string	ReadOnly	The name of the domain the server belongs to.
FeaturePackVersion	uint32	ReadOnly	The version number of the server's feature pack.
ID	uniqueID	Constant	The server's globally unique identifier.
IsAvailable	uint32	ReadOnly	True if the server is available (that is accessible on the network)
IsOnline	uint32	ReadOnly	True if the server is online (that is, able to participate in replication)
IsStale	uint32	Mutable	For internal use only.
LastAlertDateTime	dateTime	ReadOnly	The date and time of the last alert posted from this server.
LastAlertSequenceNumber	uint64	ReadOnly	The sequence number of the last alert posted from this server.
MaintenancePackVersion	uint32	ReadOnly	The maintenance pack version number of the software running on this server.
MajorBuildNumber	uint32	ReadOnly	The major build number of the software running on this server.
MajorProductVersion	uint32	ReadOnly	The major product version number of the software running on this server.
MinorBuildNumber	uint32	ReadOnly	The minor build number of the software running on this server.

Property Name	Data Type	Access	Description
MinorProductVersion	uint32	ReadOnly	The minor product version number of the software running on this server.
Modified	dateTime	ReadOnly	The date and time when the server property data was last modified.
Name	string	Constant	The name of the server.
OSBuildNumber	uint32	ReadOnly	The build number of the operating system software running on the server.
OSClass	uint32	ReadOnly	The class code of the operating system software running on the server.
OSMajorVersion	uint32	ReadOnly	The major version number of the operating system software running on the server.
OSMinorVersion	uint32	ReadOnly	The minor version number of the operating system software running on the server.
OSRevisionNumber	uint32	ReadOnly	The revision number of the operating system software running on the server.
OSServicePackMajor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSServicePackMinor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSVersion	string	ReadOnly	The version string for the operating system software running on the server.
OSWindowsSubType	uint32	ReadOnly	The windows sub-type code of the operating system software running on the server.
PatchVersion	uint32	ReadOnly	The patch version number of the software running on the server.
SpecialBuildString	string	ReadOnly	The special build string of the software running on the server.
TimeToKeepAlerts	timeSpan	Mutable	The maximum age of alerts before they get deleted automatically.

Property Name	Data Type	Access	Description
TimeToKeepLogItems	timeSpan	Mutable	The maximum amount of time to retain log entries on the RMS before automatically purging them.

Discovering Servers

Use the ***list*** or ***select*** command to find out about the servers that exist in your RMS' replication neighborhood:

```
get name, address, isAvailable of all servers
```

Creating Servers

Servers cannot be created using srTool.

Changing Servers

Use the ***set*** command to change the value of any mutable property of a server:

```
set TimeToKeepAlerts of all servers to "2 weeks" as timeSpan
```

Deleting Servers

To delete a server, uninstall VRE 3.1 from it, then use the ***delete*** command:

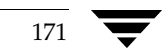
```
delete server "Milan"
```

SourceServer Objects

SourceServer objects represent servers that contain the files that are to be replicated for a job. SourceServers are no different than Server objects, except they are obtained from Job objects.

Aliases:

SourceServers



SourceServer Properties

Most SourceServer properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Access	Description
Address	string	Constant	The server's IP address.
BuildVersionString	string	ReadOnly	The build version of the RSA server software running on the server.
DefaultTargetPath	string	Mutable	The default target path for replica files copied to the server
Domain	string	ReadOnly	The name of the domain the server belongs to.
FeaturePackVersion	uint32	ReadOnly	The version number of the server's feature pack.
ID	uniqueID	Constant	The server's globally unique identifier.
IsAvailable	uint32	ReadOnly	True if the server is available (that is accessible on the network)
IsOnline	uint32	ReadOnly	True if the server is online (that is, able to participate in replication)
IsStale	uint32	Mutable	For internal use only.
LastAlertDateTime	dateTime	ReadOnly	The date and time of the last alert posted from this server.
LastAlertSequence Number	uint64	ReadOnly	The sequence number of the last alert posted from this server.
MaintenancePack Version	uint32	ReadOnly	The maintenance pack version number of the software running on this server.
MajorBuildNumber	uint32	ReadOnly	The major build number of the software running on this server.
MajorProductVersion	uint32	ReadOnly	The major product version number of the software running on this server.
MinorBuildNumber	uint32	ReadOnly	The minor build number of the software running on this server.

Property Name	Data Type	Access	Description
MinorProductVersion	uint32	ReadOnly	The minor product version number of the software running on this server.
Modified	dateTime	ReadOnly	The date and time when the server property data was last modified.
Name	string	Constant	The name of the server.
OSBuildNumber	uint32	ReadOnly	The build number of the operating system software running on the server.
OSClass	uint32	ReadOnly	The class code of the operating system software running on the server.
OSMajorVersion	uint32	ReadOnly	The major version number of the operating system software running on the server.
OSMinorVersion	uint32	ReadOnly	The minor version number of the operating system software running on the server.
OSRevisionNumber	uint32	ReadOnly	The revision number of the operating system software running on the server.
OSServicePackMajor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSServicePackMinor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSVersion	string	ReadOnly	The version string for the operating system software running on the server.
OSWindowsSubType	uint32	ReadOnly	The windows sub-type code of the operating system software running on the server.
PatchVersion	uint32	ReadOnly	The patch version number of the software running on the server.
SpecialBuildString	string	ReadOnly	The special build string of the software running on the server.

Property Name	Data Type	Access	Description
TimeToKeepAlerts	timeSpan	Mutable	The maximum age of alerts before they get deleted automatically.
TimeToKeepLogItems	timeSpan	Mutable	The maximum amount of time to retain log entries on the RMS before automatically purging them.

Discovering SourceServers

Use the ***list*** or ***select*** command to inquire about SourceServers in a particular job:

```
get name of all sourceServers of job "Foo"
```

Creating SourceServers

SourceServers can only be created by adding Replication Pairs to a Job.

Changing SourceServers

Since SourceServers are functionally identical to Servers, they can be changed using the ***set*** command. See the “Changing Servers” section for examples.

Deleting SourceServers

Since SourceServers are functionally identical to Servers, they can be deleted using the ***delete*** command. See the “Deleting Servers” section for examples.

SubFile Objects

SubFile objects represent files on a server. Subfiles are no different than Files, except that they are used to produce an entire heirarchy of files that have a common ancestor container, as opposed to just the files in a single container. SubFile objects are obtained from folder or volume objects.

Aliases:

SubFiles

SubFile Properties

All subFile properties are constant, and cannot be modified using the ***set*** command..

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the file was last accessed.
Created	dateTime	The date and time that the file was created.
Depth	uint32	The depth (valence) of the file in the hierarchy.
FullPath	string	The full path to the file.
IsArchive	uint32	True if archive bit is set.
IsCompressed	uint32	True if the file is compressed.
IsContainer	uint32	True if the file is a container (folder). This property is always false.
IsEncrypted	uint32	True if the file is encrypted.
IsHidden	uint32	True if the file is hidden.
IsOffline	uint32	True if the file is offline.
IsReadOnly	uint32	True if the file is read-only.
IsReparsePoint	uint32	True if the file is a reparse point.
IsSparseFile	uint32	True if the file is sparse.
IsSystem	uint32	True if the file is a system file.
IsTemporary	uint32	True if the file is temporary.
Modified	dateTime	The date and time that the file was last modified.
Name	string	The name of the file, including its name extensions, if any.
ServerName	string	The name of the server on which the file can be found.
Size	uint64	The size of the file in bytes.

Discovering SubFiles

Use the ***list*** or ***select*** command to inquire about SubFiles in a particular directory:

```
get fullPath of all subFiles of folder "Windows" of vol "C:" of
server "Foo"
```

Creating SubFiles

srTool cannot create SubFiles.

Changing SubFiles

srTool cannot change SubFiles.

Deleting SubFiles

srTool cannot delete SubFiles.

SubFolder Objects

SubFolders represent directories on a server. Subfolders are no different than folders, except that they are used to produce an entire heirarchy of folders that have a common ancestor container, as opposed to just the folders in a single container. SubFolders can be obtained from folder or volume objects.

Aliases:

SubFolders

SubFolder Properties

All subFolder properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the folder was last accessed.
Created	dateTime	The date and time that the folder was created.
Depth	uint32	The depth (valence) of the folder in the hierarchy.
FullPath	string	The full path to the folder.
HasContainers	uint32	True if the folder contains any folders.
HasFiles	uint32	True if the folder contains any files.
IsArchive	uint32	True if the archive bit is set.
IsCompressed	uint32	True if the folder is compressed.
IsContainer	uint32	True if the folder is a container (folder). This property is always true.
IsEncrypted	uint32	True if the folder is encrypted.
IsHidden	uint32	True if the folder is hidden.
IsOffline	uint32	True if the folder is offline.
IsSystem	uint32	True if the folder is a system folder.
IsVolume	uint32	True if the folder is a volume. This property is always false.
Modified	dateTime	The date and time that the folder was last modified.
Name	string	The name of the folder, including its name extension, if any.
ServerName	string	The name of the server on which the folder can be found.

Discovering SubFolders

Use the ***list*** or ***select*** command to inquire about all SubFolders in a given directory:

```
get fullPath of all subFolders of vol "C:" of server "Foo"
```

Creating SubFolders

srTool cannot create SubFolders.

Changing SubFolders

srTool cannot change SubFolders.

Deleting SubFolders

srTool cannot delete SubFolders.

SubItem Objects

SubItems represent folders or files on a server. Subitems are no different than Items, except that they are used to produce an entire heirarchy of folders or files that have a common ancestor container, as opposed to just those folders or files in a single container. SubItems come from folder or volume objects.

Aliases:

SubItems

SubItem Properties

All subItem properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	dateTime	The date and time that the item was last accessed.
Created	dateTime	The date and time that the item was created.

Property Name	Data Type	Description
Depth	uint32	The depth (valence) of the item in the hierarchy.
FullPath	string	The full path to the item.
HasContainers	uint32	True if the item contains any folders (if it's a container).
HasFiles	uint32	True if the item contains any files (if it's a container).
IsArchive	uint32	True if the archive bit is set.
IsCompressed	uint32	True if the item is compressed.
IsContainer	uint32	True if the item is a container (folder); false if it is a file.
IsEncrypted	uint32	True if the item is encrypted.
IsHidden	uint32	True if the item is hidden.
IsOffline	uint32	True if the item is offline.
IsReadOnly	uint32	True if the item is read-only.
IsReparsePoint	uint32	True if the item is a reparse point.
IsSparseFile	uint32	True if the item is a sparse file.
IsSystem	uint32	True if the item is a system file.
IsTemporary	uint32	True if the item is temporary.
IsVolume	uint32	True if the object is a volume. This property is always set to false.
Modified	dateTime	The date and time that the item was last modified.
Name	string	The name of the item, including its name extensions, if any.
ServerName	string	The name of the server on which the item can be found.

Discovering SubItems

Use the ***list*** or ***select*** command to inquire about subItems in a given directory:

```
get fullPath, isContainer of all subItems of folder "Windows"
of vol "C:" of server "Foo"
```

Creating SubItems

srTool cannot create SubItems.

Changing SubItems

srTool cannot change SubItems.

Deleting SubItems

srTool cannot delete SubItems.

TargetServer Objects

TargetServer objects represent servers that will receive replica files for a job. TargetServers are no different than Server objects, except they are obtained from Job objects.

Aliases:

TargetServers

TargetServer Properties

Most TargetServer properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Access	Description
Address	string	Constant	The server's IP address.
BuildVersionString	string	ReadOnly	The build version of the RSA server software running on the server.

Property Name	Data Type	Access	Description
DefaultTargetPath	string	Mutable	The default target path for replica files copied to the server
Domain	string	ReadOnly	The name of the domain the server belongs to.
FeaturePackVersion	uint32	ReadOnly	The version number of the server's feature pack.
ID	uniqueID	Constant	The server's globally unique identifier.
IsAvailable	uint32	ReadOnly	True if the server is available (that is accessible on the network)
IsOnline	uint32	ReadOnly	True if the server is online (that is, able to participate in replication)
IsStale	uint32	Mutable	For internal use only.
LastAlertDateTime	dateTime	ReadOnly	The date and time of the last alert posted from this server.
LastAlertSequence Number	uint64	ReadOnly	The sequence number of the last alert posted from this server.
MaintenancePack Version	uint32	ReadOnly	The maintenance pack version number of the software running on this server.
MajorBuildNumber	uint32	ReadOnly	The major build number of the software running on this server.
MajorProductVersion	uint32	ReadOnly	The major product version number of the software running on this server.
MinorBuildNumber	uint32	ReadOnly	The minor build number of the software running on this server.
MinorProductVersion	uint32	ReadOnly	The minor product version number of the software running on this server.
Modified	dateTime	ReadOnly	The date and time when the server property data was last modified.
Name	string	Constant	The name of the server.
OSBuildNumber	uint32	ReadOnly	The build number of the operating system software running on the server.

Property Name	Data Type	Access	Description
OSClass	uint32	ReadOnly	The class code of the operating system software running on the server.
OSMajorVersion	uint32	ReadOnly	The major version number of the operating system software running on the server.
OSMinorVersion	uint32	ReadOnly	The minor version number of the operating system software running on the server.
OSRevisionNumber	uint32	ReadOnly	The revision number of the operating system software running on the server.
OSServicePackMajor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSServicePackMinor	uint32	ReadOnly	The service pack number of the operating system software running on the server.
OSVersion	string	ReadOnly	The version string for the operating system software running on the server.
OSWindowsSubType	uint32	ReadOnly	The windows sub-type code of the operating system software running on the server.
PatchVersion	uint32	ReadOnly	The patch version number of the software running on the server.
SpecialBuildString	string	ReadOnly	The special build string of the software running on the server.
TimeToKeepAlerts	timeSpan	Mutable	The maximum age of alerts before they get deleted automatically.
TimeToKeepLogItems	timeSpan	Mutable	The maximum amount of time to retain log entries on the RMS before automatically purging them.

Discovering TargetServers

Use the ***list*** or ***select*** command to inquire about TargetServers in a particular job:

```
get name of all targetServers of job "Foo"
```

Creating TargetServers

TargetServers can only be created by adding Replication Pairs to a Job.

Changing TargetServers

Since TargetServers are functionally identical to Servers, they can be changed using the ***set*** command. See the "Changing Servers" section for examples.

Deleting TargetServers

Since TargetServers are functionally identical to Servers, they can be deleted using the ***delete*** command. See the "Deleting Servers" section for examples.

Volume Objects

A Volume object represents a logical storage volume on a server, and thus, comes from a server object. Volume objects can be used to obtain File, Folder, Item, SubFile, SubFolder, or SubItem objects.

Aliases:

Volumes, Vol, Vols

Volume Properties

All volume properties are constant or read-only, and cannot be modified using the ***set*** command.

Property Name	Data Type	Description
Accessed	TimeStamp	The date and time when the volume was last accessed.
BytesFree	uint64	The number of free bytes on the volume.
Capacity	uint64	The capacity of the volume, in bytes.

Property Name	Data Type	Description
Created	TimeStamp	The date and time of the volume's creation.
Depth	count	Always zero (0).
FileSystem	string	The file system format of the volume (for example, "NTFS").
FullPath	string	A string containing the full path name of the volume.
HasContainers	uint32	True if the root level of the volume contains any folders.
HasFiles	uint32	True if the root level of the volume contains any files.
IsContainer	uint32	True if the volume is a container. This property is always true.
IsReadOnly	uint32	True if the volume is write-protected.
IsVolume	uint32	True if the volume is a volume. This property is always set true.
Modified	TimeStamp	The date and time when the volume was last changed.
Name	string	A string containing the name of the volume.
ServerName	string	The name of the server that owns this volume.

Discovering Volumes

Use the ***list*** or ***select*** commands to inquire about volumes on a particular server:

```
get name of every volume of every server
```

Creating Volumes

srTool cannot create Volumes.

Changing Volumes

srTool cannot change Volumes.

Deleting Volumes

srTool cannot delete Volumes.

Messages and Troubleshooting A

srTool can issue many different messages to many different venues. One is the srTool console window, particularly if the global variable "**verbose**" is set **true**. Another is the "Trace..." log file(s), which are kept in the Logs folder in the VRE 3.1 home directory.

Regardless of where they end up, all messages that originate in the client interfaces, the "shared classes" library or in srTool, have an identification name, which is provided in the following form.

MMM000I

The message configuration is as follows.

Message ID Part	Description
Digits 1 to 3:	The three-letter module code that identifies the origin of the message.
"HLS":	◆ The message originated inside the high-level client interface.
"SHR":	◆ The message originated inside the shared classes library
"SRT":	◆ The message originated inside of srTool
Digits 4 to 6:	A unique three-digit unsigned decimal identification number that identifies the message.
Digit 7:	A code letter that describes the message type.
"A" (Action):	◆ The program is waiting for the user to take an action
"I" (Inform):	◆ The message contains information that may or may not be of interest to the user.
"E" (Error):	◆ The message contains information about a problem the software encountered that caused the operation to fail.
"S" (Severe):	◆ The message contains information about a serious problem the software encountered that required the software to terminate its operation.

For example, the message SRT301A is a message that originated in srTool, its message ID is 301, and it is an action message. The following message listings are generated by srTool: "Shared Classes Library Messages" on page 188, "High-Level Client Interface Messages" on page 206, and "srTool Messages" on page 243.

Shared Classes Library Messages

Message Code	Message and Description
SHR200E	**SHR200E CElementProcessor failed while processing** *entity 'entityDescription*' PROBLEM: A failure occurred while processing an expression, preventing all of its elements from being processed. *Entity* is either "operator", "operand" or "invalid element". CAUSE: If *entity* is **operator**, the failure occurred while processing an operator in the expression. If *entity* is **operand**, the failure occurred while processing an operand in the expression. The *entityDescription* identifies the operator or operand whose processing failed. If *entity* reads "invalid element", a serious internal problem exists that should be reported to VERITAS technical support. SOLUTION: This message is normally accompanied by another message indicating the cause of the failure, whether the fault of an operator or an operand. Note the *entityDescription* and determine from the other messages what aspect of the operator or operand failed. (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.)
SHR201E	**SHR201E (ProcessOperator) Binary operation failure, operator=***operator***, operands:** *leftOperand*, *rightOperand* PROBLEM: A failure occurred while processing two operands, *leftOperand* and *rightOperand* using the binary operator *operator*, preventing the posting of its result on the expression evaluator's result stack. CAUSE: There are many possible causes for operators to fail (for example, division by zero, type incompatibility, and so on). This message identifies the operator and the two operands associated with the failure. SOLUTION: This message is normally accompanied by another message indicating the underlying cause of the failure. (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.)

Message Code	Message and Description
SHR202E	**SHR202E (ProcessOperator) Unary operation failure, operator=***operator*, **operand:** *operand* PROBLEM: A failure occurred while processing the given single *operand* using the given unary *operator*, preventing the posting of its result on the expression evaluator's result stack. CAUSE: There are many possible causes for operators to fail (for example, type incompatibility, and so on). This message identifies the operator and the operand associated with the failure. SOLUTION: This message is normally accompanied by another message indicating the underlying cause of the failure. (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.)
SHR203E	**SHR203E (ProcessOperator) Evaluation result stack was empty, expected second operand -- operator=***'operator'* **(operand='***operand***')** PROBLEM: During processing of an expression, the expression evaluator's result stack became empty while the binary operator *operator* remained to be processed. The first operand is *operand*. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, and contact VERITAS technical support.
SHR204E	**SHR204E (ProcessOperator) Operator invalid, neither unary or binary** PROBLEM: During processing of an expression, an invalid operator was found that answered false to both IsUnaryOperator and IsBinaryOperator queries. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, then contact VERITAS technical support.

Message Code	Message and Description
SHR205E	**SHR205E (ProcessOperator) Evaluation result stack was empty, expected first operand** PROBLEM: During processing of an expression, the expression evaluator's result stack was found to be empty when an operator was next in line to be processed. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, then contact VERITAS technical support.
SHR207E	**SHR207E (ProcessOperand) Unable to get value of basic property reference** *'propertyRefDescription'* PROBLEM: A failure occurred while processing the basic property reference identified in the message. CAUSE: There are many possible causes for basic property references to fail to answer with a data value. The most common cause is specifying a property that doesn't belong to the kind of object being queried for. SOLUTION: Check to be sure that the property being requested in the query filter expression is a valid property of the kinds of objects being queried for.
SHR208E	**SHR208E (ProcessOperand) Unable to get value of property reference** *'propertyRefDescription'* PROBLEM: A failure occurred while processing the (absolute) property reference identified in the message. No data value was able to be returned for that property from the given object. CAUSE: There are two common causes for property references to fail to answer with a data value. The most common cause is specifying a an object that does not exist, or specifying a property that doesn't belong to the kind of object that was specified. SOLUTION: Check to be sure that the object specified in the reference actually exists. Verify that the property being requested is a valid property of the object that was specified.

Message Code	Message and Description
SHR209E	**SHR209E (ProcessOperand) Invalid operand** *'operandDescription'* **-- internal error** PROBLEM: During processing of an expression, an invalid operand identified by the *operandDescription* was encountered that was not a constant, variable reference, basic property reference, (absolute) property reference or a function call. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, then contact VERITAS technical support.
SHR210E	**SHR210E (CEvaluator::End) Result stack is empty -- expected final result** PROBLEM: After fully processing an expression, the expression evaluator's result stack was found to be empty. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, then contact VERITAS technical support.
SHR211E	**SHR211E (CEvaluator::End) Result stack is wrong size -- should have 1 element, instead found** *elementCount* PROBLEM: After fully processing an expression, the expression evaluator's result stack was found to contain *elementCount* elements, a value that exceeds 1, the expected number. Somehow, too many result values were pushed onto the result stack. This indicates an internal software problem. CAUSE: This error should not be possible to produce during normal use of the VRE 3.1 console or srTool. SOLUTION: (In srTool, be sure the *verbose* shell variable is *true* so that all available diagnostic messages are visible.) Gather as much information about how this message came about, then contact VERITAS technical support.

<table>
<tr><th>Message Code</th><th>Message and Description</th></tr>
<tr><td>SHR212E</td><td>SHR212E (ANYTYPE) GetValue: Unable to convert 'existingType' value 'dataValue' to type 'targetType'
PROBLEM:
The data value dataValue, whose data type is existingType, was requested in a different data type targetType. A failure occurred while trying to convert the data value to the target data type.
CAUSE:
This error usually occurs when trying to convert a data value into another data type that is not compatible with the original value's data type.
SOLUTION:
To avoid this error, check to be sure that the data value's existing data type is able to be converted into the desired target data type. (See “Converting Between Different Data Types” on page 33.)</td></tr>
<tr><td>SHR213I</td><td>SHR213I signalDescription (codeName) signaled
CAUSE:
This message reports that the signal signalDescription whose code name is codeName was received and acknowledged. The software may or may not respond to the signal, depending upon the type of signal and the state of the software at the time it was raised.
<table>
<tr><th>Signal Description</th><th>Code Name</th><th>Meaning</th></tr>
<tr><td>^C</td><td>SIGINT</td><td>"Control-C" interrupt. The user held down the Control key and pressed the 'C' key. In srTool, this is used to interrupt the current operation that is in progress.</td></tr>
<tr><td>Illegal Instruction</td><td>SIGILL</td><td>An illegal instruction was encountered during program execution.</td></tr>
<tr><td>Floating Point Exception</td><td>SIGFPE</td><td>A floating point exception was thrown during program execution.</td></tr>
<tr><td>Segment Violation</td><td>SIGSEGV</td><td>A segment violation occurred during program execution.</td></tr>
<tr><td>Terminate</td><td>SIGTERM</td><td>The program was asked to terminate.</td></tr>
<tr><td>Break</td><td>SIGBREAK</td><td>(Windows only) The Windows NT command prompt window was asked to close while srTool was running inside it. This signal causes srTool to quit at its earliest convenience.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Message Code</th><th colspan="3">Message and Description</th></tr>
<tr><td></td><td>Abort</td><td>SIGABRT</td><td>Execution of the program is aborting.</td></tr>
<tr><td></td><td>Unknown signal</td><td>signal code number</td><td>A signal of unknown class was encountered during program execution.</td></tr>
<tr><td>SHR221W</td><td colspan="3">SHR221W (Class::Method) Invalid index or index range 'indexOrIndexRange'
PROBLEM:
While filtering an ordered sequence of objects by indexing specification, an invalid index or an invalid index range was encountered.
CAUSE:
This error can occur when an indexing specification is used in an srTool command, and the indexing spec refers to objects that aren't in the given index position.
SOLUTION:
To avoid this error, check to be sure that the indexing specification that is used refers to objects that really do occupy those index positions. (See “Indexing Specifications” on page 70.)</td></tr>
<tr><td>SHR222E</td><td colspan="3">SHR222E (ANYTYPE) Expected expectedLength characters in GUID string -- instead found actualLength characters
PROBLEM:
While converting a data value of type string into a data value of type uniqueid, the string was found to have actualLength characters in it, whereas expectedLength characters were expected.
CAUSE:
This error is most commonly found in srTool when a string is being converted into a uniqueID. For example, this srTool command will fail with this error:
<code>echo -x "{A32Fba1E-D2D7-4583-850A-1FA58CbB9eB}" as uniqueID</code>
This srTool example will display the proper string length expected:
<code>echo -x "The proper length of a string containing a uniqueID is " + sizeOf (newID () as string) as string</code>
SOLUTION:
To avoid this error, be sure the contents of the string follows the syntax rules for specifying a uniqueID. (See “Converting a String into a UniqueID” on page 34.)</td></tr>
</table>

Message Code	Message and Description
SHR223E	**SHR223E (ANYTYPE) Expected hexadecimal digit at position** *charPos* **-- instead found** *'character'* PROBLEM: While converting a data value of type string into a data value of type uniqueid, the string was found to have a non-hexadecimal digit *character charPos* characters into the it. (A hex digit is 0 thru 9, 'a' thru 'f', or 'A' thru 'F'.) CAUSE: This error is most commonly found in srTool when a string is being converted into a uniqueID. For example, this srTool command will fail with this error: **`echo -x "{A32Fba1*-D2D7-4583-850A-1FA58CbB9eB0}" as uniqueID`** SOLUTION: To avoid this error, be sure the contents of the string follows the syntax rules for specifying a uniqueID. (See "Converting a String into a UniqueID" on page 34.)
SHR224E	**SHR224E (ANYTYPE) Expected** *'goodChar'* **character at position** *charPos* **-- instead found** *'badChar'* PROBLEM: While converting a data value of type string into a data value of type uniqueid, the string didn't have a '{', '-' or '}' character in the correct positions. Instead the character *badChar* was found *charPos* characters into the string, whereas the character *goodChar* was expected there. CAUSE: This error is most commonly found in srTool when a string is being converted into a uniqueID. For example, this srTool command will fail with this error: **`echo -x "{A32Fba1E^D2D7-4583-850A-1FA58CbB9eB0}" as uniqueID`** SOLUTION: To avoid this error, be sure the contents of the string follows the syntax rules for specifying a uniqueID. (See "Converting a String into a UniqueID" on page 34.)
SHR225I	**SHR225I (CExpression::Evaluate) Expression** *'expressionDescription'* **failed to evaluate** CAUSE: This error is reported for any expression evaluation failure and merely states that a failure occurred. This message is normally accompanied by other messages that help to indicate the cause of the failure.

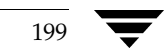
Message Code	Message and Description
SHR226E	**SHR226E (ProcessOperand) Function call** *'functionCallDescription'* **failed** PROBLEM: A failure was reported while processing the function call operand identified by *functionCallDescription* during evaluation of an expression. CAUSE: This error is most commonly found in srTool when an expression contains a call to a function that doesn't exist or that has an incorrect number of arguments. SOLUTION: To avoid this error, be sure the function being called exists (see "`show` command" on page 122) and that the proper number of arguments are passed to it.
SHR227E	**SHR227E Listing incomplete due to** *signalName* **signal** PROBLEM: The resulting list of objects was incomplete because a signal was raised while it was being displayed. CAUSE: If a large listing of objects is being emitted by the shell and the user pressed Control-C, the resulting list that's displayed is incomplete. SOLUTION: To avoid this error, do not interrupt the listing being generated or displayed.

Message Code	Message and Description
SHR228E	**SHR228E (ANYTYPE) TimeSpan specification: Unexpected** *element* **found near** *'token'* **-- Cannot switch from longhand to shorthand, or vice-versa** PROBLEM: A syntax error was encountered while converting a string into a timespan. If *element* reads "longhand time unit", the string's content up to the *token* was successfully parsed as shorthand timespan notation. If *element* reads "shorthand timespan notation", the string's content up to the *token* was successfully parsed as longhand timespan notation. In either case, the notation used inside the string is ambiguous and cannot be parsed any further. CAUSE: This error can occur in srTool when a string is being converted into a timespan and the string content does not follow the syntax rules for specifying timespans. For example, these srTool commands will both fail with this error: **`echo -x "2.5 days 4:5" as timespan`** **`echo -x "35 : 2 minutes" as timespan`** The first example starts off using longhand timespan notation, then switches to shorthand (at the colon). The second example starts off in shorthand notation, then switches to longhand (at "minutes"). SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying timespans. (See “Converting a String into a TimeSpan” on page 36.) In particular, do not mix shorthand and longhand notation elements.
SHR229E	**SHR229E (ANYTYPE) TimeSpan specification: Time unit** *'timeUnit'* **used more than once in longhand notation** PROBLEM: A syntax error was encountered while converting a string containing longhand timespan notation into a timespan. The *timeUnit* found in the string was specified more than once, making it impossible to definitively convert the string. CAUSE: This error can occur in srTool when a string is being converted into a timespan and the string content does not follow the syntax rules for specifying timespans. For example, this srTool command will fail with this error: **`echo -x "2.5 days 4 minutes 3 days 2 seconds" as timespan`** Note that the "days" time unit was specified more than once. SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying timespans. (See “Converting a String into a TimeSpan” on page 36.) In particular, when using longhand timespan notation, do not specify any time unit more than once.

Message Code	Message and Description
SHR230E	**SHR230E (ANYTYPE) Expected** *elementDescription* **-- instead got** *'token'* PROBLEM: A syntax error was encountered while converting a string into a timespan or datetime. If *elementDescription* reads "numeric value", an unsigned decimal or floating point value was expected, but *token* was found instead. If *elementDescription* reads "time unit", the string's content up to but not including *token* was successfully parsed as longhand timespan notation, and a valid time unit was expected instead of *token*. In any of these cases, the notation used inside the string is ambiguous and cannot be parsed any further. CAUSE: This error can occur in srTool when a string is being converted into a timespan or datetime, and the string content does not follow the syntax rules for specifying datetimes or timespans. The next two examples finds "foo" instead of a numeric value: **`echo -x "foo 5/5 5:30" as datetime`** **`echo -x "3/foo" as datetime`** This example finds "foo" instead of a time unit: **`echo -x "2 days 35 foo" as timespan`** SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying datetimes or timespans. (See “Converting a String into a DateTime” on page 35 and “Converting a String into a TimeSpan” on page 36.)
SHR231E	**SHR231E (ANYTYPE) TimeSpan specification: Too many numeric fields in shorthand notation, exceeds 'day' part -- found** *number* **fields** PROBLEM: A syntax error was encountered while converting a string into a timespan. The string's content was successfully parsed as a timespan using shorthand notation, but it found *number* numeric fields, which exceeded four, the maximum possible. CAUSE: This error can occur in srTool when a string is being converted into a timespan, and the string content does not follow the syntax rules for specifying timespans. This example will cause this error: **`echo -x "2:12:30:15:8" as timespan`** Note that 8 corresponds to the seconds part, 15 to the minutes part, 30 to the hours part, 12 to the days part. Because there are no parts beyond days in shorthand notation, the 2 cannot correspond to any time part, thus making it impossible to convert the string. SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying timespans. (See “Converting a String into a TimeSpan” on page 36.)

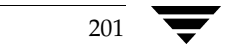
Message Code	Message and Description
SHR232E	**SHR232E (ANYTYPE) DateTime specification:** *part* **already specified** PROBLEM: A syntax error was encountered while converting a string into a datetime. The string contained more than one *part* specification, where *part* is "Time", "Date" or "AM or PM". CAUSE: This error can occur in srTool when a string is being converted into a datetime and the string content does not follow the syntax rules for specifying datetimes. For example, these srTool commands will all fail with this error: **`echo -x "4/3 5/4" as datetime`** Note that the "date" part was specified more than once. **`echo -x "4:3 5:4" as datetime`** Note that the "time" part was specified more than once. **`echo -x "4/3 5:4 am pm" as datetime`** Note that the "AM or PM" part was specified more than once. SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying datetimes. (See "Converting a String into a TimeSpan" on page 36.)
SHR233E	**SHR233E (ANYTYPE) DateTime specification: Meridian specification** *'token'* **found in date part** PROBLEM: A syntax error was encountered while converting a string into a datetime. The string contained a meridian specification in the date part. CAUSE: This error can occur in srTool when a string is being converted into a datetime and the string content does not follow the syntax rules for specifying datetimes. For example, this srTool command will fail with this error: **`echo -x "4/3 am 5:4 pm" as datetime`** Note that a meridian specification appears in the date part. SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying datetimes. (See "Converting a String into a DateTime" on page 35.) Specifically, do not put a meridian specification in the date portion of the datetime specification -- it must be in the time part.

Message Code	Message and Description
SHR234E	**SHR234E (ANYTYPE) DateTime specification: Floating point value '***value***' not allowed in field** *fieldNumber* **of** *part* **part** PROBLEM: A syntax error was encountered while converting a string into a datetime. The number found in *fieldNumber* of the *part* ("date" or "time") in the string was specified using the floating point number *value*, which is not allowed. CAUSE: This error can occur in srTool when a string is being converted into a datetime and the string content does not follow the syntax rules for specifying datetimes. For example, this srTool command will fail with this error: **`echo -x "4/3.5/2003 5 PM" as datetime`** Note that a floating point number is used in the month field of the date part. SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying datetimes. (See "Converting a String into a DateTime" on page 35.) Specifically, do not use floating point values in either the date or time portion of the datetime specification -- only unsigned positive decimal integers can be used.
SHR235E	**SHR235E (ANYTYPE) DateTime specification: Illegal value '***value***' for** *field* **field of** *part* **part --** *problem* PROBLEM: A syntax error was encountered while converting a string into a datetime. The *value* found in *field* ("year", "month", "day", "hour", "minute", or "second") of the *part* ("date" or "time") in the string is not allowed. The *problem* part of the message describes the specific problem. CAUSE: This error can occur in srTool when a string is being converted into a datetime and the string content does not follow the semantic rules for specifying datetimes. For example, these srTool commands all fail with this error: **`echo -x "13/5/2003 5 PM" as datetime (Month part exceeds 12)`** **`echo -x "5/0/2003 5 PM" as datetime (Day part is zero)`** **`echo -x "10/5/2003 15:05 PM" as datetime (Hour part implies military clock, yet meridian specified)`** **`echo -x "10/5/2003 1:65 PM" as datetime (Minute part exceeds 59)`** As should be clear now, there are many other possible ways to cause this error. SOLUTION: To avoid errors of this type, be sure the numeric values of each field in the date and/or time parts of the string being converted are legitimate. (See "Converting a String into a DateTime" on page 35.)



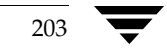
Message Code	Message and Description
SHR236E	**SHR236E (ANYTYPE) DateTime specification: Too** *fewOrMany dateOrTime* **fields** PROBLEM: A syntax error was encountered while converting a string into a datetime. Too few or too many numeric fields were found in the date or time part in the string. CAUSE: This error can occur in srTool when a string is being converted into a datetime and the string content does not follow the syntax rules for specifying datetimes. For example, these srTool commands all fail with this error: **`echo -x "1/2/5/3 5 PM" as datetime`** (Too many date fields) **`echo -x "10/5/2003 5:5:23:48 PM" as datetime`** (Too many time fields) SOLUTION: To avoid this error, be sure the content of the string being converted follows the syntax rules for specifying datetimes. (See "Converting a String into a DateTime" on page 35.) Specifically, be sure the proper number of date or time fields are specified.
SHR237E	**SHR237E (ANYTYPE) DateTime specification: No date or time part specified** PROBLEM: A syntax error was encountered while converting a string into a datetime. No date or time part was found in the string. CAUSE: It should not be possible to receive this message, since missing fields or delimiters will elicit messages **SHR236E** or **SHR230E**. SOLUTION: Be sure the content of the string being converted follows the syntax rules for specifying datetimes. (See "Converting a String into a DateTime" on page 35.)
SHR239E	**SHR239E (CDataSupplier) Unable to provide resulting value for** *entity 'name'* **since no method was supplied to obtain this result** PROBLEM: While computing the result of an expression, the *entity* (a variable, function or property specification) having the given *name* could not be obtained, resolved or computed. CAUSE: This problem should not occur during normal operation. SOLUTION: If the problem still happens, note other applicable messages that may accompany this one, then please contact Technical Support.

Message Code	Message and Description
SHR240E	**SHR240E (XMLParse) While in** *XMLState* **state, expected** *syntaxElement*, **instead got** *token* PROBLEM: A syntax error was found in the XML code being parsed. The *XMLState* can be any of **XML_Invalid, XML_InNewTag, XML_InBeginTag, XML_InBeginTagParamName, XML_InBeginTagParamValue, XML_InEndTag or XML_InContent**. The parser expected the *syntaxElement* but instead found the given *token*. CAUSE: This is typically caused by improperly coded XML. SOLUTION: Correct the syntax of the offending XML code then try the operation again.
SHR241E	**SHR241E (XMLParse) End tag** *tagName* **found without corresponding start tag** PROBLEM: A syntax error was found in the XML code being parsed. An end tag with the given *tagName* was encountered, but no begin tag with that same name preceded it. CAUSE: This is typically caused by a misspelled tag name in the XML stream. SOLUTION: Correct the syntax of the offending XML code then try the operation again.
SHR242E	**SHR242E (XMLParse) End tag** *tagName* **out of sequence -- expected end tag for** *currentTagName* PROBLEM: A syntax error was found in the XML code being parsed. An end tag with the given *tagName* was encountered, but the end tag for the *currentTagName* was expected, indicating an out-of-sequence error. CAUSE: This is typically caused by mixing up the order of end tags, or forgetting to insert an end tag in the XML stream. SOLUTION: Correct the end tag order or add the missing end tag in the offending XML code then try the operation again.



Message Code	Message and Description
SHR243E	**SHR243E (XMLParse) Invalid parse state (***stateCode***) -- internal failure** PROBLEM: The XML parser got into an invalid state. CAUSE: This problem should not occur during normal operation. SOLUTION: Note any other applicable messages that may accompany this one, then please contact Technical Support.
SHR244W	**SHR244W (FromXMLNode) In** *tag***, encountered an unexpected XML node:** *nodeDump* PROBLEM: While creating an object from its XML specification, an unexpected XML node was encountered in the given *tag*. CAUSE: This is typically caused by improperly coded XML. SOLUTION: Correct the offending XML code then try the operation again.
SHR245E	**SHR245E (FromXMLNode) Unable to construct '***parentTag***' object from XML -- missing '***childTag***' child node** PROBLEM: While creating an object from its XML specification (using the *parentTag* tag), a required XML node with the tag *childTag* was missing from the XML stream. CAUSE: This is typically caused by improperly coded XML. SOLUTION: Correct the offending XML code then try the operation again.
SHR246E	**SHR246E (FromXMLNode) Unable to construct '***parentTag***' object from XML -- invalid data '***badData***' found in '*tagName*' tag** PROBLEM: While creating an object from its XML specification (using the *parentTag* tag), the tag named *tagName* contained invalid data. CAUSE: This is typically caused by improperly coded XML. SOLUTION: Correct the offending XML code then try the operation again.

Message Code	Message and Description
SHR247E	**SHR247E (FromXMLNode) Unable to construct '*parentTag*' object from XML -- more than one '*childTag*' tag found** PROBLEM: While creating an object from its XML specification (using the *parentTag* tag), a child node tagged *childTag* appeared more than once, making it ambiguous as to which one to use. CAUSE: This is typically caused by improperly coded XML. SOLUTION: Correct the offending XML code then try the operation again.
SHR248E	**SHR248E (Dictionary) Failure in *method*, symbol name is '*name*', value is '*value'* – reason** PROBLEM: A failure occurred in the given *method* (**Add, Change, Delete, SetReadOnly** or **SetReadWrite**) of the Dictionary facility for the given symbol *name* and its corresponding data *value*. The *reason* for the failure is also given in the message. CAUSE: **Add** failures are usually because the symbol already exists. **Change** failures are usually because the symbol was marked as read-only. **Delete, SetReadOnly** and **SetReadWrite** failures happen usually because the symbol does not exist. SOLUTION: Be sure when adding a symbol that it does not already exist; or when changing a symbol's value that it exists and that it is not read-only; or when deleting or altering the access attributes of a symbol, be sure that the symbol exists.
SHR249E	**SHR249E (ProcessOperand) Unable to get value of variable '*symbolName*'** PROBLEM: The value of the symbolic variable with the name *symbolName* in an expression being evaluated could not be obtained from any of the data suppliers registered with the expression evaluator. CAUSE: The most common cause of this error is the misspelling of a variable name. SOLUTION: Be sure that all variables used in the expression are defined.



Message Code	Message and Description
SHR250E	**SHR250E (DoTypeCastOperation) While converting '***dataValue***', desired type '***dataType***' (***ordinalValue***) was unexpected** PROBLEM: The *dataValue* value was asked to convert into an illegal or unknown *dataType* that has the given *ordinalValue*. CAUSE: The most common cause of this error is the misuse of the **as** operator. SOLUTION: Be sure that the datatype used on the right-hand side of the **as** operator is valid. See "Data Types" on page 32.
SHR252E	**SHR252E (CDebugLevel) Unable to change debug level to** *newLevel* **-- level must be between** *minLevel* **and** *maxLevel***, inclusive** PROBLEM: An attempt was made to change the debug level of a configurable module or object to the given *newLevel*, which was outside of the legal range of *minLevel*, the minimum, and *maxLevel*, the maximum. CAUSE: The desired debug level *newLevel* was outside of the legal range of *minLevel*, the minimum, and *maxLevel*, the maximum. SOLUTION: Be sure that the desired debug level is greater than or equal to *minLevel* and less than or equal to *maxLevel*.
SHR253I	**SHR253I (CDebugLevel) Debug level changed from** *previousLevel* **to** *newLevel* CAUSE: The debug level of a configurable module or object was changed from its previous value *previousLevel* to the new value *newLevel*.
SHR254E	**SHR254E (CExpression::DoBinaryOperation) Divide by zero error (numerator=***numer***, denominator=***denom***)** PROBLEM: While evaluating an expression, the denominator operand *denom* passed to the binary operator OP_Divide or OP_Mod was zero, resulting in a divide-by-zero exception. CAUSE: The denominator value *denom* of the division or modulo operation was zero. SOLUTION: Try to avoid situations in which the denominator of a division or modulo division operation is zero.

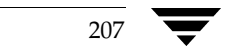
Message Code	Message and Description
SHR255I	**SHR255I** *rmsError* CAUSE: This message provides a further explanation for an RMS error that occurred. It is usually preceded by another message that provides the context in which the failure occurred.
SHR256W	**SHR256W (CConfigurable) Unable to set configuration parameter named '***paramName***' to value '***newValue***' -- no such parameter** PROBLEM: An attempt was made to change the value of a parameter with the name *paramName* of a configurable module or object to the given *newValue*, but the module or object did not recognize the parameter with that name. CAUSE: The configurable module or object did not have a parameter with that name. This is commonly due to misspelling the parameter name. SOLUTION: Be sure to spell the parameter name correctly, and that the configurable module or object has a parameter with that name.
SHR257E	**SHR257E (CConfigurable) Unable to get configuration parameter named '***paramName***' -- no such parameter** PROBLEM: An attempt was made to retrieve the value of a parameter with the name *paramName* from a configurable module or object, but the module or object did not recognize the parameter with that name. CAUSE: The configurable module or object did not have a parameter with that name. This is commonly due to misspelling the parameter name. SOLUTION: Be sure to spell the parameter name correctly, and that the configurable module or object has a parameter with that name.

High-Level Client Interface Messages

Following are the messages that originate from inside the high-level client interface module.

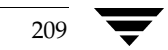
Message Code	Message and Description
HLS601E	**HLS601E (CRXObjectWithProperties::GetSubObjects)** *parentObject* **is unable to provide** *childObjectKind* PROBLEM: The given parentObject cannot provide an iterator for objects of the given *childObjectKind*. CAUSE: The parent object does not and cannot contain objects of the desired kind. For example, job objects cannot provide a destinationRule iterator. SOLUTION: Do not ask *parentObject* for *childObjectKind* iterators for object kinds that it does not and cannot contain.
HLS603I	**HLS603I** *objectKind* **cache starting, using RMS '***name***'...** CAUSE: This message means that the cache containing objects of type *objectKind* is starting, and will receive update events from the Replication Management Server named *name*. This message will appear in the "Trace_HLSOB..." log file if the cache's **debugLevel** configuration parameter is set to 1 or higher.
HLS604I	**HLS604I** *objectKind* **cache started,** *objectCount* **object(s)** CAUSE: This message means that the cache containing objects of type *objectKind* has successfully started, and initially contains *objectCount* objects. This message will appear in the "Trace_HLSOB..." log file if the cache's **debugLevel** configuration parameter is set to 1 or higher.
HLS605I	**HLS605I** *objectKind* **cache stopped** CAUSE: This message means that the cache containing objects of type *objectKind* has stopped. This message will appear in the "Trace_HLSOB..." log file if the cache's **debugLevel** configuration parameter is set to 1 or higher.

Message Code	Message and Description
HLS606E	**HLS606E** *objectKind* **cache event subscription to RMS failed,** *objectCount* **object(s),** *uncommittedCount* **uncommitted,** *iteratorCount* **iterator(s) affected** PROBLEM: The database update event subscription to the Replication Management Server failed for the cache containing objects of type *objectKind*. At the time this happened, the cache contained *objectCount* committed objects, *uncommittedCount* uncommitted objects, and had *iteratorCount* open iterators registered to it. This message is usually immediately followed by the **HLS605I** message. CAUSE: Some of the most common reasons for this kind of failure are: the RMS's network connection may have failed; or the RMS or ENL services on the RMS were stopped; or the client host machine's network connection may have failed; or the client host machine's ENL service may have stopped. SOLUTION: Check to make sure that the RMS machine is operable and its RMS and ENL services are started. Also verify that the host machine can "see" the RMS machine on the network. Then ensure that the local host machine's ENL service is started.
HLS607W	**HLS607W (FilterWithIndexingSpec) Index range** *indexRange* **ignored -- not in result set** PROBLEM: An indexing specification contained an index range that did not intersect any of the resulting set of objects. This is only a warning – it does not indicate that the filtering operation failed. CAUSE: The *indexRange* was out of bounds from the object list that was at hand. For example, this can happen in srTool with the command **`list jobs 13 thru 24`**, if there was, say, only one job defined. SOLUTION: Be sure that the index ranges that are supplied in indexing specifications are valid for the resulting object set.
HLS608W	**HLS608W (FilterWithIndexingSpec) Index range** *indexRange* **ignored -- not valid** PROBLEM: An indexing specification contained an *indexRange* that was invalid. CAUSE: This is caused by the *indexRange* having an end index value that is numerically less than the start index value. SOLUTION: Because index ranges are automatically created in a valid state, this error should not ever occur during normal operation of srTool or srConsole. If this warning persists, please contact Technical Support.



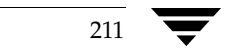
Message Code	Message and Description
HLS609W	**HLS609W (FilterWithGroupingSpec) Grouping spec** *groupingSpec* **ignored -- not valid** PROBLEM: The grouping specification *groupingSpec* that was being used to filter a set of objects was invalid. CAUSE: This is caused by an uninitialized or cleared *groupingSpec* being used to filter a set of objects. SOLUTION: This error should not ever occur during normal operation of srTool or srConsole. If this warning persists, please contact Technical Support.
HLS610E	**HLS610E** *exceptionType* **Exception caught in file** *fileName* **on line** *lineNumber* PROBLEM: A run-time exception of type exceptionType (“Memory” or “General”) was caught at line *lineNumber* of the file *fileName*. CAUSE: This usually is due to insufficient system memory. SOLUTION: Provided there is sufficient system memory for the kind of task being attempted, this error should not occur during normal operation of srTool or srConsole. If this warning persists, contact Technical Support.

Message Code	Message and Description
HLS611E	**HLS611E (CheckProperties) Property '***propertyName***' (***propertyID***) cannot be specified at creation time for '***objectKind***' objects, but was specified to be '***dataValue***'** PROBLEM: An initial value of *dataValue* for the property having the name *propertyName* (and ordinal value *propertyID*) was specified for a new object of type *objectKind* being created. That particular property is not allowed to be specified for new objects. CAUSE: This error usually manifests itself in srTool through the **add** command in which the user has specified an initial property value, and that property's "RequiredToCreate" meta-property is "CannotBeSpecified". For example, the command **`add job with type=OneToOne, pairCount=0`** will produce this error, because the **PairCount** property cannot be specified at creation time for job objects. SOLUTION: Be sure that any initial property values are for properties that can legally be set at object creation time. In the "srTool Object Reference" note the "Require to Create" column in the property tables for each object. Also, the **Property** meta-object's **RequiredToCreate** property indicates whether a property can be specified at object creation time. For example, the srTool command **`get name, requiredToCreate of all properties whose requiredToCreate NE CannotBeSpecified of objectKind ?FileReplicationJob?`** will display just those properties that are legal to specify when creating new jobs.
HLS612E	**HLS612E (CheckProperties) Property '***propertyName***' (***propertyID***) is required for creating '***objectKind***' objects, but was not specified** PROBLEM: The property having the name *propertyName* (and ordinal value *propertyID*) was not specified for a new object of type *objectKind* being created. That particular property is required to be specified for new objects. CAUSE: This error usually manifests itself in srTool through the **add** command in which the user hasn't specified an initial property value, and that property's "RequiredToCreate" meta-property is "MustBeSpecified". For example, the command **add job** will produce this error, because the **Type** property must be specified at creation time for job objects. SOLUTION: Be sure that any initial property values that are required for objectKind objects are set at object creation time. In the "srTool Object Reference" note the "Require to Create" column in the property tables for each object. Also, the **Property** meta-object's **RequiredToCreate** property indicates if a property must be specified at object creation time. For example, the srTool command **`get name, requiredToCreate of all properties whose requiredToCreate EQ MustBeSpecified of objectKind ?FileReplicationJob?`** will display just those properties that must be specified when creating new jobs.



Message Code	Message and Description
HLS613W	**HLS613W (CheckProperties) For creating '***objectKind* **' object(s), property '***propertyName*'**(***propertyID*)** was of type '***actualDataType*'**, but should be of type '***requiredDataType*'** PROBLEM: The property having the name *propertyName* (and ordinal value *propertyID*) that was specified for a new object of type *objectKind* being created had a value of type *actualDataType*, but should have been of type *requiredDataType*. CAUSE: This error usually manifests itself in srTool through the **add** command in which the user has specified one or more initial property values, and one of the property's data values is of the wrong data type. For example, the command **`add job with type = ?OneToMany?`** will produce this error, because the **Type** property must be of type **uint32**. SOLUTION: Be sure that any initial property values being specified while creating *objectKind* objects are set to values with the proper data types expected for them. In the "srTool Object Reference" note the "Data Type" column in the property tables for each object. Also, the Property meta-object's **DataType** property indicates the intrinsic data type of the property. For example, the srTool command **`get name, dataType of all properties of objectKind ?FileReplicationJob?`** will show the names and data types of all properties of jobs.
HLS614I	**HLS614I Transaction** *transactionID* **created in** *methodName* **for** *object* CAUSE: This message means that a new transaction with the globally unique identifier *transactionID* has been created for the given *object*. This usually means the object is about to be edited (changed) or deleted, although for jobs, it can also mean that a job is being created. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being edited is set to 1 or higher; for jobs being created, this message always appears in the log.
HLS615I	**HLS615I Transaction** *transactionID* **committed in** *methodName* **for** *object* CAUSE: This message means that the existing transaction with the globally unique identifier *transactionID* has been committed for the given *object*. This usually means the object creation, or its changes, or its deletion has been committed on the RMS. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being added, changed or deleted is set to 2 or higher.

Message Code	Message and Description
HLS615E	**HLS615E Transaction** *transactionID* **commit failed in** *methodName* **on object** *object* **--** *reason* PROBLEM: The existing transaction with the globally unique identifier *transactionID* could not be committed for the given *object* for the given *reason*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being added, changed or deleted is set to 1 or higher. CAUSE: This problem can be caused by several things, including bad network connections for the client machine running srTool or srConsole; bad network connections for the RMS; or a lack of free disk space on the RMS. This message is normally followed by one or more additional messages that should provide other information about the actual cause of the commit failure. SOLUTION: Use the information gleaned from the additional messages that follow this one to diagnose and correct the cause of the failure.
HLS616I	**HLS616I Transaction** *transactionID* **about to commit** *action* **of** *count* **object(s):** *objectListing* CAUSE: This message means that the *action* (**creation, change** or **deletion**) of *count* objects is about to be committed for the existing transaction with the globally unique identifier *transactionID*. A listing of the objects being added, changed or deleted immediately follows this message. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being added, changed or deleted is set to 2 or higher.
HLS617I	**HLS617I Transaction** *fromTransactionID* **merged into transaction** *toTransactionID* **in CRXRMSDBObject::MergeChanges** CAUSE: This message means that all creations, changes and deletions associated with the existing transaction with the globally unique identifier *fromTransactionID* will be merged into the existing transaction with the globally unique identifier *toTransactionID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being added, changed or deleted is set to 2 or higher.

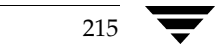


Message Code	Message and Description
HLS617E	**HLS617E Transaction** *fromTransactionID* **merge into transaction** *toTransactionID* **failed in CRXRMSDBObject::MergeChanges –** *reason* PROBLEM: The existing transaction with the globally unique identifier *fromTransactionID* (which should be "**{00000000-0000-0000-0000-000000000000}**") could not be merged into the transaction having the globally unique identifier *toTransactionID* because of the specified *reason* (which should be "RXRESULT_IllegalOperation"). This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being added, changed or deleted is set to 1 or higher. CAUSE: This is caused by an object being asked to merge its changes into another transaction when its *fromTransactionID* is **{00000000-0000-0000-0000-000000000000}**, which means the object was not being edited. This problem should never occur during normal use of srTool or srConsole. SOLUTION: If the problem persists, please contact Technical Support.
HLS618I	**HLS618I Transaction** *transactionID* **recording change in property** *propertyName* **of** *object***:** *newValue* **–** *action* **uncommitted list** CAUSE: This message means that the *object*'s property *propertyName* was changed to the value *newValue* under the transaction having the globally unique identifier *transactionID*. If *action* is "added to," this means that the change is the first change ever recorded for that object's property under the transaction. If *action* is "modified" or "patched," it means that the object's property has been changed before under the transaction. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being changed is set to 3 (the maximum).
HLS620E	**HLS620E (InternalCheckForValidPathVolume) Volume '***driveLetter***' of path '***pathSpec***' of server '***serverName***' is not NTFS** PROBLEM: The volume having the given *driveLetter* of the path *pathSpec* of the server named *serverName* does not have the NTFS volume format. CAUSE: This can be caused by attempting to add a path rule to a job and the path rule's path property refers to a directory that is on a non-NTFS volume. This error can also be caused by a destination rule that refers to a non-NTFS volume. SOLUTION: Be sure that the **path** properties of path rule and destination rule objects are rooted to NTFS-formatted volumes.

Message Code	Message and Description
HLS621E	**HLS621E (AddReplicationPair) '***ServerObject***' does not have a valid default target path** PROBLEM: While attempting to add a new replication pair to an existing job, the *ServerObject* was found to not have a valid default target path. CAUSE: This is normally caused by using a server whose DefaultTargetPath property has been mis-configured to a path that doesn't exist or to an invalid path. In rare cases, it can be due to misconfigured software on the server itself. SOLUTION: Be sure that the **DefaultTargetPath** property of the *ServerObject* in question is not empty, is valid, and actually exists on that server.
HLS622E	**HLS622E (CRXCredentialDBNT)** *operation***: Host OS function '***functionName***' failed** PROBLEM: The *operation* (Open, Get, Put, Find, Delete or Enumerate) involving credential objects failed inside the host operating system function named *functionName* (for example, OpenRegKeyIfExists, RegQueryValueEx, and so on). CAUSE: This problem should not normally occur during normal operation of srTool or srConsole. SOLUTION: If this problem persists, please contact Technical Support.
HLS623E	**HLS623E (CanConnect) Unable to connect to** *serverObject* **-- server not available** PROBLEM: The *serverObject*'s **IsAvailable** property was "false" and a request was made to connect to the server. CAUSE: This problem is most commonly seen when adding a replication pair to a job, and one of the servers of the new pair is unavailable. SOLUTION: Be sure that the server being used in a new replication pair being added to a job is powered on, connected to the network and its replication services started. Also be sure that the host machine running srTool or srConsole can "see" the other server on the network.
HLS624I	**HLS624I Transaction** *transactionID* **released in EndEdit** CAUSE: This message means that the transaction having the globally unique identifier *transactionID* has been released. No other editing can be done using that *transactionID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being edited is set to 1 or higher.

Message Code	Message and Description
HLS625I	**HLS625I Transaction** *transactionID* **aborted in DiscardChanges** CAUSE: This message means that all creations, changes and/or deletes recorded in the transaction having the globally unique identifier *transactionID* have been discarded. The transaction is not released and can be used for other creations, changes and/or deletes. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being edited is set to 1 or higher.
HLS626I	**HLS626I** *Object* **locked for exclusive editing by lock** *lockID* CAUSE: This message means that the given *object* has been locked for exclusive access under the globally unique identifier *lockID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being locked is set to 1 or higher. It will also appear in the log for newly created jobs.
HLS627I	**HLS627I** *Object* **unlocked from exclusive editing -- lock** *lockID* **released** CAUSE: This message means that the given *object* has been unlocked from exclusive access under the globally unique identifier *lockID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the object being locked is set to 1 or higher.
HLS628E	**HLS628E (CRXFileReplicationJob) Illegal value '***newName***' for Name property -- duplicated, already exists** PROBLEM: A job was being created or its **Name** property was being changed, and the specified *newName* matched the name of another existing job. CAUSE: This error occurs whenever the user attempts to create a job that has the same name as another job, or whenever the user attempts to rename a job to the same name as another. SOLUTION: Be sure that the *newName* is unique among all other jobs.

Message Code	Message and Description
HLS629E	**HLS629E (***moduleName***) Illegal value '***dataValue***' for** *propertyName* **property -- length exceeds maximum allowable value (***maxLength***) or is empty** PROBLEM: The property named *propertyName* was being specified for a new object being created or for an existing object being changed, and the *dataValue* was either empty and an empty value was not allowed, or its length exceeded *maxLength*, the maximum allowable length. CAUSE: This error can occur whenever a character-string- or path- type property is initialized while creating a new object or is set while changing an existing object. Some of these properties have different string or path length limitations, depending on the technical requirements underlying them. Some properties cannot be set to the empty string, while others can. SOLUTION: Be sure to follow the rules for setting property values for the objects of interest. Consult the property tables in “srTool Object Reference” on page 135.
HLS630E	**HLS630E (***moduleName***) Illegal value '***dataValue***' for** *propertyName* **property -- enumerated value outside legal bounds of** *minValue* **thru** *maxValue* PROBLEM: The property named *propertyName* was being specified for a new object being created or for an existing object being changed, and the *dataValue* was less than *minValue* or greater than *maxValue*. CAUSE: This error can occur whenever a property is initialized while creating a new object or is set while changing an existing object. Many of these properties have different restrictions on the values they can have, depending on the technical requirements underlying them. SOLUTION: Be sure to follow the rules for setting property values for the objects of interest. Consult the property tables in “srTool Object Reference” on page 135.



Message Code	Message and Description
HLS631E	**HLS631E (ReplicationPair) Incompatible servers: source OS is** *sourceOS*, **target OS is** *targetOS* PROBLEM: A new replication pair was being added to a job, and the servers had incompatible operating systems on them. CAUSE: This error was more prevalent in older versions of Storage Replicator, which used to operate on Windows NT version 4. This is no longer the case, so it should not be possible to encounter this message during normal use of srTool or srConsole. SOLUTION: If the problem persists, please contact Technical Support.
HLS632E	**HLS632E** (*ReplicationPair*) **Unable to add more than one pair to one-to-one job** PROBLEM: An attempt was made to add a new replication pair to an existing one-to-one job. CAUSE: This error is usually caused by srTool users trying to add a replication pair to a one-to-one job that already has one. SOLUTION: To replace the replication pair that's inside a one-to-one job in srTool, first delete the existing pair (using the **delete** command), then add the new pair to the job (using the **add** command).
HLS633E	**HLS633E (***functionName***) Duplicate** *objectKind* **--** *object* **already exists** PROBLEM: An attempt was made to add a new object of type *objectKind* but the *object* already exists. CAUSE: This error is usually caused by srTool users inadvertently trying to add an object when it already exists, or trying to change an object that would conflict with another existing object. SOLUTION: When adding pairs to jobs, be sure that no other pair has the same source and target servers. When adding path rules, be sure no other rule has the same source server and path property. When adding selection rules, be sure no other selRules in the path rule have matching **nameSpec, isExclude** and **isRecursive** properties. When changing the **targetServer** property of destination rules, be sure no other destination rules of the owning path rule have a matching target server.

Message Code	Message and Description
HLS634E	**HLS634E (AddReplicationPair) Illegal** *sourceServerOrTargetServer serverObject* **-- cannot use same server as** *publicationOrCentralization* **job's established JCD (currently** *serverName***)** PROBLEM: The specified **sourceServer** property of a new pair being added to a centralization job matches the name of the established target server of the job, or the specified **targetServer** of a new pair being added to a publication job matches the name of the established source server of the job. Either case is illegal. CAUSE: This message is most commonly seen by srTool users who are trying to add a pair to an existing centralization or publication job. For example, given a publication job "Foo" that replicates data from server "A" to servers "X" and "Y", this error would occur for the following srTool command: `add pair to job ?Foo? with sourceServer = ?Z?, targetServer = ?A?` SOLUTION: Be sure that new pairs added to publication (one-to-many) jobs use a **targetServer** that differs from the established source server of the job's other pairs. Similarly, be sure that new pairs added to centralization (many-to-one) jobs use a **sourceServer** that differs from the established target server of the job's other pairs.
HLS635E	**HLS635E (AddReplicationPair) Illegal** *sourceServerOrTargetServer serverObject* **-- cannot use** *sourceServerOrTargetServer* **that differs from** *publicationOrCentralization* **job's established JCD (currently** *serverName***)** PROBLEM: The specified **sourceServer** property of a new pair being added to a publication job doesn't match the name of the established source server of the job, or the specified **targetServer** of a new pair being added to a centralization job doesn't match the name of the established target server of the job. Either case is illegal. CAUSE: This message is most commonly seen by srTool users who are trying to add a pair to an existing centralization or publication job. For example, given a publication job "Foo" that replicates data from server "A" to servers "X" and "Y", this error would occur for the following srTool command: `add pair to job ?Foo? with sourceServer = ?B?, targetServer = ?Z?` SOLUTION: Be sure that new pairs added to publication (one-to-many) jobs use a **targetServer** that matches the established source server of the job's other pairs. Similarly, be sure that new pairs added to centralization (many-to-one) jobs use a **sourceServer** that matches the established target server of the job's other pairs.

Message Code	Message and Description
HLS636E	**HLS636E (LookupObjectByName) No such** *objectKind* **object with name '***objectName***' (using transaction ID** *transactionID***)** PROBLEM: An object of type *objectKind* could not be found that had the name *objectName*. CAUSE: This is usually caused by an srTool user mis-spelling the name of a server while adding a **pair**, **pathRule** or **destinationRule** object. SOLUTION: Be sure to correctly specify the name of a server or use an property reference in an expression instead (for example, **`name of first server whose…`**).
HLS637E	**HLS637E (GetProperty) Property '***propertyName***' (***propertyID***) does not exist in property table of** *object* PROBLEM: A property named *propertyName* (whose ordinal value is *propertyID*) was requested from the given *object* which did not have such a property. CAUSE: This error should never happen in the console, but it is relatively easy to make it happen in srTool, such as, for example, in this command: **`echo -x sourceServer of first rms`** Objects of type **RMS** do not have a **sourceServer** property. SOLUTION: In srTool, be sure to specify only those properties that truly exist for the object(s) of interest. Consult the property tables in “srTool Object Reference” on page 135.
HLS638E	**HLS638E (InternalTestForIllegalNameSpec) Illegal character '***illegalChar***' found in path specification '***pathSpec***' -- these are illegal:** *illegalCharacters* PROBLEM: The path specification *pathSpec* contained an illegal character *illegalChar*. CAUSE: This error is caused by specifying a **path** property that contains one or more illegal characters. On Windows, the characters <, >, | and / cannot comprise a file or path name. This error can readily be produced in srTool: **`add rule to job "X" with sourceserver = "Y", path = "C:\\foo>bar"`** The ‘>’ character is not allowed in the **path** property. SOLUTION: Be sure that any paths that are specified in srTool do not contain the characters that are disallowed.

Message Code	Message and Description

HLS638W	(InternalTestForIllegalNameSpec) **Possible unintended character** *suspect* **found in path specification** *pathSpec*
	PROBLEM:
	The path specification *pathSpec* contained a character *suspect* that may not have been intended by the user.
	CAUSE:
	This warning is caused by specifying a path property that contains one or more characters that are easily produced accidentally in srTool. Many users will forget that backslash characters, the path delimiter in Windows file systems, are used as an escape character for srTool string literals. Thus, they may inadvertently type the following srTool command:
	`add rule to job "X" with sourceserver = "Y", path = "c:\reports\tuesday"`
	srTool will gladly add the path anchored to the following path:
	`c : CR e p o r t s HT u e s d a y`
	...where CR is a carriage-return character, and HT is a horizontal tab character. This is probably not what most users would expect nor desire; thus, the reason for this warning message. This warning is telling the user to instead code the following:
	`add rule to job "X" with sourceserver = "Y", path = "c:\\reports\\tuesday"`
	SOLUTION:
	Be sure that any paths that are specified in srTool do not contain escaped characters unless they are explicitly needed. See the section on "string constants".

HLS639I	**HLS639I (SOB_Open) HLSOB** *version* **is now open** CAUSE: This message means that the high-level client interface to the underlying replication system, having the given *version*, was successfully initialized and opened, which can only happen if the RMS is found on the network and the local host machine's ENL service is started and running. This message will always appear in the "Trace_HLSOB..." log file each time an instance of the console or srTool is started.
HLS640I	**HLS640I (SOB_Close) HLSOB is now closed** CAUSE: This message means that the high-level client interface was properly closed. This message will always appear in the "Trace_HLSOB..." log file each time an instance of the console or srTool is closed.

Message Code	Message and Description
HLS641E	**HLS641E (CRXIterator) Open: Live updates were requested but no CRXLiveUpdater was specified** PROBLEM: An iterator was asked to open with live update support, but no live-updater instance was supplied. CAUSE: This error should never happen during normal operation of srTool or the console. SOLUTION: If the problem persists, please contact Technical Support.
HLS642W	**HLS642W (***objectKind* **Cache) Object** *objectID* **changed, but was not in the cache** PROBLEM: The cache containing objects of type *objectKind* was notified by the event redirector that one or more properties of the object having the globally unique identifier *objectID* had changed, but the object was not found in the cache. CAUSE: This can happen under rare conditions when the console or srTool is launched while thousands of changes are taking place in jobs, pairs, scripts, rules, selRules, destRules and/or servers. SOLUTION: Generally, there is no need to take any action. If the problem occurs in the absence of any RMS activity, please contact Technical Support.
HLS643E	**HLS643E (CRXServer) Server object was created with a NULL (empty) name** PROBLEM: While attempting to contact a server object for purposes of a remote procedure call (RPC), it was found that the server had no name. CAUSE: This error should not happen during normal operation of srTool or the console. SOLUTION: If the problem persists, please contact Technical Support.

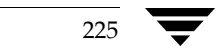
Message Code	Message and Description
HLS644E	**HLS644E (CRXObjectContainer::GetRootObjects)** *objectKind* **objects are not root-level objects** PROBLEM: An iterator for root-level objects of type *objectKind* was requested, but such objects are not root-level objects.. CAUSE: This error most commonly occurs in srTool when there is no default object specification in use. For example, these two commands will cause the error: `use none; count all jobs` Job objects come from RMS objects, not from the root-level. SOLUTION: Be sure that all object specifications, in the absence of a default object spec, are rooted with one of the root-level object types. See "srTool Object Hierarchy" on page 64.
HLS645I	**HLS645I (CRMSDBEventRedirector) Starting event redirector** CAUSE: This message means that the event redirector is starting. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 1 or higher.
HLS646I	**HLS646I (CRMSDBEventRedirector) Event redirector** *verb* CAUSE: This message means that something happened to state of the event redirector, indicated by the *verb* (which is either "paused," "resumed" or "started"). This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 1 or higher.
HLS647I	**HLS647I (CRMSDBEventRedirector) Event redirector ended,** *statisticsDump* CAUSE: This message means that the event redirector has stopped. A dump of the redirector's run-time statistics immediately follows the message. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 1 or higher.

Message Code	Message and Description
HLS648E	**HLS648E (CRXObjectWithProperties) CreateSubObject: Unable to add** *objectKind* **object to** *parentObject* PROBLEM: The *parentObject* could not add a child object of type *objectKind* because such object types don't belong to the parent type of object. CAUSE: This error most commonly occurs in srTool when trying to add an object using the **add** command but there's no "**to**" clause. For example, this command will cause this error: **`add pair to first rms`** This command will fail because pairs belong to jobs, not RMSs. SOLUTION: Be sure that when adding an object to a parent object, that the parent object can contain an object of type *objectKind*. See "srTool Object Hierarchy" on page 64.
HLS649E	**HLS649E (***objectKind***) Property '***propertyName***' expected '***dataType***' data** PROBLEM: The property named *propertyName* for a replication pair being created or changed was being set to a data value that was not of the expected type *dataType*. CAUSE: This error most commonly occurs in srTool when trying to add an object using the **add** command but there's no "**to**" clause. For example, this command will cause this error: **`add pair to first rms`** This command will fail because pairs belong to jobs, not RMSs. SOLUTION: Be sure that when adding an object to a parent object, that the parent object can contain an object of type *objectKind*. See "srTool Object Hierarchy" on page 64.
HLS650E	**HLS650E (CreateRootObjects) Unable to create root-level object of type '***objectKind***'** PROBLEM: The root-level object type *objectKind* could not be created. CAUSE: This error occurs when an srTool user tries to create any kind of root-level object other than credentials. Other than credentials, all other root-level objects cannot be created. SOLUTION: Except for credentials, do not try to create any other root-level objects.

Message Code	Message and Description
HLS652I	**HLS652I (CRMSDBEventRedirector) ENL event** *eventCount***: About to handle event of class** *eventClass***, subclass** *eventSubClass***, type** *eventType* CAUSE: This message means that the event redirector has received an ENL event of *eventClass*, *eventSubClass* and *eventType*. The *eventCount* shows the total number of events received by the redirector since it was started. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 3 (the maximum).
HLS653W	**HLS653W (CRMSDBEventRedirector) ENL event** *eventCount***: Event not handled due to invalid event type** PROBLEM: The event redirector received an ENL event that had an event type that was not expected. CAUSE: This warning should not occur during normal operation of srTool or srConsole. SOLUTION: If the warning persists, contact Technical Support.
HLS654E	**HLS654E (CRMSDBEventRedirector) ENL event** *eventCount***: Event handler returning failure –** *reason* PROBLEM: A failure of some sort occurred in the event redirector's event handler for the event whose sequence number is *eventCount*. CAUSE: The failure is due to the given *reason*. This message may be followed by one or more additional messages that can help diagnose the problem. SOLUTION: If the problem persists, contact Technical Support.
HLS656I	**HLS656I (CRMSDBEventRedirector) ENL event** *eventCount***: Enqueued successfully** CAUSE: This message means that the event redirector has successfully enqueued the ENL event whose sequence number is *eventCount* (which is the total number of events received by the redirector since it was started). This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 3 (the maximum).

Message Code	Message and Description
HLS656E	**HLS656E (CRMSDBEventRedirector) ENL event** *eventCount***: Failed to enqueue –** *reason* PROBLEM: Because of the given *reason*, the event redirector's event handler was not able to enqueue the ENL event whose sequence number is *eventCount* (which is the total number of events received by the redirector since it was started). CAUSE: The failure is due to the given *reason*, which, in most cases is caused by insufficient memory resources. This message may be followed by one or more additional messages that can help diagnose the problem. SOLUTION: If the problem persists, contact Technical Support.
HLS657S	**HLS657S (CRMSDBEventRedirector) Unable to acquire name of RMSDB event publisher** PROBLEM: The event redirector was not able to get the name of the Replication Management Server, which prevented the redirector from starting. CAUSE: The name of the Replication Management Server (RMS) was unknown at the time the event redirector was asked to start. SOLUTION: Check to ensure that the RMS is running and connected to the network. If the problem persists, contact Technical Support.
HLS658S	**HLS658S (CRMSDBEventRedirector) Thread::resume failed** PROBLEM: The event redirector could not resume its thread function.. CAUSE: This is probably due to a lack of resources available to srTool or the console. SOLUTION: Check to ensure that there are sufficient memory, thread and handle resources available on the local host machine. If the problem persists, contact Technical Support.
HLS659I	**HLS659I (CNotifier) Distributing notification '***notification***' to** *subscriberCount* **subscribers** CAUSE: This message means that the event redirector, a kind of notifier, is about to distribute the *notification* to a number (*subscriberCount*) of subscribers. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the redirector object is set to 1 or higher.

Message Code	Message and Description
HLS660W	**HLS660W (CNotifier) Subscriber** *subscriberInstance* **update notification reported problem for '***notification'***— resultCode** PROBLEM: The notifier's *subscriberInstance* received the *notification* but returned the given *resultCode*. CAUSE: This warning message indicates that the subscriber had a minor problem to report. This message may be followed by one or more messages that can help diagnose the underlying problem. SOLUTION: If the problem persists, contact Technical Support.
HLS660E	**HLS660E (CNotifier) Subscriber** *subscriberInstance* **update notification failed for '***notification***'—***reason* PROBLEM: The notifier's *subscriberInstance* failed while receiving the *notification* because of the given *reason*. CAUSE: This message may be followed by one or more messages that can help diagnose the underlying problem. SOLUTION: If the problem persists, contact Technical Support.
HLS661I	**HLS661I (***objectKind* **Cache) FillObjectCache: Using packed enumeration protocol** CAUSE: This message means that the cache containing objects of type *objectKind* is starting and is about to be filled using the packed enumeration protocol. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 2 or higher.
HLS662E	**HLS662E (***objectKind* **Cache) FillObjectCache: Failed** – *reason* PROBLEM: The cache containing objects of type *objectKind* could not be filled due to the given *reason*. CAUSE: This message may be followed by one or more messages that can help diagnose the underlying problem. SOLUTION: If the problem persists, contact Technical Support.



Message Code	Message and Description
HLS663W	**HLS663W (***object***) InternalSyncFromPropValueMap: Unexpected property** *propertyName* **(***propertyID***) received, value =** *dataValue* PROBLEM: The property named *propertyName* (whose ordinal value is *propertyID*) was being set to the given *dataValue* and the property was not already in the *object's* property cache. This is not expected behavior, but is not fatal. The property is not added to the *object*'s property cache. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* is set to 1 or higher. CAUSE: This problem should not occur during normal operation of srTool or srConsole. SOLUTION: If the problem persists, contact Technical Support.
HLS664I	**HLS664I (RMSDBObject) InternalMergeTransactions: Moved uncommitted changed property** *propertyName* **(***propertyID***) for** *object* **from transaction** *sourceTransactionID* **to** *destTransactionID* CAUSE: This message means that the changed but not-yet-committed property named *propertyName* (whose ordinal value is *propertyID*) of the given *object* was successfully transferred from the transaction whose globally unique identifier is *sourceTransactionID* to the transaction *destTransactionID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* is set to 2 or higher.
HLS666I	**HLS666I (***objectKind* **Cache) MakeNewCacheEntry succeeded for transaction** *transactionID***:** *object* CAUSE: The cache containing objects of type *objectKind* has successfully added the given *object* under the transaction whose globally unique identifier is *transactionID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).

Message Code	Message and Description
HLS666E	**HLS666E (***objectKind* **Cache) MakeNewCacheEntry failed for transaction** *transactionID: object* | *propertyList* **--** *reason* PROBLEM: The cache containing objects of type *objectKind* failed while, under the transaction having the globally unique identifier of *transactionID*, trying to add the given *object* or add the object that it created from the given *propertyList* due to the given *reason*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* is set to 1 or higher. CAUSE: This problem should not occur during normal operation of srTool or srConsole. If it occurs, it is most likely due to a lack of memory resources on the host machine. SOLUTION: If the problem persists, contact Technical Support.

HLS667E	**HLS667E (***objectKind* **Cache) LookupObject:** *objectKind* **object** *objectID* **not found**
	PROBLEM:
	The cache containing objects of type *objectKind* was asked to find an object whose **ID** property (a globally unique identifier) is *objectID* and there was no such object with that ID in the cache.
	CAUSE:
	The most common way for this problem to occur is when an srTool user tries to start, stop or cancel a job that has no pairs. For example,
	`add job with name = "X", type = OneToOne;`
	`start job "X"`
	Job control operations are directed to the job's JCD server. In this example, the job has a "null" JCD, which causes the server cache to respond with this error.
	SOLUTION:
	Be sure that a job has at least one pair before trying to start, stop or cancel it.

Message Code	Message and Description
HLS668E	**HLS668E** *job* **has no JCD (job control delegate) server -- perhaps it has no pairs** PROBLEM: The Job Control Delegate server for the given *job* object was requested, but the globally unique identifier of the JCD server was **`{00000000-0000-0000-0000-000000000000}`**. CAUSE: The most common way for this problem to occur is when an srTool user tries to start, stop or cancel a job that has no pairs. For example, **`add job with name = "X", type = OneToOne;`** **`start job "X"`** Job control operations are directed to the job's JCD server. In this example, the job has a "null" JCD, which causes the server cache to respond with this error. SOLUTION: Be sure that a job has at least one pair before trying to start, stop or cancel it.
HLS669E	**HLS669E (CCSJacket) Failure in dynamic_cast: Client connection object** *ptr* **not of type** '*class*' PROBLEM: The client connection object having the instance pointer *ptr* could not be dynamically cast to a pointer to the given *class*. CAUSE: This problem should not occur during normal operation of srTool or srConsole. SOLUTION: If the problem persists, contact Technical Support.
HLS670E	**HLS670E (CRXIterator) Update failure, possible dropped live update: object=***object***, what=***updateType***, propIDs=***propertyIDs***,** *iteratorInfo* **--** *resultCode* PROBLEM: An object iterator was being notified about the addition, deletion or change (specified by *updateType*) of the given *object*.which can cause the console to lose information and cause its list views to become out-of-date. CAUSE: This problem should not occur during normal operation of srTool or srConsole. SOLUTION: If the problem persists, contact Technical Support.

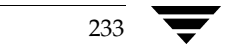
Message Code	Message and Description
HLS671E	**HLS671E (UnlockControllingObject) Unlock failed for** *object* **using lock token** *lockID* **–** *reason* PROBLEM: The *object* was being unlocked from exclusive write access under the lock with the globally unique identifier *lockID* when the unlock operation failed due to the given *reason*. CAUSE: This problem is usually caused by network connectivity problems between the host machine and the Replication Management Server (RMS) while editing a job or any of the objects it contains. SOLUTION: Be sure the network connection between the local host machine and the RMS is reliable. If the problem persists, contact Technical Support.
HLS672E	**HLS672E (CRXIterator)** *iteratorInfo* **update queue overflowed at** *updateCount* **updates** PROBLEM: The update queue of the object iterator having the given *iteratorInfo* overflowed when the number of updates in the queue reached the *updateCount* value. CAUSE: This problem is caused by an iterator that was opened with a live updater attached to it, and live updates from the iterator have been suspended, and the iterator has been receiving thousands of add/change/delete updates since the suspension began. This can happen in srTool by monitoring an active set of objects, pausing the monitor and then walking away from the local host, leaving srTool running. Given enough system activity, the iterator that backs the monitor will eventually overflow. SOLUTION: Be sure that any paused monitors get resumed or stopped before they overflow. See “`monitor` command” on page 112.

Message Code	Message and Description
HLS673E	**HLS673E (CRXIterator)** *iteratorInfo* **update queue overflowed at** *updateCount* **updates -** *missedCount* **update(s) missed** PROBLEM: The update queue of the object iterator having the given *iteratorInfo* overflowed when the number of updates in the queue reached the *updateCount* value. The number of updates that have been missed since overflow occurred is *missedCount*. CAUSE: This message is usually seen in srTool when executing the **`monitor -resume`** command when the monitor being resumed had overflowed. This message follows the playback of the recorded updates. SOLUTION: Be sure that any paused monitors get resumed or stopped before they overflow. See "`monitor` command" on page 112.
HLS674E	**HLS674E (CRXIterator) Open: '***objectKind***' iterator being opened with '***specObjectKind***' object specification** PROBLEM: An iterator that iterates over objects of type *objectKind* was being opened with an object specification that results in objects of type *specObjectKind*. The two object kinds are not the same. CAUSE: This problem should not happen during normal operation of srTool or srConsole. SOLUTION: It the problem persists, contact Technical Support.
HLS675I	**HLS675I (CCSJacket) Retrying RPC to server** *serverName* **using credential '***credential***' on RxBaseClient** *clientID* CAUSE: A Remote Procedure Call to the server named *serverName* using the client object having the given *clientID* failed previously with an "access denied" error and is being retried with the given *credential*. This message will always appear in the "Trace_HLSOB..." log file.

Message Code	Message and Description
HLS676E	**HLS676E (CCSJacket) Unable to set authorization information on RxBaseClient** *clientID* **–** *reason* PROBLEM: A Remote Procedure Call to the server managed by the client object having the given *clientID* failed previously with an "access denied" error and was being retried with another credential. The client communications object failed while being set with the new authentication info. CAUSE: This problem should not happen during normal operation of srTool or srConsole. It may be an "out of resource" issue. SOLUTION: It the problem persists, contact Technical Support.
HLS677E	**HLS677E (CCSJacket) Unable to connect to server** *serverName* **using RxBaseClient** *clientID* PROBLEM: A Remote Procedure Call to the server named *serverName* that is managed by the client object having the given *clientID* failed previously with an "access denied" error and had been retried with all other available credentials to no avail. CAUSE: The access rights of the user running srTool or the console are insufficient to access the given server and there are no other stored credentials that permit access to that server. SOLUTION: Be sure the user running the console or srTool has sufficient access rights to the server, or has stored credentials that will permit access to the server.
HLS678E	**HLS678E (CCSJacket) Unable to open local credential cache –** *reason* PROBLEM: A Remote Procedure Call to a server failed previously with an "access denied" error and was about to be retried with other available credentials, but the local host machine's credential repository could not be opened. CAUSE: This problem should not happen during normal use of the console or srTool. This message may be followed by one or more additional messages that may help diagnose the underlying problem. SOLUTION: If the problem persists, contact Technical Support.

Message Code	Message and Description
HLS679E	**HLS679E (CRMSDBEventRedirector) Failure while copying queued notifications -- continuing distribution of** *count* **notification(s) –** *reason* PROBLEM: A failure occurred while copying *count* notifications just prior to distributing them to subscribers due to the given *reason*. The notifications that were successfully copied will be distributed. CAUSE: This problem should not happen during normal use of the console or srTool and may be due to a lack of available memory resources on the local host machine. This message may be followed by one or more additional messages that may help diagnose the underlying problem. SOLUTION: If the problem persists, contact Technical Support.
HLS680E	**HLS680E (***CreateOrSetProperty***) The job** *nameOrDescription* **(***actualLength* **characters long) exceeds** *maximumLength* **characters in length** PROBLEM: During the *Create* or *SetProperty* operation, the *name* or *description* property of a job was being set to a string value whose *actualLength* exceeded the maximum allowable length of *maximumLength* characters. CAUSE: This error is typically seen in srTool, where it's very easy to overflow the string property of an object. For example, the command **`description = MakeString ("abc", 1000) for first job`** will produce this error. SOLUTION: Be sure that the string value being assigned to an object's property is not too large. Heed the length limit displayed in this message.
HLS682E	**HLS682E (SetProperty) Dynamic replication is not allowed for** *job* **-- it's either using merge replication, or its dynamic journaling is disabled** PROBLEM: The new value for the **IsRealTime** property of the given *job* was not allowed because the job's **TargetReplicaType** property was set to "Merge," or its **NoDynamicJournal** property was "true" – or both. CAUSE: This error is typically seen in srTool, where it is easy to attempt to set conflicting job configuration settings. SOLUTION: Be sure that the job's configuration property settings do not conflict. See the section "Selecting Replication Options" in the *Replication Exec* Administrator's Guide for a description of these settings.

Message Code	Message and Description
HLS683E	**HLS683E (SetProperty) Dynamic replication is not allowed for jobs using merge replication** PROBLEM: The new value of "Merge" for the **TargetReplicaType** property of a job being edited was not allowed because the job's **IsRealTime** property was set to "true." CAUSE: This error is typically seen in srTool, where it is easy to attempt to set conflicting job configuration settings. SOLUTION: Be sure that the job's configuration property settings do not conflict. See the section "Selecting Replication Options" in the *Replication Exec* Administrator's Guide for a description of these settings.
HLS684E	**HLS684E (SetProperty) Disabling dynamic journaling is not allowed for jobs that do dynamic replication** PROBLEM: Setting a new value for the **NoDynamicJournal** property of a job being edited was not allowed because the job's **IsRealTime** property was set to "true." CAUSE: This error is typically seen in srTool, where it's easy to attempt to set conflicting job configuration settings. SOLUTION: Be sure that the job's configuration property settings do not conflict. See the section "Selecting Replication Options" in the *Replication Exec* Administrator's Guide for a description of these settings.
HLS685E	**HLS685E (SetProperty) The schedule mask specified is the wrong type (***actualDataType***) or the wrong length (***actualLength***) -- expected** *expectedDataType* **of length** *expectedLength* PROBLEM: Setting a new value for the **Schedule** property of a job being edited was not allowed because the *actualDataType* differed from the *expectedDataType* or the *actualLength* of the data value differed from the *expectedLength*. CAUSE: This error is typically seen in srTool, where it is relatively easy to assign bad values to certain properties of replication system objects. SOLUTION: Be sure that the data value being assigned to the job's **Schedule** property is of the correct type (**byteArray**) and has the proper size. Heed the expected data type and lengths displayed in this message.

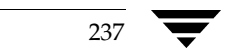


Message Code	Message and Description
HLS686I	**HLS686I (***objectKind* **Cache) Cache state at TransactionCollectChanges:** *cacheDump* CAUSE: The cache containing objects of type *objectKind* is about to report all uncommitted data for a particular transaction. A dump of the cache contents immediately follows this message. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS687E	**HLS687E (LockControllingObject)** *object* **cannot be locked for exclusive access** PROBLEM: A request to lock the controlling object of the given *object* for exclusive write access could not be done. CAUSE: The *object* had no owning object. This problem should never happen during normal use of srTool or srConsole. SOLUTION: If the problem persists, contact Technical Support.
HLS688E	**HLS688E (***module***) Unable to set property '***propertyName***' of '***object***' to '***dataValue***' -- access type (***accessType***) is not 'Mutable'** PROBLEM: The property named *propertyName* of the given *object* could not be set to the new *dataValue* because the access type of the property was something other than "Mutable." CAUSE: The access type of the given property was something other than Mutable. The most common cause of this problem is using srTool to change the value of a Constant or Read-Only property of an object. For example, the srTool command **`set type = OneToMany for job "Foo"`** will fail with this error. SOLUTION: Be sure to use the **set** command to change just those properties of objects that are Mutable.

Message Code	Message and Description
HLS689E	**HLS689E (CRXEditableObject)** *object* **is not locked for editing** PROBLEM: The given *object* was not locked for exclusive write access when it should have been. CAUSE: This error should not happen during normal operation of srTool or the console. It happens if an editable *object* was not first edited (via BeginEdit), but it was asked to change one of its Mutable properties, or it was asked to save (via CommitChanges) or discard (via DiscardChanges) any changes made to it. SOLUTION: If the problem persists, contact Technical Support.
HLS690E	**HLS690E (CRXEditableObject)** *object* **is already locked for editing** PROBLEM: The given *object* was asked to be locked for exclusive write access when it was already locked. CAUSE: This error should not happen during normal operation of srTool or the console. It happens if an editable *object* was first edited (via BeginEdit), and it was asked again to be edited (via BeginEdit). SOLUTION: If the problem persists, contact Technical Support.
HLS691E	**HLS691E (***methodName***) Illegal path specification '***pathSpec***' -- empty or blank** PROBLEM: The given path specification *pathSpec* was empty (blank), which is not allowed. CAUSE: This error is most common in srTool. For example, the srTool command **`add pathRule to job "Foo" with sourceServer = "S1", path = ""`** will produce this error. SOLUTION: Be sure that the string values assigned to properties that expect path specifications are not empty.

Message Code	Message and Description
HLS692E	**HLS692E (CRXRMSDBObject)** *object* **is already being edited under transaction** *transactionID* PROBLEM: The given *object* was asked to be edited when it was discovered that it was already being edited under the auspices of the transaction having the globally unique identifier of *transactionID*. CAUSE: This error should not happen during normal operation of srTool or the console. It happens if an editable *object* was first edited (via BeginEdit), and it was asked again to be edited (via BeginEdit). SOLUTION: If the problem persists, contact Technical Support.
HLS693E	**HLS693E (CRXRMSDBObject)** ***object*** **is not being edited -- no transaction** PROBLEM: The given RMS database-backed *object* was asked to EndEdit, CommitChanges or DiscardChanges, and it was discovered that it did not have an associated transaction. CAUSE: This error should not happen during normal operation of srTool or the console. SOLUTION: If the problem persists, contact Technical Support.
HLS694I	**HLS694I (***method***) Property '***propertyName***' of** *object* **set to** *dataValue* **under transaction** *transactionID* CAUSE: The value of the property named *propertyName* in the property cache of the given *object* was patched to the new dataValue under the auspices of the transaction having the globally unique identifier *transactionID*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS695I	**HLS695I (CRMSDBEventRedirector::AugmentAtomic) Update ID** *updateID***, event number** *eventNumber***,** *notificationKind***:** *status* CAUSE: This message reports the *status* ("added okay" or "update not found") for the update having the globally unique identifier *updateID* and the given *eventNumber* and *notificationKind*.

Message Code	Message and Description
HLS696I	**HLS696I (CRMSDBEventRedirector::EndAtomic) Update ID** *updateID***:** *whatHappened* CAUSE: This message reports *whatHappened* ("Enqueue failed" or "GetAtomicUpdate failed") for the update having the globally unique identifier *updateID*.
HLS697E	**HLS697E (ObjectPtrList::Remove) Failure removing** *object* **from list of** *count* **object(s)** PROBLEM: A list of *count* objects was asked to remove the given *object*, but that *object* wasn't in the list. CAUSE: This error should not happen during normal operation of srTool or the console. SOLUTION: If the problem persists, contact Technical Support.
HLS698I	**HLS698I (***objectKind* **Cache)** *object* **removed from uncommitted list (***itemCount* **items) under transaction** *transactionID* CAUSE: The given *object* (under the transaction having the globally unique identifier *transactionID*) was removed from the uncommitted list of the cache containing objects of type *objectKind*. The *object* was added, the "add" was never committed, and then it was deleted, thus resulting in this message. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS699I	**HLS699I (***objectKind* **Cache)** *object* **remains in uncommitted list (***itemCount* **items, this:** *uncommittedEntry***), now deleted under transaction** *transactionID* CAUSE: The given deleted *object* (under the transaction having the globally unique identifier *transactionID*) was left in the uncommitted list of the cache containing objects of type *objectKind*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS700I	**HLS700I (***objectKind* **Cache)** *object* **added to uncommitted list (***itemCount* **items) as deleted under transaction** *transactionID* CAUSE: The given deleted *object* (under the transaction having the globally unique identifier *transactionID*) was added to the uncommitted list of the cache containing objects of type *objectKind*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).



Message Code	Message and Description
HLS701I	**HLS701I (***objectKind* **Cache)** *object* **removed from main cache,** *itemCount* **items remain** CAUSE: The given deleted *object* was removed from the cache containing objects of type *objectKind*, leaving *itemCount* items in the cache. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS702I	**HLS702I** *class***::***method object***...** or… **HLS702I** *objectKind status* CAUSE: The first form of this message is a simple trace log entry, indicating that the *method* of the *class* was called for the given *object*. The second form of this message is a trace log for an iterator for objects of type *objectKind*, and indicates its *status* (for example, "iterator opening"). This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* or *objectKind* iterator is set to 3 (the maximum).
HLS703E	**HLS703E** *class***::***method* **failed for** *object* **--** *reason* or… **HLS703E** *objectKind operation* **--** *reason* PROBLEM: The first form of the message indicates that the *method* of the *class* failed for the given *object* due to the given *reason*. The second form of the message indicates that the *objectKind* iterator failed while performing the given *operation* (for example, "iterator opening") due to the given *reason*. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* or *objectKind* iterator is set to 1 or higher. CAUSE: The cause of the failure should be described by the given *reason*. SOLUTION: If the problem persists, contact Technical Support.

Message Code	Message and Description
HLS704I	**HLS704I** *class*::*method object* **succeeded** or… **HLS704I** *objectKind operation* CAUSE: The first form of this message is a simple trace log entry, indicating that the *method* of the *class* was completed successfully for the given *object*. The second form of this message is a trace log for an iterator for objects of type *objectKind*, and indicates that its *operation* (for example, "iterator opening") completed successfully. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the *object* or *objectKind* iterator is set to 3 (the maximum).
HLS705I	**HLS705I (***objectKind* **Cache) Uncommitted** *addOrDelete* **of** *object* **transaction changed from** *sourceTransactionID* **to** *destTransactionID* CAUSE: The add/change/delete items of the source transaction whose globally unique identifier is *sourceTransactionID* are being transferred into the destination transaction *destTransactionID*. In the cache that stores objects of type *objectKind*, the *addOrDelete* information for the given *object* was successfully transferred to the new (destination) transaction. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS706I	**HLS706I (***iterator***) Update: Applying** *object* **to filter '***expression***' evaluated to '***dataValue***'** CAUSE: The given *iterator* was notified of an update involving the given *object*, and the given filter *expression* was applied to the *object* resulting in the given *dataValue*. (A non-zero *dataValue* indicates the *object* matches the filter *expression*.) This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).
HLS707I	**HLS707I (***iterator***) Update: UPDATETYPE_Changed for** *object* **changed to UPDATETYPE**_*AddedOrDeleted* CAUSE: The given *iterator* was notified that one or more properties of the given *object* changed, and because the *iterator* was using a filter, the "change" notification was effectively translated into an "added" or "deleted" notification. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 2 or higher.
HLS708I	**HLS708I (***iterator***) Update:** *propertyNames* **properties of** *object* **changed** CAUSE: The given *iterator* was notified that the properties having the given *propertyNames* of the given *object* changed. This message will appear in the "Trace_HLSOB..." log file if the **debugLevel** configuration parameter of the cache is set to 3 (the maximum).

Message Code	Message and Description
HLS709E	**HLS709E (***objectKind* **cache) Unable to set configuration parameter '***paramName***' to new value '***newDataValue***'** PROBLEM: An attempt was made to change the value of a parameter with the name *paramName* of the cache that contains objects of type *objectKind*, but the cache did not recognize the parameter with that name. CAUSE: This message is usually accompanied by the **SHR256W** message. The cache did not have a configurable parameter with that name. This is commonly due to misspelling the parameter name. SOLUTION: Be sure to spell the parameter name correctly, and that the cache has a parameter with that name.
HLS710E	**HLS710E (***objectKind* **Cache) Unable to get configuration parameter '***paramName***'** PROBLEM: An attempt was made to retrieve the value of a parameter with the name *paramName* of the cache that contains objects of type *objectKind*, but the cache did not recognize the parameter with that name. CAUSE: This message is usually accompanied by the **SHR257E** message. The cache did not have a configurable parameter with that name. This is commonly due to misspelling the parameter name. SOLUTION: Be sure to spell the parameter name correctly, and that the cache has a parameter with that name.
HLS711E	**HLS711E (SetProperty of** *destRule***) TargetServer: There is no pair in** *job* **having a target server that matches** *serverID (dataValue)* PROBLEM: An attempt was made to change the value of the **TargetServer** property of the destination rule object *destRuleObject* with the new value *dataValue* (the name of the proposed new target server), but there was no replication pair in the job that ultimately owns the destination rule that had a matching target server. CAUSE: This is commonly due to changing the target server of a destination rule without first ensuring that there is a pair in the owning job with the same target server. SOLUTION: Be sure to verify that there is a pair in the owning job that has the same target server as the one being set in the destination rule.

Message Code	Message and Description
HLS712E	**Built-in function** *functionName* **usage:** *description* PROBLEM: The built-in function named *functionName* was called with the wrong number of arguments, or one or more of the arguments were of the wrong data type. The proper usage of the function is shown in the given *description*. CAUSE: The built-in function named *functionName* was called with the wrong number of arguments, or one or more of the arguments were of the wrong data type. As an aid to the user, a *description* of the function's purpose and its arguments is included in the text of the message. SOLUTION: Be sure the built-in function is called with the proper number of arguments, and that each argument is passed the proper type of data.
HLS713E	*moduleName:* **Expected** '*expectedType*' **-- instead got** '*actualType*' PROBLEM: The *moduleName*, an implementation of a built-in function, was passed an argument that had the *actualType* data type instead of the *expectedType*. CAUSE: This message usually results from calling a built-in function with an argument that has the wrong kind of data. SOLUTION: Be sure that all built-in functions are called with the proper data passed to each argument of the functions.
HLS714E	*(moduleName)* **Error in path specification** *pathSpec* PROBLEM: In *moduleName*, the given *pathSpec* was found to be invalid. CAUSE: This message usually results from calling a built-in function with an argument that was expected to be a string that contains a path specification, and the path was invalid. SOLUTION: Be sure that any path specifications passed to built-in functions are correctly specified for the platform and file system being used.

Message Code	Message and Description
HLS715E	**Built-in function** *functionName:* **Illegal data type** '*actualType*' **specified -- use** '*expectedType*' **instead** PROBLEM: In the built-in function named *functionName*, the *actualType* data type that was specified as an argument to that function was invalid. The function expected the *expectedType* data type. CAUSE: This message usually results when the built-in function **ReadFile** is called with the optional third argument set to something other than **byteArray** (aka **blob**) or **string** (or their equivalent integral ordinal values). SOLUTION: Be sure that the arguments passed to built-in functions contain the proper data types and values.

HLS716E	**(InternalCheckForValidPathVolume) Path** '*pathSpec*' **is not absolute (i.e., has no root volume or drive)**
	PROBLEM:
	In the software module *InternalCheckForValidPathVolume*, the given *pathSpec* that was specified was not rooted to a drive letter (i.e., it was a relative path instead of an absolute one).
	CAUSE:
	This message results when a relative path is used as the **path** property of a new **pathRule** being created. For example, this srTool command would produce this error:
	`add rule to job "Foo" with sourceServer = "Src", path = "folderA\\folderB"`
	SOLUTION:
	Be sure that the **path** property of new **pathRules** is always a rooted (absolute) path. For example:
	`add rule to job "Foo" with sourceServer = "Src", path = "D:\\folderA\\folderB"`

srTool Messages

Message Code	Message and Description
SRT301A	**SRT301A (Shell** *shellID***) Ready** CAUSE: srTool emits this message to the standard output stream when the command shell with the given *shellID* is still running, and its command queue is empty, and it's reading commands from the standard input stream. (Note that the text of this prompt string can be changed by setting the **PromptString** shell variable.) SOLUTION: Type in one or more valid srTool commands, separating them with semicolons, then press the Enter (or Return) key.
SRT302I	**SRT302I (Shell** *shellID***) Command shell exited with error,** *details* CAUSE: Appearing in the diagnostic output stream, the command shell with the given *shellID* terminated with a result code of something other than RXRESULT_Success. If the shell's ***verbose*** variable was set ***true***, additional information describing the error should follow the *details*.
SRT303I	**SRT303I Welcome to srTool version** *version* **Copyright (C) 1999 - 2004 by VERITAS Software, Inc. All rights reserved worldwide.** CAUSE: This is the welcome banner message that gets displayed to the diagnostic output stream only if srTool was started with the **–v** (*verbose*) option and the **–nobanner** option was not specified. srTool's version number is displayed in the message.
SRT304E	**SRT304E Client interface initialization failed,** *reason* PROBLEM: When srTool tried to initialize the client interface, it failed for the given *reason*. The *reason* may include a number of additional messages that follow this one in the diagnostic output stream. CAUSE: The cause can be discerned from any additional messages that follow this one in the diagnostic output stream. The most common cause of this error is the local machine that's hosting srTool cannot communicate with the RMS machine. SOLUTION: Be sure that the local machine that's hosting srTool has a working network connection. Also verify that the RMS machine is powered on, connected to the network, and its replication services have been started.

Message Code	Message and Description
SRT305E	**SRT305I** *module***: Failure in deinitialization –** *reason* CAUSE: Appearing in the diagnostic output stream, the deinitialization of *module* failed for the given *reason*. If the **–v** (verbose) option was specified on the command line that invoked srTool, additional messages may follow this one to further describe the *reason* for the failure.
SRT306E	**SRT306E Command line processing failed,** *reason* PROBLEM: Command line argument processing failed for the given *reason*. CAUSE: The command line arguments passed to srTool most likely did not follow srTool's syntax rules. Perhaps an invalid option was specified, or two options that are mutually exclusive were both specified. SOLUTION: Be sure that the arguments passed to srTool from the Windows command line meet the srTool command line syntax rules (see "Starting srTool" on page 23).
SRT307E	**SRT307E (Shell** *shellID***)** *command***: Invalid option '***option***' specified** PROBLEM: The *command* issued in the srTool command shell *shellID* had an invalid option, which is shown inside the single quote marks in the message. CAUSE: This is typically caused by mis-spelling or mis-specifying the keyword that follows the hyphen when specifying an option "switch" for a given command. For example, the command **`list -noTabel first rms`** will produce this error ("noTabel" should be spelled "noTable"). SOLUTION: Be sure the syntax rules for the *command* are followed, paying close attention to the spelling of the keywords that are valid as options for that *command*.

Message Code	Message and Description
SRT308I	**SRT308I SYNTAX**: **srTool [-help]** **[-h[l[sob]] | -n[o[interface]]]** **[-u[ser[id | name]] {userName}]** **[-p[ass[word]] {clearTextPassword}]** **[-d[om[ain]] {*domainName*}]** **[-v[erbose]]** **[-nofirst]** **[-nobanner]** **[-stay]** **[{-command | -cmd} {srToolCommands}...]** CAUSE: This message commonly appears after the **SRT307E** or **SRT310E** message. It shows, in summary form, the syntax of the srTool command line.
SRT309E	**SRT309E (Shell** *shellID***)** *command***: Option '***keyword***' already specified** PROBLEM: An option *keyword* was specified more than once from the Windows command line used to start srTool, or in the srTool command line for the given *command* in the command shell *shellID*. CAUSE: If *shellID* is not zero and *command* is not "(command line)": The *command* issued in the srTool command shell *shellID* had an option *keyword* that was specified more than once. If *shellID* is zero and *command* is "(command line)": The specified srTool command line option *keyword* was specified more than once in the argument list obtained from the Windows command line. SOLUTION: If *shellID* is not zero and *command* is not "(command line)": Be sure to follow the syntax rules for the given *command*, particularly with regard to the spelling of its option keywords. If *shellID* is zero and *command* is "(command line)": Be sure that the arguments passed to srTool from the Windows command line meet the srTool command line syntax rules (see "Starting srTool" on page 23).

Message Code	Message and Description
SRT310E	**SRT310E Invalid command line parameter '***token***'** PROBLEM: One of the arguments used to start srTool from the Windows command line was invalid. CAUSE: This message is typically caused by failing to precede an optional parameter with a hyphen. SOLUTION: Be sure that the arguments passed to srTool from the Windows command line meet the srTool command line syntax rules (see "Starting srTool" on page 23). In particular, be sure that the keyword used for any option switches is preceded with a hyphen.
SRT311E	**SRT311E Missing or invalid '***keyword***' parameter -- '***token***'** PROBLEM: A command line option that required parameter data did not actually have any parameter data specified after the option *keyword*, or the data was of the wrong type. CAUSE: This message is caused by failing to follow an option parameter keyword with valid data when it was expected. For example, srTool **–d –v** would cause this error because the **–d** option requires a domain name after the '**d**' keyword. SOLUTION: Be sure that the arguments passed to srTool from the Windows command line meet the srTool command line syntax rules (see "Starting srTool" on page 23). In particular, be sure that command line options that require parameter data after them actually have valid data specified on the command line after the keyword.
SRT316E	**SRT316E (Shell) External command shell 'system' call failed, returned** ***errorCode*****, was passed '***commandLine***'** PROBLEM: srTool made a 'system' call to run an external program *commandLine* on the native host machine, and received a non-zero result code *errorCode* from that call, indicating that some kind of failure occurred. CAUSE: The cause of this message can usually be discerned from the *errorCode* and the *commandLine* being attempted to execute. SOLUTION: Be sure the command to be run is a valid command on the local machine's host operating system, that its syntax is specified correctly, and that it is semantically correct as well.

Message Code	Message and Description
SRT317I	**SRT317I (Shell** *shellID***)** *command***: No help available for '***topic***'** CAUSE: In the shell identified by *shellID*, while executing the given *command* (which should be **Help**), no help was available for the given *topic*.
SRT318W	**SRT318W (Shell** *shellID***)** *command***: Unable to get '***module***' version,** *resultCode* PROBLEM: In the shell identified by *shellID*, while executing the given *command* (which should be **Show**), srTool received a failure *resultCode* from a function that is supposed to acquire the version information of a particular software *module*. CAUSE: This indicates a problem in the software itself. SOLUTION: Please notify VERITAS technical support.
SRT319E	**SRT319E (Shell** *shellID***)** *command***: Unable to create '***objectKind***' object(s),** *resultCode* PROBLEM: In the shell identified by *shellID*, while executing the given *command* (which should be **Create, Make** or **Add**), srTool failed with *resultCode* while attempting to create one or more objects of type *objectKind*. CAUSE: This message, if the shell variable ***verbose*** is set true, will normally be followed by other messages that will help to explain the cause of the problem. There are several ways that object creation can fail, including trying to add objects to another object that cannot contain them, or omitting the initial value of a required property. For example, the command **add pair to first RMS** will fail in this manner. SOLUTION: Be sure that the shell variable **verbose** is set true, and take note of the message(s) that follow this one. Also be sure to follow the "srTool Object Hierarchy" on page 64, which determines which objects can be added to other objects.

Message Code	Message and Description
SRT320E	**SRT320E (Shell** *shellID***)** *command***: Unable to lock '***object***' for editing –** *reason* PROBLEM: In the shell identified by *shellID*, while executing the given *command*, srTool was unable to lock the *object* for editing for the *reason* given. CAUSE: This situation is typically caused by another console or srTool instance that is already editing the *object*. Note that such console or srTool instances could potentially be running on a machine anywhere on the network that encompasses the replication neighborhood. SOLUTION: Determine which VRE console or srTool instance is editing the object of interest, and close the properties dialog (for srConsole), or use the **quit** command (for srTool).
SRT321E	**SRT321E (Shell** *shellID***)** *command***: Unable to save and unlock '***object***' –** *reason* PROBLEM: In the shell identified by *shellID*, while executing the given *command*, srTool was unable to save and unlock the *object* for the *reason* given. CAUSE: This message is not usually seen during normal use of srTool. To determine the cause of this problem, be sure the shell variable ***verbose*** is set ***true***, which will cause srTool to emit one or more messages immediately after this one. These follow-up messages should help in diagnosing the true cause of the problem. SOLUTION: Based on the messages that follow this one (assuming the shell variable **verbose** is set **true**), correct the problem and try the *command* again.
SRT324I	**SRT324I (Shell** *shellID***) About to execute:** *commandLine* CAUSE: The shell identified by *shellID* is about to execute the *commandLine* (displayed as an ordered sequence of tokens separated by the vertical bar character). This message will appear in the shell's diagnostic output stream if the shell variable ***echoCommands*** is set ***true***. This message will appear in the srTool log files if the shell variable ***debugLevel*** is set to 2 or higher.
SRT325I	**SRT325I (Shell** *shellID*) *command: methodName: resultCode – data* CAUSE: The shell identified by *shellID*, while executing the *command*, is about to return from the given *method* with the given *resultCode*. The data being returned by the *method* is displayed at the end of this message. This message only appears in the srTool log files if the shell variable ***debugLevel*** is set to 3 (the maximum level).

Message Code	Message and Description
SRT326E	**SRT326E Syntax error:** ***command*** PROBLEM: srTool was not able to execute the *command* due to a syntax error detected somewhere in the command line. The *command* is displayed on the line immediately beneath this message. The offending token where the syntax error was detected is highlighted with a sequence of caret characters (^^^^) on the line immediately below the *command* line. CAUSE: The syntax of the command is incorrect. SOLUTION: Taking note of the offending token where the syntax error was detected, correct the *command* such that it follows the syntax rules documented elsewhere in this manual.
SRT328E	**SRT328E (Shell** *shellID***)** *command***: Error while evaluating expression** PROBLEM: In the shell identified by *shellID*, while executing the given *command*, an expression failed to evaluate. CAUSE: There are many possible reasons for expression evaluation to fail. To determine the cause of this problem, be sure the shell variable **verbose** is set **true**. Additional messages will follow this one, and should shed light on the actual cause of the failure. SOLUTION: Based on the cause that was determined above, correct the problem and try the command again.
SRT329E	**SRT329E (Shell** *shellID***)** *command***: Error while converting count value (***dataType***) to a numeric value, *reason*** PROBLEM: In the shell identified by *shellID*, while executing the given *command*, the *dataType* result of an expression would not convert to an unsigned integer value that is to be used as an object count. CAUSE: This is typically caused by an expression that results in a data type that cannot convert to an unsigned integer value being used in a grouping spec's object count, or as the object count in the **add** command. SOLUTION: Be sure that anywhere an object count is expected that the expression used to compute that count results in a data type that can be converted to an unsigned integer. See "Converting Between Different Data Types" on page 33.

Message Code	Message and Description
SRT330E	**SRT330E (Shell** *shellID***)** *command***: Object count value must be at least 1** PROBLEM: In the shell identified by *shellID*, while executing the given *command*, the unsigned integer that resulted from the evaluation of an expression was zero. An object count of zero is not permitted. CAUSE: An expression used in an object count of a grouping spec or the **add** command resulted in zero. SOLUTION: Be sure the object count expression results in a value of one or more, then try the command again.
SRT334E	**SRT334E (Shell** *shellID***)** *command: 'syntaxElement*' **specified more than once** PROBLEM: In the shell identified by *shellID*, while executing the given *command*, srTool discovered that the given *syntaxElement* was specified more than once, creating an ambiguity where one is not allowed. CAUSE: Usually this error occurs while parsing the **add** command or property assignment lists. For example, the command **`add job with type = oneToOne, type = OneToMany`** will produce this error. SOLUTION: Carefully check the *command*, and remove the duplicitous syntax element(s), then retry the *command* again.
SRT335E	**SRT335E (Shell** *shellID***)** *command***: Unexpected token ('***token***') found past end of command** PROBLEM: In the shell identified by *shellID*, after successfully parsing the entire *command*, srTool found the given *token* past the logical end of the *command*. CAUSE: This is usually due to the user inadvertently entering some characters at the end of a valid command line. For example, the command **`list every rms extra`** will elicit this message. SOLUTION: Check the syntax of the ***command***, remove the extra (and erroneous) tokens from the end of the command line, then retry the *command* again.

Message Code	Message and Description
SRT343E	**SRT343E (Shell** *shellID***)** *command: 'token*' **is not a valid property name** PROBLEM: In the shell identified by *shellID*, while executing the given *command*, srTool expected to find a valid property name, but instead found the given *token*. CAUSE: This error is most commonly caused by misspelling a property name. For example, **`add job with foo = 3`** will produce this error. SOLUTION: Correct the spelling of the property to be specified, then retry the *command* again.
SRT345E	**SRT345E (Shell** *shellID***)** *command***: Expected '***syntaxElement***' -- instead got '***token***'** PROBLEM: In the shell identified by *shellID*, while parsing the given *command*, srTool expected to find the given *syntaxElement*, but instead found the given *token* (or nothing if the end of the command has passed). CAUSE: While parsing the command, srTool expected something on the command line, but didn't get it. This is usually due to the user entering a command with an incorrect syntax. SOLUTION: Check the syntax of the *command*, correct any errors, then retry the *command* again.
SRT348E	**SRT348E (Shell** *shellID***)** *command***: Error in path specification** *data* PROBLEM: In the shell identified by *shellID*, while executing or parsing the given *command*, srTool encountered an invalid path specification. This message may be followed by additional information in *data*. CAUSE: This is usually due to the user entering a path whose syntax is incorrect. SOLUTION: Check the syntax of the path specification, correct any errors, then retry the *command* again.

Message Code	Message and Description
SRT350E	**SRT350E (Shell** *shellID***)** *command***: Only one** *syntaxElement* **may be specified -- instead there are** *count* **of them** PROBLEM: In the shell identified by *shellID*, while executing or parsing the given *command*, srTool encountered *count syntaxElements*, when there should have only been one specified. CAUSE: This is usually due to specifying more than one property in the **set** command, like this: **`set name, description of first job = "foo"`** SOLUTION: Be sure to specify only one *syntax element* where it is called for, then retry the *command* again.
SRT352E	**SRT352E (Shell** *shellID***)** *command***: Error while adding or changing symbol value -** *resultCode* PROBLEM: The shell identified by *shellID*, while executing or parsing the given *command* (which should be the **set** command), was not able to add or change the value of a variable. CAUSE: This is usually caused by an attempt to change the value of a read-only variable, whether it be a global variable or it belongs to one of the shell's execution contexts. For example, the command **`whenShellStarted = now()`** can produce this error. In very rare cases, it can also be caused by a lack of available system memory. SOLUTION: Be sure that the variable being **set** is not marked as read-only.
SRT354E	**SRT354E (Shell** *shellID***)** *command***: Unable to obtain default object specification -** *resultCode* PROBLEM: The shell identified by *shellID*, while executing or parsing the given *command*, was not able to obtain the shell's default object specification because of *resultCode*. CAUSE: This problem should never occur during normal operation of srTool. SOLUTION: If the problem persists, be sure that the ***verbose*** shell variable is set ***true***, then please contact Technical Support.

Message Code	Message and Description
SRT355E	**SRT355E (Shell** *shellID***)** *command***: Unable to set default object specification --** *reason* PROBLEM: The shell identified by *shellID*, while executing or parsing the given *command*, was not able to change the shell's default object specification due to the given *reason*. CAUSE: This problem should never occur during normal operation of srTool. SOLUTION: If the problem persists, be sure that the ***verbose*** shell variable is set ***true***, capture all messages that precede and follow this one, then please contact Technical Support.
SRT356E	**SRT356E (Shell** *shellID***)** *command***: Unable to add or set** *variableName* **--** *reason* PROBLEM: The shell identified by *shellID*, while executing or parsing the given command, was not able to add or change the variable identified by *variableName* because of the given reason. If *variableName* is shown as "parameter values", the shell was not able to set one or more parameter value variables. CAUSE: This message is usually followed by one or more additional messages that should point to the actual cause of the problem. SOLUTION: Be sure that the ***verbose*** shell variable is set ***true***, then read the additional message(s) that follow(s) this one, then correct the indicated problem(s).
SRT356W	**SRT356W (Shell** *shellID***)** *methodOrFunction***: Unable to add or set inherited symbols --** *reason* PROBLEM: The shell identified by *shellID*, while executing given *methodOrFunction*, was not able to add or change one or more variables inherited from its parent shell because of the given *reason*. CAUSE: This indicates a condition similar to that reported by message **SRT356E**, except this one is not fatal, thus is only a warning. This message is usually followed by one or more additional messages that should point to the actual cause of the problem. SOLUTION: Be sure that the ***verbose*** shell variable is set ***true***, then read the additional message(s) that follow(s) this one, then correct the indicated problem(s).
SRT357I	**SRT357I (Shell** *shellID***) Spawn: Shell task [***taskNumber***] completed** CAUSE: This message indicates that the spawned shell with the given *shellID* that was running under the given *taskNumber* completed normally.

Message Code	Message and Description
SRT358E	**SRT358E (Shell** *shellID***) Shift: Shift count value must be at least 1** PROBLEM: While the shell *shellID* was parsing the **shift** command, it came up with a shift count of zero, which is not allowed. CAUSE: The expression that was specified in the **shift** command resulted in a value of zero. SOLUTION: Be sure that the expression that is used in the **shift** command results in a value that is greater than or equal to one (1).
SRT359E	**SRT359E (Shell** *shellID***)** *command***: Command failed,** *reason* PROBLEM: The shell *shellID* encountered a failure while parsing or executing the *command*, which was due to *reason*. When this message is present, it means that all other enqueued commands will be flushed and not executed if the shell variable ***continueOnError*** is set to ***false***. CAUSE: This message is usually caused by a semantic or other execution error (for example, the syntax rules were correctly followed, but the command's meaning was incorrect). If the shell variable ***verbose*** is set to **true**, this message is followed by one or more additional messages that should indicate the actual cause of the problem. SOLUTION: Be sure that the ***verbose*** shell variable is set ***true***, then read the additional message(s) that follow(s) this one, then correct the indicated problem(s).
SRT362E	**SRT362E (Shell** *shellID***)** *command***: No such function '***functionName***'** PROBLEM: While executing the *command*, the shell *shellID* encountered a failure while trying to obtain the commands that were stored for a user-defined function that was defined with the name *functionName*. CAUSE: This problem is usually caused by mis-typing a function name in an expression (RXRESULT_NotFound). On extremely rare occasions, this same message can result from srTool being operated under exceedingly low system memory conditions (RXRESULT_Memory). SOLUTION: Be sure that any function(s) used in any expression(s) in the *command* truly exist under the reported *functionName*, whether they be built-in or user-defined.

Message Code	Message and Description
SRT363E	**SRT363E (Shell** *shellID***)** *command***: Missing argument '***argumentName***' for function '***functionName***'** PROBLEM: While executing the *command*, the shell *shellID* discovered that the argument named *argumentName* was missing in the call to the user-defined function named *functionName*. CAUSE: This problem is caused by omitting an argument that must be passed to the named user-defined function. SOLUTION: Be sure that when using any user-defined function(s) in any expression(s) in the *command*, that all required function arguments are supplied in the expression(s).
SRT364W	**SRT364W (Shell** *shellID***)** *command***: Superfluous argument passed to function '***functionName***':** *expression* PROBLEM: While executing the *command*, the shell *shellID* discovered that the function argument using *expression* was not needed in the call to the user-defined function named *functionName*. CAUSE: This problem is caused by supplying an argument that was not expected by the user-defined function. SOLUTION: Be sure that when using any user-defined function(s) in any expression(s) in the *command*, that only the required function arguments are supplied – no more and no less.
SRT366E	**SRT366E (Shell** *shellID***)** *command***: No corresponding '***contextKind***' kind of execution context** PROBLEM: While executing the *command*, the shell *shellID* encountered an execution context control command that would have been legal for a context of type *contextKind*. CAUSE: This problem is typically caused by using a **break** or **continue** statement when there is no active **loop** or **repeat** context; or by using an **else** or **elseIf** statement when there is no active **if** context. SOLUTION: Be sure that **break** or **continue** statements are used inside **loop** or **repeat** contexts, and that **else** or **elseIf** statements are used inside **if** contexts.

Message Code	Message and Description
SRT367I	**SRT367I (Shell** *shellID***)** *command***: Command execution aborted** CAUSE: This message indicates that the *command* that was executing in the shell *shellID* was aborted, typically by a "Control-C" interrupt. When this message is displayed, it also means that all execution contexts other than the default one are discarded, as are all queued commands.
SRT370E	**SRT370E (Shell** *shellID***) end: Command not preceded by 'loop', 'if', 'function' or 'begin' command** PROBLEM: The **end** command was executing in the shell *shellID* without being preceded by a **loop, if, function** or **begin** command. CAUSE: This problem is typically caused by using an **end** statement when there is no active execution context. SOLUTION: Be sure that **end** statements always terminate execution contexts.
SRT372E	**SRT372E (Shell** *shellID***)** *command***: Unable to convert '***originalDataType***' value to '***requiredDataType***'** PROBLEM: While executing the *command*, the shell *shellID* evaluated an expression to obtain a data value that should have been of type *requiredDataType* but instead ended up with a data value of type *originalDataType*, and wasn't able to convert it to the *requiredDataType*. CAUSE: This problem usually results from the expressions that produce data values used to determine the 1) wait time or the "until" condition in the **wait** command, or 2) index values in **indexing specifications**. The expressions result in data values that can't be converted to the data types required in those two situations. SOLUTION: For the **wait** command, be sure the "until" expression can be converted to an **uint64** or **uint32** value, and that the "wait time" expression is convertible to a **timespan**. For indexing specifications, be sure the expression results will convert to **uint64** values. See "Converting Between Different Data Types" on page 33.

Message Code	Message and Description
SRT373E	**SRT373E (Shell** *shellID***)** *command***: Unable to obtain '***propertyName***' property from the current object** PROBLEM: While executing the *command*, the shell *shellID* was asked for the value of the property of an object under consideration during the evaluation of an expression, and the object did not have the property that was requested. CAUSE: This problem typically occurs when a **select** command is being used and a property reference in one of the expressions refers to a property that does not exist in the object(s) that result(s) from the specified or implied compound object specification. SOLUTION: Be sure that the property specification in the expression refers to a property that exists for the kind of object that will result from the compound object specification. See "Property Specifications" on page 55, and "`select` command" on page 117.
SRT375E	**SRT375E (Shell** *shellID***)** *command***: Unable to obtain '***propertyName***' property from '***objectSpec***' -- there are** *count* **objects from which to get it** PROBLEM: While evaluating an expression during the execution of *command*, the shell *shellID* was asked for the value of the property of the object that results from the query *objectSpec*, and instead of one, there were *count* objects from which to obtain the property value. CAUSE: This problem can occur anytime an expression is used that contains a property specification that incorporates an object specification that results in no objects or more than one object. For example, the command **`echo -x name of job ""`** will produce this error. If two or more jobs exist, the command **`echo -x name of all jobs`** will also produce this error. If *count* exceeds one, this message will be accompanied by the **SRT465I** message. SOLUTION: Be sure that any property specifications that are used in expressions always follow the rules as documented in "Property Specifications" on page 55.
SRT377I	**SRT377I (Shell** *shellID***)** *command***: Command execution stopped due to error** CAUSE: An error occurred while the shell *shellID* was executing a prior command, and the shell variable ***continueOnError*** was set ***false***. Because of this, the shell has stopped execution of all other pending commands.

Message Code	Message and Description
SRT378E	**SRT378E (Shell** *shellID***) Macro or embedded command nesting level exceeds** *level* PROBLEM: The maximum nesting *level* was exceeded while shell *shellID* was replacing macros or embedded commands. CAUSE: This problem can happen if macro or embedded command nesting exceeds 32 levels of depth. SOLUTION: Be sure that macro or embedded command nesting does not exceed 32 levels of depth.
SRT378S	**SRT378S (Shell** *shellID***) Nesting level exceeds** *level* **-- unable to continue** PROBLEM: A maximum nesting *level* was exceeded in shell *shellID*. CAUSE: This problem can happen if an srTool script calls itself (recursively) too many times, or if too many srTool shells have called down too many levels, or if an expression is too complex. SOLUTION: Be sure that any expressions that are specified in any commands do not contain any sub-expressions that exceed a depth of 64, and that command shells are not nested (via the **call** command) to a depth that exceeds 32 levels.
SRT379I	**SRT379I (Shell** *shellID***) RunShell starting at level** *depth* CAUSE: The shell *shellID* has started reading, enqueing and executing commands at the nesting level *depth*. This message will appear in the "Trace_HLSOB..." log file in the **debugLevel** shell variable is set to 2 or higher.
SRT380I	**SRT380I (Shell** *shellID***) RunShell exiting from level** *depth* CAUSE: The shell *shellID* has finished executing commands at the nesting level *depth*. This message will appear in the "Trace_HLSOB..." log file in the **debugLevel** shell variable is set to 2 or higher.

Message Code	Message and Description
SRT381S	**SRT381S (Shell** *shellID***)** *command***: Unable to record command -- fatal, cannot continue** PROBLEM: The shell *shellID* encountered a failure while trying to record the *command*. CAUSE: This problem can only happen under extremely tight system memory conditions and should never happen during normal use of srTool. SOLUTION: Be sure that there is sufficient system memory available for use by srTool.
SRT382S	**SRT382S (Shell** *shellID***) Input text line exceeds** *length* **character buffer capacity – cannot continue** PROBLEM: The input stream of shell *shellID* returned NULL from its **getline** method for a condition other than end-of-file, which indicates that the line of text that was just read exceeded the capacity of its line buffer (which is *length* characters). CAUSE: This can happen if an srTool script file is executed, called or spawned (via the **exec**, **call** or **spawn** command, respectively) and the file doesn't contain standard ASCII or Unicode text data, or if it actually contains a run of more than 32,767 characters without an intervening line break (which, on Windows is the carriage-return, line-feed character pair). SOLUTION: Be sure that srTool only executes script files containing standard ASCII or Unicode text data, and that the line lengths in the files are less than 32K characters.
SRT384E	**SRT384E (Shell** *shellID***) wait: Failure in wait command --** *reason* PROBLEM: The **wait** command running in the shell *shellID* failed for the given *reason*. CAUSE: One or more messages should follow this one that describe what caused the failure. Typically this is caused by the use of an "until" expression that cannot be evaluated or whose resulting data cannot be converted into an unsigned integer value. SOLUTION: Be sure that the expression used in the "until" clause of the **wait** command will result in a data value that can convert into an unsigned integer value.

Message Code	Message and Description
SRT385E	**SRT385E (Shell** *shellID***)** *command***: Error in 'sortedBy' clause -- missing 'ascending' or 'descending' keyword, or property name** PROBLEM: While the shell *shellID* was parsing the 'sortedBy' clause used in an object specification in *command*, a property name was missing, or the keywords **ascending** or **descending** were expected but missing. CAUSE: This problem is caused by syntax errors in the 'sortedBy' clause of an object specification. SOLUTION: Be sure to follow the syntax rules of 'sortedBy' clauses of the object specification(s) being used in *command*. See "SortedBy Clause" on page 71.
SRT387E	**SRT387E (Shell** *shellID***)** *command***: Unable to obtain module list --** *reason* PROBLEM: The shell *shellID* was executing *command* (which should only be the **configure** command) and was not able to obtain a list of configurable software modules for the given *reason*. CAUSE: This problem should not happen during normal operation of srTool. This message may be followed by one or more other messages that may give a clue as to the exact origin of the failure. SOLUTION: If this problem recurs, note any other messages that follow this one, then please contact Technical Support.
SRT388I	**SRT388I (Shell** *shellID***) configure: Registered modules:** CAUSE: A list of registered software modules is about to follow this message in the shell *shellID's* standard output stream. This message is a consequence of having the ***verbose*** shell variable set to ***true*** and using the **configure** command without any other arguments or parameters.
SRT389W	**SRT389W (Shell** *shellID***)** *command***: There are no registered modules, or none were specified** CAUSE: The shell *shellID* was executing *command* (which should be the **configure** command) and noticed that there were no registered software modules, and none were specified as arguments to the *command*.

Message Code	Message and Description
SRT390I	**SRT390I (Shell** *shellID***) delete:** *functionOrGlobal name* **deleted** CAUSE: The shell *shellID* successfully deleted the user-defined function or global variable named *name*. The *functionOrGlobal* part of the message will indicate either "function" or "global variable." This message will be seen only if the ***verbose*** variable of shell *shellID* is set to ***true***.
SRT393E	**SRT393E (Shell** *shellID***) configure: Unable to obtain information about module** *'moduleName'* **--** *reason* PROBLEM: The shell *shellID* was executing the **configure** command and was not able to obtain information about the software module named *moduleName* because of *reason*. CAUSE: If the *reason* is RXRESULT_Invalid, one of the module names specified in the *command* wasn't the name of an object kind. If the *reason* is RXRESULT_NotFound, one of the module names specified in the command was invalid, probably misspelled. SOLUTION: Be sure to use only module names from this list: **FileReplicationJob, ReplicationPair, Script, PathRule, SelectionRule, DestinationRule, Server** and **EventRedirector.**
SRT395E	**SRT395E (Shell** *shellID***) use: Unable to set the new default object specification** PROBLEM: The shell *shellID* was executing the **use** command and was not able to change its default object specification. CAUSE: This error should not occur during normal use of srTool. SOLUTION: Please contact Technical Support.
SRT396W	**SRT396W (Shell** *shellID***) quit: Shell exiting with count unterminated execution contexts:** *contextList* PROBLEM: The shell *shellID* was about to exit and discovered that it still had *count* active execution contexts. The *contextList* shows the ones that were still active. CAUSE: This error can occur if the quit command is issued inside of a **begin, if/else/elseIf** or **loop** context. SOLUTION: Be sure to end all active contexts (using the **end** command) prior to issuing the **quit** command. To determine the contexts that are active, use the **show contexts** command.

Message Code	Message and Description
SRT398E	**SRT398E (Shell** *shellID***)** *command***:** *'identifierName'* **is undefined** PROBLEM: The function or variable whose name is identifierName was undefined in shell *shellID*. The *command* was currently executing when this was discovered. CAUSE: This error most commonly occurs in commands containing one or more expressions, at least one of which was using an undefined variable, or that had a function call, but a function with that name was never defined. SOLUTION: Be sure that expressions use only variables or functions that have already been defined.
SRT399E	**SRT399E (Shell** *shellID***)** *command***: 'tty' or 'con' file specification not allowed with this command** PROBLEM: The *command* being executed in shell *shellID* did not allow the use of a 'tty' or 'con' file specification. CAUSE: This error is reported whenever the 'tty' or 'con' file specification is used with the **spawn** or **exec** command. Such a file specification is only valid with the **call** command. SOLUTION: Be sure that the 'tty' or 'con' file specification is only used with the **call** command.
SRT400I	**SRT400I (Shell** *shellID***)** *command***: Context *contextName* started, expression=***condition***** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that a new execution context with the name *contextName* was started in the shell *shellID* by the *command*. The new context had the given *condition* expression attached to it.
SRT401I	**SRT401I (Shell** *shellID***)** *command***: Context** *contextName* **changed, command=***tokens***, expression=***condition* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the existing execution context with the name *contextName* was changed in the shell *shellID* by the *command*. The command causing the context change had the given *tokens* and *condition* expression associated with it.

Message Code	Message and Description
SRT402I	**SRT402I (Shell** *shellID***)** *command***: Context** *contextName* **ended** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the existing execution context with the name *contextName* was ended in the shell *shellID* by the *command*.
SRT404I	**SRT404I (Shell** *shellID***)** *command***: Context** *contextName* **-- SetExecute is now** *trueOrFalse* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the existing execution context with the name *contextName* had its "SetExecute" state set to *trueOrFalse* in the shell *shellID* by the *command*.
SRT405I	**SRT405I (Shell** *shellID***) Prepended command** *tokens*, *count* **queued** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the shell *shellID* prepended the listed *tokens* to its command queue, bringing the number of commands on the queue up to *count*.
SRT406I	**SRT406I (Shell** *shellID***) Appended command** *tokens*, *count* **queued** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the shell *shellID* appended the listed *tokens* to its command queue, bringing the number of commands on the queue up to *count*.
SRT407I	**SRT407I (Shell** *shellID***) Removed command** *tokens*, *count* **remain** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the shell *shellID* removed the listed *tokens* from the end of its command queue, bringing the number of remaining commands on the queue down to *count*.
SRT408I	**SRT408I (Shell** *shellID***) Command queue cleared,** *count* **command(s) were discarded** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the shell *shellID's* command queue was cleared, discarding *count* commands in the process.

Message Code	Message and Description
SRT409I	**SRT409I (Shell** *shellID***)** *command***: Context** *contextName* **not yet ended, commands=***recordedCommands***, vars=***variables* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the state of the existing loop context named *contextName* was changed in the shell *shellID* by the (**end**) *command*. The context's recorded commands are listed, as are its currently defined variables.
SRT410I	**SRT410I (Shell** *shellID***)** *command***: Context** *contextName* **recorded command** *tokens* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the existing context named *contextName* in the shell *shellID* recorded the *command's tokens*.
SRT411I	**SRT411I (Shell** *shellID***)** *command***: Context** *contextName* **command recording now** *trueOrFalse* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that the command-recording state of the existing context named *contextName* in the shell *shellID* is now set to *trueOrFalse*.
SRT412E	**SRT412E (Shell** *shellID***) loop: Illegal loop variable** *'name' – reason* PROBLEM: The variable named *name* being defined in the **loop** command being executed in shell *shellID* is illegal because of the given *reason*. CAUSE: This error is reported whenever a loop variable being defined in a "**loop for**" command matches the name of a property or is already defined to be global in scope or is read-only. SOLUTION: Be sure to not use loop variables that have the same name as any object properties or existing global variables or existing variables that are read-only.

Message Code	Message and Description
SRT413W	**SRT413W (Shell** *shellID***) delete: Unable to delete** *functionOrGlobal 'name'* **--** *reason* PROBLEM: The user-defined function or global variable named *name* could not be deleted by the shell *shellID* because of the given *reason*. CAUSE: If the shell *shellID's* ***verbose*** variable is set to *true*, there may be other messages that follow this one that can indicate the underlying reason for the deletion failure. The *reason* **RXRESULT_NotFound** indicates the specified function or global does not exist and could have been misspelled. The *reason* **RXRESULT_IllegalOperation** typically means the specified function name matched the name of a built-in function. SOLUTION: Be sure that the names of the functions or variables to be deleted are for existing user-defined functions or globals, respectively.
SRT414I	**SRT414I (Shell** *shellID***)** *command***: Calling function** *name***, args=***argumentList***, tokens=***cmdTokens* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the user-defined function with the given *name* is about to be called with the given arguments shown in the *argumentList*. The command tokens stored for the function are shown in the *cmdTokens*.
SRT 418E	**SRT418E (Shell** *shellID***) spawn: Unable to spawn task --** *reason* PROBLEM: The shell *shellID* was not able to create a new task for the given *reason*. CAUSE: Generally, this problem should never occur during normal operation of srTool. If it does, it most likely would be due to a resource issue (for example, lack of available system memory). If the shell *shellID's* ***verbose*** variable is set to *true*, there may be other messages that follow this one that can indicate the underlying reason for the failure. SOLUTION: Be sure to set the shell variable **verbose** to **true** and try the command again. If the problem recurs, note any additional messages that follow this one and please contact Technical Support.

Message Code	Message and Description
SRT419E	**SRT419E (Shell** *shellID***)** *command***: Missing option -- expected** *suggestion* PROBLEM: While parsing *command*, the shell *shellID* discovered an option keyword was missing, when it was expecting those in the *suggestion*. CAUSE: This problem is commonly caused by omitting the option keyword after the hyphen that normally precedes it. For example, **`list -`** will produce this error. Note that if *shellID* is zero (and *command* is "srTool"), the option keyword was missing in the command line used to start srTool from the Windows NT command prompt. SOLUTION: Be sure to follow the syntax rules for the *command*, specifying a valid option keyword immediately after the hyphen.
SRT420E	**SRT420E (Shell** *shellID***)** *command***: Option '***optionSwitch***' illegal for** *situation* PROBLEM: While parsing *command*, the shell *shellID* discovered that the option *optionSwitch* was illegal in the given *situation*. CAUSE: This problem can be caused by one of two things: using the **–target** option for a job control command (**start, stop, pause, resume** or **cancel**) other than **cancel**, which is not allowed; or using any of the **–src, -test** or **–x** options with the **–from** option in the **xml** command, which also is not allowed. SOLUTION: Be sure to follow the syntax and semantic rules that are documented for the *command*, recognizing that some *command* options cannot be used in combination with certain others.
SRT422E	**SRT422E (Shell** *shellID***) xml: Unable to process '***tagName***' tag** PROBLEM: In shell *shellID* was executing the **xml** command, trying to "reanimate" replication system objects from XML code, and didn't know how to create an object from the XML code that had the tag named ***tagName***. CAUSE: This version of srTool only knows how to reanimate objects of type *ObjectData* from XML. SOLUTION: There is no solution for this problem.

Message Code	Message and Description
SRT423E	**SRT423E Built-in function '***functionName***' failed: '***reason***'** PROBLEM: The built-in function named *functionName* failed for the given *reason*. CAUSE: The *reason* given in the message should state why the failure occurred (for example, disk full). SOLUTION: Based on the cause of the failure, correct the problem as indicated then try again.
SRT424I	**SRT424I (Shell** *shellID***) use: Default object specification changed from '***oldObjectSpec***' to '***newObjectSpec***'** CAUSE: This message means that the default object specification of shell *shellID* was changed from the *oldObjectSpec* to the *newObjectSpec*. This message only appears if the shell's ***verbose*** variable is set to ***true***.
SRT425E	**SRT425E (Shell** *shellID***)** *command***: Command option '***keyword***' valid only for '***commandName***' command** PROBLEM: While parsing *command*, the shell *shellID* encountered an option *keyword* that is legal only for the command(s) *commandName*, which isn't the current situation. CAUSE: This problem is caused by the mis-use of certain options in one of the shell invocation commands (**exec, call** or **spawn**). For example, using the **`-list`** option with **call** or **exec** is not allowed. SOLUTION: Be sure to follow the syntax and semantic rules that are documented for the *command*, recognizing that some *command* options can only be used with certain commands.

Message Code	Message and Description
SRT426E	**SRT426E (Shell** *shellID***)** *command***: Conflicting command options:** *options* PROBLEM: While parsing *command*, the shell *shellID* detected one or more *options* that were in conflict with each other. CAUSE: This problem is caused by specifying one or more mutually exclusive options, which is commonly found in the shell invocation commands (**exec, call** or **spawn**) or, if the *shellID* is zero (and *command* is "srTool"), this same problem was detected in the srTool command line itself, as issued from the Windows NT command prompt. For example, using both the **`-list`** and **`-c`** options together in the **spawn** command will produce this error. SOLUTION: Be sure to follow the syntax and semantic rules that are documented for the *command*, recognizing that some *command* options can only be used with certain commands.
SRT427I	**SRT427I (Shell** *shellID***)** *command***: About to resolve** *objectSpec* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the simple or compound object specification *objectSpec* is about to be resolved into objects or object iterators that can be used to iterate over the resulting objects.
SRT428I	**SRT428I (Shell** *shellID***)** *command***: Resolved** *objectSpec* **into** *count* **object(s) --** *resultCode* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that the simple or compound object specification *objectSpec* was resolved into *count* objects while executing *command*. If the resolution was successful, the *resultCode* will be **RXRESULT_Success**; otherwise, it will be another code designating the type of failure that occurred.
SRT429I	**SRT429I (Shell** *shellID***)** *command***: GetQualifiedObjects final results --** *count* ***object(s)*** *listOfObjects* CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 3 (the maximum), this message means that a simple object specification was successfully resolved into *count* objects while executing *command*. The objects are subsequently listed in the log immediately after this message.

Message Code	Message and Description
SRT431W	**SRT431W (Shell** *shellID***)** *command***: Query resulted in no objects** CAUSE: Appearing only in the HLSOB or srConsole logs when the shell *shellID's* **debugLevel** shell variable is set to 2 or higher, this message means that a request was made to resolve an empty compound object specification into iterators or objects. A message of this type should be considered an unusual event in the log.
SRT433E	**SRT433E (Shell** *shellID***) delete: Unable to delete** *object* **-- failure in** *methodName*, *reason* PROBLEM: The shell *shellID* was unable to delete the given *object* due to the given *reason*. The failure was detected in the method *methodName*, which should be one of the following: "Open", "DeleteAll" or "Delete". CAUSE: There are many potential reasons that an object (or set of objects) cannot be deleted. To help determine the cause, this message is always followed by one or more additional messages that should pinpoint the reason for the failure. These other messages can always be seen if the shell variable ***verbose*** is set to ***true***. SOLUTION: Using the other messages that accompany this one, note the reasons for the failure. If the reason is something that can be corrected (for example, a job being edited in a console), correct the problem, then try the **delete** command again.
SRT434I	**SRT434I (Shell** *shellID***) delete:** *object* **deleted** CAUSE: This message means that the specified *object* was successfully deleted by the **delete** command executing in shell *shellID*. This message only appears in the standard output stream if the shell's ***verbose*** variable is set to ***true***.
SRT435I	**SRT435I (Shell** *shellID***) delete:** *count* **object(s) deleted** CAUSE: This message means that *count* object(s) was (were) successfully deleted by the **delete** command executing in shell *shellID*. This message only appears in the standard output stream if the shell's ***verbose*** variable is set to ***false***.

Message Code	Message and Description
SRT437E	**SRT437E (Shell** *shellID***) set: Unable to set property in object** *objectNameOrID* **-- object is not editable** PROBLEM: The shell *shellID* was unable to change the value of a property in the given *objectNameOrID* because the object is not editable. CAUSE: Many objects in the srTool object hierarchy are intrinsically not editable, and thus cannot have any of their property values changed (for example, **file** objects). SOLUTION: Be sure that only editable objects are specified in the object specification used in the **set** command.
SRT438E	**SRT438E (Shell** *shellID***) set: Unable to set property** *propertyName* **in object** *objectNameOrID – reason* PROBLEM: The shell *shellID* was unable to change the value of the property named *propertyName* in the given *objectNameOrID*. CAUSE: There are many possible *reasons* why a certain property of a particular object cannot be changed to a certain value. Some properties are Constant or ReadOnly. Others can only be changed to a limited set of data values. Still others can only be changed to a particular value if some other condition is true. This message is commonly followed by one or more other messages that describe the particular *reasons* for the failure. SOLUTION: Note the reasons for the failure from the messages that follow this one, then correct those conditions that can be corrected, and then try the **set** command again.
SRT439I	**SRT439I (Shell** *shellID***) set:** *count* **object(s) set** CAUSE: This message means that *count* object(s) had one or more properties that were successfully changed by the **set** command executing in shell *shellID*. This message only appears in the standard output stream if the shell's ***verbose*** variable is set to ***false***.
SRT440I	**SRT440I (Shell** *shellID***) set:** *propertyNames* **property(s) of object** *objectNameOrID* **set** CAUSE: This message means that the specified object had the listed *propertyNames* successfully changed by the **set** command executing in the shell *shellID*. This message only appears in the standard output stream if the shell's ***verbose*** variable is set to ***true***.

Message Code	Message and Description
SRT441E	**SRT441E Built-in function** *functionName* **usage:** *description* CAUSE: This message means that the built-in function named *functionName* was called with the wrong number (or data types) of arguments. As an aid to the user, a *description* of the function's purpose and its arguments is included in the text of the message.
SRT442E	**SRT442E (Shell** *shellID***)** *command***: Unable to** *operation* **object** *nameOrID – reason* PROBLEM: The shell *shellID*, while executing *command*, was unable to perform *operation* on the object with the given *nameOrID*. CAUSE: This problem is commonly seen when using job control commands (**start, stop, cancel, pause** or **resume**), **enable** or **disable**, or the **promote** or **demote** commands. This message is commonly followed by one or more other messages that describe the particular *reasons* for the failure. As an example, attempting to **promote** a **selectionRule** that is already at the top will produce this error. SOLUTION: Note the reasons for the failure from the messages that follow this one, setting the shell variable ***verbose*** to ***true***, if necessary to see the additional messages, then correct those conditions that can be corrected, and then try the *command* again.
SRT443I	**SRT443I (Shell** *shellID***)** *command: nameIDOrCount* **object(s) successfully commanded to** *operation* CAUSE: This message means that the specified object was successfully commanded to do *operation*, or the given number of objects were successfully commanded to do the same, while executing the *command* in the shell *shellID*. If the shell's ***verbose*** variable is set to ***true***, this message will appear once in the standard output stream for each object that is successfully commanded; otherwise, it will appear once to show the total number of objects that were successfully commanded.

Message Code	Message and Description
SRT444E	**SRT444E (Shell** *shellID***)** *command***: Cannot perform '***operation***' operation on '***nameOrID***' object -- expected object of type '***objectKind***'** PROBLEM: The shell *shellID*, while executing *command*, was unable to perform the given *operation* on the object identified by *nameOrID* because the object was not of type *objectKind*. CAUSE: This problem is commonly seen when using the **start, stop, cancel, pause** or **resume** commands, which expect **fileReplicationJob** objects, or the **enable** or **disable** commands, which expect **server** objects, or the **promote** or **demote** commands, which expect **selectionRule** objects. For example, the command **`promote first job`** will produce this error. SOLUTION: Check the documentation for the specific *command* and be sure that the *command* verb can be applied to the kinds of objects that will result from the object specification used in the command.
SRT445I	**SRT445I (Shell** *shellID***) Object added:** *object* CAUSE: This message means that monitoring was taking place in the shell *shellID* and the specified *object* met the filter criteria for one of the shell's active monitors, and is giving notice that the *object* was successfully added. This message appears in the shell's standard output stream.
SRT446I	**SRT446I (Shell** *shellID***) Object changed:** ***object (propertyChanges)*** CAUSE: This message means that monitoring was taking place in the shell *shellID* and the specified *object* met the filter criteria for one of the shell's active monitors, and is giving notice that one or more properties of the *object* changed. The new values of the changed properties are shown in the *propertyChanges*. This message appears in the shell's standard output stream.
SRT447I	**SRT447I (Shell** *shellID***) Object deleted:** *object* CAUSE: This message means that monitoring was taking place in the shell *shellID* and the specified *object* met the filter criteria for one of the shell's active monitors, and is giving notice that the *object* was successfully deleted. This message appears in the shell's standard output stream.

Message Code	Message and Description
SRT449I	**SRT449I (Shell** *shellID***) All elements removed -- list reset** CAUSE: This message means that monitoring was taking place in the shell *shellID* and the object list associated with one of the active monitors was reset. All preceding messages associated with that monitor should be disregarded as stale. This message appears in the shell's standard output stream.
SRT450W	**SRT450W (Shell** *shellID***) flush:** *objectKind* **objects have no cache that can be flushed** PROBLEM: The shell *shellID*, while executing the **flush** command, was unable to flush the internal cache for the given *objectKind*. CAUSE: This problem is caused by specifying an *objectKind* that has no internal cache. The only *objectKinds* that have an internal cache are **fileReplicationJobs, replicationPairs, scripts, pathRules, selectionRules, destinationRules** and **servers**. For example, the command **`flush alert`** would produce this message. SOLUTION: Note that this error is only a warning; thus, the caches for other kinds of objects that were also specified will be flushed. If one or more *objectKind*(s) are to be specified in the **flush** command, be sure to specify just those listed above.
SRT451I	**SRT451I (Shell** *shellID***) ReadCommandLineTokens returned** *resultCode*, *quantity* **token(s):** *tokenList* CAUSE: This message means that the given quantity of tokens in the *tokenList* were being returned to the shell *shellID* for further processing. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 3 (the maximum level).
SRT452I	**SRT452I (Shell** *shellID***) ReadCommandLine returned** *resultCode*, *length* **character(s):** *commandLine* CAUSE: This message means that the given *commandLine*, which contains *length* characters, was being returned to the shell *shellID* for further processing. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 3 (the maximum level).

Message Code	Message and Description
SRT453E	**SRT453E (Shell** *shellID***) Macro substitution failure, variable '***identifierName***' does not exist** PROBLEM: The shell *shellID* could not perform a macro substitution that was found for a variable named *identifierName* that did not exist in any of the current execution contexts or the global one. CAUSE: This problem is usually caused by misspelling the variable name used in a macro replacement in the command line. SOLUTION: Correct the spelling of the macro, taking care to ensure that the variable name it uses really does exist, then try the command again.
SRT454E	**SRT454E (Shell** *shellID***) Macro substitution problem, variable '***identifierName***', unable to tokenize replacement text** *replacementText*, *reasonCode* PROBLEM: The shell *shellID* could not perform a macro substitution for the variable named *identifierName* because replacing the macro with the *replacementText* in the command line failed because of the given *reasonCode*. CAUSE: This message most likely indicates a shortage of memory, and should not occur during normal operation of srTool. SOLUTION: If the problem recurs even with sufficient system memory, please contact Technical Support.
SRT455I	**SRT455I (Shell** *shellID***) monitor: Attempting to** *operation* **monitor** *number* **for** *objectSpec* CAUSE: This message means that the shell *shellID* was trying to start, stop, pause or resume the monitor with the given *number* for the given *objectSpec*. If the shell's ***verbose*** variable is set to ***true***, this message will appear once in the standard output stream for each object specification for which a monitor is being started, stopped, paused or resumed.
SRT456I	**SRT456I (Shell** *shellID***) monitor: Monitor *number operation* for** *objectSpec* CAUSE: This message means that the shell *shellID* had successfully started, stopped, paused or resumed the monitor with the given *number* for the given *objectSpec*. If the shell's **verbose** variable is set to **true**, this message will appear once in the standard output stream for each object specification for which a monitor successfully started, stopped, paused or resumed. It usually appears immediately after the **SRT455I** message.

Message Code	Message and Description
SRT457E	**SRT457E (Shell** *shellID***) monitor: Duplicate monitor, same as [***number***]** *objectSpec* PROBLEM: The shell *shellID* could not add a new monitor for the given *objectSpec* because it already had a monitor (with the given *number*) for that exact same *objectSpec*. CAUSE: The shell *shellID* is already monitoring that same object specification. SOLUTION: Be sure that the object specification being used in the **monitor** command is unique amongst all other currently active monitors.
SRT458E	**SRT458E Built-in function** *functionName***: Illegal data type '***actual***' specified -- use '***desired***' instead** PROBLEM: The built-in function *functionName* was called with an argument of type *actual* when it was expecting an argument of type *desired*. CAUSE: The most common source of this error is the incorrect use of the built-in function **`ReadFile`**, which, if called with a second parameter, it must be equal to the ordinal value of the srTool data types **byteArray** (**blob**) or **string**. SOLUTION: Be sure that the arguments passed to built-in functions are what the functions expect. See the documentation for the Built-in Functions under "Functions" on page 57.
SRT459W	**SRT459W (Shell** *shellID***) Execution failed in embedded command** `*command*` **--** *reason* PROBLEM: The shell *shellID* could not substitute the output of the *command* into the command line because of the given *reason*. CAUSE: This message can be followed by one or more additional messages and most likely indicates a shortage of memory, and should not occur during normal operation of srTool. SOLUTION: If the problem recurs even with sufficient system memory, please contact Technical Support.
SRT460I	**SRT460I (Shell** *shellID***) Macro substitution(s) completed –** *commandLine* CAUSE: This message means that the given *commandLine* resulted from macro replacement being performed by the shell *shellID*. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 2 or higher.

Message Code	Message and Description
SRT461I	**SRT461I (Shell** *shellID***) Embedded command substitution(s) completed --** *commandLine* CAUSE: This message means that the given *commandLine* resulted from embedded command replacement being performed by the shell *shellID*. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 2 or higher.
SRT462E	**SRT462E (Shell** *shellID***)** *command***: Unable to define variable '***name***' -- a property (***propertyID***) already uses that name** PROBLEM: While executing *command* in shell *shellID*, the identifier *name* could not be defined because it matches the name of an object property (whose ordinal value is *propertyID*). CAUSE: There are two ways to produce this error. One is by creating a new ***global*** variable with that *name*; the other is to specify a **loop** variable with that *name*. Either way, if the *name* matches the name of a property – any property of any object – this error will result. SOLUTION: Be sure that any variable names to be defined are unique names that do not collide with any of the existing object property names. See "srTool Object Reference" on page 135, or use the command **`get name of all properties of all objectkinds`**.
SRT463I	**SRT463I (Shell** *shellID***)** *command***: Resolved** *objectSpec* **into** *count* **iterator(s) --** *resultCode* CAUSE: This message means that the shell *shellID*, while executing *command*, resolved the given *objectSpec* into *count* iterators. If successful, the *resultCode* will be **RXRESULT_Success**. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 3 (the maximum setting).
SRT464I	**SRT464I (Shell** *shellID***) delete:** *count* **object(s) deleted using '***objectSpec***'** CAUSE: This message means that the shell *shellID*, while executing the **delete** command, was able to successfully delete *count* objects at once using a single operation, rather than deleting each object individually. This message only appears in the standard output stream if the shell's ***verbose*** variable is set ***true***.

Message Code	Message and Description
SRT465I	**SRT465I To get a single result from** *propertyName*, **use 'minimum', 'average', 'median', 'maximum' or 'total' keyword before** *propertyName* CAUSE: This message appears with the **SRT375E** message whenever an expression is used that contains a property specification that incorporates an object specification that yields more than one object. This message is a reminder that a single data value can be obtained from multiple objects if one of the special keywords is used in the property specification. See "Property Specifications" on page 55.
SRT466W	**SRT466W (Shell** *shellID*) **shift: Desired parameter shift** *desiredCount* **computed from '***expression***' exceeded existing parameter count of** *paramCount* **and was truncated** PROBLEM: While executing the **shift** command in shell *shellID*, the desired *shiftCount* exceeded the *paramCount* and was truncated to that value. CAUSE: This is not a serious error, thus the "W" (warning) designation. It only means that the desired parameter shift exceeded the number of parameters, so the actual shift to be performed by the **shift** command will match the number of parameters. SOLUTION: To avoid this warning, be sure that the shift count expression results in a value that is less than or equal to the existing parameter count.
SRT469S	**SRT469S Local ENL service has stopped or failed -- srTool cannot continue** PROBLEM: srTool has been notified that the local ENL service was commanded to stop or it was terminated due to some failure condition. CAUSE: This is a serious error, thus the "S" (serious) designation. This can be caused by an administrator stopping the ENL service on the local host machine while one or more srTool instances are running in the command prompt windows on the same machine. SOLUTION: To avoid this problem, be sure that all *Replication Exec* console and srTool instances have been quit from before stopping the local ENL service.

Message Code	Message and Description
SRT470E	**SRT470E (Shell** *shellID***) use: Default object specification originates with non-root-level object (***objectKind***)** PROBLEM: While executing the **use** command in shell *shellID*, it was discovered that the desired default object specification was rooted in an *objectKind* that is not a root-level object. A default object specification must be rooted at a root-level object. CAUSE: This is caused by specifying a non-root-level object at the "root" of the compound object specification used in the **use** command. For example the command **`use first job`** would produce this error because **job** objects are not root-level objects. (**Job** objects are obtained from **`rms`** objects.) SOLUTION: Be sure that the "root" of the compound object specification used in the **use** command refers to a root-level object. The left-most objects listed in the "srTool Object Hierarchy" on page 64 are the root-level objects.
SRT471I	**SRT471I (Shell** *shellID***)** *command***: Parent execution context changed from '***childContext***' to '***parentContext***'** CAUSE: This message means that the current execution context of shell *shellID* was changed from the *childContext*, which is being destroyed, to the *parentContext*, while executing *command*. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 3 (the maximum setting).
SRT472I	**SRT472I (Shell** *shellID***)** *command***: Context change 'DoChange' not handled in context '***thisContext***' -- passing on to context '***parentContext***'** CAUSE: This message means that the *command* (typically **else, elseIf, break** or **continue**), executing in the shell *shellID*, was trying to alter the characteristics of an execution context in the context stack. *ThisContext* refused to handle the alteration, and was passing the request on to the *parentContext*. This message only appears in the srTool log files if the shell variable **debugLevel** is set to 3 (the maximum setting).

Message Code	Message and Description

SRT473E	**SRT473E (Shell** *shellID***)** *command: 'successorCommand***' expected after '***predecessorCommand***'**
	PROBLEM:
	The shell *shellID*, having already seen the *predecessorCommand*, requires the *successorCommand* to come after it, but instead it found *command*, which is illegal.
	CAUSE:
	This is usually caused by having more than one **else** clause in an **if** command context, or an **elseif** (or **else if**) clause after an **else** clause in an **if** command context. For example, the following **if** command context will produce this error:
	`if true`
	`echo true`
	`else`
	`echo false`
	`else`
	`echo Huh?`
	`end if`
	SOLUTION:
	Check to make sure that no other **else, elseif** (or **else if**) clauses follow an **else** clause in an **if** command context.

SRT474E	**SRT474E (Shell** *shellID***)** *command***: Unable to obtain '***objectSpec***' because no** *objectKind* **resulted from '***lastObjectSpec***'** PROBLEM: The shell *shellID*, while executing *command*, wasn't able to obtain any objects for the given *objectSpec* because there were no objects of type *objectKind* that resulted from the *lastObjectSpec*. In other words, there were no parent objects from which subordinate objects could be obtained. CAUSE: This is a common problem that can happen when composing compound object specifications to make inqueries of the replication system. For example, if the "first job" of the RMS had only one pathRule, then the following command would produce this error: `list all selRules of rule 5 of first job` srTool would not be able to obtain any selectionRule objects, because no pathRule objects would result from "rule 5 of first job" – there is no "rule 5". SOLUTION: Be sure that intermediate queries will produce objects from which subsequent queries can be made.

Index