



HVR

HIGH VOLUME REPLICATOR

HVR User Manual

Version 4.2
August 2009



HVR User Manual

PSB High Volume Replication bv
Haaksbergweg 45
1101 BR Amsterdam
The Netherlands

Telephone: (+31) 20 312 7512
<http://www.hvr.biz>

Copyright © 2009 PSB High Volume Replication bv
Printed in The Netherlands.
All rights reserved.

Table of Contents

Table of Contents	3
Conventions	5
1. Introduction	7
1.1. Terminology	7
1.2. Process Architecture and Network Connections	7
1.3. Overview of Steps to Setup a Channel	8
1.4. Starting Replication	9
1.5. Refresh and Compare	10
1.6. Calling HVR on the Command Line	11
2. Before Installing HVR	13
2.1. Where Should HVR be Installed?	13
2.2. Which Unix or Linux Permissions are Needed?	14
2.3. Which Microsoft Windows Privileges are Needed?	14
2.4. Which Ingres Permissions are Needed?	15
2.5. Which Oracle Permissions are Needed?	17
2.6. Which Microsoft SQL Server Permissions are Needed?	18
2.7. How Much Disk Room is Needed?	18
2.8. Which Network Ports Must be Open Through the Firewall?	19
2.9. How Much Network Bandwidth Will Be Used?	20
3. Installing HVR	21
3.1. New Installation on Unix or Linux	21
3.2. New Installation on Windows	23
3.3. Installing HVR Monitor on Unix or Linux	25
3.4. Installing HVR Monitor for Apache on Windows	28
3.5. Installing HVR Monitor for Microsoft IIS	30
3.6. Installing an HVR Upgrade	32
4. After Installing HVR	35
4.1. Configuring User Profiles	35
4.2. Restarting HVR after Reboot	35
4.3. Adapting a Channel due to Data Model Changes and DDL	37
4.4. Scheduling Regular Maintenance Scripts	39
4.5. Checkpointing an Ingres Capture Database	39
4.6. Managing Oracle Archive/Redo Logfiles	40
5. Command Reference	41
5.1. hvr runtime engine	41
5.2. hvrcatalogexport, hvrcatalogimport	44
5.3. hvrcrypt, hvrcryptdb	47
5.4. hvrgui	49
5.5. hvrload	51
5.6. hvrlogrelease	54
5.7. hvrmaint	57
5.8. hvrmonitor	61
5.9. hvrping, hvrtestlistener, hvrtestlocation, hvrschedport	62
5.10. hvrrefresh, hvrcompare	63
5.11. hvrremotelistener	67
5.12. hvrretryfailrows, hvrretryblockloc, hvrhistorypurge	69
5.13. hvrrouterview	71
5.14. hvrscheduler	72
5.15. hvrsslgen	80
5.16. hvrstatistics	83
5.17. hvrsuspend	85
5.18. hvrtrigger	86
6. Action Reference	88
6.1. DbCapture	90
6.2. DbIntegrate	94

6.3. TableProperties	96
6.4. ColumnProperties	97
6.5. Restrict	99
6.6. CollisionDetect	102
6.7. DbObjectGeneration	104
6.8. Agent	108
6.9. Environment	110
6.10. FileCapture	111
6.11. FileIntegrate	112
6.12. FileFormat	113
6.13. LocationProperties	114
6.14. Scheduling	115
7. Objects Used Internally by HVR	117
7.1. Catalog Tables	117
7.2. Naming of HVR Objects Inside Database Locations	124
7.3. Extra Columns for Capture, Fail and History Tables	125
7.4. Integrate Receive Timestamp Table	126
A:Quick Start for Ingres	128
B: Quick Start for Oracle	134
C: Quick Start for SQL Server	141
D: Example Replication between Different Table Layouts	147

Conventions

This document uses a number of conventions that will be explained in this section.

Computer Words and Examples

Words displayed in **bold** indicate computer ‘words’ that are fixed, or code examples in running paragraphs, such as ‘**grant execute permission**’. Words displayed in *italics* indicate computer words that are variable. For example, in a directory path such as `$HVR_CONFIG/log/hubdb` the word ‘*hubdb*’ is in italics indicating that it is variable and should be replaced by the word that is applicable to you. Code examples are displayed in a courier typeface. For example:

```
$ hvrsuspend -u hubdb SYSTEM
```

Menus

Menu selection sequences are displayed in bold and each selection is divided by the ‘>’ sign. For example: select **Tools > Data > Entity > Organization**. This means select the menu option **Tools** in the menu bar, then select menu option **Data** in the **Tools** menu, followed by selecting menu option **Entity** in the **Data** submenu and finally click on menu option **Organization** in the **Entity** submenu.

Platform Specific Functionality

- Oracle** Indicates functionality only supported on Oracle
- Ingres** Indicates functionality only supported on Ingres
- SQL Server** Indicates functionality only supported on SQL Server
- Unix & Linux** Indicates functionality only supported on Unix and Linux
- Windows** Indicates functionality only supported on Microsoft Windows

File Pathnames

Many pathnames are shown using Unix conventions, e.g. using a forward slash ‘/’ as file pathname delimiter. For the Microsoft Windows platform this can be understood as a backward slash ‘\’; generally HVR converts between forward and backwards slashes as appropriate, so the two can be used interchangeably.

1. Introduction

HVR replicates transactions between databases which HVR calls ‘locations’. Each change it captures is applied to the target locations. It can also replicate between directories (file locations) or replicate between databases and directories.

1.1. Terminology

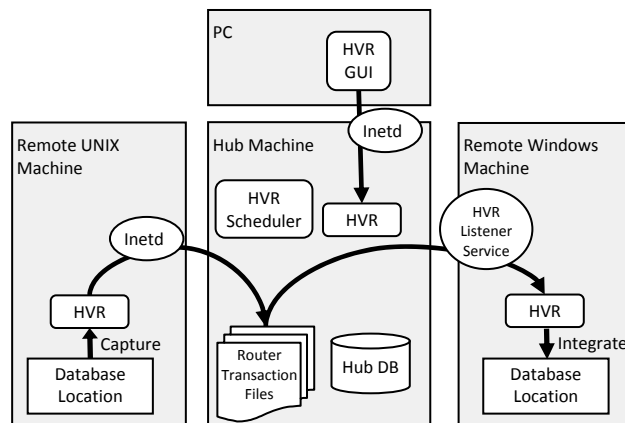
The HUB DATABASE is a small database from which HVR controls replication of the other databases. It can either be an Ingres database, Oracle_schema (username) or a SQL Server database. It is created especially for the HVR installation and can have any name (in most examples in this manual it is called *hubdb*). It contains HVR catalog tables which hold all specifications of replication such as the names of the replicated databases, the replication direction and the list of tables to be replicated. These catalog tables are created in the hub database during installation (see also sections [3.1 New Installation on Unix or Linux](#) and [3.2 New Installation on Windows](#)).

A LOCATION is a database which HVR will replicate to or from. A location can also be a directory (a file location) from which HVR replicates files.

A CHANNEL is an object in HVR which defines everything about replication set; which locations should be connected together and which tables should be replicated. It also contains actions which control how replication should be done. For example to capture changes a **DbCapture** action must be defined on a database location. Channels are defined in the hub database.

1.2. Process Architecture and Network Connections

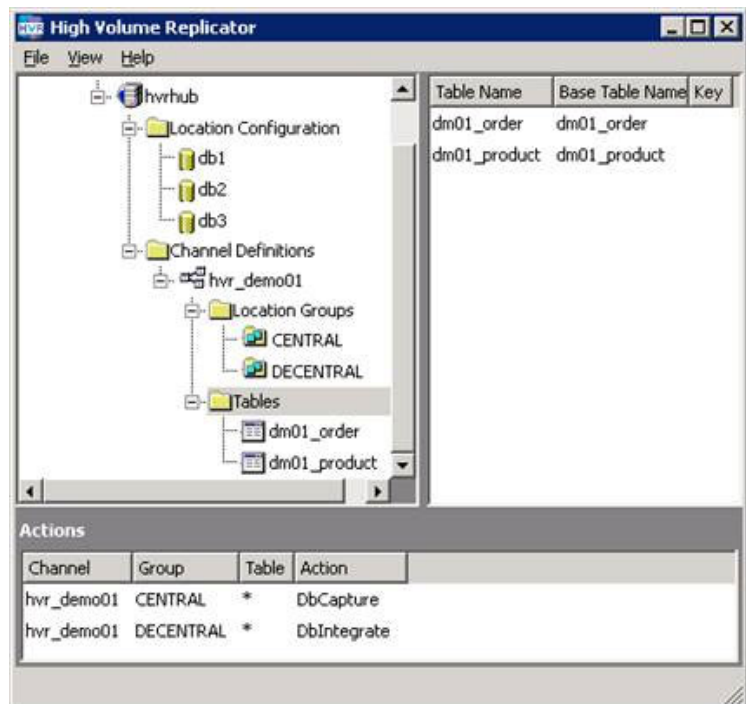
The hub machine contains the HVR hub database and an HVR scheduler. Process which controls the replication jobs and all the log files. Locations can either be local (i.e. on the hub machine), or remote. To access a remote location HVR normally connects to an HVR installation on that remote machine using a special TCP/IP port number. If the remote machine is Unix then the INETD daemon is configured to listen to this TCP/IP port. If it is a Windows machine then HVR listens with its own HVR Remote Listener (a Windows Service). Alternatively HVR can connect to a remote database location using the DBMS protocol such as Oracle TNS. HVR on remote machines is quite passive. The HVR Scheduler on the hub machine starts capture and integrate jobs which connect out to the remote location and either capture or apply (‘integrate’) changes to the remote location. Once installed on a remote machine, all of the replication is entirely controlled from the hub machine.



1.3. Overview of Steps to Setup a Channel

HVR must first be installed on the various machines involved. For the installation steps, see sections [3.1 New Installation on Unix or Linux](#) or [3.2 New Installation on Windows](#). These installation steps also create a hub database containing empty catalog tables.

Once HVR is installed it can be managed using a Graphical User Interface (GUI). The GUI can just run directly on the hub machine if the hub machine is Windows or Linux. Otherwise it should be run on the user's PC and connect to the remote hub machine.



To start the GUI double-click on its shortcut or command [hvrGui](#) on Linux. See section [5.4 hvrGui](#) for more information. The HVR GUI allows a channel to be defined in the hub database. The channel must contain at least two locations (e.g. an Oracle schema, or an Ingres or SQL Server database). It must further contain location groups. This is a collection of locations belonging to a channel. HVR actions are typically defined on the channel's location group. Once a location is defined the [Table Select](#) option in the GUI can be used to import a list of tables into the channel for replication. Action [DbCapture](#) is defined on the source database to capture database changes and [DbIntegrate](#) is defined on the target database to apply ('integrate') changes. This behavior can be reconfigured by specifying other parameters on the actions or by adding other actions.

1.4. Starting Replication

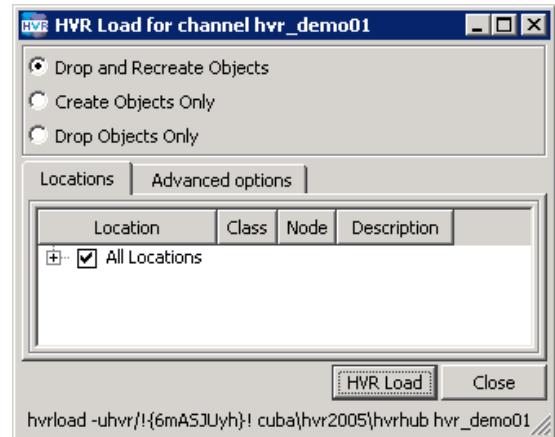
The runtime replication system is generated by command **HVR Load** in the GUI or **hvrload** from the command line on the hub machine. **HVR Load** checks the channel and then creates the objects needed for replication, plus replication jobs in the HVR Scheduler. Also for trigger-based capture (as opposed to log-based capture) HVR creates database objects such as triggers (or ‘rules’) for capturing changes.

Once **HVR Load** has been performed the process of replicating changes from source to target location occurs in the following steps:

1. Changes made by a user are captured. In case of log-based capture these are automatically recorded by the DBMS logging system. For trigger-based capture, this is done by HVR triggers inserting rows into capture tables during the user’s transaction.
2. When the ‘capture job’ runs, it transports changes from the source location to router transaction files on the hub machine. Note that when the capture job is suspended, changes will continue being captured (step 1).
3. When the ‘integrate job’ runs, it reads from the router transaction files and insert, update and delete statements on the target location to mimic the original change made by the user.

Runtime replication requires that the HVR Scheduler is running. Right-click on the hub database to create and start the HVR Scheduler.

HVR Load creates jobs in suspended state. These must be activated using command **hvrsuspend** with option **-u** (for unsuspend).



Three examples:

```
$ hvrsuspend -u hubdb SYSTEM           # unsuspend all channels in hub
$ hvrsuspend -u hubdb CHN              # unsuspend all jobs of channel CHN
$ hvrsuspend -u hubdb CHN-CAP          # unsuspend channel CHN's capture jobs
```

Note the use of capitals. The last argument in fact refers to job groups, into which similar jobs are assembled.

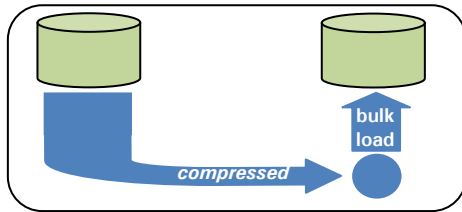
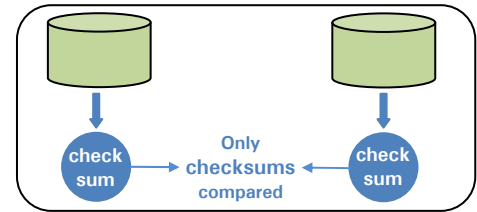
Command **hvrsuspend** requires the HVR Scheduler to be running. For more information, see section 5.17 [hvrsuspend](#).

The HVR Scheduler collects output and errors from all its jobs in several log files in directory **\$HVR_CONFIG/log/hubdb**. Each replication job has a log file for its own output and errors and there are also log files containing all output and only errors for each channel and for the whole of HVR.

1.5. Refresh and Compare

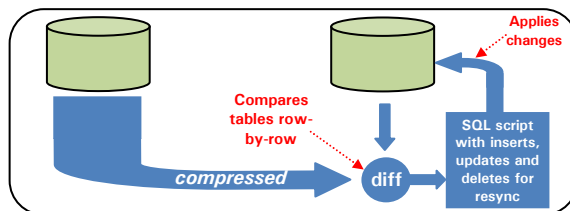
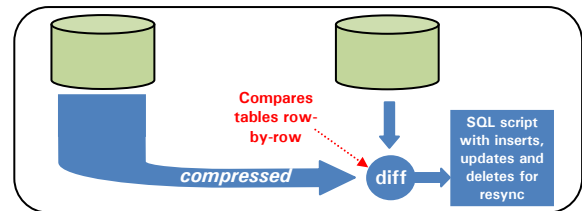
As well as actual replication (capturing each change and applying it to another database), HVR supports refresh and compare. HVR supplies two flavours of refresh and compare.

Bulk compare means that HVR performs checksums on each of the tables in the replication channel. The actual data does not travel across the network so this is very efficient for large databases over a WAN.



Bulk Refresh means that HVR extracts the data from all the tables in one database, compresses it, brings it across the network, uncompresses it and loads that data into the target database. After the data has been bulk loaded into the target database, indexes are reinitialized.

Row-wise compare means that HVR extracts the data from one database, compresses it and on the the target database it compares those changes row by row with the changes in the target database. For each difference detected an SQL statement is written; an insert, update or delete.



Row-wise refresh is the same as row-wise compare. But the resulting SQL statements are applied directly to the target database in order to resynchronize the tables.

1.6. Calling HVR on the Command Line

Many HVR commands take a hub database name (*hubdb*) as their first argument. This is either an Ingres database, an Oracle schema (username) or a SQL Server database, depending on the form:

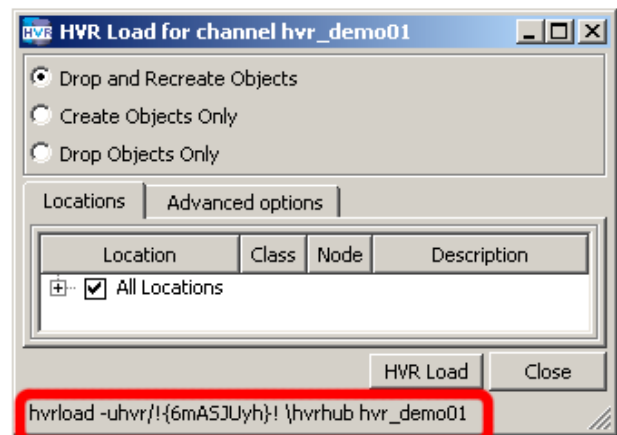
Example	Description
<code>hvrload hubdb hvr_demo01</code>	Ingres hub database hubdb , because the database name contains neither a forward nor backslash.
<code>hvrload hubdb/pwd hvr_demo01</code> <code>hvrload hubdb/pwd@tnsname hvr_demo01</code>	Oracle schema hubdb with password pwd , because the name contains a forward slash.
<code>hvrload \hubdb hvr_demo01</code> <code>hvrload inst\hubdb hvr_demo01</code> <code>hvrload node\inst\hubdb hvr_demo01</code> <code>hvrload -uuser/pwd \hubdb hvr_demo01</code> <code>hvrload -hsqlserver hubdb hvr_demo01</code>	SQL Server hub database hubdb , because the name contains a backslash. A SQL Server node and SQL Server instance can be added with extra backslashes. A username and password can be supplied with option -u .

The examples above are for command **hvrload**, but other commands like **hvrrefresh** and **hvrcompare** also behave the same.

The hub database class can be defined explicitly either using option **-hclass** or with environment variable **\$HVR_HUB_CLASS**. Valid values are **ingres**, **oracle** or **sqlserver**.

For an Oracle or SQL Server hub database a DBMS password may be required as a command line argument. These passwords can be provided in an encrypted form using command **hvincrypt** to prevent them being visible in the process table (for example with Unix command **ps**). See also section 5.3 [hvincrypt](#), [hvincryptdb](#).

HVR commands can be performed either inside the GUI or on the command line. Each GUI dialog displays the equivalent command line at the bottom of the dialog window.



2. Before Installing HVR

Before proceeding, check the following are present:

- ❖ The HVR distribution CD-Rom or file. Read the COMPATIBILITY section of the accompanying RELEASE NOTES to ensure this version is compatible with the target Operating System and DBMS. The release notes are in `$HVR_HOME/hvr.rel`.
- ❖ An HVR license file (named `hvr.lic`) which is normally delivered separately to each customer by HVR Technical Support.
- ❖ Perl (version 5 or higher) must be installed on the hub machine. Most operating systems already have Perl installed; if it is not installed then a binary distribution can be downloaded from <http://www.cpan.org/ports>.
- ❖ It is recommended that HVR is installed under its own login account (e.g. `hvr`) on the hub machine and each machine containing the replicated database or directory. However HVR can also use the account that owns the replicated tables (e.g. the DBA's account) or it can use the DBMS owner account (e.g. `ingres` or `oracle`). Any login shell is sufficient.
- ❖ An Ingres, Oracle or Microsoft SQL Server database on the hub. Initially this may be empty.

2.1. Where Should HVR be Installed?

The HVR distribution must be installed on the hub machine. If a database involved in replication is not located on the hub machine HVR can connect to it in two ways; either using the network database's own protocol (e.g. SQL*Net or Ingres/Net) or using an HVR remote connection (recommended).

For an HVR remote location the full HVR distribution should also be installed on the remote machine, although a few of its files are used. If the DBMS's own protocol is used then no HVR files need be installed on the remote machine.

If the HVR replication configuration is altered, all the reconfiguration steps are performed on the hub machine; installation files on the remote machine are unchanged.

HVR makes no assumptions about there being only one installation per machine, and locations sharing a single machine need only have HVR installed once and HVR can replicate data between these locations without difficulty.

The HVR distribution files are installed under environment variable `$HVR_HOME`. Files created by HVR itself are created underneath `$HVR_CONFIG`. Both these variables must be configured.

2.2. Which Unix or Linux Permissions are Needed?

Unix & Linux

If the HVR GUI must connect from a PC to the hub machine or if the hub machine must connect using HVR's protocol to a remote machine then root permission is required to edit the `inetd` configuration files.

The HVR Monitor (the HVR Scheduler's JAVA-based user interface) requires a HTTP daemon (typically Apache) on the hub machine. If the HVR Monitor is to be used, then this HTTP daemon can operate under the same account as HVR or under a different account. If it is to operate under a different account then the following is required:

- ❖ The account under which the HTTP daemon must be a member of the default Unix group (as defined in `/etc/group`) of the account under which the HVR Scheduler.
- ❖ The account under which the HTTP daemon must have permission to use the Unix `setuid(2)` system call. On HP-UX, this account must be in a Unix group with `SETRUGID` privilege, which is granted using command `setprivgrp(1m)` or by adding a line with group name `SETRUGID` to the file `/etc/privgroup`, which is applied by HP-UX at reboot.

Ingres

Log-based capture requires that HVR has permission to read the internal DBMS logging files which are owned by Ingres. Installing this requires a special step to create a trusted while logged in as `ingres`. This installation step is not needed if HVR runs under login `ingres`, and is also not needed if trigger-based capture is used or on Microsoft Windows.

Oracle

If HVR must perform Oracle log-based capture, the Unix username that HVR uses must be in Oracle's `dba` Unix group (file `/etc/group`).

2.3. Which Microsoft Windows Privileges are Needed?

Windows

On Microsoft Windows, HVR must be installed under a user which is a member of the `Administrator` group and must be granted `SeTcbPrivilege` ('Act as part of the operating system') and `SeServiceSid` ('Log on as a service').

On Microsoft Windows 2000 the screen to grant this privilege can found under `Settings > Control Panel > Administrative Tools > Local Security Policy > Local Policies > User Rights Assignment`.

On Microsoft Windows XP the screen to grant this privilege can found under `Start > Control Panel > Administrative Tools > Local Security Policy`. Alternatively use the command prompt with command: `C:\Windows\System32\secpol.msc`.

2.4. Which Ingres Permissions are Needed?

Ingres

For an Ingres hub and for any Ingres HVR database locations, each account must have permission to use Ingres, as well as permission to use the **-u** flag ('security administrator' privilege) and also Ingres trace points. Other privileges ('maintain locations', 'operator' etc.) are not needed.

Accessdb permission screen

```

ACCESSDB - User Information

  User Name: hvr
  Profile for User: _____
  Default Group: _____
  Expire Date: _____

Permissions:      Create Database: y      Operator: y
                  Security Administrator: y      Set Trace Flags: y
                  Maintain Locations: n      Maintain Users: n
    
```

Databases Owned	Authorized Databases
hubdb	

Save(F10) Help(Help) End(PF3) Password(F14) Privileges(F17) >

DBA permission screen

Alter User on malta

User Name: hvr
 Default Group: (No Group)
 Default Profile: (No Profile)

Privilege	Users	Default
Create Database	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Trace	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Security	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Operator	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Maintain Location	<input type="checkbox"/>	<input type="checkbox"/>
Auditor	<input type="checkbox"/>	<input type="checkbox"/>
Maintain Audit	<input type="checkbox"/>	<input type="checkbox"/>
Maintain Users	<input type="checkbox"/>	<input type="checkbox"/>

Security Audit
 All Events
 Query Text

Expire Date: _____
 Limiting Security Label: _____

Old Password: _____
 Password: _____ External Password
 Confirm Password: _____

Remote Command (mcmd) Privileges

For trigger-based capture from Ingres databases the isolation level must be set to serializable.

```

CBF Configure DBMS Server Definition

  Host: guam
  II_SYSTEM: /distr/ingres/303
  II_INSTALLATION: LL

  Definition Name: (default)
    
```

DBMS Server Parameters		
Name	Value	Units
stack_caching	OFF	boolean
stack_size	153600	bytes
system_isolation	serializable	string
system_lock_level	default	string
system_maxlocks	50	integer
system_readlocks	shared	string
system_timeout	0	seconds

Edit(2) Databases(3) Cache(4) Derived(5) Restore(6) >

Ingres**Unix & Linux**

Log-based capture requires that HVR has permission to read the internal DBMS logging files which are owned by Ingres. Installing this requires a special step to create a trusted while logged in as **ingres**. This installation step is not needed if HVR runs under login **ingres**, and is also not needed if trigger-based capture is used or on Microsoft Windows.

2.5. Which Oracle Permissions are Needed?

Oracle

HVR needs its own username on the hub machine, where it keeps its control information. All Oracle usernames used by HVR need the following system privileges: **create session**, **create table**, **create sequence**, **create procedure**, **create trigger** and **any procedure**. The hub username on the hub and any user names used for trigger-based capture must also be given a "grant on dbms_alert to user" by the **SYS** account.

Example

```
$ sqlplus system/manager
SQL> create user hvrhub identified by hvr
  2 default tablespace users
  3 temporary tablespace temp
  4 quota unlimited on users
  5 /
SQL> grant create session to hvrhub
  2 /
SQL> grant create table to hvrhub
  2 /
SQL> grant create sequence to hvrhub
  2 /
SQL> grant create procedure to hvrhub
  2 /
SQL> grant create trigger to hvrhub
  2 /
SQL> grant create view to hvrhub
  2 /
SQL> grant any procedure to hvrhub
  2 /
SQL> exit
```

Login as operating system user **oracle** (or another user with **sysdba** privilege) and perform the following extra statements. This must be done for the hub database username and for any user where **hvr** will do trigger-based capture:

```
$ sqlplus
Enter user-name: / as sysdba
SQL> grant execute on dbms_alert to hvrhub
  2 /
SQL> exit
```

Log-based capture requires that:

- On Unix and Linux the user name that HVR uses must be in Oracle's **dba** Unix group (file **/etc/group**).
- On Windows the database user name that HVR uses must have **sysdba** privilege. This is done with statement **grant sysdba to hvr**.
- The Oracle instance must have archiving enabled. This can be done by running the following statement as **sysdba** against a mounted but unopened database **alter database archive log**. The current state of archiving can be checked with query **select log_mode from v\$database**.

2.6. Which Microsoft SQL Server Permissions are Needed?

The account used for HVR should be defined with [SQL Server authentication](#) instead of [Windows authentication](#) and should be a member of the SQL Server [system administration](#) role.

For SQL Server 2000 databases the extended stored procedure [hvrevent](#) must also be installed. See also section [3.2 New Installation on Windows](#). This step requires system administrator access to the master database.

2.7. How Much Disk Room is Needed?

The full HVR distribution occupies 150Mb. For a simple situation such as channel [hvr_demo01](#) about 10Mb more disk room is needed, plus 2Mb for the hub database and 2Mb for each of the replicated databases. The following illustrates the disk usage on the hub machine:

📁 HVR_HOME	Smaller than 150 Mb.
📁 HVR_CONFIG	
├─ files	} Smaller than 1 Mb.
├─ jobs	
├─ sqlgen	
├─ wwwgen	
├─ jnl	Compressed data files, grows every time data is replicated.
├─ log	Contains a record for each action such as transport, routing and integration.
├─ router	Compressed data files, grows if replication has a backlog.
└─ work	Temporary files for sorting large amounts of data.

The following illustrates the disk usage on a remote location:

📁 HVR_HOME	Smaller than 150 Mb.
📁 HVR_CONFIG	
├─ files	Smaller than 1 Mb.
└─ work	Temporary files for sorting large amounts of data.

For replicating files, HVR also requires diskroom in its 'state directory'. This is normally located in subdirectory [_hvr_state](#) which is inside the file location, but this can be changed using parameter [LocationProperties/StateDirectory](#) (see section [6.13 LocationProperties](#)). When capturing changes the files in this directory will be less than 1 Mb, but when integrating changes HVR can make temporary files containing data being moved at that moment.

2.8. Which Network Ports Must be Open Through the Firewall?

To configure connections between the hub machine, the remote HVR locations and an operator's PC it is important to understand which TCP/IP ports HVR uses at runtime. The operator can control replication by logging into the hub machine and using command line commands or running the GUI from the hub. Alternatively the HVR GUI and/or the HVR Monitor can be used from his PC, in which case several TCP/IP ports must be opened from his PC to the hub machine.

Additionally the HVR may need to connect from the hub machine to remote machines. This connection can be established using its own protocol (typically port 50000), but to connect to a database location it can also use a DBMS protocol such as Oracle TNS. Connections are never needed straight from the operator's PC to a remote machine or from a remote machine to a hub machine or the operator's PC.

Connection	TCP/IP Port Number	Network Protocol (unencrypted)	Network Protocol (encrypted)
HVR GUI connecting from PC to hub machine.	Arbitrary port, typically 50000. On Unix and Linux the listener is the inetd daemon. On Windows the listener is the HVR Remote Listener service.	HVR internal protocol.	
Loading HVR applet to PC from hub machine.	Typically port 80 or port 8080.	HTTP.	HTTPS.
Running HVR Monitor applet from PC to hub machine.	Normally the HVR Scheduler listens on a TCP/IP port which derives (using a hash algorithm) from the name of its database. The resulting port number will be between 50000 and 59999. A fixed TCP/IP port (typically 50010) can be configured by defining variable \$HVR_PUBLIC_PORT in the Operating System environment of the HVR Scheduler, or using a set attribute in its SYSTEM job group.		
Replicating from hub machine to remote location using HVR remote connection.	Arbitrary port, typically 50000. On Unix and Linux the listener is the inetd daemon. On Windows the listener is the HVR Remote Listener service.	HVR internal protocol.	SSL encryption can be enabled using HVR action LocationProperties/SslCertificate .
Replicating from remote location using DBMS protocol.	DBMS dependant.	Oracle's TNS protocol, or Ingres/NET or SQL Server protocol	Not available.

2.9. How Much Network Bandwidth Will Be Used?

HVR is extremely efficient over WAN and uses the minimum possible bandwidth. Examination of the compression ratios reported by capture jobs in the HVR can be used to accurately determine the amount of bandwidth used. These logfiles can be found under [\\$HVR_CONFIG/log](#). Action [LocationProperties /ThrottleKbytes](#) can be used to reduce the fraction of available bandwidth used.

See also section [6.13 LocationProperties](#), action [/ThrottleKbytes](#) for more information.

Note: The environment variable [HVR_NET_STATS=1](#) traces the number of bytes sent and the number of packet send/receives. The bytes reported may be slightly higher than expected because they include internal 'control packets'. The number of such packets is constant for a given HVR script and does not depend on the number of rows transported. A typical HVR script may send 10 such packets, each containing about 20 bytes: so these 'control packets' can be ignored for network sizing purposes.

3. Installing HVR

This chapter describes the steps to install HVR. The first section describes the steps to [install HVR on Unix and Linux](#). The second section describes the install steps for [Windows](#). Subsequent sections describe installing HVR Monitor (an optional user interface for HVR Scheduler) on [Unix or Linux](#), [Apache on Windows](#) and [Microsoft IIS](#), and the final section describes [how to upgrade HVR](#).

3.1. New Installation on Unix or Linux

Unix & Linux

1. Choose directories for read-only and read-write parts of HVR.

```
$ HVR_HOME=/usr/hvr/hvr_home
$ HVR_CONFIG=$HOME/usr/hvr/hvr_config
$ export HVR_HOME HVR_CONFIG
```

Modify your **\$PATH**. For Bourne Shell and Korn Shell this is:

```
$ PATH=$HVR_HOME/bin:$PATH
```

If this installation is for a hub machine then this should be repeated in the shell startup script (e.g. **.profile**).

2. Read and uncompress the distribution file:

```
$ umask 0
$ mkdir $HVR_HOME $HVR_CONFIG
$ cd $HVR_HOME
$ zcat < /tmp/hvr44444xx.taz | tar xf -
```

3. Install HVR license file when this installation is for the hub machine. This file is named **hvr.lic** and is normally delivered to each customer by HVR Technical Support.

```
$ cp /tmp/hvr.lic $HVR_HOME/lib
```

This license file is only required on the hub machine.

4. If HVR must perform log-based capture from Ingres then a trusted executable must be created so it can read from the Ingres logging system. Perform the following steps while logged in as the DBMS owner (**ingres**):

```
Ingres
$ cd /usr/hvr/hvr_home
$ cp bin/hvr/sbin/hvr_ingres
$ chmod 4755/sbin/hvr_ingres
```

This step is not needed if capture will be trigger-based or if capture will be from another machine. It is also not needed if HVR is already running as the DBMS owner (**ingres**).

If HVR and **ingres** share the same Unix group, then the permissions can be tightened from 4755 to 4750.

5. For a remote machine or for a hub machine to which operators will be connecting using the HVR GUI an HVR listen port must be configured.

Pick an arbitrary TCP/IP port number between **1024** and **65535** which is not already in use. Number **50000** is recommended.

Login as **root** and configure the **inetd** daemon to invoke HVR. For an HVR installed under **/usr/hvr/hvr_home** with TCP/IP port **50000**, add the following line to **/etc/services**:

```
hvr 50000/tcp
```

For Unix the following line should be added to `/etc/inetd.conf`:

```
hvr stream tcp nowait root /usr/hvr/hvr_home/bin/hvr hvr -r
-EHVR_HOME=/usr/hvr/hvr_home -EHVR_CONFIG=/usr/hvr/hvr_config
```

On Linux however a new file should be created named `/etc/xinetd.d/hvr` which should contain the following contents:

```
service hvr
{
    socket_type          = stream
    wait                = no
    user                 = root
    server               = /usr/hvr/hvr_home/bin/hvr
    server_args          = -r
    env                  += HVR_HOME=/usr/hvr/hvr_home
    env                  += HVR_CONFIG=/usr/hvr/hvr_config
    disable              = no
    cps                  = 10000 30
}
```

Now send signal **SIGHUP (kill -1)** to the process-id of the **inetd** daemon to force it to reread its configuration information.

Notes on the options used above:

- Option **-r** tells **hvr** to run as a remote slave process.
- The values used for **HVR_HOME** and **HVR_CONFIG** are for the current machine not the hub machine.
- On some machines **hvr** needs the PAM option **-p** (Pluggable Authentication Module so it can check the passwords properly), for example **-plogin**.
- For encryption options **-C** and **-K** see section [5.1 hvr runtime engine](#).

On some machines where restricted security is configured it may be necessary to add the following line to file `/etc/hosts.allow`:

```
hvr: ALL
```

For Solaris version 10 and higher, file `/etc/inetd.conf` must be imported into SMF using command **inetconv(8)**.

Test the connection from a different machine using command **hvrtestlistener**.

```
$ hvrtestlistener node 50000
```

For an alternative to connecting with the **inetd** daemon see section [5.11 hvrremotelistener](#).

6. Oracle

For an Oracle database HVR requires that the **NLS_LANG** in its environment matches the instance's **NLS_CHARACTERSET**. This can be done by adding an **Environment** action to the channel:

chn_name	grp_name	tbl_name	act_name	act_parameters
chn	*	*	Environment	/Name=NLS_LANG /Value=AMERICAN_AMERICA.AL32UTF8

7. HVR GUI can either run on a separate PC and connect using a HVR remote connection to the hub or it can run on the hub machine itself and connect using X to an X server on the user's pc. In the first case HVR GUI will run on the user's PC and another hvr install must be performed there. HVR GUI will only run on Windows and Linux.

3.2. New Installation on Windows

Windows

1. Login to Microsoft Windows under normal user account.
2. Install distribution. This can be done using the installation wizard as follows:

```
C:\> d:\windows\hvr44444win32setup.exe
```

Then answer the installation wizard's questions to install the runtime objects.

3. Install HVR license file when this installation is for the hub machine. This file is named **hvr.lic** and is normally delivered to each customer by HVR Technical Support.

```
C:\> copy c:\tmp\hvr.lic %HVR_HOME%\lib
```

This file need only be installed in directory **%HVR_HOME%\lib** on the hub machine.

SQL Server

4. If HVR uses Microsoft SQL Server as hub database or for database capture, a special extended stored procedure must be created called **hvrevent**. This DLL must be registered in the master database with the username under which HVR will run. This requires membership of the **system administration** role. Use the following SQL command:

```
$ osql -dmaster -Uuser -Ppwd
> exec sp_addextendedproc 'hvrevent', 'c:\hvr_home\bin\hvrevent.dll'
> go
```

Note that to remove this extended stored procedure use the following SQL statement:

```
exec sp_dropextendedproc 'hvrevent'
```

Replace pathname **c:\hvr_home** with the correct value of **HVR_HOME** on that machine.

If HVR does not actually run on the machine that contains the database (either the hub database is not on the hub machine or HVR is capturing from a database without running on the machine that contains this database) then this step should be performed on the database machine, not the HVR machine. If the HVR machine is 32-bit Windows and the other database machine is 64-bit Windows then copy file **hvrevent64.dll** instead of file **hvrevent.dll**. If both machines are 32-bit Windows or both are 64-bit Windows then the file is just named **hvrevent.dll**.

5. Use environment variable **\$HVR_HUB_CLASS** to define a default DBMS type for the hub database. This can be done with **Start > Control Panel > System > Advanced > Environment Variables**. See also [1.6 Calling HVR on the Command Line](#).
6. Pick an arbitrary TCP/IP port number between 1024 and 65535 which is not already in use. Number 50000 is recommended. Create and start the HVR Remote Listener service with this port number:

```
C:\> hvrremotelistener -acs 50000
```

See also section [5.11 hvrremotelistener](#) for more details.

Test the connection from a different machine using command **hvrtestlistener**.

```
C:\> hvrtestlistener node 50000
```

7. **Oracle**

For an Oracle database HVR requires that the **NLS_LANG** in its environment matches the instance's **NLS_CHARACTERSET**. This can be done by adding an **Environment** action to the channel:

chn_name	grp_name	tbl_name	act_name	act_parameters
chn	*	*	Environment	/Name=NLS_LANG /Value=AMERICAN_AMERICA.AL32UTF8

3.3. Installing HVR Monitor on Unix or Linux

1. If no HTTP product exists yet on the hub machine, then one must be installed.

A binary distribution of Apache can be found on the HVR distribution CD-ROM. It can also be downloaded from <http://www.apache.org>.

The HTTP daemon can either be installed to run under a generic account (e.g. **root** or **apache**) or it can run under the same account as the HVR will use at runtime. An advantage of using the same account as the HVR is that the user will be free to install, start and stop the Apache daemon without needing extra privileges or passwords.

If the installation steps of HTTPD product automatically perform reconfiguration (e.g. changing the **httpd.conf**), then just perform a standard installation.

If the installation steps of HTTPD product do not automatically perform reconfiguration, then certain directives must be configured manually in the configuration file **httpd.conf**. These directives are **ServerRoot** (the directory where the HTTP product is installed), **ServerAdmin** (the operator's email address), **ServerName** (the hostname) and **DocumentRoot** (the default directory for HTML files). Normal values should be manually assigned to these directives.

Example:

```
ServerRoot /usr/local/apache
ServerAdmin jim@bilbo
ServerName bilbo
DocumentRoot /usr/local/apache/htdocs
```

2. Recognize files with extension **.cgi** as CGI scripts. Add the following line (if not already present).


```
AddHandler cgi-script .cgi
```
3. Configure the HTTP configuration file for the HVR Monitor URLs. For Apache this file is named **httpd.conf** and is located in the Apache conf directory (e.g. **/etc/httpd/conf/httpd.conf**).

For HVR installed with **HVR_HOME=/usr/hvr/hvr_home**, the following directives should be added:

```
ScriptAlias /hvr_home/cgi-bin/ /usr/hvr/hvr_home/cgi-bin/
ScriptAlias /hvr_home/cgi-sys/ /usr/hvr/hvr_home/cgi-sys/

# Note: Alias directives must be below the ScriptAlias directives

Alias /hvr_home/ /usr/hvr/hvr_home/
Alias /hvr_config/ /usr/hvr/hvr_config/

<Directory "/usr/hvr/hvr_home/cgi-sys/">
    AllowOverride All
</Directory>
```

The **AllowOverride All** directive at the end instructs the HTTP daemon to respect directives in file **\$HVR_HOME/cgi-sys/.htaccess**. These directives specify that the CGI in directory **\$HVR_HOME/cgi-sys** is password protected, and may only be run by HTTP users which are members of group **cgi-sys**.

4. Configure the TCP/IP port on which the HTTP daemon will listen.

By default an HTTP daemon will listen on port 80. It is customary (but not mandatory) for the HTTPD used for the HVR Monitor to listen on port 8080. If the HTTPD is to be run under an unprivileged account (e.g. **hvr**) then Unix will not allowed it to use a port number lower than 1024.

EITHER:

- a. Alter the HTTPD configuration file to use port 8080.

For Apache 1.3, change:

```
Port 80
```

To this:

```
Port 8080
```

For Apache 2.0, change:

```
Listen 80
```

To this:

```
Listen 8080
```

OR:

- b. Use a different port number than 8080, e.g. keep using port 80.

In this case set variable `$HVR_HTTP_PORT` to value 80. This variable can be added to the environment of each HVR Scheduler or can be defined as a `set` attributes in the `SYSTEM` job group of each hub database. This variable does not need to be set in the environment of Apache or the internet browser.

5. Create an HTTP daemon user-name which is a member of group `cgi-sys`, in files `passwd` and `group` in the HTTP daemon conf directory. For an Apache installed in `/usr/local/apache` the commands are as follows:

```
$ cd /usr/local/apache/bin
$ ./htpasswd -c /usr/local/apache/conf/passwd hvr
New password:
Re-type new password:
$ echo cgi-sys: hvr >> /usr/local/apache/conf/group

$ chmod 755 /usr/local/apache/conf
$ chmod 644 /usr/local/apache/conf/passwd
$ chmod 644 /usr/local/apache/conf/group
```

6. If the HTTP daemon will run under a different account than the HVR, then the following is required:

- The account under which the HTTP daemon executes must be a member of the default Unix group (as defined in `/etc/group`) of the account under which the HVR Scheduler executes.
- The account under which the HTTP daemon executes must have permission to use the Unix `setuid(2)` system call. Certain Unix machines restrict which users can perform this system call. On HP-UX this permission is granted by putting this account in a Unix group with `SETRUGID` privilege, which is granted using command `setprivgrp(1m)` or by adding a line with group name `SETRUGID` to the file `/etc/privgroup`, which is applied by HP-UX at reboot.

If these privileges are enabled, the HVR Monitor will ensure that all CGI is performed under the correct Unix account.

7. Start the HTTP daemon.

For Apache the command is as follows:

```
$ apachectl start
```

8. Start the HVR Scheduler. The HVR Monitor applet needs a running server to connect to:

```
$ hvrscheduler hubdb
```

The above steps would result in the following URL to startup an HVR Monitor on a Unix or Linux hub machine named **bilbo** with an HTTP daemon configured to listen on port 8080:

```
http://bilbo:8080/hvr_home/www/
```

The HVR Monitor started with this URL can be used to access any HVR Scheduler in the HVR installation. The URL to startup the HVR Monitor so that it automatically connects to the specific HVR Scheduler for database **myhubdb** is as follows:

```
http://bilbo:8080/hvr_config/wwwgen/myhubdb
```

3.4. Installing HVR Monitor for Apache on Windows

1. Install Apache on the hub machine. A binary distribution of Apache for Microsoft Windows can be found on the HVR distribution CD-ROM. It can also be downloaded from <http://www.apache.org/>.

The installation is done using an install wizard. The regular/default options can be chosen.

2. Arrange that Apache will run under the same account as HVR will use at runtime (e.g. **hvr**). This is done using **My Computer > Apache > Manage > Services and Applications > Services > Apache > Properties > Logon > This Account**. Then supply the username and password of the account under which HVR will run.
3. Configure the Apache configuration file for the HVR Monitor URLs. Choose **Programs > Apache HTTP Server > Configure Apache Server > Edit the Apache httpd.conf Configuration File**. For an HVR Scheduler whose executables are installed under **d:\hvr\hvr_home**, the following directives should be added:

```
ScriptAlias /hvr_home/cgi-bin/ d:/hvr/hvr_home/cgi-bin/
ScriptAlias /hvr_home/cgi-sys/ d:/hvr/hvr_home/cgi-sys/

# Note: Alias directives must be below the ScriptAlias directives

Alias /hvr_home/ d:/hvr/hvr_home/
Alias /hvr_config/ d:/hvr/hvr_config/
Alias /c:/ "c/"
Alias /C:/ "c/"
Alias /d:/ "d/"
Alias /D:/ "d/"

<Directory />
    AllowOverride All
</Directory>
```

The **AllowOverride All** directive at the end instructs the HTTP daemon to respect directives in file **hvr_home\cgi-sys\htaccess**. These directives specify that the CGI in directory **hvr_home\cgi-sys** is password protected, and may only be run by HTTP users which are members of group **cgi-sys**.

4. Recognize files with extension **.cgi** as CGI scripts. Add the following line (if not already present).

```
AddHandler cgi-script .cgi
```

5. Configure the TCP/IP port on which Apache will listen. By default Apache will listen on port 80. It is customary (but not mandatory) for the HTTP Service used for the HVR Monitor to listen on port 8080.

EITHER:

- a. Alter the HTTPD configuration file to use port 8080.

For Apache 1.3, change:

```
Port 80
```

To this:

```
Port 8080
```

For Apache 2.0, change:

```
Listen 80
```

To this:

Listen 8080

OR:

- b. Use a different port number than 8080, e.g. keep using port 80.
In this case set variable `$HVR_HTTP_PORT` to value 80. This variable can be added to the environment of each HVR Scheduler or can be defined as a `set` attributes in the `SYSTEM` job group of each hub database. This variable does not need to be set in the environment of Apache or the internet browser.

6. Create an Apache username `HVR` for the password-protected HVR Monitor operations which change the HVR Scheduler scheduling. The read-only operations are not password protected. This username must be a member of group `cgi-sys`.

For Apache 1.3 installed in `C:\Program Files\Apache Group` the commands are as follows:

```
C:\> cd C:\Program Files\Apache Group\APACHE
C:\> bin\htpasswd -c conf\passwd hvr
New password:
Re-type new password:

C:\> echo cgi-sys: hvr > conf\group
```

For Apache 2.0 installed in `C:\Program Files\Apache Group` the commands are as follows:

```
C:\> cd C:\Program Files\Apache Group\APACHE2
C:\> bin\htpasswd -c conf\passwd hvr
New password:
Re-type new password:

C:\> echo cgi-sys: hvr > conf\group
```

7. Start the Apache service with command

```
C:\> net start apache
```

For Apache version 2 the command should be

```
C:\> net start apache2
```

8. Start the HVR Scheduler. The HVR Monitor applet needs a running server to connect to:

```
C:\> hvrscheduler -acs -Ppwd hubdb
```

The above steps result in the following URL to startup an HVR Monitor installed on a Windows hub machine named `dosbox` where Apache is configured to listen on port `8080`:

```
http://dosbox:8080/hvr_home/www
```

The HVR Monitor started with this URL can be used to access any HVR Scheduler in that HVR installation. The URL to startup the HVR Monitor so that it automatically connects to the specific HVR Scheduler for database `myhubdb` is as follows:

```
http://dosbox:8080/hvr_config/wwwgen
```

3.5. Installing HVR Monitor for Microsoft IIS

Use the following steps to configure Microsoft Internet Information Services for a single HVR installation.

1. Install Microsoft Internet Information Services on the Windows hub machine.
2. Open the IIS configuration window using **Start > Programs > Administrative Tools > Computer Management > Services and Applications > Internet Information Services > Default Website**. The remaining steps in this section are performed from this point in the IIS configuration window.
3. Add the **hvr_home** directory. This is done from the IIS configuration window using menu option **Actions > New > Virtual Directory**. Ensure no existing directory was selected; otherwise the **New** option will not be available.

A wizard will now start up:

- In field **Alias** supply value **hvr_home**
 - In field **Directory** supply the Windows directory which is defined as **HVR_HOME** for the HVR installation, e.g. **C:\hvr\hvr_home**. Note that forward slashes (/) are illegal.
 - Enable only options **Read** and **Execute** (such as ISAPI applications or CGI).
 - Click **Finish**.
4. Add the **hvr_config** directory.

Use menu option **Actions > New > Virtual Directory**. Ensure no existing directory was selected; otherwise the **New** option will not be available.

A wizard will now start up:

- In field **Alias** supply value **hvr_config**
 - In field **Directory** supply the Windows directory which is defined as **HVR_CONFIG** for the HVR installation, e.g. **c:\hvr\hvr_config**. Note that forward slashes (/) are illegal.
 - Enable option **Read** only.
 - Click **Finish**.
5. Configure that the HVR Monitor's CGI scripts should be executed with the Perl interpreter. Right click on **hvr_home**, then choose **Properties > Virtual Directory > Configuration > Add**. Configure the following:
 - In field **Executable** add the full path of the Perl executable followed by **%s** two times, e.g. **c:\perl\bin\perl.exe %s %s**.
 - In field **Extension** add value **.cgi**.
 - In section **Verbs** enable **Limit To** and add values **GET, POST**.
 6. Configure that the HVR Monitor's CGI scripts will run under the same account as the HVR Scheduler. Right click on **hvr_home** then choose **Properties > Directory Security > Anonymous access and authentication control > Edit**. Configure the following:
 - Ensure that option **Anonymous access** is enabled.
 - Choose **Edit** and supply the username and password of the account under which the HVR Scheduler will run.
 7. Configure password protection for HVR Monitor operations which change the HVR Scheduler scheduling. The read-only operations are not password protected. Double click on **hvr_home**, then right-click on **cgi-sys** then choose **Properties > Directory Security > Anonymous access and authentication control > Edit**. Configure the following:
 - Disable **Anonymous access**,

- Enable **Basic authentication**,
 - Choose **Edit** and select an authentication domain.
8. Extend the list of HTML directory indices recognized by Microsoft Internet Information Services. Right click on **Default Web Site**, choose **Properties**, then under tab **Documents** enable option **Enable Default Document** and add **index.htm** to the list.
 9. Configure the TCP/IP port on which IIS will listen. By default IIS will listen on port 80. It is customary (but not mandatory) for the HTTP Service used for the HVR Monitor to listen on port 8080.

EITHER:

- a. Configure IIS to listen on port 8080.
Right click on **Default Web Site**, choose **Properties**, then under tab **Web Site**, fill field **TCP Port** with value 8080.

OR:

- b. Use a different port number than 8080, e.g. keep using port 80.
In this case set variable **\$HVR_HTTP_PORT** to value 80. This variable can be added to the environment of each HVR Scheduler or can be defined as a **set** attributes in the **SYSTEM** job group of each hub database. This variable does not need to be set in the environment of Apache or the internet browser.

10. Start the HVR Scheduler. The HVR Monitor applet needs a running server to connect to:

```
C:\> hvrscheduler -acs -Ppwd hubdb
```

It may be necessary to reboot the Windows hub machine at this point, in order to reset Microsoft IIS.

The above steps result in the following URL to startup an HVR Monitor installed on a Windows hub machine named **dosbox** where IIS is configured to listen on port **8080**:

```
http://dosbox:8080/hvr_home/www
```

The HVR Monitor started with this URL can be used to access any HVR Scheduler in that HVR installation. The URL to startup the HVR Monitor so that it automatically connects to the specific HVR Scheduler for database **myhubdb** is as follows:

```
http://dosbox:8080/hvr_config/wwwgen/myhubdb
```

3.6. Installing an HVR Upgrade

When existing channels connect hundreds of databases, upgrading the HVR versions can be a large undertaking. For this reason HVR defines five levels of upgrade severity: type {1} is the fastest and easiest; type {5} is the most severe and far reaching. The upgrade type for each version is reported in its Release Notes.

Upgrade type	Hub executables changed?	Upgrade remote executables?	Regenerate hvr scripts?	Layout of hub catalogs changed?	Regenerate HVR objects in database locations?
{1}	✓	✗	✗	✗	✗
{2}	✓	✓	✗	✗	✗
{3}	✓	✓	✓	✗	✗
{4}	✓	✓	✓	✓	✗
{5}	✓	✓	✓	✓	✓

More severe upgrade steps can always be used, for example a type {5} upgrade can be performed instead of a type {1}, {2}, {3} or {4} upgrade; in a non-production environment this may be the only upgrade type performed. The following upgrades are required:

Step	Upgrade Type	Instruction
A.	{5}	Arrange all users suspend work that could result in captured data. For database locations this could be checked by stopping and starting the DBMS, or by attempting an exclusive database lock.
B.	{1}, {2}, {3}, {4}, {5}	On the hub, stop runtime processing: \$ hvr scheduler -q0 hubdb
C.	{1}, {2}, {3}, {4}, {5}	Unix & Linux On AIX, uninstall the old HVR runtime executables using command: \$ rm \$HVR_HOME/lib/* Alternatively remove all cached shared libraries from the kernel by performing command slibclean as root . This step is not needed for other flavors of Unix or Linux. This step is also needed on remote machines unless this is a type {1} upgrade.
D.	{5}	Flush all replication tables for existing channels by running the capture-integrate script. \$ hvr \$HVR_CONFIG/job/hubdb/chn/chn-cap_integ_all
E.	{5}	Drop all existing channels. \$ hvrload -d hubdb chn
F.	{4}, {5}	Backup the contents of the HVR catalogs. \$ hvrcatalogexport hubdb hvrbackup.xml
G.	{4}, {5}	Drop the contents of the HVR catalogs: Ingres \$ sql hubdb <\$HVR_HOME/sql/ingres/hvrcatalogdrop.sql or Oracle \$ sqlplus user/pwd <\$HVR_HOME/sql/oracle/hvrcatalogdrop.sql
H.	{1}, {2}, {3}, {4}, {5}	Read and uncompress the distribution file. Unix & Linux Under Unix and Linux this is done by: \$ cd \$HVR_HOME \$ mask 0 \$ zcat < /tmp/hvr44444xx.taz tar xf - Windows On Microsoft Windows this is performed using the installation wizard: C:\> d:\hvr44444win32setup.exe

Step	Upgrade Type	Instruction
I.	{2}, {3}, {4}, {5}	Upgrade remote executables on remote locations. This can be done by installing the new distributions on top of the old executables (as step K) or by using files from the hub machine using FTP (if the platform is the same). If HVR GUI is used on a PC to connect to the hub machine then HVR on this PC must also be upgraded. Any running executables (e.g. <code>hvr.exe</code>) must be halted during this refresh.
J.	{1}, {2}, {3}, {4}, {5}	<p>Ingres</p> <p>If HVR must perform log-based capture, it needs a trusted executable to read from the DBMS logging system. Perform the following steps while logged in as ingres:</p> <pre>\$ cd /usr/hvr/hvr_home \$ cp bin/hvr sbin/hvr_ingres \$ chmod 4755 sbin/hvr_ingres</pre> <p>This step is not needed if capture will be trigger-based or if capture will be from another machine.</p> <p>If HVR and the ingres share the same Unix group, then the permissions can be tightened from 4755 to 4750.</p> <p>This step is also needed on remote machines unless this is a type {1} upgrade.</p>
K.	{3}, {4}	<p>Recreate the jobs and job scripts for all channels in the HVR installation:</p> <pre>\$ hvrload -oj hubdb chn</pre>
L.	{4}, {5}	<p>Recreate the HVR catalogs and state tables in the hub database:</p> <p>Ingres</p> <pre>\$ sql hubdb <\${HVR_HOME}/sql/ingres/hvrcatalogcreate.sql</pre> <p>or</p> <p>Oracle</p> <pre>\$ sqlplus user/pwd <\${HVR_HOME}/sql/oracle/hvrcatalogcreate.sql</pre>
M.	{4}, {5}	<p>After adjusting the definition and configuration channel information so it fits the new catalog structure, fill the catalogs.</p> <pre>\$ hvrcatalogimport hubdb hvrbackup.xml</pre>
N.	{1}, {2}, {3}, {4}, {5}	<p>Reload the channels and restart the runtime processes.</p> <pre>\$ hvrscheduler hubdb</pre>

4. After Installing HVR

After installing HVR the operator may need to make changes to the user profiles and to start HVR after a reboot. A special maintenance script can also be installed for housekeeping HVR's scheduler. There may also be changes necessary for how backup and archiving of the [Ingres](#) or [Oracle](#) database.

4.1. Configuring User Profiles

❖ [Unix & Linux](#)

Under Unix and Linux, ensure HVR users on the hub machine must have environment variables `$HVR_HOME` and `$HVR_CONFIG` set and have directory `$HVR_HOME/bin` in their `$PATH`. This can be done by adding lines to their `.profile` files.

❖ [Ingres](#)

For users with trigger-based capture from an Ingres database and HVR action `DbCapture /QuickToggle`, variable `ING_SET` must be defined. This variable should force `readlock=nowlock` on the quick toggle table. Under Unix, Linux or Microsoft Windows this can be done with an environment variable or using `ingsetenv`.

Example:

```
$ ingsetenv ING_SET 'set lockmode on hvr_qtogmychn where readlock=nowlock'
```

4.2. Restarting HVR after Reboot

[Unix & Linux](#)

Under Unix and Linux, after reboot, the HVR service must be restarted. This can be done by configuring file `/etc/hvrtab` and then copying script `hvr.boot` from `$HVR_HOME/lib` into the `init.d` directory.

Each boot filename requires either a start sequence or a stop sequence. The start sequence for the HVR Scheduler processes must be bigger than the DBMS start sequence, otherwise the connection to its hub database will fail. Likewise the stop sequence must be smaller than the DBMS stop sequence. The step below use start sequence 97 and stop sequence 03 (except HP-UX which uses 997 and 003 because it expects 3 digits). Use of `rc.d/K*` script to stop HVR at system shutdown is optional; it will stop anyway!

Each line of `/etc/hvrtab` contains four or more fields, separated by spaces. These fields have the following meaning:

```
username hub_or_port hvr_home hvr_config [options]
```

The first field is the Unix username under which HVR runs. The second field is the name of the hub database (for an HVR Scheduler) or a TCP/IP port (for a HVR Remote Listener). The third and fourth fields are the values of `$HVR_HOME` and `$HVR_CONFIG`. The other fields are optional and are passed to the HVR process as options. For more details see the comment at the start of script `hvr.boot`.

EXAMPLE

This `/etc/hvrtab` file starts two HVR schedulers (one for an Ingres hub and the other for an Oracle hub) and also starts one Remote Listener. Note use of command `hvrencrypt` to encrypt the Oracle password.

```
hvr inghub /opt/hvr/hvr_home /opt/hvr/hvr_config -EII_SYSTEM=/opt/ingres -E
HVR_PUBLIC_PORT=50010
hvr user/!{DszmZY}! /opt/hvr/hvr_home /opt/hvr/hvr_config -EORACLE_HOME=/opt/oracle
-EORACLE_SID=prod
root 50000 /opt/hvr/hvr_home /opt/hvr/hvr_config
```

INSTALLATION

On HP-UX to start and stop HVR for run level 3;

```
$ cp hvr.boot /sbin/init.d/hvr
$ ln -s /sbin/init.d/hvr /sbin/rc3.d/S97hvr
$ ln -s /sbin/init.d/hvr /sbin/rc3.d/K003hvr
```

On Solaris to start and stop HVR for run level 2 (also implies level 3);

```
$ cp hvr.boot /etc/init.d/hvr
$ ln -s /etc/init.d/hvr /etc/rc2.d/S97hvr
$ ln -s /etc/init.d/hvr /etc/rc2.d/K03hvr
```

On AIX to start and stop HVR for run level 2;

```
$ cp hvr.boot /etc/rc.d/init.d/hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc2.d/S97hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc2.d/K03hvr
```

On Linux to start HVR for run levels 3 and 5 and stop for all run levels;

```
$ cp hvr.boot /etc/rc.d/init.d/hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc3.d/S97hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc5.d/S97hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc0.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc1.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc2.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc3.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc4.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc5.d/K03hvr
$ ln -s /etc/rc.d/init.d/hvr /etc/rc.d/rc6.d/K03hvr
```

4.3. Adapting a Channel due to Data Model Changes and DDL

HVR only captures inserts, updates and deletes (known as DML statements) and replicates these to each integrate database. But DDL statements, such as create, drop and alter table to change the data model, or bulk copy (copy from) and truncate table (modify to truncated) are not captured by HVR. When DDL statements are used, the following points must be considered:

- These statements are not replicated by HVR, so they must be applied manually on both the capture and integrate databases.
- The HVR channel which replicates the database must be changed ('adapted') so it contains the new list of tables and columns, and the enroll information contains the correct internal table id number.

There are two ways to adapt a channel:

1. 'Online Adapt'. This is the least disruptive, because during this procedure users can still make changes to all tables.
2. 'Offline Adapt'. This is the most disruptive, because users are not allowed to make changes to any of the replicated tables.

The procedure to be followed depends on which DDL statements are used and whether the tables they affected have a uniqueness constraint. A uniqueness constraint is a table structure such a **unique index** or a **btree unique** which makes it impossible to insert duplicate rows into a given table.

Online Adapt can only be used for a database with log-based capture and if the DDL statements are limited to the following:

- Drop table statements.
- Create table statements for tables with a uniqueness constraint.
- Truncate table or bulk copy (copy from) statements for tables with a uniqueness constraint, but only if no inserts, updates or deletes are done (either in the DDL script or by end-users) between the start of the online adapt steps and these truncate/bulk copy statements .

Offline Adapt must be used for any other DDL statements and for trigger-based capture. This includes alter table and also any DDL affecting a table without a uniqueness constraint.

ONLINE ADAPT STEPS FOR INGRES LOG-BASED CAPTURE

1. Suspend the capture jobs

```
$ hvrsuspend hubdb CHN_NAME-CAP
```
2. Wait until the integrate jobs have finished integrating all the changes (so no transaction files are left in the router directory). Then suspend the integrate jobs.

```
$ hvrsuspend hubdb CHN_NAME-INTEG
```
3. Run the SQL script with the DDL statements against both the source and target databases.
4. Use the HVR GUI **Table Select** connected to the captured database to incorporate the changes in the channel definition.
 - a) Use **Add** to add the tables that are **Only in Database** or **In other Channel**.

- b) Use **Replace** to change the definition of the tables that have **Different Keys**, **Different Columns** or **Different Datatypes**.
- c) Use **Delete** for tables that have **Only in Channel**.
5. Use **Table Select** to the integrate locations to check that all the tables have value **Same** in the **Match** column.
6. Use **hvrrefresh** to synchronize all tables that are affected by DDL statements (except for tables which were only dropped) in the SQL script in step 3. Tables which were only affected by DML statements in this script do not need to be refreshed.

```
$ hvrrefresh -rsource -ttable1 -ttable2 ... hubdb chn_name
```

This is called an Online Refresh. See Online Refresh in section [5.10 hvrrefresh](#), [hvrcompare](#) of the HVR User Manual for more information. An Online Refresh requires integrate resilience (e.g. **/OnErrorSaveFailedRow**) and a unique key on the target table. After an Online Refresh, the integration job will give errors like "duplicate insert" or "failed delete" when it runs for the first time. These errors are expected and will disappear after the first run and the target database will then be consistent.

7. If tables were added or changed (new columns) perform **ckpdb** on all databases to enable journaling.
8. Regenerate the enroll information and the replication jobs & scripts

```
$ hvrload -oej hubdb chn_name
```

9. Unsuspend the capture and integrate jobs:

```
$ hvrsuspend -u hubdb CHN_NAME
```

OFFLINE ADAPT STEPS

1. Start downtime. Make sure that users cannot make changes to any of the replicated tables.
2. Suspend the replication jobs


```
$ hvrsuspend hubdb CHN_NAME
```
3. Unload the channel:


```
$ hvrload -d hubdb chn_name
```
4. Run the SQL script with the DDL statements against both the source and target databases.
5. Use the HVR GUI **Table Select** connected to the captured database to incorporate the changes in the channel definition.
 - a) Use **Add** to add the tables that are **Only in Database** or **In other Channel**.
 - b) Use **Replace** to change the definition of the tables that have **Different Keys**, **Different Columns** or **Different Datatypes**.
 - c) Use **Delete** for tables that have **Only in Channel**.
6. Use **Table Select** to the integrate locations to check that all the tables have value **Same** in the **Match** column.
7. Use **hvrrefresh** to synchronize all the tables that are affected by the SQL script in step 4 (except for the tables which were only dropped). This includes tables that were only affected by DML statements in this script.

```
$ hvrrefresh -r source -ttable1 -ttable2 ... hubdb chn_name
```

The **-t** options can also just be omitted, which will cause all replicated tables to be refreshed.

8. For an Ingres database: If tables were added or changed (new columns) perform **ckpdb** on all databases to enable journaling.

9. Reload the channel:

```
$ hvrload -c hubdb chn_name
```

10. Unsuspend the capture and integrate jobs:

```
$ hvrsuspend -u hubdb CHN_NAME
```

11. Finish downtime.

4.4. Scheduling Regular Maintenance Scripts

Periodically it is necessary to restart the HVR Scheduler, purge its old logfiles and check for errors. The maintenance script **hvrmaint** is a standard Perl script for ‘regular’ nightly or weekly housekeeping of the HVR Scheduler. It can be scheduled on Unix or Linux using **crontab** or on Windows as a **Scheduled Task**. See also section [5.7 hvrmaint](#).

4.5. Checkpointing an Ingres Capture Database

❖ Trigger-based Capture

If trigger-based capture is defined for an Ingres database, HVR uses SQL DDL statement **modify to truncated** to empty the capture tables. The locks taken by this statement conflicts with locks taken by an on-line checkpoint. This can lead to HVR jobs hanging or deadlocking. These problems can be solved by creating file **\$HVR_CONFIG/files/dbname.avoidddl** just before checkpointing database *dbname* and deleting it afterwards. HVR will check for this file, and will avoid DDL when it sees it.

Under Unix, do this as follows:

```
$ touch $HVR_CONFIG/files/mydb.avoidddl
$ sleep 5
$ ckpdb mydb
$ rm $HVR_CONFIG/files/mydb.avoidddl
```

❖ Log-based Capture

If log-based capture is defined for an Ingres database (action **DbCapture /LogBased**) then HVR may need to go back to reading the Ingres journal files. But each site has an existing backup/recovery regime that depends on the Ingres checkpoint and journal files. Command **hvrlogrelease** can purge these checkpoint and journal files only when they are no longer needed by the backup/recovery regime and have also been released by the HVR capture jobs.

4.6. Managing Oracle Archive/Redo Logfiles

HVR can capture changes straight from Oracle's redo and archive files.

If log-based capture is defined for an Oracle database (action [DbCapture/Logbased](#)) then HVR may need to go back to reading the Oracle archive/redo logfiles. But each site has an existing backup/recovery regime that depends on the Oracle archive redo logfiles.

Command [hvrlogrelease](#) can purge these archive redo logfiles only when they are no longer needed by the backup/recovery regime and have also been released by the HVR capture jobs.

5. Command Reference

HVR can be configured and controlled either using the GUI or by using the command line interface. This chapter contains a section describing each HVR command and its parameters. Most HVR commands take a hub database name as first argument. For more conventions see section [1.6 Calling HVR on the Command Line](#).

5.1. hvr runtime engine

NAME

hvr - HVR runtime engine.

SYNOPSIS

hvr [*script* [-*scropts*][*scrargs*]]

hvr -r [-*A*] [-*En=v*]... [-*Cpubcert* -*Kprivkey*] [-*N*] [-*ppamsrv*] [-*User*]...

hvr -s*lbl*

DESCRIPTION

The command **hvr** is an interpreter for HVR's internal script language. These scripts are generated by HVR itself. Inspection of these scripts can improve transparency and assist debugging, but it is unnecessary and unwise to use the internal script language directly because the syntax is liable to change without prior notice between HVR versions.

If no arguments are supplied or the first argument is '-' then input is read from **stdin**. Otherwise script is taken as input. If *script* begins with '.' or '/' it is opened as an absolute pathname, otherwise a search for the **HVR** script is done in the current directory '.' and then in **\$HVR_HOME/script**.

Command **hvr** with option **-r** is used to provide an HVR slave process on a remote machine. Its validation of passwords at connection time is controlled by options **-A**, **-p**, **-N** and **-U**.

OPTIONS

- A** Unix & Linux Remote HVR connections should only authenticate login/password supplied from hub, but should not change from the current Operating System username to that login. This option can be combined with the **-p** option (PAM) if the PAM service recognizes login names which are not known to the Operating System. In that case the **inetd** service should be configured to start the HVR slave as the correct Operating System user (instead of **root**).
- Cpubcert** Network encryption of remote HVR connection using public certificate *pubcert* generated by command **hvrsslgen**. File *pubcert* should be an absolute pathname.
- En=v** Set environment variable *n* to value *v* for this process and its children.
- Kprivkey** Network encryption of remote HVR connection using private key *privkey*. This private key should correspond to the *pubcert* file supplied using option **-C**. File *privkey* should be an absolute pathname.
- N** Do not check passwords for authentication. Disabling password authentication is a security hole, but may be useful as a temporary measure for testing.












- ppamsrv** Use Pluggable Authentication Module *pamsrv* for login password authentication of remote HVR connections. PAM is a service provided by several Operation Systems as an alternative to regular login/password authentication, e.g. checking the `/etc/passwd` file. Often **-plogin** will configure hvr slaves to check passwords in the same way as the operating system. Available PAM services can be found in file `/etc/pam.conf` or directory `/etc/pam.d`.
- r** HVR slave to service remote HVR connections. On Unix the `hvr` executable is invoked with this option by the `inetd` daemon. On Windows `hvr.exe` is invoked with this option by the HVR Remote Listener Service. Remote HVR connections are authenticated using the login/password from column `loc_remote_login` and `loc_remote_pwd` from HVR catalog `hvr_location`.
- slbl** Internal slave co-process. HVR sometimes uses slave co-processes internally to connect to database locations. Value `lbl` has no effect other than to appear next to the process id in the process table (e.g. from `ps -ef`) so that operators can distinguish between slave processes.
- User** Limits the HVR slave so it only accepts connections which are able to supply Operating System password for account *user*. This reduces the number of passwords that must be kept secret. Multiple **-U** options can be supplied.









EXAMPLE

To run hvr script `foo` with arguments `-x` and `bar` and to redirect `stdout` and `stderr` to file `log`:

```
$ hvr foo -x bar >log 2>&1
```

FILES

 HVR_HOME		
 bin		
	hvr	HVR executable (Unix and Linux).
	hvr.exe	HVR executable (Microsoft Windows).
	hvr_iiN.dll	Ingres version N shared library (Microsoft Windows).
	hvr_orN.dll	Oracle version N shared library (Microsoft Windows).
 lib		
	hvr_iiN.sl or .so	Ingres shared library (Unix and Linux).
	hvr_orN.sl or .so	Oracle shared library (Unix and Linux).
	hvrvalidpw	Optional executable for password validation. If HVR detects this executable, it invokes it for password validation instead of attempting this validation internally.
	hvrscripthelp.html	Description of HVR's internal script syntax and procedures.

 HVR_CONFIG		
 job		HVR scripts generated by hvrload.
 files		
	<i>dbname.avoidddl</i>	File indicating HVR should avoid using DDL (modify to truncated) during trigger-based capture from Ingres database dbname.
	<i>[hubnode]-hub-chn-loc.logrelease</i>	Status of HVR log-based capture jobs, for command hvrlogrelease.
 work		
 <i>hubnode-hubdb</i>		
	<i>store*.tmp</i>	Temporary files created during large sorts and when transporting large long datatype values.

SEE ALSO

Command [5.4 hvrgui](#).

5.2. hvrcatalogexport, hvrcatalogimport

NAME

hvrcatalogexport - Export from hub database into HVR catalog document.

hvrcatalogimport - Import from HVR catalog document into hub database.

SYNOPSIS

hvrcatalogexport [-*cchn...*] [-C] [-d] [-g] [-*hclass*] [-l] [-*uuser*] *hubdb catdoc*

hvrcatalogimport [-*hclass*] [-*uuser*] *hubdb catdoc*

DESCRIPTION

Command **hvrcatalogexport** extracts information from the HVR catalog tables in the hub database and writes it into file *catdoc*. HVR catalog document *catdoc* is an XML file which follows the HVR catalog Document Type Definition (DTD).

Command **hvrcatalogimport** loads the information from the supplied HVR catalog document into the HVR catalogs in the hub database.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

An HVR catalog document file can either be created using command **hvrcatalogexport** or it can be prepared manually, provided it conforms to the HVR catalog DTD.

OPTIONS

- cchn* Only export catalog information for channel *chn*.
- C Only export information from configuration action catalog **hvr_config_action**.
- d Only export information from channel definition catalogs **hvr_channel**, **hvr_table**, **hvr_column**, **hvr_loc_group** and **hvr_action**.
- g Only extract information from group membership catalog **hvr_loc_group_member**.
- hclass* Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).
- l Only export information from location catalog **hvr_location**.
- uuser* Connect to hub database using Ingres DBMS account *user*.
Ingres
- uuser/pwd* Connect to Microsoft SQL Server hub database as *user* with password *pwd*.
SQL Server

HVR CATALOG DTD

HVR catalog documents are XML files which must conform to the HVR catalog Document Type Definition (DTD). A formal specification of this DTD can be found in file [HVR_HOME/lib/hvr_catalog.dtd](#). Most XML tags in the DTD are directly equivalent to a table or row of the HVR catalog tables (see also section [7.1 Catalog Tables](#)).

The root tag of the HVR catalog DTD is `<hvr_catalog>`. This root tag contains “table” tags named `<hvr_channels>`, `<hvr_tables>`, `<hvr_columns>`, `<hvr_loc_groups>`, `<hvr_actions>`, `<hvr_locations>`, `<hvr_loc_group_members>` and `<hvr_config_actions>`. Most table tags contain a special optional attribute `chn_name`. This special attribute controls the amount of data that is deleted and replaced as the HVR catalog document is loaded into the catalog tables. For example, a document that contains `<hvr_actions chn_name="hvr_demo01">` would imply that only rows for channel `hvr_demo01` should be deleted when the document is imported. If the special attribute `chn_name` is omitted then all rows for that catalog table are deleted.

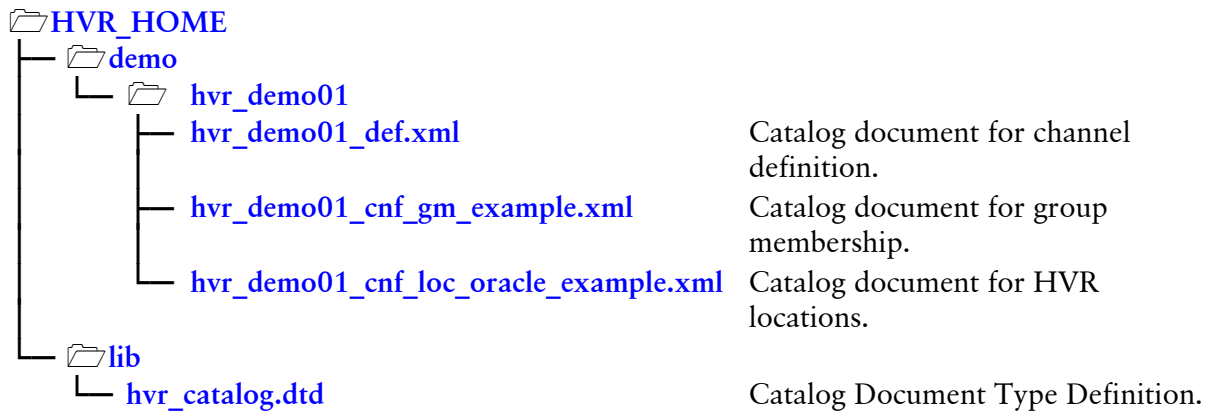
Each table tag contains tags which correspond to rows in the catalog tables. These ‘row’ tags are named `<hvr_channel>`, `<hvr_table>`, `<hvr_column>`, `<hvr_loc_group>`, `<hvr_action>`, `<hvr_location>`, `<hvr_loc_group_member>` and `<hvr_config_action>`. Each of these row tags has an attribute for each column of the table. For example, tag `<hvr_tables>` could contain many `<hvr_table>` tags, which would each have attributes `chn_name`, `tbl_name` and `tbl_base_name`.

Some attributes of a row tag are optional. For example, if attribute `col_key` of `<hvr_column>` is omitted it defaults to 0 (false), and if attribute `tbl_name` of tag `<hvr_action>` is omitted then it defaults to “*” (affect all tables).

EXAMPLE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hvr_catalog SYSTEM "http://www.hvr.biz/dtd/1.0/hvr_catalog.dtd">
<hvr_catalog version="1.0">
  <hvr_channels chn_name="hvr_demo01">
    <hvr_channel chn_name="hvr_demo01" chn_description="Simple reference channel."/>
  </hvr_channels>
  <hvr_tables chn_name="hvr_demo01">
    <hvr_table chn_name="hvr_demo01" tbl_name="dm01_order" tbl_base_name="dm01_order"/>
    <hvr_table chn_name="hvr_demo01" tbl_name="dm01_product" tbl_base_name="dm01_product"/>
  </hvr_tables>
  <hvr_columns chn_name="hvr_demo01">
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_order" col_sequence="1"
      col_name="prod_id" col_key="1" col_datatype="integer4"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_order" col_sequence="2"
      col_name="ord_id" col_key="1" col_datatype="integer4"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_order" col_sequence="3"
      col_name="cust_name" col_datatype="varchar" col_length="100"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_order" col_sequence="4"
      col_name="cust_addr" col_datatype="varchar" col_length="100" col_nullable="1"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_product" col_sequence="1"
      col_name="prod_id" col_key="1" col_datatype="integer4"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_product" col_sequence="2"
      col_name="prod_price" col_datatype="float8"/>
    <hvr_column chn_name="hvr_demo01" tbl_name="dm01_product" col_sequence="3"
      col_name="prod_descrip" col_datatype="varchar" col_length="100"/>
  </hvr_columns>
  <hvr_loc_groups chn_name="hvr_demo01">
    <hvr_loc_group chn_name="hvr_demo01" grp_name="CEN" grp_description="Headquarters"/>
    <hvr_loc_group chn_name="hvr_demo01" grp_name="DECEN" grp_description="Decentral"/>
  </hvr_loc_groups>
  <hvr_actions chn_name="hvr_demo01">
    <hvr_action chn_name="hvr_demo01" grp_name="CEN" act_name="DbCapture"/>
    <hvr_action chn_name="hvr_demo01" grp_name="DECEN" act_name="DbIntegrate"/>
  </hvr_actions>
</hvr_catalog>
```

FILES



5.3. hvrcrypt, hvrcryptdb

NAME

hvrcrypt, **hvrcryptdb** - Encrypt passwords for databases and remote connections.

SYNOPSIS

hvrcrypt *key* *pwd*

hvrcryptdb [*options*] *hubdb*

DESCRIPTION

Command **hvrcrypt** can be used to interactively encrypt a password for a hub database when starting HVR on the command line. It is not needed for commands started with the HVR GUI.

Command **hvrcryptdb** will encrypt all unencrypted passwords in column **loc_remote_pwd** and **loc_db_user** in catalog **hvr_location** of the hub database, using column **loc_name** as key. Passwords entered using the HVR GUI will already be encrypted.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

Passwords are encrypted using an encryption key. Each password is encrypted using a different encryption key, so that if two passwords are identical they will be encrypted to a different value. The encryption key used for hub database passwords is the name of the hub database, whereas the key used to encrypt the login passwords and database passwords for HVR locations is the HVR location name. This means that if an HVR location is renamed, the encrypted password becomes invalid.

Regardless of whether **hvrcrypt** and **hvrcryptdb** are used, **hvrload** will always generate job scripts containing encrypted passwords (it will encrypt any passwords not yet encoded in the catalogs). The passwords will be sent encoded over the network, and will only be decrypted during authorization checking on the remote location.

OPTIONS

-hclass Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).

-uuser Connect to hub database using Ingres DBMS account *user*.

Ingres

-uuser/pwd Connect to Microsoft SQL Server hub database as *user* with password *pwd*.

SQL Server

ENCRYPTING COMMAND LINE PASSWORDS

HVR commands **hvrload** and **hvrscheduler** may require that a DBMS password has a command line argument. These passwords can be provided in encrypted form using command **hvrcrypt**.

For example, the following steps can be used to start the HVR Scheduler at reboot:

```

Unix & Linux
$ DBUSER=hvrhubaw
$ DBPWD=mypassword
$ DBPWD_CRYPT= `hvrcrypt $DBUSER $DBPWD`
$ hvrscheduler $DBUSER/$DBPWD_CRYPT

```

Use of Unix command `ps|grep hvrscheduler` will give the following:

```
hvr 21852 17136 0 15:50:59 pts/tf 00:03 hvrscheduler -i  
hvrhubaw/!{CLCIfCSy6Z7AUUya}!
```

The above techniques also work for the hub database name supplied to `hvrload`.

NOTES

Although the password encryption algorithm is reversible, there is deliberately no decryption command supplied.

Commands `hvincrypt` and `hvincryptdb` are intended only for password encryption. Secure network encryption of remote HVR connections is provided using command `hvrsslgen` and action `LocationProperties /SslCertificate`.

SEE ALSO

Commands [5.4 hvrgui](#) and [5.15 hvrsslgen](#).

Parameter `/SslCertificate` in section [6.13 LocationProperties](#).

5.4. hvrgui

NAME

HVR GUI - HVR Graphical User Interface.

SYNOPSIS

hvrgui

DESCRIPTION

HVR GUI is a Graphical User Interface used to configure replication. The GUI can just be run on the hub machine, but it can also run on the user's PC and connect to a remote hub machine. To start the GUI double-click on its Windows shortcut or execute command **hvr**gui on Linux. HVR GUI does not run on Unix machines; instead it must connect from the user's PC to such a hub machine.

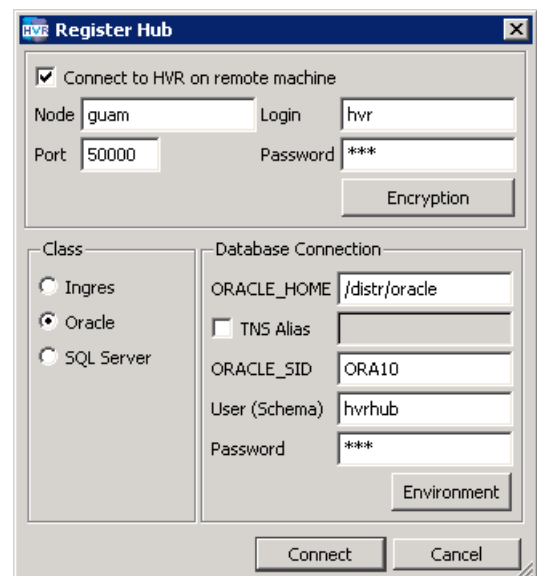
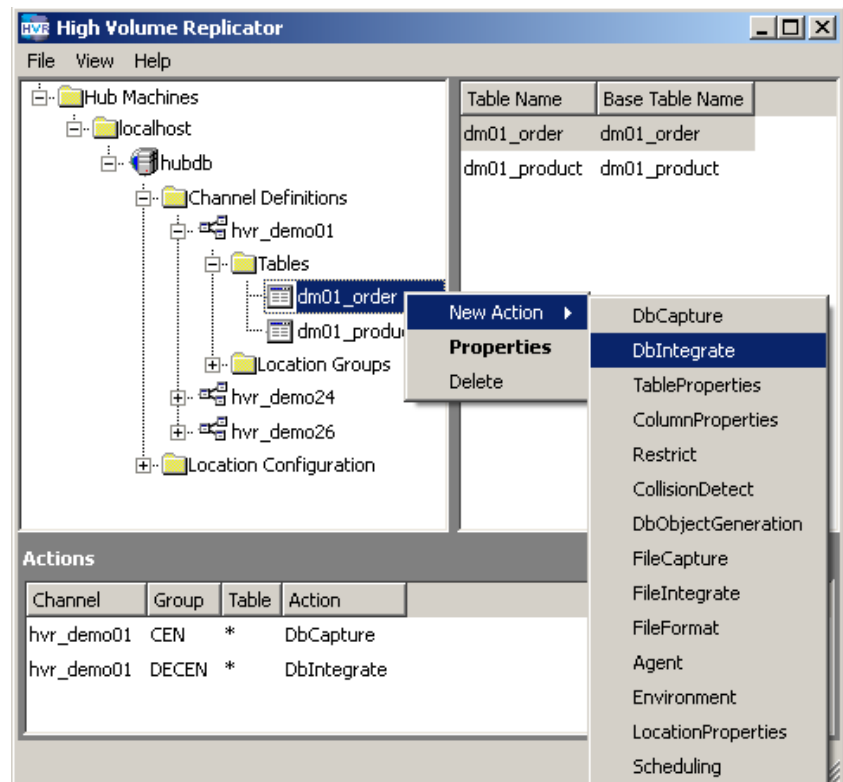
At startup the Register Hub dialog window

requires you to input data for connecting to the hub database. The main window consists of four panes. The top-left pane contains a treeview with channel definitions and location configuration. The top-right pane displays details from the node selected in the treeview. The Actions pane lists the actions configured for the channel selected in the treeview. The bottom pane provides logfile information and error information.

The first window that appears is Register Hub (displayed left). This provides the connection with the hub database.

After registering a hub, you can see folders for Channel Definitions and Location Configuration. A right-click on these (or on the actual channels, locations, etc. under it) reveals a menu that allows you to do things like:

- Create a new instance of that object
- Perform commands like an HVR Load, Export, Import
- Create an Action for that object



NOTE

In the HVR GUI you can change an action that is defined for all tables (**Table="*"**) with a set of actions (each for a specific table) as follows. Right-click in the action's **Table field** > **Expand**. You can expand actions on any field that has a "*" value.

5.5. hvrload

NAME

hvrload - Load a replication channel.

SYNOPSIS

hvrload [-options] *hubdb chn*

DESCRIPTION

hvrload encapsulates all steps required to generate and load the various objects needed to enable replication of channel *chn*. These objects include replication jobs and scripts as well as database triggers/rules for trigger-based capture and table enrollment information for log-based capture.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

OPTIONS

- c** Create objects only. With neither this flag nor flag **-d**, **hvrload** will attempt to drop and recreate some objects (such as database triggers and scripts) but will leave other objects unchanged (such as transaction files containing data and Scheduler job groups).
- d** Drop objects only. With neither this flag nor flag **-c**, **hvrload** will attempt to drop and recreate some objects (such as database triggers and scripts) but will leave other objects unchanged (such as transaction files containing data and Scheduler job groups).
- hclass** Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).
- itime** Capture rewind. Initialize channel to start capturing changes from a specific time in the past, rather than only changes made from the moment the **hvrload** command is run. The parameter must have form **YYYY-MM-DD HH:MM:SS**, or a negative number of seconds (relative to the current date and time) or **min** (which means '1970-01-01 00:00:00' UTC). Capture rewind is only supported for database log-based capture using parameter **/LogBased** (not for trigger-based capture) and for capture from file locations when parameter **/DeleteAfterCapture** is not defined.
- lx** Only affect objects for locations specified by *x*. Values of *x* may be one of the following:
 - loc* Only affect location *loc*.
 - l1-l2* Affects all locations that fall alphabetically between *l1* and *l2* inclusive.
 - !*loc*** Affect all locations except *loc*.
 - !*l1-l2*** Affects all locations except for those that fall alphabetically between *l1* and *l2* inclusive.
 Several **-lx** instructions can be supplied together to **hvrload**.
- oS** Operations limited to objects indicated by *S*. Values of *S* may be:
 - s** State tables in database locations, e.g. the toggle table.

c	Tables containing changes, e.g. capture, history and fail tables.
t	Database triggers/rules for tables with trigger-based capture.
p	Database procedures in database locations.
r	Router directory's transaction files and capture timestamps.
e	Enroll information for tables with log-based capture.
l	Supplemental logging for log-based capture tables.
j	Job scripts and scheduler jobs and job groups.
f	Files inside file location's state directory.

Several **-oS** instructions can be supplied together (e.g. **-octp**) which causes **hvrload** to effect all object types indicated. Not specifying a **-o** option implies all objects are affected (**-sctpreljf**).

- pN** Indicates that SQL for database locations should be performed using *N* subprocesses running in parallel. Output lines from each subprocess is preceded by a symbol indicating the corresponding location. This option cannot be used with option **-S**.
- Ro** Resilience of SQL statements. Values of *o* may be one of the following:
- | | |
|----------|---|
| a | No failures of SQL statements are considered fatal. |
| d | Failed drop statements are not considered fatal, but other failed statements are fatal. The default. |
| n | Any failed SQL statement is fatal and causes the script to exit. |
- S** Write SQL to **stdout** instead of applying it to database locations. This can either be used for debugging or as a way of generating a first version of an SQL include file (see action **DbObjectGeneration /IncludeSqlFile**), which can later be customized. This option is often used with options **-lloc -otp**.
- ty** Only affect objects referring to tables specified by *y*. Values of *y* may be one of the following:
- | | |
|---------------|---|
| <i>tbl</i> | Only affects table <i>tbl</i> . |
| <i>t1-t2</i> | Affects all tables that fall alphabetically between <i>t1</i> and <i>t2</i> inclusive. |
| !tbl | Affects all tables except <i>tbl</i> . |
| !t1-t2 | Affects all tables except for those that fall alphabetically between <i>t1</i> and <i>t2</i> inclusive. |

Several **-ty** instructions can be supplied together to **hvrload**.

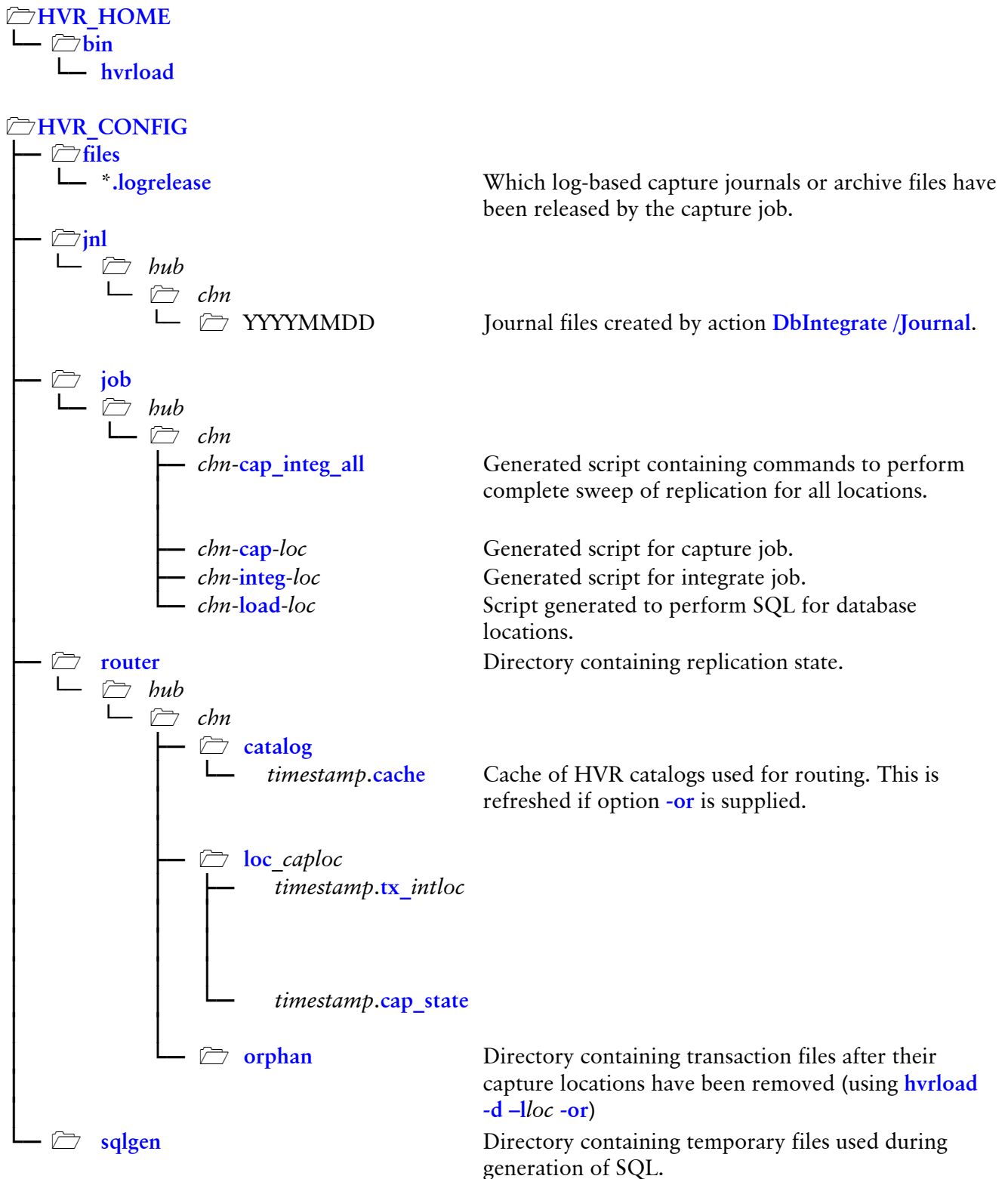
- uuser** Connect to hub database using Ingres DBMS account *user*.

Ingres

- uuser/pwd** Connect to Microsoft SQL Server hub database as *user* with password *pwd*.

SQL Server

FILES



SEE ALSO

Commands [5.3 hvrcrypt](#), [hvrcryptdb](#), [5.10 hvrrefresh](#), [hvrcompare](#), [5.14 hvrscheduler](#).

5.6. hvrlogrelease

NAME

hvrlogrelease – Manage DBMS logging files when not needed by log-based capture.

SYNOPSIS

hvrlogrelease [*optfile*] [-*options*]

DESCRIPTION

hvrlogrelease can be used to manage DBMS logging files (Ingres journals and checkpoints or Oracle archive redo logs) when they are no longer needed by HVR log-based capture. The default behavior is just to list the files which are not needed, but options **-purge** or **-compress** can be specified to delete or compress them.

It relies on ‘log release’ files which created by log-based capture jobs. These files are created in **\$HVR_CONFIG/files** by HVR log-based capture jobs and contain timestamps which allows **hvrlogrelease** to safely purge or compress DBMS logging files and backups without affecting HVR replication.

Either option **-ingres_db** or **-oracle_sid** must be specified.

Note that **hvrlogrelease** must be scheduled to run under the DBMS owner’s login (e.g. **ingres** or **oracle**), whereas **hvrmaint** must run under the HVR’s login.

OPTIONS

- ingres_db=db** Only check ‘log release’ files for Ingres database *db*. If value *db* contains an asterisk (*) then all Ingres databases are matched. This option can be supplied several times for different databases. Note that **hvrlogrelease** extracts the path **\$II_SYSTEM** from matching log release files, so it could affect journals files which are not in the current Ingres installation.
- oracle_sid=sid** Only check for ‘log release’ files for Oracle instance *sid*. If value *sid* contains an asterisk (*) then all Oracle instance are matched. This option can be supplied several times for different instances. Note that **hvrlogrelease** extracts the value of **\$ORACLE_HOME** from matching log release files, so it could affect archived redo files which are not in the current Oracle installation.
- purge** Purge (i.e. delete) old DBMS logging files (Ingres journals and Oracle archived redo logfiles) and backup files (e.g. Ingres checkpoints) once the ‘log release’ files indicate that they are no longer needed for any HVR replication. Options **-backup_num** or **-backup_days** further delay purging until the files are no longer needed for DBMS recovery. If the ‘log release’ file is absent or unchanged then nothing is purged. This means that if replication is not running then operators must purge journals/archived redo files manually. If there are multiple channels which capture from the same database then **hvrlogrelease** must not be run until all these capture jobs have run once.

- compress**=*cmd* Compress old DBMS logging files (Ingres journals or Oracle archived redo logfiles) as soon as 'log release' files indicate that they are no longer needed for HVR replication. Pattern *%f* contained in *cmd* is substituted with the filename to be compressed (e.g. **compress -f %f**). The compressed files are kept in the same directory. If they are needed again for DBMS recovery they must be uncompressed manually.
- logrelease_expire**=*Nunits* Value *units* can be **days**, **hours** or **minutes**.
- backup_num**=*N* Do not purge the most recent *N* database backups (e.g. Ingres checkpoints) and their corresponding logging files (Ingres journals or Oracle archive redo-logs), even if they are no longer needed by HVR log-based replication.
- backup_age**=*N units* Do not purge database backups (e.g. Ingres checkpoints) and their corresponding logging files (Ingres journals or Oracle archive redo-logs) until they are older than *N* units, even if they are no longer needed by HVR log-based replication. Value *units* can be **days**, **hours** or **minutes**.
- env**=*NAME=VALUE* Set environment variable. This option can be repeated to set multiple variables such as **\$HVR_HOME**, **\$II_SYSTEM**, **\$ORACLE_HOME** etc.
- hvr_config**=*dir* Check for 'log release' files in this **\$HVR_CONFIG** directory. This option must be supplied at least once. It can also be supplied several times if multiple HVR installations capture log changes from a single database. **hvrlogrelease** will then purge DBMS logging files only after they have been released by all the HVR installations. If value *dir* contains an asterisk (*) then all matching directories are searched.
- output**=*fil* Append **hvrlogrelease** output to file *fil*. If this option is not supplied then output is sent to **stdout**. Output can also be sent to an operator using option **-email**.
- email**=*addr1[;addr2]* Email output from **hvrlogrelease** output to *addr1* [and *addr2*]. Requires either option **-smtp_server** or option **-mailer**. Multiple email addresses can be specified, either using a single **-email** option with values separated by a semicolon or using multiple **-email** options.
- email_only_errors** Only send an email if **hvrlogrelease** encounters an error.
- email_activity** Only send an email if **hvrlogrelease** encounters an error or purged or compressed any file.
- smtp_server**=*server* SMTP server to use when sending email. Value *server* can be either a node name or IP address. Requires option **-email**.
- email_from**=*from* Specify a *from* address in email header.
- mailer**=*cmd* Mailer command to use for sending emails, instead of sending them via an SMTP server. Requires option **-email**. String *%s* contained in *cmd* is replaced by the email subject and string *%a* is replaced by the intended recipients of the email. The body of the email is piped to *cmd* as **stdin**.

-backup_command=*cmd* Invoke command *cmd*, for example to make a new backup or checkpoint of a capture or hub database. This option can be supplied several times for multiple backup commands.

FILES

```

└─ HVR_CONFIG
  └─ files
    └─ hubdb
      └─ [node-]chn-loc.logrelease

```

Log release file, containing the time. Created by log-based capture jobs and used by **hvrlogrelease**.

EXAMPLES

Suppose an Ingres database **mydb** is owned by user **dba** and was backed up each night with command **ckpdb** and options **+j** (enable journaling) and **-d** (delete previous checkpoint and journals). That command could delete journal files before HVR log-based capture had finished reading from them, so it should be replaced by a call **hvrlogrelease** each night which runs as **ingres** and uses the following option file:

```

-env=HVR_HOME=/opt/hvr410/hvr_home
-hvr_config=/opt/hvr410/hvr_config
-backup_command=ckpdb +j -udba mydb      # Removed -d option
-ingres_db=mydb
-purge
-backup_num=1                            # Keep one set of checkpoints
-email=bob@mycorp.com

```

Suppose that other Ingres databases **hisdb** and **herdb** do not need regular backup but get very frequent changes which are captured from several HVR channels in several HVR installations. Journaling is enabled once and **hvrlogrelease** is called every hour to purge journal files which are no longer needed by any HVR log-based capture. This prevents the journaling disk filling up. The following option file is used:

```

-env=HVR_HOME=/opt/hvr410/hvr_home
-hvr_config=/opt/*/hvr_config
-ingres_db=hisdb
-ingres_db=herdb
-purge

```

List any Oracle archive redo log-files in instance **ORA1020** which are no longer needed by HVR log-based capture and which contains changes older than 2 days:

```

-env=HVR_HOME=/opt/hvr410/hvr_home
-hvr_config=/opt/hvr410/hvr_config
-oracle_sid=ORA1020
-backup_age=2days

```

5.7. hvrmaint

NAME

hvrmaint - Housekeeping script for HVR jobs

SYNOPSIS

hvrmaint [*optfile*] [-*options*]

DESCRIPTION

Command **hvrmaint** is a standard Perl script for regular housekeeping of the HVR. The behavior of **hvrmaint** is controlled by its options. These can be supplied on its command line, or they can be bundled into a file *optfile*, which must be its first argument.

One way to use **hvrmaint** is to schedule it on its own with options **-stop** and **-start**, perhaps nightly or each weekend. These options instruct **hvrmaint** to restart the HVR Scheduler. Often other options would be supplied such as **-scan_hvr_out**, (scan log files for HVR errors) or **-archive_dir** (move old log files to archive directory). Option **-email** can be used to send an email to operator(s) which summarizes the status. When used in this way **hvrmaint** could be scheduled on Unix using **crontab**, and on Windows as a Windows Scheduled Task.

A second way to use **hvrmaint** is to run it frequently (perhaps every half hour) with options **-ping**, **-check_logfile_growth** and **-scan_hvr_out**. These options check that the HVR Scheduler is responding, that its logfile is growing and that no errors have been written. Running **hvrmaint** this way does not interrupt the HVR Scheduler. Option **-email_only_errors** can be supplied so that emails are only sent if a problem is found.

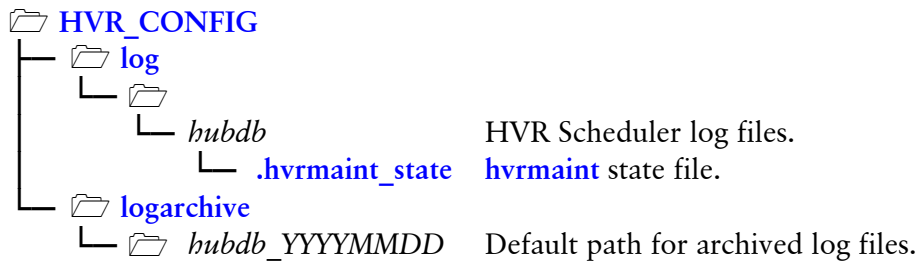
The last way to use **hvrmaint** is as part of a larger nightly or weekly batch script, perhaps a script which halts all server processes (including the DBMS) does a system backup and then restarts everything again. In this case **hvrmaint** would be called at the top of the batch script with option **-stop** (stop the HVR Scheduler) and would then be called again near the bottom with option **-start** (restart the HVR Scheduler).

OPTIONS

-hub = <i>hub</i>	Hub database for HVR Scheduler. This value has form <i>inghub</i> (for an Ingres hub database), <i>user/pwd</i> (for an Oracle hub) or <i>hub</i> for a (SQL Server hub database). For Oracle, passwords can be encrypted using command hvcrypt .
-stop	Stops HVR Scheduler.
-start	Starts HVR Scheduler.
-ping	Check that HVR Scheduler is actually running using command hvrping . If option -stop is also defined then this 'ping' is performed before the HVR Scheduler is stopped. If option -start is supplied then hvrmaint always checks that the HVR Scheduler is running using a 'ping', regardless of whether or not option -ping is defined.
-sched_option = <i>schedopt</i>	Extra startup parameters to the HVR Scheduler server, for example to pass hvr scheduler the options -h (hub class) or -u (username).
-quiesce_grace = <i>secs</i>	If jobs are still running when the HVR Scheduler must stop, allow secs seconds grace before killing them. The default is 60 seconds. This parameter is passed the HVR Scheduler using the -q option.

-env = <i>NAME=VALUE</i>	Set environment variable. This option can be repeated to set multiple variables such as \$HVR_HOME , \$HVR_CONFIG , \$HII_SYSTEM , \$ORACLE_HOME etc...
-output = <i>fil</i>	Append hvrmaint output to file <i>fil</i> . If this option is not supplied then output is sent to stdout . Output can also be sent to an operator using option -email .
-scan_hvr_out	Scan Scheduler log file hvr.out . Command hvrmaint writes a summary of HVR errors detected in this file to its output and to any emails that it sends.
-check_logfile_growth	Check that logfile hvr.out has grown in size since the last time hvrmaint was run. If this file has not grown than an error message will be written.
-archive_files = <i>patt</i>	Move any files in directory \$HVR_CONFIG/log/hub which pattern <i>patt</i> to the archive directory (see option -archive_dir below). Files which are not matched by <i>patt</i> are deleted. The option -stop must be supplied.
-archive_dir = <i>archdir</i>	Archive directory name. Files matched by option -archive_files are moved to <i>archdir/hubdb_YYYYMMDD</i> where <i>YYYY</i> is the year, <i>MM</i> the month number and <i>DD</i> the day of the month. The default value for <i>archdir</i> is \$HVR_CONFIG/logarchive .
-archive_keep_days = <i>N</i>	Removed files in archive directory after <i>N</i> days. Requires option -archive_files . If this option is not specified, then archived files are kept indefinitely.
-archive_compress = <i>cmd</i>	Compress HVR scheduler log files while moving them to the archive directory. Pattern %f contained in <i>cmd</i> is substituted with the filename to be compressed (e.g. compress -f %f)
-email = <i>addr1[;addr2]</i>	Email output from hvrmaint output to <i>addr1</i> [and <i>addr2</i>]. Requires either option -smtp_server or option -mailer . Multiple email addresses can be specified, either using a single -email option with values separated by a semicolon or using multiple -email options.
-email_only_errors	Only send an email if hvrmaint encountered an error itself or detected an HVR error while scanning hvr.out .
-error_limit = <i>N</i>	Limit the number of HVR errors reported to <i>N</i> . This option prevents the generated emails becoming too large. Default is 1000.
-email_from = <i>addr</i>	Specify a sender address in email header.
-smtp_server = <i>server</i>	SMTP server to use when sending email. Value <i>server</i> can be either a node name or IP address. Requires option -email .
-mailer = <i>cmd</i>	Mailer command to use for sending emails, instead of sending them via an SMTP server. Requires option -email . String %s contained in <i>cmd</i> is replaced by the email subject and string %a is replaced by the intended recipients of the email. The body of the email is piped to <i>cmd</i> as stdin .

FILES



EXAMPLES

Unix & Linux

On Unix **hvrmaint** could be scheduled using **crontab** to restart the HVR Scheduler each night at 9 o'clock. The environment for such batch programs is very limited, so many **-env** options are needed to pass it sufficient environment variables. Output is redirected to a log file using Unix output redirection (although an **-output** option could also have been used).

The following option file (**/opt/hvrmaint.opt**) is prepared; only most important (**-stop** and **-start**) are kept on the command line (for extra prominence).

```

-hub=hvr/!{s8Dhx./gsuWHUt}! # Encrypted Oracle password
-env=HVR_HOME=/usr/hvr/hvr_home
-env=HVR_CONFIG=/usr/hvr/hvr_config
-env=ORACLE_HOME=/distr/oracle/OraHome817
-env=ORACLE_SID=ORA817
-scan_hvr_out
-archive_files=hvr.out # Only archive log file hvr.out
-archive_compress="compress -f %f"
-archive_keep_days=14 # Delete files after 2 weeks
-mailer=/bin/mailx -s%s %a # Mailer suitable for HP-UX
-email=bob@mycorp.com;jim@mycorp.com

```

The following line is added to **crontab** for user **hvr** (this should be a single line, without wrapping):

```

0 21 * * * sh -c /usr/hvr/hvr_home/bin/hvrmaint /opt/hvrmaint.opt -stop -start
>>/tmp/hvrmaint.out 2>&1

```

Alternatively the following line could be added to **crontab** for **root**:

```

0 21 * * * su hvr -c /opt/hvr_home/bin/hvrmaint /opt/hvrmaint.opt -stop -start
>>/tmp/hvrmaint.out 2>&1

```

Instead of scheduling **hvrmaint** on its own, it could also be used as part of a larger nightly batch script run by **root** which halts the HVR Scheduler and DBMS before doing a system backup. This batch script would look like roughly this:

```

su hvr -c /opt/hvr_home/bin/hvrmaint /opt/hvrmaint.opt -stop -scan_hvr_out
-archive_files=hvr.out

su ingres -c /opt/ingres/utility/ingstop # Stop DBMS
backup -f/dev/rmt/0m # Perform system backup

su ingres -c /opt/ingres/utility/ingstart # Restart DBMS

su hvr -c /opt/hvr_home/bin/hvrmaint /opt/hvrmaint.opt -start

```

Windows

On Windows, **hvrmaint** can be run as a Windows Scheduled Task.

The configuration steps are as follows:

1. Click **Start > Settings > Control Panel > Scheduled Tasks**.
2. Create a new Scheduled Task by clicking **File > New > Scheduled Task**.
3. Right click on this new task for Properties.
4. On the Tasks tab enter the hvrmaint command line in the Run field, for example:

```
c:\hvr_home\bin\hvrmaint c:\hvrmaint.opt -stop -start
```
5. On the **Schedule** tab, configure when the **hvrmaint** script should run.
6. When ready click the **OK** button.
7. A dialog now appears requesting Windows account information (username and passwords). Enter this information as requested.

A sample option file for Windows could be:

```
-hub=hvr/!{s8Dhx./gsuWHUt}!      # Encrypted Oracle password
-output=c:\hvr_home\hvrmaint.log
-scan_hvr_out
-archive_files=.                  # Archive all log files
-archive_keep_days=14             # Delete files after 2 weeks
-smtp_server=bambi.mycorp.com
-email=bob@mycorp.com;jim@mycorp.com
```

SAMPLE OUTPUT

```
From: root@bambi.mycorp.com
To: bob@mycorp.com; jim@mycorp.com
Subject: hvrmaint detected 7 errors (323 rows in fail tables) for hub hvr/ on bambi

Feb 9 21:00:01 hvrmaint: Starting hvrmaint c:\tools\hvrmaint.opt -hub=hvr/ -stop -start
Feb 9 21:10:21 hvrmaint: Stopping HVR Scheduler 4.0.0 (win/5).
Feb 9 21:10:33 hvrmaint: Scanning d:\hvr_config\log\hvr\hvr.out (Feb 08 21:00:03).
Feb 9 21:11:13 hvrmaint: 7 errors (323 rows in fail tables) were detected during scan.
Feb 9 21:12:33 hvrmaint: 3 capture jobs for 1 location did 606 cycles.
Feb 9 21:12:59 hvrmaint: 6 integrate jobs for 2 locations did 400 cycles and integrated 50
changes for 3 tables.
Feb 9 21:13:53 hvrmaint: Archiving 9 log files to d:\hvr\archivelog\hvr_20050209.
Feb 9 21:16:23 hvrmaint: Purging 0 archive directories older than 14 days.
Feb 9 21:18:29 hvrmaint: Starting HVR Scheduler 4.0.0 (win/5).

----- Summary of errors detected during scan-----

F_JD1034_RAISE_ERROR_P3 occurred 6 times between Feb 09 19:43:52 and Feb 09 20:14:24
F_JJ106E_TIMEO_DB occurred 1 time at Feb 08 21:10:03

----- Errors detected during scan-----
Feb 9 21:10:03: hvr_scheduler: F_JJ106E_TIMEO_DB: Wait for response from database
slave exceeded the 600 seconds defined by 'timeo_db' attribute. Waiting...
Feb 9 19:43:52: channel-cap-d01[3210]: F_JD1034_RAISE_ERROR_P3: Error as raised by user
during pl/sql procedure statement on Oracle SID.
----- End of errors detected during scan -----
Feb 9 21:19:01 hvrmaint: Sending e-mail to bob@mycorp.com; jim@mycorp.com
```

5.8. hvrmonitor

NAME

HVR Monitor - Monitor for HVR Scheduler.

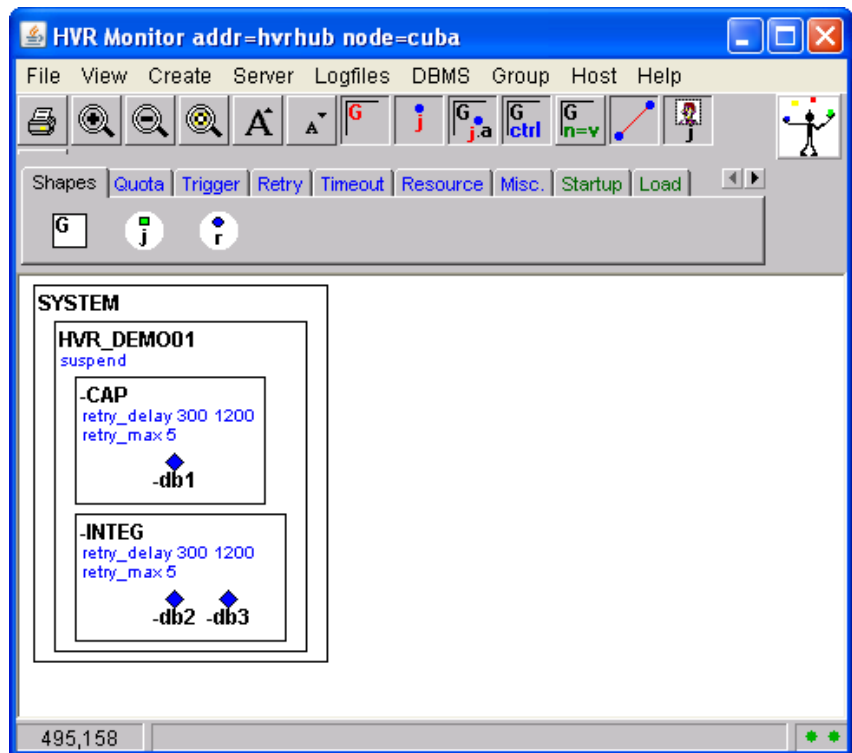
SYNOPSIS

http://node:8080/hvr_home/www/

DESCRIPTION

HVR Monitor is a JAVA based Graphic User Interface which provides an animated view of replication jobs.

HVR Monitor requires an HTTP daemon or service (known as an HTTPD) to be running on the hub machine. This HTTPD is used at startup time to send the HVR Monitor JAVA applet to the user's internet browser. The HVR Monitor then connects directly to the HVR Scheduler, but also continues to



connect and send requests to the HTTPD for various menu options that the user can request. A common choice is Apache (see www.apache.org), although any HTTP software can be configured for the HVR Monitor.

HVR Monitor graphically displays which replications jobs are running, idle or require attention. It serves as an interface between a user and the HVR Scheduler. HVR Scheduler can nonetheless also be used without HVR Monitor, relying instead on command line instructions for the HVR Scheduler, and monitoring jobs via HVR log files and the job catalog tables.

HVR MONITOR SESSION ACCESS

The HVR Monitor's default access regime is as follows:

- ❖ Read-only access to the HVR Monitor's HTTP daemon is unrestricted. This includes showing the state of running jobs and links, showing logfiles, etc.
- ❖ Read-write access is password protected. This includes links which allow objects on the hub machine to be modified, for example the state of job can be changed from **FAILED** to **PENDING**.

This access regime is implemented using the standard HTTP password cookie system. The regime can be modified by customizing file `cgi-access` in `$HVR_HOME/lib`, file `.htaccess` in `$HVR_HOME/www`, `$HVR_HOME/cgi-bin` and `$HVR_HOME/cgi-sys` in `$HVR_HOME/lib`. The HTTP passwords, groups and realms can be modified with Apache commands `htpasswd` or by directly accessing Apache files `passwd` and `group`.

5.9. **hvrping, hvrtestlistener, hvrtestlocation, hvrschedport**

NAME

hvrping - Check HVR Scheduler is running.

hvrtestlistener - Test listening TCP/IP port for HVR remote connection

hvrtestlocation - Test connection to HVR location

hvrschedport - Map a hub database name to an HVR Scheduler port

SYNOPSIS

hvrping [-*nnode*][-*tN*] *hubdb*

hvrtestlistener [-*Cpubcert*] [-*Llogin/pwd*] [-*tN*] *node port*

hvrtestlocation [-*lloc*]... [-*tN*] [-*uuser*] *hubdb chn loc*

hvrschedport *hubdb*

DESCRIPTION

Command **hvrping** checks if the HVR scheduler is running.

Command **hvrtestlistener** tests that an HVR process is listening on a TCP/IP port for a HVR remote connection. If option **-L** is supplied then it also tests the authentication for that login and password.

Command **hvrtestlocation** tests a connection to an HVR location.

Command **hvrschedport** maps the name of a hub database to the TCP/IP port number that the HVR Scheduler would listen on for that hub database.

OPTIONS

-C*pub_cert* Public certificate for testing encrypted SSL connection.

-l*loc* Only test specific location(s).

-L*login/pwd* Test authentication of login/password on remote machine.

-n*node* Connect to HVR Scheduler running on node.

-t*N* Time-out after *N* seconds if network is hanging or HVR Scheduler takes too long to reply.

-u*user* Connect to hub using specific database username.

5.10. hvrefresh, hvcompare

NAME

hvrefresh, **hvcompare** - Refresh or compare tables.

SYNOPSIS

hvrefresh [-options] *hubdb chn*

hvcompare [-options] *hubdb chn*

DESCRIPTION

Commands **hvrefresh** and **hvcompare** perform the refresh and compare of objects in the locations of channel *chn*.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

Refresh and compare operations can be performed a row by row or as a bulk operation, depending on which **-g** option is supplied. Bulk refresh means that the target object is truncated and then bulk copy is used to refresh the data from the read location. During bulk refresh table indexes and constraints will be temporarily dropped or disabled and will be reset after the refresh is complete. Bulk compare involves calculating the checksum for each object and reporting whether the replicated tables are identical. Row-wise compare causes data to be piped from the read location to the write location whereby each individual row is compared. This results in a list of a minimal number of inserts, updates or deletes needed to resynchronize the tables. Row-wise refresh works the same as row-wise compare, except that the SQL statements which are calculated to resynchronize the tables are applied during the refresh process.

OPTIONS

- d** Remove (drop) scripts and scheduler jobs & job groups generated by previous **hvrefresh** or **hvcompare** commands.
- f** Fire database triggers/rules while applying SQL changes for refresh. The default behavior is that database trigger/rule firing is disabled during refresh. Even if option **-f** is supplied the capture triggers/rules created by HVR will only be fired if a capture session name does not match the integrate session name. The session name defaults to **hvr_integrate** and can be set on the capture side with action **DbCapture /IgnoreSession** and on the integrate side with **DbIntegrate /SessionName**. If this option is not defined then **hvrefresh** avoids rule firing using statement **set no rules** for Ingres database, trigger disable and enable for an Oracle database and dropping and recreating the triggers on a SQL Server database.
- F** Allow refresh into database which is not defined for replication. Normally **hvrefresh** will give an error if the target database does not have action **DbIntegrate** defined. This option suppresses that error.
- gx** Granularity of refresh or compare operation in database locations. Valid values of *x* are:
 - b** Bulk refresh or compare using checksums. This is the default.
 - r** Row-wise refresh or compare of tables.

- hclass** Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).
- H** Transport data through the hub machine instead of taking a shortcut straight from the source to the target machine. This option only has an effect if the hub, source and target are on different machines.
- lx** Target location of compare or refresh. The other (read location) is specified with option **-r**. If this option is not supplied then all locations except the read location are targets. Values of *x* maybe one of the following:
- loc* Only location *loc*.
 - l1-l2* All locations that fall alphabetically between *l1* and *l2* inclusive.
 - !*loc* All locations except *loc*.**
 - !*l1-l2* All locations except for those that fall alphabetically between *l1* and *l2* inclusive.**
- Several **-lx** instructions can be supplied together.
- mmask** Mask (ignore) some differences between the tables that are being compared. Valid values of *mask* can be:
- d** Mask out delete differences.
 - u** Mask out update differences.
 - i** Mask out insert differences.
- Letters can be combined, for example **-mid** means mask out inserts and deletes. If a difference is masked out, then the verbose option (**-v**) will not generate SQL for it and **hvrrefresh** will not rectify it. The **-m** option can only be used with row-wise granularity (option **-gr**).
- pN** Perform refresh or compare operations on different locations in parallel using *N* sub-processes. This cannot be used with option **-s**.
- qd** Purge queued replication changes during **hvrrefresh**. Valid values for *d* are:
- rw** Read/Write. Purge queued replication changes on both read and write locations.
 - wo** Write only. Only purge queued replication changes on write location. If changes are being replicated from the read location to multiple targets and only one of these targets is being refreshed, then this value will avoid purging changes that are still needed by the other targets.
- The objects purged are the contents of capture tables in a database with trigger-based capture and the transaction files in the **router** directory (files matching ***.tx_***). If this option is not supplied then no replication changes are purged.
- rloc** Read location. For refresh this means that data will be read from location *loc* and written to the other location(s). For compare however this means that location *loc* is passive; the data from here is piped to the other location(s) and the work of comparing the data is performed there instead.

- s** Schedule invocation of refresh (or compare) scripts using the HVR Scheduler. Without this option the default behavior is to perform the refresh or compare operations immediately. The jobs created by this option are named *chn-refr-l1-l2* (or *chn-cmp-l1-l2*) and are created in job group *CHN-REFR* (or *CHN-CMP*). This job group is initially inactive as it is created with scheduler attribute **suspend**. These jobs can be invoked using command **hvrtrigger** as in the following example:

```
$ hvrtrigger -u -w hubdb MYCHN-REFR
$ hvr $HVR_CONFIG/job/hubdb/mychn/mychn-refr-loc1-loc2
```

The previous commands remove the **suspend** attribute from job group **MYCHN-REFR** and instructs the scheduler to run all jobs it contains (when quotas permit). Output from the jobs is copied to the **hvrtrigger** command's **stdout** and the command finishes when all jobs have finished.

Jobs created are cyclic which means that after they have run they go back to **PENDING** state again. They are not generated by a **trig_delay** attribute which means that once they complete they will stay in **PENDING** state without getting retriggered.

- ty** Only affect objects referring to table codes specified by *y*. Values of *y* may be one of the following:
- tbl* Only affects table name *tbl*.
 - t1-t2* Affects all table codes that fall alphabetically between *t1* and *t2* inclusive.
 - !tbl** Affects all table names except *tbl*.
 - !t1-t2** Affects all table codes except for those that fall alphabetically between *t1* and *t2* inclusive.

Several **-ty** instructions can be supplied together.

- Ttsk** Specify alternative task for naming scripts and jobs. The default task name for **hvrefresh** is **refr** and for **hvrcompare** is **cmp**, so for example **hvrefresh** without this **-T** option generates scripts and jobs named *chn-refr-l1-l2* which are placed in scheduler job group *CHN-REFR*.

- uuser** Connect to hub database using Ingres DBMS account *user*.

Ingres

- uuser/pwd** Connect to Microsoft SQL Server hub database as *user* with password *pwd*.

SQL Server

- v** Verbose. This causes row-wise compare and refresh to display each difference detected. Differences are presented as SQL statements. This option requires that option **-gr** (row-wise granularity) is supplied.

EXAMPLE

For bulk refresh of table **order** from location **cen** to location **decen**:

```
$ hvrefresh -rcen -ldecen -torder hubdb sales
```

ON-LINE REFRESH

Normally when a database is being refreshed end-users should be prevented from making changes to the original until the refresh is complete. This is done to prevent the copy of the database becoming inconsistent. For large databases however this can be inconvenient because it may take many hours to make the copy. On-line refresh is a technique to avoid this downtime. On-line refresh consists of performing the following steps:

1. Enable capture on the source database. This may require activating replication with [hvrload -c](#).
2. Ensure that replication jobs are not running. Either stop the HVR Scheduler or suspend the replication jobs with [hvrsuspend](#).
3. Start refresh from the original database to the target database.

```
$ hvrrefresh -rcen -ldecen hubdb sales
```

4. When refresh is complete, restart the replication jobs.

```
$ hvrtrigger -u hubdb SALES-CAP SALES-INTEG
```

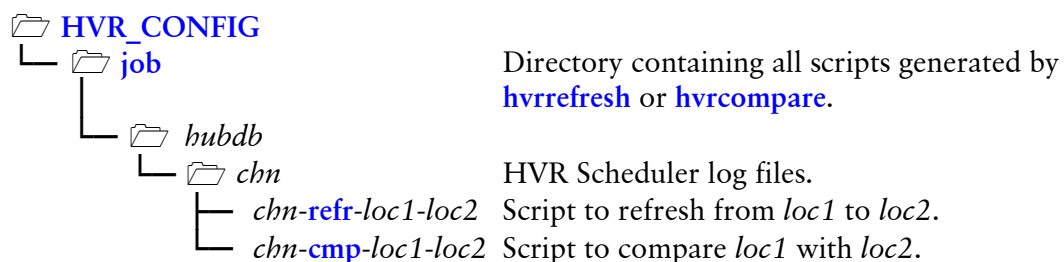
The target database will be consistent after replication jobs have run for the first time. This involves that the first time the replication jobs run (step 4) the integration of some changes may fail. The errors encountered will be "duplicate insert" or "row does not exist". These errors should be expected the first time the replication jobs run. But after the first run the target database should be consistent and these errors should no longer occur.

The online refresh technique can only be used if the following is true:

- ❖ Action [DbIntegrate](#) is defined with parameters [/OnErrorSaveFailedRow](#) or [/Resilient](#).
- ❖ All the target tables have a unique key constraint (so that duplicate inserts will fail).

Foreign key constraints or secondary unique indexes on the target database can cause online refresh to fail.

FILES



SEE ALSO

Commands [5.3 hvrencrypt](#), [hvrencryptdb](#) and [5.4 hvrgui](#).

5.11. hvrremotelistener

NAME

hvrremotelistener - HVR Remote Listener.

SYNOPSIS

hvrremotelistener [-options] portnum

DESCRIPTION

HVR Remote Listener listens on a TCP/IP port number and invokes an **hvr** process for each connection. The mechanism is the same as that of the Unix daemon **inetd**.

On Windows HVR Remote Listener is a Windows Service which is administered with option **-a**. The account under which it is installed must be member of the Administrator group, and must be granted privilege to act as part of the Operating System (**SeTcpPrivilege**). The service can either run as the default system account, or (if option **-P** is used) can run under the HVR account which created the Windows Service.

On Unix and Linux HVR Remote Listener runs as a daemon which can be started with option **-d** and killed with option **-k**.

OPTIONS

- ax** Administration operations for Microsoft Windows system service. Values of *x* can be:
- Windows**
- c** Create the HVR Remote listener system service.
 - s** Start the HVR Remote listener system service.
 - h** Halt (stop) the system service.
 - d** Destroy the system service.
 - D** Destroy all HVR Remote Listener system services in registry.
- Several **-ax** operations can be supplied together; allowed combinations are e.g. **-acs** (create and start) or **-ahd** (halt and destroy). Operations **-as** and **-ah** can also be performed from the **Manage>Services and Applications>Services** screen of Windows.
- cclus\clusgrp** The Windows service should be created, started, stopped and deleted in the Windows Cluster service named *clus* in the cluster group *clusgrp*. This service will act as a ‘Generic Service’ resource within the cluster. This option must be used with option **-a**, but not with option **-aD**.
- Windows**
- Cpubcert** Require network encryption for remote HVR connections using public certificate *pubcert* generated by command **hvrsslgen**. This public certificate should correspond to the *privkey* file supplied using option **-K**. File *pubcert* should be an absolute filename.
- d** Start **hvrremotelistener** as a daemon process.
- En=v** Set environment variable *n* to value *v* for the HVR processes started by this service.
- i** Interactive invocation. HVR Remote Listener stays attached to the terminal instead of redirecting its output to a logfile.
- k** Stop **hvrremotelistener** daemon using the process-id in **\$HVR_CONFIG/files/hvrremotelistenerport.pid**.

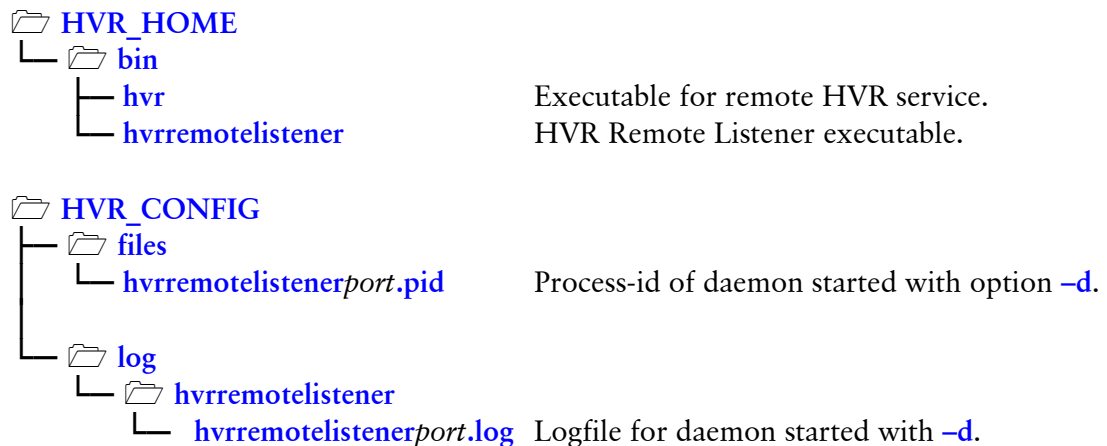
- Kprivkey** Require network encryption for remote HVR connections using private key *privkey*. This private key should correspond to the *pubcert* file supplied using option **-C**. File *privkey* should be an absolute filename.
- N** Do not check passwords for authentication. Disabling password authentication is a security hole, but may be useful as a temporary measure for testing.
- ppamsrv** Use Pluggable Authentication Module *pamsrv* for login password authentication of remote HVR connections. PAM is a service provided by several Operation Systems as an alternative to regular login/password authentication, e.g. checking the */etc/passwd* file. Often **-plogin** will configure hvr slaves to check passwords in the same way as the operating system. Available PAM services can be found in file */etc/pam.conf* or directory */etc/pam.d*.
Unix & Linux
- Ppwd** Configure HVR Remote Listener service to run under the current login HVR account using password *pwd*, instead of under the default system login account. May only be supplied with option **-ac**. Empty passwords are not allowed. The password is kept (hidden) within the Microsoft Windows OS and must be re-entered if passwords change.
Windows
- Uuser** Limits the HVR slave so it only accepts connections which are able to supply the password for account *user*. Multiple **-U** options can be supplied.

NOTES

HVR Remote Listener is supported on Unix and Linux but it is more common on these machines to start remote HVR executables using the **inetd** process.

When running as a Windows service errors are written to the Windows event log. See screen **Programs > Administrative Tools > Event Viewer > Log > Application**.

FILES



SEE ALSO

Commands [5.1 hvr runtime engine](#) and [5.15 hvrsslgen](#).

5.12. hvrretryfailrows, hvrretryblockloc, hvrhistoricalpurge

NAME

hvrretryfailrows - Retry changes written to fail tables due to integration errors.

hvrretryblockloc - Retry changes from locations blocked due to integration errors.

hvrhistoricalpurge - Purge old data from collision handling history tables.

SYNOPSIS

hvrretryfailrows [-d] [-hclass] [-tbl]... [-uuser] [-wsqlrestr] [-v] *hubdb chn loc*

hvrretryblockloc [-hclass] [-lcaploc] [-uuser] *hubdb chn intloc*

hvrhistoricalpurge [-a] [-freq] [-hclass] [-tbl]... [-uuser] *hubdb chn loc*

DESCRIPTION

Command **hvrretryfailrows** causes HVR to reattempt integration of changes in the fail tables for location *loc*. HVR integration jobs write changes to the fail tables if integration fails and action **DbIntegrate /OnErrorSaveFailedRow** is defined. The integration is retried immediately, instead of being delayed until the integrate job next runs.

Command **hvrretryblockloc** unblocks integration into location *intloc* of changes from capture locations that have become blocked due to an integration error. A captured location becomes blocked if an error occurs during integration while action **DbIntegrate /OnErrorBlockLocation** is defined. The actual retry is delayed until the next time the integration job runs, rather than being done immediately by the **hvrretryblockloc** command.

Command **hvrhistoricalpurge** will purge old data from collision handling history tables. This is done automatically after integration if **CollisionDetect /AutoHistoryPurge** is defined.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

OPTIONS

- a** Purge all history, not just changes older than receive stamps.
- d** Only delete rows, do not retry them. If no **-w** option is supplied then the fail table is also dropped.
- ffreq** Commit frequency. Default is commit every 100 deletes.
- hclass** Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).
- lcaploc** Only unblock location *caploc*. If this option is not supplied all blocked capture locations are unblocked.
- tbl** Only failed rows from table *tbl*. If this option is not supplied, rows from all fail tables will be processed. Value *tbl* may be one of the following:
- tbl* Only affects table *tbl*.
 - t1-t2* Affects all tables that fall alphabetically between *t1* and *t2* inclusive.
 - !tbl** Affects all tables except *tbl*.
 - !t1-t2** Affects all tables except for those that fall alphabetically between *t1* and *t2* inclusive.
- Several **-ty** instructions can be supplied together.
- uuser** Connect to hub database using Ingres DBMS account *user*.
Ingres
- uuser/pwd** Connect to Microsoft SQL Server hub database as *user* with password *pwd*.
SQL Server
- v** Verbose output for **hvrretryfailrows**.
- wsqlrestrict** Where clause. Only failed rows where *sqlrestr* is true will be processed. For example to only retry recent changes SQL restriction would be **hvr_cap_tstamp >= '25/5/2007' and col1=22**.

5.13. hvrrouterview

NAME

hvrrouterview - View contents of router transaction and journal files in XML format.

SYNOPSIS

hvrrouterview [-options] *hubdb chn*

hvrrouterview -*ffile*

DESCRIPTION

This command shows the contents of transaction and journal files in directory **\$HVR_CONFIG/router/hub/chn/loc_caploc** in XML format. The output is sent to **stdout**.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

OPTIONS

- btime** Only changes after time, with format *YYYY-MM-DD HH:MI:SS*
- cloc** Only changes from specific capture location.
- d** Show column datatype information.
- etime** Only changes before time, with format *YYYY-MM-DD HH:MI:SS*
- ftxfile** Contents of specific transaction file. This option can be specified multiple times.
- F** Need for transaction file(s) for 'blob file' channel which has no table information.
- h** Print data values in hexadecimal format.
- iloc** Only changes for specific integrate location.
- j** Show the contents of HVR journal files in directory **\$HVRCONFIG/jnl/hubdb/chn**. These files are kept if parameter **/Journal** is defined for **DbIntegrate** or **FileIntegrate**.
- n** Newest. Only most recent transaction file(s).
- tbl** Only changes for specific table. This option can be specified multiple times.

EXAMPLE

```
cd $HVR_CONFIG/router/hubdb/hvr_demo01/loc_cen/
hvrrouterview -f 4a82a8e8_c31e6.tx_dec01
```

5.14. hvrscheduler

NAME

hvr_scheduler - HVR Scheduler server

SYNOPSIS

hvr_scheduler [-options] *hubdb*

DESCRIPTION

The HVR Scheduler is a process which runs jobs defined in catalog **hvr_job**. This catalog is a table found in the hub database. These jobs are generated by commands **hvrload**, **hvrrefresh** and **hvrcompare**. After they have been generated these jobs can be controlled by attributes defined by the jobs themselves and on the job groups to which they belong. These attributes control when the jobs get scheduled.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

On Unix the HVR Scheduler runs as a daemon, and on Microsoft Windows it runs as a system service. On Windows the HVR Scheduler system service is installed and administered using the **hvr_scheduler** command line option **-a** and can also be controlled using the normal Windows service dialog.

HVR Scheduler uses a concept of 'Job Space', a two-dimensional area containing jobs and job groups. A job group may contain jobs and other job groups. In Job Space, jobs are represented as points (defined by X and Y coordinates) and job groups are represented as boxes (defined by 4 coordinates minimum X, maximum X, minimum Y and maximum Y). All jobs and job groups are contained within the largest job group, which is called **SYSTEM**.

Attributes can be defined on jobs or job groups to specify scheduling. Some attributes are known as control attributes others as environment attributes. Control attributes specify quotas, retry and timeout. Environment attributes tell the scheduler how to start the job. The **set** attribute is an example of an environment attribute.

Since the same attribute with different values can be assigned to, for example, a job and the job group that contains the job, the scheduler uses a hierarchy which determines the effects of any given attribute. Specifically, attributes take precedence in Job Space from the smallest object to the largest (i.e. a job attribute takes precedence over the same attribute type assigned to a job group).

OPTIONS

-ax

Windows

Administration operations for the HVR Scheduler Microsoft Windows system service. Allowed values of *x* are:

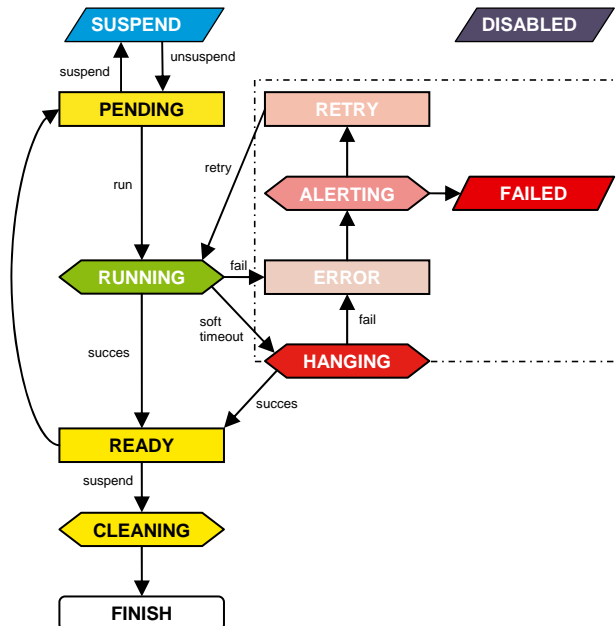
- c** Create the HVR Scheduler service and configured it to start automatically at system reboot. The service will run under the default system unless **-P** option is given.
- s** Start the HVR Scheduler service.
- h** Halt (stop) the system service.
- d** Destroy the system service.
- D** Destroy all HVR Scheduler services in registry.

Several **-ax** operations can be supplied together, e.g. **-acs** (create and start) and **-ahd** (halt and destroy). Operations **-as** and **-ah** can also be performed from screen **Settings>ControlPanel>Services** of Windows.

- cclus\clusgrp** Windows The Scheduler Windows service should be created, started, stopped and deleted in the Windows Cluster service named *clus* in the cluster group *clusgrp*. This service will act as a ‘Generic Service’ resource within the cluster. This option must be used with option **-a**, but not with **-aD**.
- En=v** Set environment variable *n* to value *v* for this process and its children.
- hclass** Specify hub database. Valid values are **ingres**, **oracle** and **sqlserver**. See also section [1.6 Calling HVR on the Command Line](#).
- i** Interactive invocation. HVR Scheduler does not detach itself from the terminal and job output is written to **stdout** and **stderr** as well as to the regular logfiles.
- k** Kill old HVR Scheduler and any jobs which it may be running at that moment.
- Ppwd** Windows Configure the HVR Scheduler system service to run under the current login account using password *pwd*, instead of under the default system login account. May only be supplied with option **-ac**. Empty passwords are not allowed. The password is stored (hidden) within the Microsoft Windows OS and must be re-entered if passwords change.
- qg** Quiesce with *g* seconds ‘grace’. Old scheduler is instructed to cease starting new jobs and wait until all currently running jobs have finished before terminating itself. If *g* is zero (**hvscheduler -q0**) this command will block indefinitely while the old scheduler waits until all jobs are finished. If *g* is non-zero the old scheduler must shut itself down within *g* seconds, after which it and any jobs still running will be abruptly killed as by option **-k**. After the old scheduler has had a quiesce request it will still accept new sessions from HVR Monitor, **hvrtrigger**, **hvrping**, but will not start any more jobs.
- s/bl** Internal slave co-process. The scheduler uses two co-processes at runtime; one to make SQL changes to the hub database (**-swork**), and the other for listening for database events (**-slisten**).
- t** Timeout for connection to old HVR Scheduler process. The default is 10 seconds.
- uuser** Ingres Connect to hub database using Ingres DBMS account *user*.
- uuser/pwd** SQL Server Connect to Microsoft SQL Server hub database as *user* with password *pwd*.

JOB STATES

The HVR Scheduler schedules jobs. Each job performs a certain task. At any moment a job is in a certain state. For instance, when a job is waiting to be run, it is in state **PENDING**; when a job is running, it is in state **RUNNING**.



Jobs can be either acyclic or cyclic. Acyclic jobs will only run once, whereas cyclic jobs will rerun repeatedly. When a cyclic job runs, it goes from state **PENDING** to **RUNNING** and then from state **RUNNING** to **READY**, from where it automatically changes to state **PENDING**. In this state it waits to receive a signal (trigger) in order to run again. When an acyclic job runs, it goes from state **PENDING** to **RUNNING**, then to **READY** and disappears.

If for some reason a job fails to run successfully the scheduler will change its state first to **ERROR** and then to **ALERTING**. While in **ALERTING** state script `hvr.alert` will be called, which decides what the job should do next. If the job is allowed to retry it will go to state **RETRY** and will eventually run again. Otherwise the job will be marked **FAILED**, where it will remain until its state is manually changed. If a job stays in state **RUNNING** for too long it may be marked with state **HANGING**; if it finishes successfully it will just become **READY**, otherwise it becomes state **ERROR**.

DISABLING AND SUSPENDING JOBS

There are two ways to prevent jobs in the HVR Scheduler from running:

- ❖ Abruptly, by explicitly setting the state of an individual job to **DISABLED**. Once in **DISABLED** state a job is immune to triggering, retry and the presence or absence of suspend attribute; it can only be reactivated by explicitly resetting its state (e.g. back to **PENDING**).
- ❖ Gracefully, by adding a **suspend** attribute to a job group containing the job. **PENDING** jobs will immediately assume state **SUSPEND**, and jobs which are **RUNNING** or **HANGING** will continue running but assume state **SUSPEND** when they finish. Jobs in other states (**DISABLED**, **FAILED** and **RETRY**) ignore the **suspend** attribute, except that it inhibits job retrying.

The **SUSPEND** state means a job has stopped running because it is affected by a **suspend** attribute. Stopping jobs by defining the suspend attribute allows the scheduler to stop them gracefully the next time they are passive, instead of abruptly killing a job while it is processing changes. If you try to set a running job's state to **SUSPEND** the HVR Scheduler

may kill it and force it back to **SUSPEND** but the job will then spring back to **PENDING** state when it sees it is not affected by a **suspend** attribute.

There are 3 methods of adding a **suspend** attribute:

1. In HVR Monitor by drag & drop from attribute toolbar.
2. In HVR Monitor using the **Group > Suspend** right-click menu option.
3. Using command line utility **hvrsuspend**.

Methods 2 and 3 also allow the definition of a suspend timeout (using option **Group > Suspend > Allow 1 minute timeout** or the **hvrsuspend -t** option); if running jobs do not finish within that amount of time they are killed and forced into **SUSPEND** state. This suspend timeout feature differs from the timeout of the **hvrtrigger** command because if a triggered job does not complete in time it is left running and only **hvrtrigger** stops.

TRIGGERING JOBS

Although jobs can be made to run immediately by forcing their state to directly **RUNNING**, this can be dangerous because quotas are ignored so attributes (such as memory or database connections) may become exhausted. Instead jobs are typically started by setting the job's trigger (column **job_trigger** of catalog **hvr_job**). The server starts triggered jobs as soon as it can do so without exceeding a quota. Jobs must be triggered: a **PENDING** job whose trigger is not set will not be run. A job can be triggered in the following ways:















- ❖ Triggered a number of seconds after it finished running using attribute **trig_delay**.
- ❖ Triggered at a particular date and time using attribute **trig_at**.
- ❖ Triggered at a certain times of the day, week or month, using attribute **trig_crono**.
- ❖ From the command line, using command **hvrtrigger**.
- ❖ Directly in SQL, i.e.

```
update hvr_job set job_trigger=1 where job_name='myjob';
```

A restriction is that attributes **trig_delay**, **trig_crono** and **trig_at** can be set on job groups or directly on jobs.

Parameter *crono* has a format that closely resembles the format of Unix's crontab. See parameter **/CaptureTimes** in [6.14 Scheduling](#).

ATTRIBUTE TYPES

	Attribute	Arg1	Arg2	Description
	suspend			All affected jobs in PENDING state move into SUSPEND state, jobs in RETRY state are inhibited from being re-run and all resources in KEEPAIVE or LISTEN state become FREE . This continues until this suspend attribute is manually removed.
	quota_run	<i>n</i>		Maximum number of jobs in RUNNING or HANGING state in job group.
	quota_children	<i>n</i>		Maximum number of child processes associated with jobs in job group, including running jobs, (alert, clean etc.) and keeplive resources, but excluding database slaves hvr scheduler -sh .
	quota_speed	<i>n</i>	<i>secs</i>	Limit speed number of jobs made RUNNING within <i>secs</i> seconds of each other to <i>n</i> . The starting of other processes (alert processes, keeplive resources) is not affected by this attribute.
	trig_delay	<i>secs</i>		Trigger cyclic job <i>secs</i> after it last finished running (column job_last_run_end of hvr_job). If a cyclic job is not affected by any trig_delay attribute it will remain PENDING indefinitely.
	trig_crono	<i>crono</i>		At moment <i>crono</i> , trigger job.
	trig_at	<i>tim</i>		Trigger job at specific moment <i>tim</i> has form dd/mm/eeyy HH:MM:SS (interpreted using the local-time).
	retry_max	<i>n</i>		Allow <i>n</i> retries of unsuccessful job. If <i>n</i> is zero no retry is allowed. Jobs unaffected by a retry_max attribute are not retried: on error they become FAILED directly.
	retry_delay	<i>isecs</i>	<i>fsecs</i>	Initially retry job after <i>isecs</i> and double this delay for each unsuccessful retry until <i>fsecs</i> is reached. (Default value is 60 3600)
	timeo_soft	<i>secs</i>		After job has been in RUNNING state for <i>secs</i> seconds, write time-out message and change its job state to HANGING . If <i>secs</i> is zero then no time-out will occur.
	timeo_hard	<i>secs</i>		Kill job in RUNNING or HANGING state after it has run <i>secs</i> seconds. If <i>secs</i> is zero then no time-out will occur.
	set	<i>name</i>	<i>val</i>	Set variable <i>name</i> to value <i>val</i> in job's runtime environment and replace occurrences of <i>\$name</i> in the arguments certain other attribute types are replaced with value <i>val</i> .
	log_to	<i>sev</i>	<i>dest [flgs]</i>	Redirect output of severity <i>sev</i> to destination <i>dest</i> .
	log_cc	<i>sev</i>	<i>dest [flgs]</i>	Carbon-copy output of severity <i>sev</i> to destination <i>dest</i> .

OUTPUT REDIRECTION

Each message written by an HVR job is redirected by the scheduling server to various destinations. The destination depends on the severity of the error and which **log_cc** and **log_to** attributes are defined on the job. The three severities are:

critical Critical error messages. The following messages are critical:

- ❖ Errors which affect the HVR Scheduler server itself.
- ❖ Errors in attribute definitions or the operating system which prevent a job running.
- ❖ Any message matching a pattern defined in the file **critical.pat**.
- ❖ Messages written to **stderr** by **hvr alert**.

error Output by a running job to **stderr**.

output Output by a running job to **stdout** or **stderr**.

Output from a job for channel *chn* moving data between locations *l1* and *l2* is written to the following logfiles in directory **\$HVR_CONFIG/log/hubdb**:

Logfile	Filename	Severity		
		Output	Error	Critical
Per-job logfile, with all output and errors.	<i>jobname.out</i>	✓	✓	✓
Channel logfile with all output and errors.	<i>chn.out</i>	✓	✓	✓
Channel logfile, with normal errors and critical errors.	<i>chn.err</i>	✗	✓	✓
Channel logfile, with critical errors only.	<i>chn.crit</i>	✗	✗	✓
Server logfile with all output and errors.	<i>hvr.out</i>	✓	✓	✓
Server logfile, with normal errors and critical errors.	<i>hvr.err</i>	✗	✓	✓
Server logfile, with critical errors only.	<i>hvr.crit</i>	✗	✗	✓

If the HVR Scheduler itself fails, a critical error prefixed by a message-id will also be written to all above logfiles.

The filenames specified by parameters **log_to** and **log_cc** can be one of the following:

Destination	Description
<i>file</i>	File in \$HVR_CONFIG/log/db into which messages are appended.
<i>/path/file</i>	Absolute pathname of file into which messages are appended.
syslog id lvl Unix & Linux	Send message to Unix syslog(3) system as events for the USER facility.
eventlog Windows	Send messages as application events to Operating System event log. On Windows these are displayed in screen Programs > Administrative Tools > Event Viewer > Log > Application.

The filenames for **log_cc** and **log_to** can be prefixed by one or more of the following:

Flag	Description
-wX	Abbreviate output to <i>X</i> flags width.
-cL	Remove letter <i>L</i> from message.
-pPREF	Prefix each line with <i>PREF</i> .
-iID Unix & Linux	Send Unix syslog event with identifier <i>ID</i> .
-llvl Unix & Linux	Send message to Unix syslog with priority <i>lvl</i> . Valid values are: LOG_EMERG , LOG_ALERT , LOG_CRIT , LOG_ERR , LOG_WARNING , LOG_NOTICE , LOG_INFO or LOG_DEBUG .
-s	Use file locking and stuttering when writing to file.

ENVIRONMENT VARIABLES

HVR_HTTP_CONFIG	HTTP alias for \$HVR_CONFIG used for the URLs which the HVR Scheduler supplies to the HVR Monitor. The default is /hvr_config .
HVR_HTTP_HOME	HTTP alias for \$HVR_HOME used for the URLs which the HVR Scheduler supplies to the HVR Monitor. The default is /hvr_home .
HVR_HTTP_NODE	Explicit node name for the URLs which the HVR Scheduler supplies to the HVR Monitor. The default is the hostname of the hub as specified by the PC.
HVR_HTTP_PORT	Explicit TCP/IP port number for the URLs which the HVR Scheduler supplies to the HVR Monitor. The default is 8080.

HVR_ITO_LOG

Causes the HVR Scheduler to write a copy of each critical error to the file named in the variable's value. This can be used to ensure all HVR error messages from different hub databases on a single machine can be seen by scanning a single file. Long messages are not wrapped over many lines with a '\', but instead are written on a single line which is truncated to 1024 characters. Each line is prefixed with "**HVR_ITO_AIC** *hubnode:hubdb locnode*", although **HVR** is used instead of **\$HVR_ITO_AIC** if that variable is not set.

HVR_PUBLIC_PORT

Instructs the HVR Scheduler to listen on an additional (public) TCP/IP port number.

FILES

HVR_HOME	
bin	Perl script used by scheduler to decide if jobs should be retried.
cgi-bin	CGI for read-only HVR Monitor operations.
cgi-sys	HTTPD access control file.
.htaccess	CGI for trusted HVR Monitor operations.
hvrmonitor.cgi	
lib	
critical.pat	Patterns indicating which messages are critical. Adding patterns to this file does not affect which errors cause a job to be retried.
retriable.pat	Patterns indicating which errors can be handled by retrying a job.
www	
docs	
hvrum.pdf	This manual.
.htaccess	HTTPD access control file.
index.html	HTML to start HVR Monitor for any database.
HVR_CONFIG	
files	
scheddb.pid	Process-id file.
log	
hubdb	
jobname.out	All messages for job <i>jobname</i> .
chn.out	All messages for channel <i>chn</i> .
chn.err	All error messages for channel <i>chn</i> .
chn.crit	All critical error messages for channel <i>chn</i> .
hvr.out	All messages for all jobs.
hvr.err	All error messages for all jobs.
hvr.crit	All critical error messages for all jobs.
hvr.ctrl	Log file for actions from control sessions.
work	
hubdb	
*	Working directory for executing jobs.
wwwgen	
hubdb	
index.html	HTML to start HVR Monitor for database <i>hubdb</i> .

EXAMPLE

Create a Windows listener service to listen on port number 50000.

```
c:\> hvrremotelistener -acs -EHVR_PORT=50000
```

NOTES

When starting the HVR Scheduler it is important that a database password is not exposed to other users. This can be encrypted using command [hvrcrypt](#).

SEE ALSO

[5.3 hvrcrypt](#), [hvrcryptdb](#), [5.17 hvrsuspend](#), [5.18 hvrtrigger](#)

5.15. hvrsslgen

NAME

hvrsslgen - Generate a private key and public certificate.

SYNOPSIS

```
hvrsslgen [-ddir] [-abits][-sbits] name subj
```

DESCRIPTION

An HVR connection to a remote HVR location can be configured so that communication over the network is encrypted.

This command generates two files, a file name *name.priv_key* and a file containing the corresponding public certificate named *name.pub_cert*. The second argument *subj* is written into the subject field of the X509 public certificate file.

If necessary, each HVR location in an HVR channel can be given its own private key and public certificate pair.

Both the file containing private key and the public certificate must be placed on the remote machine, and supplied to the HVR slave process as command line arguments. The file containing the public certificate must also be placed on the hub machine, and supplied to the HVR channel using a special [LocationProperties /SslCertificate](#) action.

HVR can generate the private key and public certificate on the hub machine, or on the remote location. Generation on the remote machine has the advantage that the private key (which has to be kept secret) does not then have to be transported across the [insecure] network.

OPTIONS

- a*bits* Generate asymmetric (RSA) key pair with length *bits*. The default is 1024.
- d*dir* The generated files are created in directory *dir* instead of in the current directory.
- s*bits* During symmetric encryption a symmetric key with length *bits* should be used. The default is 128.

CONFIGURATION STEPS

When configuring a secure HVR network connection, the following steps are needed:

1. Configure a remote connection to an HVR location. The setup steps for this are described in either section [3.1 New Installation on Unix or Linux](#) or [3.2 New Installation on Windows](#) depending on which operating system is used.
2. Generate a private key and public certificate pair. The command to do this is as follows:

```
$ hvrsslgen [-opts] nm subj
```

This command generates two files, a file named *nm.priv_key* and a file containing the corresponding public certificate named *nm.pub_cert*. The second argument *subj* is written into the subject field of the X509 public certificate file.

After generation, it is wise to restrict read access to the private key files using an Operating System command, e.g.

```
Unix & Linux  
$ chmod 600 nm.priv_key
```

- Setup HVR on the remote HVR location to use the newly generated private key and public certificate. HVR on the remote machine consists of an HVR executable with option `-r` which is configured to listen on a specific port number. The command line for this process will be configured in one of the following ways, depending on the Operating System and how the existing remote connection is configured.

Edit file `/etc/inetd.conf`:

Unix & Linux

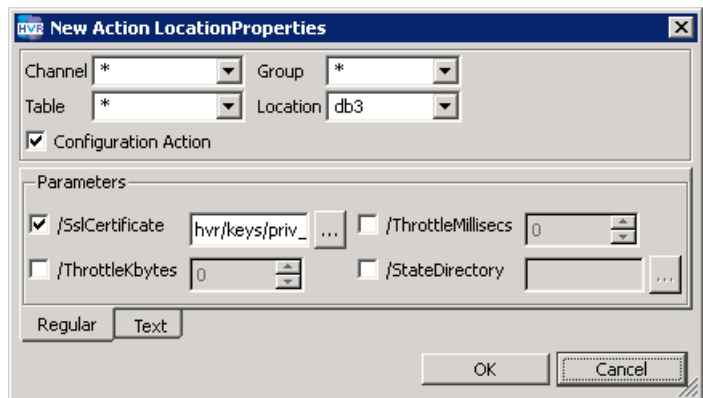
```
hvr stream tcp nowait root /usr/hvr/hvr_home/bin/hvr hvr -r
  -EHVR_HOME=/usr/hvr/hvr_home -EHVR_CONFIG=/usr/hvr/hvr_config -K
  /usr/hvr/keys/john.priv_key -C /usr/hvr/keys/john.pub_cert
```

Windows

```
C:\> hvrremotelistener -acs -K c:\hvr\keys\john.priv_key -C
  c:\hvr\keys\john.pub_cert
```

- Place the public certificate file on the hub machine, and register it in the channel configuration using the action `LocationProperties/SslCertificate`.
- Regenerate the relevant job scripts:

```
$ hvrload -oj -ldec02 hubdb
  mychn
```



ENCRYPTION IMPLEMENTATION

When encryption is activated for an HVR location, every byte sent over the network (in either direction) is encrypted with the Transport Layer Security (TLSv1) protocol. An RSA public/private key pair is used for authentication and session startup. The public key is embedded in an X509 certificate, and the private key is encrypted using an internal password with a 3DES algorithm. After the session is initiated, encryption switches to a symmetric cipher using a RC5 algorithm. By default the keys used for this asymmetric negotiation are 1024 bits long and the symmetric key is 128 bits long, although longer key lengths can be specified when the public/private keys are being generated.

The hub database guards against third parties impersonating the remote HVR location (e.g. by spoofing) by comparing the SHA1 checksums of the certificate used to build the secure connection and its own copy of the certificate.

Public certificates are self-signed. HVR checks that the hub at the remote machines' copies of this certificate are identical so signing by a root authority is unnecessary.

For network encryption, HVR uses OpenSSL, which is developed by the OpenSSL Project (<http://www.openssl.org>).

NOTES

After generation, it is wise to restrict read access to the private key files using an Operating System command, e.g.:

```
$ chmod 600 nm.priv_key
```

Unix & Linux

HVR's encryption will attempt to exploit the entropy (randomness) generation capability supplied by the `/dev/urandom` device. This is not absolutely necessary, but for optimal

security it is recommended that this Operating System option is installed, especially on the machine used to generate public/private key pairs.

SEE ALSO

Commands [5.3 hvcrypt](#), [hvcryptdb](#).

5.16. hvrstatistics

NAME

hvrstatistics - Extract statistics from HVR scheduler logfiles

SYNOPSIS

hvrstatistics [-options]... [*hubdb*]

DESCRIPTION

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

OPTIONS

- btime** Only lines after *time*. Argument must have format YYYY-MM-DD HH:MI:SS.
- cchn** Only parse output for specific channel. This option can be specified multiple times.
- etime** Only lines until *time*. Argument must have format YYYY-MM-DD HH:MI:SS.
- file** Parse scheduler log file *file*, instead of default [\\$HVR_CONFIG/log/hubdb/hvr.out](#).
- gcol** Summarize totals grouped by col which can be either channel, location, job, table, year, month, day, hour, minute or second. This option can be specified multiple times.
- i** Incremental. Only lines added since previous run of [hvrstatistics](#).
- jjob** Only parse output for specific job. This option can be specified multiple times.
- lloc** Only parse output for specific location. This option can be specified multiple times.
- ssep** Print parsed output in CSV-matrix with field separator *sep*. This allows the input to be imported into a spreadsheet.
- tbl** Only parse output for specific table. This option can be specified multiple times.
- v** Verbose trace messages.

EXAMPLES

To count total rows replicated for each location and table, use the following:

```
$ hvrstatistics -g location -g table myhub
Location cen
  Table order
    Captured rows      :    25
  Table product
    Captured rows      :   100
    Capture cycles     :     6
    Routed bytes       :  12053
Location dec01
  Table order
    Integrated updates  :    25
    Integrated changes  :    25
  Table product
    Integrated updates  :   100
    Integrated changes  :   100
    Integrate cycles    :     7
    Integrate transactions :    4
```

To create a CSV file with the same data use option `-s` as follows:

```
$ hvrstatistics -g location -g table -s", " myhub > stats.csv
```

If this CSV file was imported into a spreadsheet (e.g. Excel) it would look this:

Location	Table	Capture cycles	Captured rows	Routed bytes	Integrate cycles	Integrate transactions	Integrated updates	Integrated changes
cen	order		25					
cen	product		100					
cen		6		12053				
decen	order						25	25
decen	product						100	100
decen					7	4		

FILES

```

└─ HVR_CONFIG
   └─ files
      └─ hvrstatistics.offset    State file for incremental statistics (option -i).
```

5.17. hvrsuspend

NAME

hvsuspend - Suspend (or un-suspend) all jobs in a job group

SYNOPSIS

hvsuspend [-tN] *hubdb* GRP...

hvsuspend -u *hubdb* GRP...

DESCRIPTION

In the first form **hvsuspend** will add a **suspend** attribute to the job group(s) whose names are supplied as arguments and then wait until no more jobs are active.

The second form will remove any **suspend** attributes from the job group(s) specified and from any job groups which are contained by those specified job groups.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

Adding a **suspend** attribute means that jobs in state **PENDING** will immediately assume state **SUSPEND**, and jobs which are **RUNNING** or **HANGING** will continue running but assume state **SUSPEND** when they finish. Jobs in other states (**DISABLED**, **FAILED** and **RETRY**) ignore the suspend attribute, except that it inhibits job retrying. The **SUSPEND** state means a job has stopped running because it is affected by a **suspend** attribute. Stopping jobs by defining the **suspend** attribute allows the scheduler to stop them gracefully the next time they are passive, instead of abruptly killing a job while it is processing changes.

This command connects to the HVR Scheduler so the scheduler must be already running.

OPTIONS

- tN Time-out after N seconds if affected jobs take too long to become inactive. The default is to time-out after 10 seconds. Option -t0 means no time-out. If time-out occurs then any **RUNNING** or **HANGING** jobs will be killed, re-triggered and sent to state **SUSPEND**, and an error message will be written. This option cannot be used with option **-u**.
- u Un-suspend. Remove **suspend** attribute from job group named G and from any job groups it contains. This option cannot be used with option **-tN**.

EXAMPLE

A change has been made to the HVR catalogs for location **d01** but for the change to take effect the job's script must be regenerated.

```
$ hvrsuspend hubdb MYCHN-CAP MYCHN-INTEG      # Halt all replication jobs
$ hvrload -oj -ld01 hubdb mychn              # Regenerate script for location d01
$ hvrsuspend -u hubdb MYCHN-CAP MYCHN-INTEG  # Reactivate replication
```

5.18. hvrtrigger

NAME

hvrtrigger - Trigger jobs.

SYNOPSIS

hvrtrigger [-options] *hubdb* *jobname*...

hvrtrigger [-options] *hubdb* *GRP*...

DESCRIPTION

The first form causes job(s) *jobname* in hub database *hubdb* to be triggered; the second form causes all jobs in job group(s) *GRP* to be triggered. When a triggered job reaches **PENDING** state it will be run by the HVR Scheduler, providing relevant quotas would not be exceeded. A message will be written if any jobs are not in an appropriate state for triggering; jobs in state **DISABLED** are immune; jobs in **SUSPEND** state are immune if no **-u** option has been used; jobs in state **FAILED** or **RETRY** are immune if no **-r** option is used.

The first argument *hubdb* specifies the connection to the hub database, this can be an Ingres, Oracle or SQL Server database depending on its form. See further section [1.6 Calling HVR on the Command Line](#).

This command connects to the HVR Scheduler, so the scheduler process must already be running.

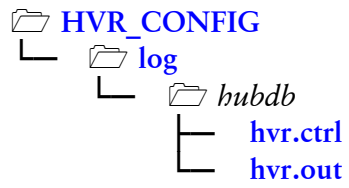
OPTIONS

- nnode** Connect to hub database located on *node*.
- r** Retry **FAILED** or **RETRY** jobs by triggering them with value **2** in column **job_trigger** of catalog **hvr_job**.
- tN** Time-out after *N* (> 0) seconds if scheduler or job takes too long, or if network is hanging. Job execution is not interrupted by this client timeout. If no **-t** option is supplied then **hvrtrigger** will wait indefinitely.
- u** Un-suspend. Remove [any] **suspend** attribute from job group named *G* and from job groups contained in *G*. This option cannot be used with option **-tN**.
- w** Wait until all triggered jobs have finished running and, in the meantime, carbon-copy their output to the **hvrtrigger** command's **stdout** and **stderr**. Any jobs which were already in **RUNNING** or **HANGING** state must finish running before being run again.

EXIT CODES

- 0 Successful communication with the HVR Scheduler server.
- 1 Error sending instruction to HVR Scheduler server, job did not exist, etc.
- 2 Time-out.
- 3 Jobs existed in state **FAILED**, **RETRY ERROR**, **ALERTING**, **SUSPEND** or **DISABLED** which were not affected by the command.

FILES



Audit log containing **hvrtrigger** actions.

Log of job output and errors.

Action	Parameter	Value	Description
6.6. CollisionDetect	/TreatCollisionAsError /TimestampColumn /IntegerTimestamp /NoHistoryTable /AutoHistoryPurge	<i>col_name</i>	Do not resolve collisions automatically Exploit timestamp column col for collision detection Use an integer timestamp for timestamp columns Collision detection without history table Delete history table rows older than receive stamp
6.7. DbObjectGeneration	/NoCaptureInsertTrigger /NoCaptureUpdateTrigger /NoCaptureDeleteTrigger /NoCaptureDbProc /NoIntegrateDbProc /IncludeSqlFile /IncludeSqlDirectory /CaptureTableCreateClause /StateTableCreateClause /FailTableCreateClause /HistoryTableCreateClause /GrantToPublic	<i>file</i> <i>dir</i> <i>sql_expr</i> <i>sql_expr</i> <i>sql_expr</i> <i>sql_expr</i>	Inhibit generation of capture insert trigger Inhibit generation of capture update trigger Inhibit generation of capture delete trigger Inhibit generation of capture database procedures Inhibit generation of integrate database procedures Include file for customizing db objects Search directory for include SQL file Clause for capture table creation statement Clause for state table creation statement Clause for fail table creation statement Clause for history table creation statement Allow all users to access HVR database objects
6.8. Agent	/Command /DbProc /UserArgument /ExecOnHub /Order /Path	<i>file</i> <i>dbproc</i> <i>str</i> <i>int</i> <i>dir</i>	Call OS command during replication jobs Call db procedure during replication jobs Pass argument to each agent execution Execute agent on hub instead of location's machine Specify order of agent execution Search directory dir for agent
6.9. Environment	/Name /Value	<i>env_var</i> <i>path</i>	Name of environment variable Value of environment variable
6.10. FileCapture	/DeleteAfterCapture /Pattern /IgnorePattern /IgnoreUnterminated /AccessDelay	<i>pattern</i> <i>pattern</i> <i>pattern</i> <i>secs</i>	Delete file after capture Only capture files whose names match pattern Ignore files whose names match pattern Ignore files whose last line does not match pattern Delay read for N seconds to ensure writing is complete
6.11. FileIntegrate	/RenameExpression /ErrorOnOverwrite /MaxFileSize /Journal	<i>expression</i> X	Expression to name new files containing substitutions Error if a new file has same name as an existing file Limit each XML file to maximum file size Journal old transaction files after they are processed
6.12. FileFormat	/HvrOpField		Adds hvr_op integer to changes written to XML file
6.13. LocationProperties	/SslCertificate /ThrottleKbytes /ThrottleMillisecs /StateDirectory	<i>file</i> <i>int</i> <i>int</i> <i>path</i>	Secure Socket Layer (SSL) encryption Restrain net bandwidth into packets of <i>int</i> bytes Restrain net bandwidth by msec wait between packets Directory for file location state files.
6.14. Scheduling	/CaptureStartTimes /IntegrateStartAfterCapture /IntegrateStartTimes /RefreshStartTimes	<i>times</i> <i>times</i> <i>times</i> <i>times</i>	Triggers capture jobs to run at a given time Integrate job will run after capture job routes new data Triggers integrate jobs to run at a given time Triggers refresh jobs to run at a given time



6.1. DbCapture

DESCRIPTION

Action **DbCapture** instructs HVR to capture changes to a database table. Various parameters are available to modify the functionality and performance of capture.

PARAMETERS

Parameter	Argument	Description
/LogBased		Capture changes directly from the DBMS logging system. If this parameter is not defined then database triggers will be used for capture. Log-based gives improved performance and low latency but limits which other DbCapture parameters can be used. This parameter cannot be used with SQL Server databases. Configuring log-based capture requires that journaling is enabled on Ingres tables.
/QuickToggle		Allows end-user transactions to avoid lock on toggle table. The toggle table is changed by HVR during trigger-based capture. Normally all changes from user transactions before a toggle are put into one set of capture tables and changes from after a toggle are put in the other set. This ensures that transactions are not split. If an end-user transaction is running when HVR changes the toggle then HVR must wait, and if other end-user transactions start then they must wait behind HVR. Parameter /Quicktoggle allows these other transactions to avoid waiting, but the consequence is that their changes can be split across both sets of capture tables. During integration these changes will be applied in separate transactions; in between these transactions the target database is not consistent.
/ToggleFrequency	<i>secs</i>	This parameter instructs HVR trigger-based capture jobs to wait for a fixed interval before toggling and reselecting capture tables, instead of dynamically waiting for a capture trigger to raise a database alert. Raising and waiting for database alerts is an unnecessary overhead if the capture database is very busy.
/KeyOnlyCaptureTable		Improve performance for capture triggers by only writing the key columns into the capture table. The non-key columns are extracted using an outer join from the capture table to the replicated table. Internally HVR uses the same outer join technique to capture changes to long columns (e.g. long varchar). This is necessary because DBMS rules/triggers do not support long datatypes. The disadvantage of this technique is that 'transient' column values can sometimes be replicated, for example if a delete happens just after the toggle has changed, then the outer join could produce a NULL for a column which never had that value.
/NoIntermediateValues		Causes coalescing of multiple operations on the same row into a single operation. For example an insert and an update can be replaced by a single insert; five updates can be replaced by one update, or an insert and a delete of a row can be filtered-out altogether. The disadvantage of not replicating these intermediate values is that some consistency constraints may be violated on the target database.
/NoBeforeUpdate		Do not capture 'before' row for an update. By default when an update happens HVR will capture both the 'before' and 'after' version of the row. This lets integration only update columns which have been changed and also allows collision detection to check the target row has not been changed unexpectedly. Defining this parameter can improve performance but means that integrate will update all columns (regardless if they were changed).

/IgnoreSessionName	<i>sess_name</i>	This action changes the session name which the HVR capture triggers/rules expect integration to use. Normally HVR's capture avoids recapturing changes made during HVR integration by ignoring any changes made by sessions named <code>hvr_integrate</code> . This prevents looping during bidirectional replication but means that different channels ignore each other's changes. The session name actually used by integration can be changed using DbIntegrate /SessionName . For more information see section SESSION NAMES AND RECAPTURING below.
/IgnoreCondition	<i>sql_expr</i>	Ignore (do not capture) any changes that satisfy expression <i>sql_expr</i> . This logic is added to the HVR capture rules/triggers and procedures. This parameter differs from the Restrict /CaptureCondition as follows: <ul style="list-style-type: none"> • The SQL expression is simpler, i.e. it cannot contain subselects. • The sense of the SQL expression is reversed (changes are only replicated if the expression is false). • No 'restrict conversion'. Restrict conversion means if an update changes a row which did not satisfy the condition into a row that does satisfy the condition then the update is converted to an insert.
/IgnoreUpdateCondition	<i>sql_expr</i>	Ignore (do not capture) any update changes that satisfy expression <i>sql_expr</i> . This logic is added to the HVR capture rules/triggers and procedures.
/HashBuckets 	<i>int</i>	This implies that Ingres capture tables have a hash structure. This reduces the chance of locking contention between parallel user sessions writing to the same capture table. It also makes the capture table larger and I/O into it more sparse, so it should only be used when such locking contention could occur. Row level locking (default for Oracle and SQL Server and configurable for Ingres) removes this locking contention too without the cost of extra I/O.
/HashKey 	<i>col_list</i>	Specify different key for capture table hashing. The default hash key is the replication key for this table. The key specified does not have to be unique; in some cases concurrency is improved by choosing a non-unique key for hashing.

SESSION NAMES AND RECAPTURING

Replication recapturing is when changes made by integration are captured again.

Recapturing is controlled using session names. Depending on the situation recapturing can be useful or unwanted. The following are some examples:

- Bi-directional replication.
During bidirectional replication if integrated changes are captured again then they can boomerang back to the original capture database. This would form an infinite loop. For this reason capture triggers check the session name and avoid recapturing integration.
- Cascade replication.
Cascade replication is when changes from one channel are captured again by a different channel and replicated onto a different group of databases. Recapturing is necessary for cascade replication. Recapturing can be configured using action [Integrate /SessionName](#) so that integration is not recognized by the cascade channel's capture triggers.
- Batch work to purge old data.
Sometimes replication of large blocks of batch work is too expensive. It may be more efficient to repeat the same batch work on each replicated database. Capturing of the batch work can be disabled by setting the session name so that the capture triggers see the changes as belonging to an integrate session.
- Application triggering during integration.
Sometimes an application will have database triggers on the replicated tables. These triggers will have already been fired on the capture database so firing them again during HVR integration is unnecessary and can cause consistency problems. For Ingres

databases this rule firing can be avoided with action **DbIntegrate /NoTriggerFiring**, but for other DBMS's the application triggers can be modified so that they have no affect during replication integration.

Session names are implemented in different ways, depending on the DBMS.

Ingres

For Ingres the session name is recognized as an Ingres role or a Ingres username. The database rules check the session name with **dbmsinfo('role')** and **dbmsinfo('username')**. Integration jobs assign the role by connecting to the integrate database with **sql** option **-R**. Log-based capture only recognizes the Ingres username (not the role) and detects HVR integration using internal table-ids.

Oracle

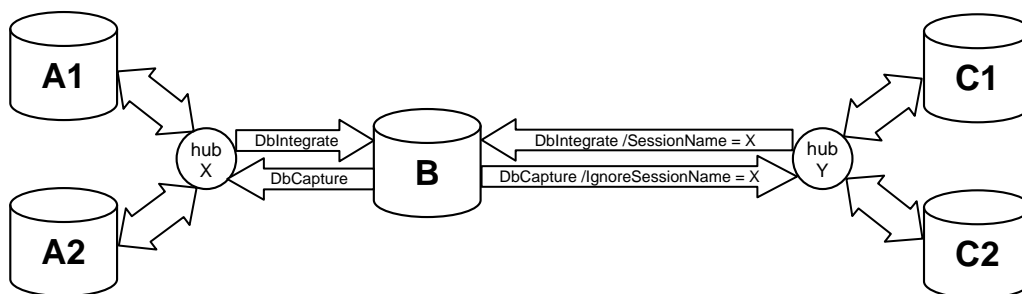
For Oracle the session is implemented as client information or an Oracle username. The Oracle triggers check the session name with **userenv('CLIENT_INFO')** and keyword **user**. The integration job assigns the session name using SQL statement **dbms_application_info.set_client_info('hvr_integrate')**. Log-based capture only recognizes the Oracle username (not the **userenv**) and detects HVR integration using internal table-ids.

SQL Server

For SQL Server the session name is an application name or a SQL Server username. The triggers check the session name with **app_name()** and keyword **user**.

EXAMPLE

Changes made to databases **A1** and **A2** must replicate via **hub X** to **B** and then cascade via **hub Y** to **C1** and **C2**. Changes made to **B** must replicate to **A2** and **A2** and to **C1** and **C2** but must not boomerang back to **B**. Normally changes from **A1** and **A2** to **B** would not be cascade replicated onto **C1** and **C2** because they all use the same session name. This is solved by adding parameters **/SessionName** and **/IgnoreSessionName** to the channel in **hub Y**.



RESTRICTIONS

Parameter **/LogBased** cannot be used with **/QuickToggle**, **/KeyOnlyCaptureTable**, **/HashBuckets**, **/IgnoreCondition** or **/IgnoreUpdateCondition**.

Parameter **/HashKey** requires that **/HashBuckets** is also defined.

EXAMPLES

For parameter **/LogBased** see channel **hvr_demo50**, **hvr_demo51** and **hvr_demo70** in **\$HVR_HOME/demo**.

For parameters **/HashBuckets** and **/HashKey** see channel **hvr_demo16** in **\$HVR_HOME/demo**.

6.2. DbIntegrate

DESCRIPTION

Action **DbIntegrate** instructs HVR to integrate changes into a database table. Various parameters are available to tune the integration functionality and performance.

PARAMETERS

Parameter	Argument	Description
/DbProc		Apply database changes by calling integrate database procedures instead of using direct SQL statements (insert, update and delete). The database procedures are created by hvrload and called tbl__ii , tbl__iu , tbl__id . This parameter cannot be used on tables with long column datatypes.
/OnErrorSaveFailedRow		On integrate error, write the failed change into 'fail table' tbl__f and then continue integrating other changes. Changes written into the fail table can be retried afterwards using command hvrretryfailrows . If this parameter is not defined the default behavior if an integrate error occurs is to write a fatal error and to stop the job.
/OnErrorBlockLocation		On integration error, block further changes from the current capture location but continue integrating changes from the other capture locations. Integration of changes from blocked capture locations can be restarted with command hvrretryblocklock .
/TxBundleSize	<i>int</i>	Bundle small transactions together for improved performance. For example if the bundle size is 10 and there were 5 transactions with 3 changes each, then the first 3 transactions would be grouped into a transaction with 9 changes and the others would be grouped into a transaction with 6 changes. Transaction bundling does not split transactions. The default transaction bundle size is 100.
/TxSplitLimit	<i>int</i>	Split very large transactions to limit resource usage. For example, if a transaction on the master database affected 10 million rows and the remote databases has a small rollback segment then if the split limit was set to 1000000 the original transaction would split into 10 transactions of 1 million changes each. The default is 0, which means transactions are never split.
/Resilient		Resilient integration of inserts, updates and deletes. This modifies the behavior of integration if a change cannot be integrated normally. If a row already exists then an insert is converted to an update, an update or a non-existent row is converted to an insert, and a delete of a non-existent row is discarded. Existence is checked using the replication key known to HVR (rather than checking the actual indexes or constraints on the target table). Resilience is a simple way to improve replication robustness but the disadvantage is that consistency problems can go undetected. Parameter /Resilient is equivalent to defining /ResilientInsert , /ResilientUpdate and /ResilientDelete individually.
/ResilientInsert		Resilient integration of inserts. If a row already exists then the insert is converted to an update. Resilient insert is also implied by parameter /Resilient .
/ResilientUpdate		Resilient integration of updates. An update of a nonexistent row is converted to an insert. Resilient update is also implied by parameter /Resilient .
/ResilientDelete		Resilient integration of deletes. A delete of a nonexistent row is discarded. Resilient delete is also implied by parameter /Resilient .
/ResilientDeleteCondition	<i>sql_expr</i>	Resilient integration of deletes only if <i>sql_expr</i> is true. The argument is an SQL expression which can contain column names enclosed in braces (e.g. {mycol}=5). The column name in braces is substituted for the column value. Conditional delete resilience can be used if a cascade delete constraint can sometimes cause a delete to fail.
/ResilientWarning		Write warning message for each resilient conversion. Normally conversion (for example a lost update being converted to an insert) occurs silently.

Parameter	Argument	Description
/NoTriggerFiring <input type="checkbox"/> Ingres		Integrate changes without firing database rules using Ingres SQL statement set norules . This can be used to prevent replication rules which have already been fired on the capture database from firing again during integration. Another way to do the same is to change the replication rules to recognize the integrate session name (see parameter /SessionName). This statement is only allowed if integration connects as the database owner to the integrate database.
/SessionName	<i>sess_name</i>	Integrate changes with specific session name. The default session name is hvr_integrate . Capture triggers/rules check the session name to avoid recapturing changes during bidirectional replication. For a description of recapturing and session names see action DbCapture .
/Journal		Move processed transaction files to journal directory \$HVR_CONFIG/jnl/hub/chn/YYYYMMDD on the hub machine. Normally an integrate job would just delete its processed transactions files. The journal files are compressed, but their contents can be viewed using command hvrouterview .

RESTRICTIONS

Parameters [/OnErrorSaveFailedRow](#) and [/OnErrorBlockedLocation](#) cannot be used together.

Parameters [/ResilientDelete](#) and [/ResilientDeleteCondition](#) cannot be used together.

Parameter [/ResilientWarning](#) requires that one of the other resilient parameters is defined.

Parameter [/DbProc](#) cannot be used on tables with long datatypes, for example [long varchar](#) or [blob](#).

EXAMPLE

For parameter [/Journal](#) see channel [hvr_demo20](#) in [\\$HVR_HOME/demo](#).

6.3. TableProperties

DESCRIPTION

Action [TableProperties](#) defines properties of a replicated table in a database location. The action has no effect other than that of its parameters. These parameters typically affect replication (on the capture and integrate side) as well as HVR refresh and compare.

PARAMETERS

Parameter	Argument	Description
/BaseName	<i>tbl_name</i>	Defines the actual name of the table in the database. The database table name defaults to the column of tbl_base_name in catalog hvr_table . This parameter is useful if the replication entity has different names in different replicated databases.
/DuplicateRows		Replication table has no unique key and can contain duplicate rows. HVR replicates the table by treating all columns as being key columns, regardless of the replication key provided in hvr_column . This means all updates are treated as key updates and are replicated as a delete and an insert. Each delete is integrated using a special SQL subselect which ensures only a single row is deleted, not multiple rows.
/Schema	<i>schema</i>	Name of database schema or user which owns the base table. By default the base table is assumed to be owned by the database username that HVR uses to connect to the database.
/IgnoreCoerceError		No error will be reported if an error is outside the boundary of a destination datatype or if a conversion is impossible because of a datatype difference (e.g. string 'hello' must be converted into an integer. Instead HVR will silently round an integer or date up or down so it fits within the boundary, truncate long strings or supply a default value if a value cannot be converted to a target datatype. The default value used for date is 1-JAN-0001 .
/TrimWhiteSpace		Remove trailing whitespace from varchars
/TrimTime	<i>BOOL</i>	Trim time when converting Oracle and SQL Server datatypes to the Ingres date datatype. Value YES means the result has a date but no time component. Value NO means the result has both date and time component. The default is that the time component is trimmed if the time is midnight (00:00:00).
/MapEmptyStringToSpace		Convert empty Ingres or SQL Server varchar values to an Oracle varchar2 containing a single space and vice versa.
/MapEmptyDateToConstant		Convert an empty Ingres date to 31-DEC-2383 when replicating to a different DBMS.

6.4. ColumnProperties

DESCRIPTION

Action **ColumnProperties** defines properties of a column (named by mandatory parameter **/Name**). The action itself has no effect other than that of its parameters. These parameters affect replication (capture and integration) as well as refresh and compare.

PARAMETERS

Parameter	Argument	Description
/Name	<i>col_name</i>	Name of column in hvr_column catalog
/BaseName	<i>col_name</i>	Database column name differs from hvr_column catalog
/Extra		Column exists in database but not in hvr_column catalog. If a column has /Extra then its value is not captured and not read during refresh or compare. If the value is omitted then appropriate default value is used (null, zero, empty string, etc.).
/Absent		Column does not exist in database table. If no value is supplied with /CaptureExpression then an appropriate default value is used (null, zero, empty string, etc.). When replicating between two tables with a column that is in one table but is not in the other there are two options: either register the table in the HVR catalogs with all columns and add parameter /Absent ; or register the table without the extra column and add parameter /Extra . The first option may be slightly faster because the column value is not sent over the network.
/CaptureExpression	<i>sql_expr</i>	SQL expression for column value when capturing or reading. This value may be a constant value or an SQL expression. Possible SQL expressions include null , 5 or 'hello' . If <i>sql_expr</i> contains <i>{colname}</i> this is replaced with the actual value from that column of the replicated table. For Oracle and SQL Server a subselect can be supplied, for example (select descrip from lookup where id={id}).
/IntegrateExpression	<i>sql_expr</i>	SQL expression for column value when integrating. For possible values of <i>sql_expr</i> see parameter /CaptureExpression .
/NoUpdate		The value for this column should never be changed by HVR when integrating an update statement . This means for example that an expression supplied using /IntegrateExpression will only be evaluated during an insert statement. By default HVR integration will only change the column's value if the captured update also changed that value or parameters /IgnoreDuringCompare or DbCapture/NoBeforeUpdate , or DbIntegrate/DbProc are defined. Otherwise the column's value is preserved.
/IgnoreDuringCompare		Ignore values in this column during compare and refresh. Also during integration this parameter means that this column is overwritten by every update statement, rather than only when the captured update changed this column.
/Datatype	<i>data_type</i>	Datatype in database if it differs from hvr_column catalog
/Length	<i>int</i>	String length in database if /Length is defined in hvr_column catalog.
/Precision	<i>int</i>	Integer precision in database if /Precision is defined in hvr_column catalog.
/Scale	<i>int</i>	Integer scale in database if /Scale is defined in hvr_column catalog.
/Nullable		Nullability in database if /Nullable is defined in hvr_column catalog.
/Identity [SQL Server]		Column has SQL Server identity attribute.
/Key		Add column to table's replication key.

RESTRICTIONS

Parameters **/BaseName**, **/Extra** and **/Absent** cannot be used together.

Parameters **/Extra** and **/Absent** cannot be used on columns which are part of the replication key. They cannot both be defined in a given database on the same column, nor can either be combined on a column with parameter **/BaseName**.

Parameter **/Length**, **/Precision**, **/Scale**, **/Nullable** and **/Identity** can only be used if parameter **/Datatype** is defined.

EXAMPLES

For parameter **/IgnoreDuringCompare** see channel **hvr_demo07** in **\$HVR_HOME/demo**.

See also channel **hvr_demo24** and **hvr_demo26** in **\$HVR_HOME/demo**.

6.5. Restrict

DESCRIPTION

Action **Restrict** defines that only rows that satisfy a certain condition should be replicated. The restriction logic is enforced during capture and integration and also during compare and refresh.

PARAMETERS

Parameter	Argument	Description
/CaptureCondition	<i>sql_expr</i>	<p>Only rows where this condition is TRUE are captured. If pattern <code>{col_name}</code> occurs in <i>sql_expr</i> it is replaced with the actual value from that column of the replicated row. Additionally, pattern <code>{hvr_cap_loc}</code> is replaced with the name of the capture location. The capture condition is embedded inside the trigger-based capture procedures.</p> <p>This parameter does 'update conversion'. Update conversion is when (for example) an update changes a row which did satisfy a condition into a row that does not satisfy the condition. Such an update would be captured as a delete. If only the value satisfied the condition then the update would be converted to a delete instead. Parameter DbCapture /IgnoreCondition has a similar effect to this parameter but does not do update conversion.</p>
/IntegrateCondition	<i>sql_expr</i>	<p>Integrate restrict condition. Only rows where this condition evaluates as TRUE are integrated. Restriction on both the capture side and integration side is unnecessary; restriction on the capture side is generally more efficient. If pattern <code>{col_name}</code> occurs in <i>sql_expr</i> it is replaced with the actual value from that column of the replicated row. If pattern <code>{hvr_cap_loc}</code> or <code>{hvr_cap_tstamp}</code> occurs they are replaced with the capture location or timestamp respectively.</p> <p>This parameter does 'update conversion'. Update conversion is when (for example) an update changes a row which did satisfy a condition into a row that does not satisfy the condition. If only the value satisfied the condition then the update would be converted to a delete instead. Such an update would be integrated as a delete.</p>
/RefreshCondition	<i>sql_expr</i>	<p>Only rows where this condition evaluates as TRUE are refreshed or compared. For compare this parameter controls which rows are selected for comparison (it can be defined on both databases or just on one). For refresh it can also be defined both on the source or on the target side. If defined on the source side it affects which rows are selected for refreshing. If defined on the target side during bulk refresh it protects non-matching rows from bulk delete. If defined for row-wise refresh it prevents some rows being selected for comparison with the source rows.</p> <p>If parameter <code>{hvr_opposite_loc}</code> occurs in the condition for the source database it is replaced with the target location name and if it occurs in the condition for the target database it is replaced with the source location name. This parameter can also be used to make compare and refresh aware of horizontal partitioning.</p>
/RefreshWithGroup	<i>grp_name</i>	<p>Only apply restrict condition if the opposite location is a member of location group <i>grp_name</i>. Normally if a refresh restriction is defined on a location then it is used during compare and refresh with any other location. But if a pair of locations contain a subset of data which is in the main location, then this parameter could say that the restrict is only needed when comparing the pair's data with the main location but not with each other.</p>

Parameter	Argument	Description
/HorizColumn	<i>col_name</i>	<p>Horizontal partitioning column. The contents of the column of the replicated table is used to determine the integrate address. If parameter /HorizLookupTable is also defined then the capture will join using this column to that table. If it is not defined then the column's value will be used directly as an integrate address. An integrate address can be one of the following:</p> <ul style="list-style-type: none"> • A integrate location name, such as dec01. • A location group name containing integrate locations, such as DECEN. • An alias for an integrate location, defined with /AddressSubscribe (see below), for example 22. • A pattern to match one of the above, such as dec*. • A list of the above, separated by a semicolon, colon or comma, such as cen,22. <p>This parameter should be defined with DbCapture.</p>
/HorizLookupTable	<i>tbl_name</i>	<p>Lookup table for value in column specified by parameter /HorizColumn. The lookup table should have a column which has the name of the /HorizColumn parameter. It should also have a column named hvr_address. The capture logic selects rows from the lookup table and for each row found stores the change (along with the corresponding hvr_address) into the capture table. If no rows match then no capture is done. And if multiple rows match then the row is captured multiple times (for different destination addresses).</p>
/DynamicHorizLookup		<p>Dynamic replication of changes to lookup table. Normally only changes to the horizontally partitioned table are replicated. This parameter causes changes to the lookup table to also trigger capture. This is done by creating extra rules/triggers which fire when the lookup table is changed. These rules/triggers are name tbl_li, tbl_lu, tbl_ld.</p> <p>Changes are replicated in their actual order, so for example if a transaction inserts a row to a lookup table and then a matching row to the main replicated table then perhaps the lookup table's insert would not cause replication because it has no match (yet). But the other insert would trigger replication (because it now matches the lookup table row).</p> <p>This dynamic lookup table replication feature is suitable if the lookup table is dynamic and there are relatively few rows of the partitioned replicated table for each row of the lookup table. But if for example a huge table is partitioned into a few sections which each correspond to a row of a tiny lookup table then this dynamic feature could be expensive because an update of one row of the lookup table could mean millions of rows being inserted into the capture table. A more efficient alternative could be to perform an HVR Refresh whenever the lookup table is changed and use parameter /RefreshCondition with pattern {hvr_opposite_loc} in the condition so that the refresh is aware of the partitioning.</p>
/AddressTo	<i>addr</i>	<p>Captured changes should only be sent to integrate locations that match integrate address <i>addr</i>. The address can be one of the following:</p> <ul style="list-style-type: none"> • An integrate location name, such as dec01. • A location group name containing integrate locations, such as DECEN. • An alias for an integrate location, defined with /AddressSubscribe (see below), for example 22 or Alias7. • A pattern to match one of the above, such as dec*. • A column name enclosed in braces, such as {mycol}. The contents of this column will be used as an integrate address. This is similar to parameter /HorizColumn. • A list of the above, separated by a semicolon, colon or comma, such as cen,{col3}. <p>This parameter should be defined with DbCapture or FileCapture.</p>

Parameter	Argument	Description
/AddressSubscribe	<i>addr</i>	<p>This integrate location should be sent a copy of any changes that match integrate address <i>addr</i>. The address can be one of the following:</p> <ul style="list-style-type: none"> • A different integrate location name, such as dec01. • A location group name containing other integrate locations, such as DECEN. • A pattern to match one of the above, such as dec*. • An alias to match an integrate address defined with /AddressTo or /HorizColumn or matched by {hvr_address} in FileCapture /Pattern. An alias can contain numbers, letters and underscores, for example 22 or Alias7. • A list of the above, separated by a semicolon, colon or comma, such as dec*,CEN. <p>This parameter should be defined with DbIntegrate or FileIntegrate.</p>

HORIZONTAL PARTITIONING

Horizontal partitioning means that different parts of a table should be replicated into different directions. Logic is added inside capture to calculate the destination address for each change, based on the row's column values. The destination is put in a special column of the capture table named [hvr_address](#). Normally during routing each capture change is sent to all other locations which have a [DbIntegrate](#) action defined for that row, but this [hvr_address](#) column overrides this. The change is sent instead to only the destinations specified.

Column [hvr_address](#) can contain a location name, a location group name (UPPERCASE) or an asterisk (*). An asterisk means send to all locations with [DbIntegrate](#) defined. It can also contain a comma separated list of the above.

When necessary horizontal partitioning will 'convert' updates. For example, an update to a lookup column or a lookup table could lead to a delete being replicated to one database and an insert replicated to another.

RESTRICTIONS

Parameter [/RefreshWithGroup](#) can only be used with [/RefreshCondition](#).

EXAMPLE

To replicate only rows of table product use parameters [/CaptureCondition](#) and [/RefreshCondition](#). Also only rows of table order for products which are in state 16 need to be captured. This is implemented with another [/CaptureCondition](#) parameter.

[hvr_action](#)

chn_name	grp_name	tbl_name	act_name	act_parameters
mychannel	CAP	product	Restrict	/CaptureCondition ="{id}> 1000000 and {id} < 2000000" /RefreshCondition ="{id}> 1000000 and {id} < 2000000"
mychannel	CAP	order	Restrict	/CaptureCondition ="exists (select 1 from product where status = 16 and prod_id = {prod_id})"

See [hvr_demo26](#) in [\\$HVR_HOME/demo](#).

6.6. CollisionDetect

DESCRIPTION

Action **CollisionDetect** causes HVR's integration to detect 'collisions', i.e. when a target row has been changed after the change that is being applied was captured. Collisions can happen during bi-directional replication; if the same row is changed to different values in databases A and B so quickly that there is no time to replicate the changes to the other database then there is danger that the change to A will be applied to B while the change to B is on its way to A. Collisions can also happen without bi-directional replication; if changes are made first to A and then to B, then the change to B could reach database C before the change from A reaches C. Undetected collisions can lead to inconsistencies; the replicated database will become more different as time passes.

The default behavior for **CollisionDetect** is automatic resolution using a simple rule; the most recent change is kept and the older changes discarded. The timestamps used have a granularity of one second; if the changes occur in the same second then one arbitrary location (the one whose name sorts first) will 'win'. Parameters are available to change this automatic resolution rule and to tune performance.

Collision detection requires extra timestamp information for each tuple which is typically held in a special history table (named *tbl_h*). This table is created and maintained by both capture and integration for each replicated table. The old rows in this history table are periodically purged using timestamp information from the integrate receive timestamp table (see also section [7.4 Integrate Receive Timestamp Table](#)).

PARAMETERS

Parameter	Argument	Description
/TreatCollisionAsError		Treat a collision as an error instead of performing automatic resolution using the 'first wins' rule. If DbIntegrate /OnErrorSaveFailedRow is defined then the collision will be written to the fail table and the integration of other changes will continue. If DbIntegrate /OnErrorBlockLocation is defined then the location from which this change originated will be blocked. If neither of these are defined then the integrate job will keep failing until the collision is cleared, either by deleting the row from the history table or by deleting the transaction file in the HVR_CONFIG/router directory.
/TimestampColumn	<i>col_name</i>	Exploit a timestamp column named <i>col_name</i> in the replicated table for collision detection. By relying on the contents of this column collision detection can avoid the overhead of updating the history table. Deletes must still be recorded. One disadvantage of this parameter is that collision handling relies on this column being filled accurately by the application. Another disadvantage is that if there are more than one database where changes can occur then if a change occurs in the same second, then collision cannot be detected properly.
/IntegerTimestamp		The timestamp column named by parameter /TimestampColumn is an integer (number of seconds since 1-Jan-1970 GMT) rather than a date datatype.
/NoHistoryTable		Detect collisions without a history table. This can only be used if parameter /TimestampColumn is defined. Detection overhead is further reduced but additional types of collisions (involving deletes from multiple databases) can go undetected. But for many tables detecting collisions involving deletes is not important because deletes only occur in big batches (purges) and because after a row is deleted the key value is never reused.
/AutoHistoryPurge		Delete rows from history table once the receive stamp table indicates that they are no longer necessary for collision detection. These rows can also be deleted using command hvrhistorypurge .

RESTRICTIONS

Parameters `/IntegerTimestamp` and `/NoHistoryTable` can only be used if `/TimestampColumn` is defined.

EXAMPLES

See channels `hvr_demo11` and `hvr_demo23` in `$HVR_HOME/demo`.

6.7. DbObjectGeneration

DESCRIPTION

Action **DbObjectGeneration** allows control over the database objects which are generated by HVR in the replicated databases. The action has no effect other than that of its parameters.

Parameters **/NoCapture*** can either be used to inhibit capturing of changes for trigger-based capture or can be used with parameter **/IncludeSqlFile** to replace the procedures that HVR would normally generate with new procedures containing special logic.

PARAMETERS

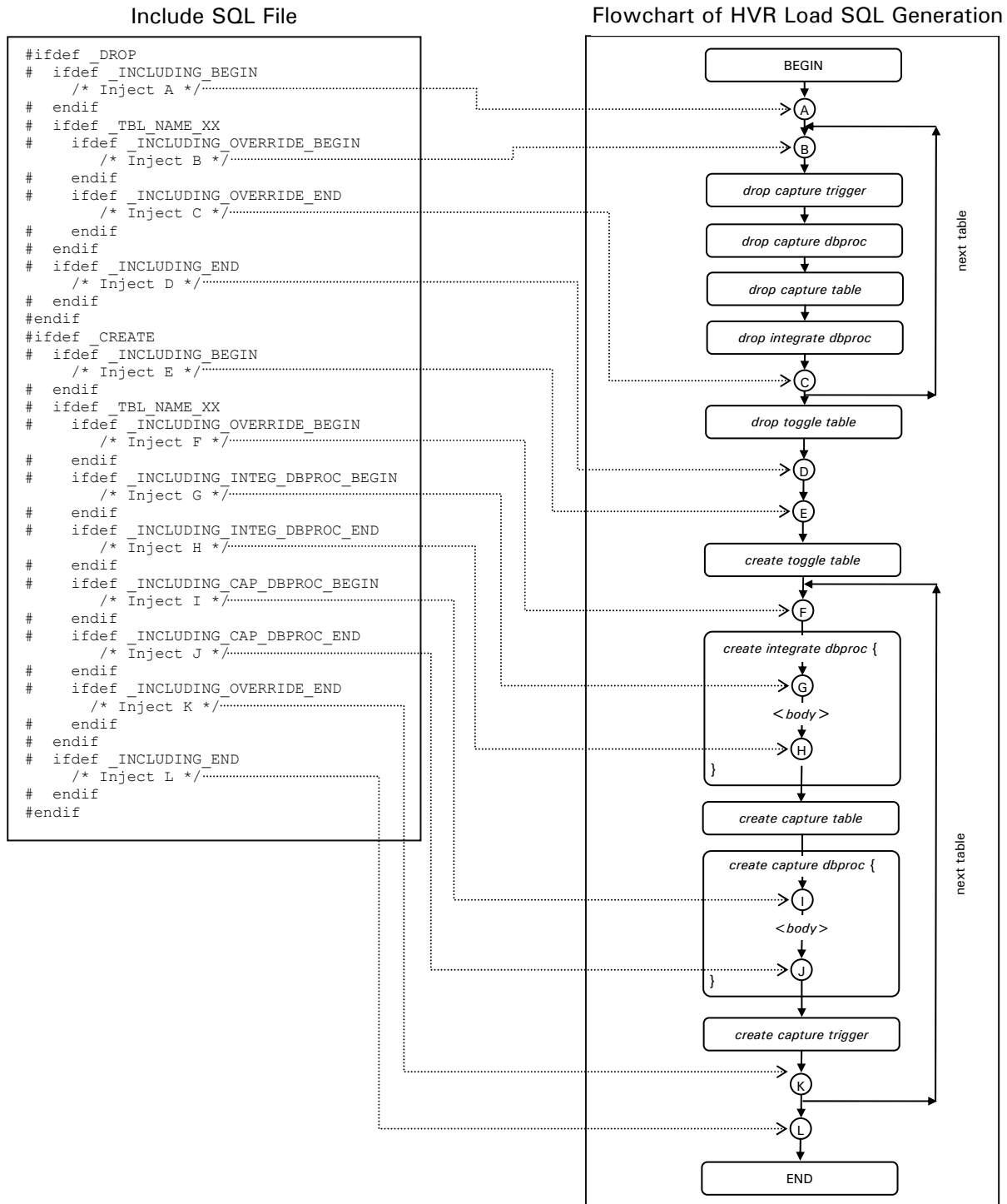
Parameter	Argument	Description
/NoCaptureInsertTrigger		Inhibit generation of capture insert trigger/rule.
/NoCaptureUpdateTrigger		Inhibit generation of capture update trigger/rule.
/NoCaptureDeleteTrigger		Inhibit generation of capture delete trigger/rule.
/NoCaptureDbProc		Inhibit generation of capture database procedures.
/NoIntegrateDbProc		Inhibit generation of integrate database procedures.
/IncludeSqlFile	<i>file</i>	Include file for customizing database objects. Argument <i>file</i> can be an absolute pathname or a relative path in a directory specified with /IncludeSqlDirectory . Option -S of hvrload can be used to generate the initial contents for this file.
/IncludeSqlDirectory	<i>dir</i>	Search directory <i>dir</i> for include SQL file.
/CaptureTableCreateClause	<i>sql_expr</i>	Clause for capture table creation statement.
/StateTableCreateClause	<i>sql_expr</i>	Clause for state table creation statement.
/FailTableCreateClause	<i>sql_expr</i>	Clause for fail table creation statement.
/HistoryTableCreateClause	<i>sql_expr</i>	Clause for history table creation statement.
/GrantToPublic		Allow all users to access HVR database objects.
Ingres		

INJECTING SQL INCLUDE FILES

Parameter **/IncludeSqlFile** can be used to inject special logic inside standard SQL which is generated by **hvrload**. The SQL that HVR would normally generate can be seen with **hvrload** option **-S**. Conditions (using **#ifdef** syntax lent from the C preprocessor), control where abouts this SQL is injected. There are twelve inject points (see diagram below). SQL code will be injected at a specific point depending on the **#ifdef** conditions specified for macros **_INCLUDING_***, **_CREATE**, **_DROP** and **_TABLE_NAME_***. If a file contains none of these conditions then its content will be injected in all twelve injections points.

These sections will not always be generated:

- ❖ Triggers are only generated for trigger-based capture locations (**/LogBased** not defined)
- ❖ Integrate database procedures are only defined if **DbIntegrate/DbProc** is defined.
- ❖ The **_DROP** section is omitted if **hvrload** option **-c** is defined without **-d**.
- ❖ The **_CREATE** section is omitted if **hvrload** option **-d** is defined without **-c**.
- ❖ Sections for specific tables are omitted if **hvrload** option **-t** is specified for different tables.
- ❖ Database procedures are only generated if **hvrload** option **-op** is defined or no **-o** option is supplied.
- ❖ Database procedures and triggers are only generated if option **-ot** is defined or no **-o** option is supplied.



The following macros are defined by **hvrload** for the contents of the file specified by parameter **/IncludeSqlFile**. These can also be used with **#if** or **#ifdef** directives.

Macro	Description
_CREATE	Defined when hvrload option -c is supplied or -d is not supplied.
_DROP	Defined when hvrload option -d is supplied or -c is not supplied.
_FLAG_OC	Defined when hvrload option -oc or no -o option is supplied.
_FLAG_OP	Defined when hvrload option -op or no -o option is supplied.
_FLAG_OS	Defined when hvrload option -os or no -o option is supplied.
_FLAG_OT	Defined when hvrload option -ot or no -o option is supplied.
_HVR_VER	HVR version number.

Macro	Description
<code>_HVR_OP_VAL</code>	Defined when <code>_INCLUDING_INTEG_DBPROC_*</code> is defined with value 0, 1 or 2. It means the current database procedure is for delete, insert or update respectively.
<code>_INCLUDING_BEGIN</code>	Defined when <code>hvrload</code> is including the SQL file at the beginning of its SQL.
<code>_INCLUDING_END</code>	Defined when <code>hvrload</code> is including the SQL file at the end of its SQL.
<code>_INCLUDING_CAP_DBPROC_BEGIN</code>	Defined when <code>hvrload</code> is including the SQL file at the beginning of each capture database procedure.
<code>_INCLUDING_CAP_DBPROC_END</code>	Defined when <code>hvrload</code> is including the SQL file at the end of each capture database procedure.
<code>_INCLUDING_INTEG_DBPROC_BEGIN</code>	Defined when <code>hvrload</code> is including the SQL file at the beginning of each integrate database procedure.
<code>_INCLUDING_INTEG_DBPROC_END</code>	Defined when <code>hvrload</code> is including the SQL file at the end of each integrate database procedure.
<code>_INCLUDING_OVERRIDE_BEGIN</code>	Defined as <code>hvrload</code> is including the SQL file at a point where database objects can be dropped or created. Each SQL statement in this section must be preceded by macro <code>_SQL_BEGIN</code> and terminated with macro <code>_SQL_END</code> .
<code>_INCLUDING_OVERRIDE_END</code>	Defined as <code>hvrload</code> is including the SQL file at a point where database objects can be dropped or created. Each SQL statement in this section must be preceded by macro <code>_SQL_BEGIN</code> and terminated with macro <code>_SQL_END</code> .
<code>_INGRES</code>	Defined when the current location is an Ingres database.
<code>_LOC_DBNAME</code>	Database name.
<code>_LOC_NAME</code>	Name of current location.
<code>_ORACLE</code>	Defined when the current location is an Oracle database.
<code>_TBL_NAME_X</code>	Indicates that a database procedure for table <code>x</code> is generated. This macro is only defined when <code>_INCLUDING_*_DBPROC_*</code> is defined.
<code>_SQL_BEGIN</code>	Macro marking the beginning of an SQL statement in a section for <code>_INCLUDING_OVERRIDE</code> .
<code>_SQL_END</code>	Macro marking the end of an SQL statement for an <code>_INCLUDING_OVERRIDE</code> section.
<code>_SQLSERVER</code>	Defined when the current location is an SQL Server database.

EXAMPLE

Ingres

The following uses action `DbObjectGeneration` to inject some special logic into the integrate database procedure for table `scenario2`. This logic changes the value of column `status` or deletes the target row if the status has a certain value.

Parameter `/DbProc` must also be added to action `DbIntegrate` so that integrate database procedures are generated.

```
#if defined _INCLUDING_INTEG_DBPROC_BEGIN && defined _TBL_NAME_SCENARIO2 && _HVR_OP_VAL == 2
if      :status = 'Status Two' then
        :status = 'Status Three';
elseif :status = 'Status Four' then
        :status = 'Status Five';
elseif :status = 'Status Six' then
        delete from scenario2 where id = :id;
        return;
endif;

#endif
```

hvr_action

chn_name	grp_name	tbl_name	act_name	act_parameters
mychannel	CAP	*	DbCapture	
mychannel	INTEG	*	DbIntegrate	/DbProc
mychannel	INTEG	scenario2	DbObjectGeneration	/IncludeSqlFile="inject.sql"

6.8. Agent

DESCRIPTION

An agent is a block of user-supplied logic which is executed by HVR during replication. An agent can be an Operating System command or a database procedure. Each time HVR executes an agent it passes parameters to indicate what stage the job has reached (e.g. start of capture, end of integration etc.).

PARAMETERS

Parameter	Argument	Description
<code>/Command</code>	<i>path</i>	Name of agent command. This can be a script or an executable. Scripts can be shell scripts on Unix and batch scripts on Windows or can be files beginning with a 'magic line' containing the interpreter for the script e.g. <code>#!/usr/bin/perl</code> or <code>#!/c:/perl/bin/perl.exe</code> . Argument <i>path</i> can be an absolute or a relative pathname. If a relative pathname is supplied the agents should be located in <code>\$HVR_HOME/bin</code> or in a directory specified with parameter <code>/Path</code> .
<code>/DbProc</code>	<i>dbproc</i>	Call database procedure <i>dbproc</i> during replication jobs. The database procedures are called in a new transaction; changes that do not commit themselves will be committed after agent invocation by the HVR job.
<code>/UserArgument</code>	<i>userarg</i>	Pass extra argument <i>userarg</i> to each agent execution.
<code>/ExecOnHub</code>		Execute agent on hub machine instead of location's machine.
<code>/Order</code>	<i>int</i>	Specify order of agent execution.
<code>/Path</code>	<i>dir</i>	Search directory <i>dir</i> for agent.

AGENT ARGUMENTS

If an agent is defined it is called several times at different points of the replication job. On execution the first argument that is passed indicates the position in the job, for example `cap_begin` for when the agent is called before capture. Argument *mode* is either `cap_begin`, `cap_end`, `integ_begin`, `integ_end`, `refr_read_begin`, `refr_read_end`, `refr_write_begin` or `refr_write_end` depending on the position in the replication job where the agent was called.

Modes `cap_end` and `integ_end` are passed information about whether data was actually replicated. Command agents can use `$HVR_TBL_NAMES` or `$HVR_FILE_NAMES` and database procedure agents can use parameter `hvr_changed_tables`. An exception if an integrate job is interrupted; the next time it runs it does not know anymore which tables were changed so it will set these variables to an empty string or `-1`.

Agents specified with `/Command` are called as follows:

```
agent mode chn_name loc_name userarg
```

Database procedure agents (specified with modifier `/DbProc`) are called as follows. The last parameter (`hvr_changed_tables`) specifies the number of tables that were changed.

```
Ingres
execute procedure agent (hvr_agent_mode='integ_end', hvr_chn_name='chn',
    hvr_loc_name='xx', hvr_agent_arg='hello', hvr_changed_tables=N);
```

```
Oracle
agent (hvr_agent_mode$=>'integ_end', hvr_chn_name$=>'chn', hvr_loc_name$=>'xx',
    hvr_agent_arg$=>'hello', hvr_changed_tables$=N);
```

```
SQL Server
execute agent @hvr_agent_mode='integ_end', @hvr_chn_name='chn', @hvr_loc_name='xx',
    @hvr_agent_arg='hello', @hvr_changed_tables=N;
```

AGENT ENVIRONMENT

An agent inherits the environment of its parent process. On the hub the parents' process is the HVR Scheduler. On a remote Unix machine it is the `inetd` daemon. On a remote Windows machine the parent is the HVR Remote Listener service. Differences with the environment of the parent process are as follows:

- ❖ Environment variable `$HVR_TBL_NAMES` is set to a colon-separated list of tables for which the job is replicating (for example `HVR_TBL_NAMES=tbl1:tbl2:tbl3`). Also variable `$HVR_BASE_NAMES` is set to a colon-separated list of table 'base names', which are prefixed by a schema name if `/Schema` is defined (for example `HVR_BASE_NAMES=base1:sch2.base2:base3`). For modes `cap_end` and `integ_end` these variables are restricted to only the tables actually processed.
- ❖ For database locations, environment variable `$HVR_LOC_DB_NAME`, `$HVR_LOC_DB_USER` (unless no value is necessary) and `$II_SYSTEM` are set and `$II_SYSTEM/ingres/bin` is added to the path.
- ❖ For Oracle locations, the environment variables `$HVR_LOC_DB_USER`, `$ORACLE_HOME` and `$ORACLE_SID` are set and `$ORACLE_HOME/bin` is added to the path.
- ❖ For SQL Server locations, the environment variables `$HVR_LOC_DB_SERVER`, `$HVR_LOC_DB_NAME`, `$HVR_LOC_DB_USER` and `$HVR_LOC_DB_PWD` are set (unless no value is necessary).
- ❖ For file locations variables `$HVR_FILE_LOC` and `$HVR_LOC_STATEDIR` are set to the file location's top and state directory respectively. For modes `cap_end` and `integ_end` variable `$HVR_FILE_NAMES` is set to a colon-separated list of replicated files, unless this information is not available because of recovery. If an extremely large number of files are replicated then this value could be abbreviated and suffixed with "...".
- ❖ Any variable defined by action `Environment` is also set in the agent's environment.
- ❖ The current working directory for file locations is the top directory of the file location and for database locations it is `$HVR_CONFIG`, except for agents with `/ExecOnHub` which are executed in the job's working directory (under `$HVR_CONFIG/work`).
- ❖ `stdin` is closed and `stdout` and `stderr` are redirected (via network pipes) to the job's logfiles.

If a command agent encounters a problem it should write an error message and return with exit code 1. If the agent does not want to do anything for a mode or does not recognize the mode (new modes may be added in future HVR versions) then the agent should return exit code 2, without writing an error message.

RESTRICTIONS

Exactly one of the following parameters must be defined: `/Command`, `/DbProc` and `/Path`.

Parameter `/Path` cannot be combined with `/UserArgument`, `/ExecOnHub` or `/Order`.

EXAMPLE

```
#!/perl

# Exit codes: 0=success, 1=error, 2=ignore_mode

if($ARGV[0] eq "integ_end") {
    print "Hello World\n";
    exit 0;
}
else {
    exit 2;
}
```

6.9. Environment

DESCRIPTION

This action sets an operating system environment variable for the HVR process which connects to the affected location. It also affects an agent called for this location.

PARAMETERS

Parameter	Argument	Description
/Name	<i>env_var</i>	Name of the environment variable.
/Value	<i>path</i>	Value of the environment variable.

EXAMPLES

Variable	Description
HVR_DOUBLE_PRECISION	Accuracy of double floating point numbers. Used when converting double floats to other datatypes. The default is 13. A higher value can be used on hardware whose double floating point precision exceeds the IEEE standard.
HVR_FLOAT_PRECISION	Accuracy of floating point numbers. Used when converting floats to other datatypes. The default is 6. A high value can be used for hardware whose floating point precision exceeds the IEEE standard.
HVR_SORT_BYTE_LIMIT	Amount of memory to use before sorting large data volumes in temporary files. The default limit is 67108864.
HVR_SORT_COMPRESS	When set to value 1 the sorting of large amounts of data will be compressed on the fly to reduce disk room.
HVR_SORT_ROW_LIMIT	Number of rows to keep in memory before sorting large amounts of data using temporary files. The default limit is 1038336.
NLS_LANG	Used so that HVR's environment matches an Oracle database's instance nls_charset.

6.10. FileCapture

DESCRIPTION

Action **FileCapture** instructs HVR to capture files from a file location's directory.

PARAMETERS

Parameter	Argument	Description
/DeleteAfterCapture		Delete file after capture, instead of capturing recently changed files. If this parameter then the channel moves files from the location. Without it, the channel copies files if they are new or modified.
/Pattern	<i>pattern</i>	<p>Only capture files whose names match <i>pattern</i>. Possible patterns are:</p> <ul style="list-style-type: none"> • '*.c' - Files ending with .c • 'a?b[de]' - Files with first letter a, third letter b and fourth letter d or e. • '*.lis *.xml' - Files ending with .lis or .xml • '**/*txt' - Search subdirectories recursively for file ending with *txt. • If a pattern contains a name in braces (such as '{office}_*.txt') then this matches the same as an asterisk ('*_*.txt') but also sets property {office} for a rename expression (see parameter /RenameExpression of action FileIntegrate). • If a pattern contains {hvr_address} then the file is only replicated to integrate locations specified by the matching part of the filename. Locations can be specified as follows: <ul style="list-style-type: none"> ○ An integrate location name, such as dec01. ○ A location group name containing integrate locations, such as DECEN. ○ An alias for an integrate location, defined with Restrict /AddressSubscribe, for example 22 or Alias7. • A list of the above, separated by a semicolon, colon or comma, such as cen,dec01.
/IgnorePattern	<i>pattern</i>	Ignore files whose names match <i>pattern</i> .
/IgnoreUnterminated	<i>pattern</i>	Ignore files whose last line does not match <i>pattern</i> . This ensures that incomplete files are not captured.
/AccessDelay	<i>secs</i>	Delay read for <i>N</i> seconds to ensure writing is complete.

NOTES

Bi-directional replication (replication in both directions with changes happening in both file locations) is not currently supported.

6.11. FileIntegrate

DESCRIPTION

Action **FileIntegrate** instructs HVR to integrate files into a file location's directory.

PARAMETERS

Parameter	Argument	Description
/RenameExpression	<i>expression</i>	Expression to name new files, containing substitutions. A rename expression can contain constants strings with 'brace substitutions' such as {hvr_cap_loc} , {hvr_cap_tstamp} and {hvr_integ_tstamp} . If the new file was captured from a file location then substitutions {hvr_cap_subdirs} and {hvr_cap_filename} are also allowed, as well any values set using brace patterns (e.g. '{office}_*.txt'). If no rename expression is defined then new files are named '{hvr_cap_subdirs}/{hvr_cap_filename}' if they were captured from another file location and '{hvr_integ_tstamp}.xml' if they are for database changes and channel is defined with tables.
/ErrorOnOverwrite		Error if a new file has same name as an existing file.
/MaxFileSize	X	Limit each XML file to no more than X bytes. This parameter cannot be used for 'blob' file channels which contain no table information and only replicated files as 'blobs'. When this size is reached, HVR will start writing rows to a new file whose name is found by re-evaluating parameter /RenameExpression (or {hvr_integ_tstamp}.xml if that parameter is not specified). XML files written by HVR always contain at least one row, which means that specifying a number between 1 and 500 will cause each file to contain a single row. Note that for efficiency reasons HVR's decision to start writing a new file uses the XML length of the previous row, not the current row. This means that sometimes the actual file size may slightly exceed the value specified.
/Journal		Move processed transaction files to journal directory \$HVR_CONFIG/jnl/hub/chn/YYYYMMDD on the hub machine. Normally an integrate job would just delete its processed transactions files. The journal files are compressed, but their contents can be viewed using command hvrrouterview with option -F .

6.12. FileFormat

DESCRIPTION

Action **FileFormat** defines the format of the files which HVR reads and writes into a file location.

If a channel contains table information then it can replicate both database and file location. In this case any changes replicated to the file location are integrated as records in XML format. The XML schema used by HVR can be found in `$HVR_HOME/lib/hvr.dtd`.

Alternatively a channel can contain only file locations and no table information. In this case each file captured is treated as a 'blob' and is replicated to the integrate file locations without HVR recognizing its format.

If such a 'blob' file channel is defined with only actions **FileCapture** and **FileIntegrate** (no parameters) then all files in the capture location's directory (including files in subdirectories) are replicated to the integrate location's directory. The original files are not touched or deleted, and in the target directory the original file names and subdirectories are preserved. New and changed files are replicated, but empty subdirectories and file deletions are not replicated.

PARAMETERS

Parameter	Argument	Description
<code>/HvrOpField</code>		<p>Defines that each change written to an XML file contains an <code>hvr_op</code> integer which specifies the change type. It also ensures that all change rows are written to the file. If this action is not defined then only inserts and updates value are shown, whereas deletes and 'before update' rows are hidden.</p> <p>Possible values of <code>hvr_op</code> are:</p> <ul style="list-style-type: none"> 0 : Delete 1 : Insert 2 : Values after update 3 : Values before a key update 4 : Values before a non-key update. <p>An update is replicated either as a single row (with <code>hvr_op=2</code>), or as a pair of rows (first <code>hvr_op=3</code> or <code>hvr_op=4</code>, then <code>hvr_op=2</code>).</p> <p>The field appears as a XML tag, for example: <code><column name="hvr_op">2</column></code>.</p>

6.13. LocationProperties

DESCRIPTION

Action [LocationProperties](#) defines properties of a remote location. This action has no affect other than that of its parameters.

PARAMETERS

Parameter	Argument	Description
/SslCertificate	<i>path</i>	Secure Socket Layer (SSL) network encryption. Ensures data send over network cannot be deciphered by third parties. Encryption relies on a private key on the hub and a public certificate on both the hub and the remote machine. These files are both generated by command hvrsslgen and are supplied to the remote hvr executable with options -C and -K . The argument <i>path</i> should be the absolute pathname of the public certificate on the remote machine.
/ThrottleKbytes	<i>int</i>	Restrain network bandwidth usage by grouping data sent to/from remote connection into packages, each containing <i>int</i> bytes, followed by a short sleep. The duration of the sleep is defined by /ThrottleMillisecs . Carefully setting these parameters will prevent HVR being an ‘anti-social hog’ of precious network bandwidth. This means it will not interfere with interactive end-users who share the link for example. For example if a network link can handle 64 KB/sec then a throttle of 32 KB with a 500 millisecond sleep will ensure HVR would be limited to no more than 50% bandwidth usage (when averaged-out over a one second interval).
/ThrottleMillisecs	<i>int</i>	Restrict network bandwidth usage by sleeping <i>int</i> millisecs between packets. See /ThrottleKbytes for setting the package size.
/StateDirectory	<i>path</i>	Directory for internal files used by for HVR file replication state. By default these files are created in subdirectory _hvr_state which is created inside the file location top directory.

RESTRICTIONS

If one of the parameters [/ThrottleMilliSecs](#) or [/ThrottleKbytes](#) is defined, then the other must also be defined.

6.14. Scheduling

DESCRIPTION

Action **Scheduling** controls how the replication jobs generated by **hvrload** will be scheduled by **hvrscheduler**. By default (if this **Scheduling** action is not defined) HVR schedules capture and integrate jobs to run continuously. This means after each replication cycle they will keep running and wait for new data to arrive. Other parameters also affect the scheduling of replication jobs, for example **DbCapture /ToggleFrequency**.

PARAMETERS

Parameter	Argument	Description
/CaptureStartTimes	<i>times</i>	Defines that the capture jobs should be triggered at the given times, rather than cycling continuously. For the format of <i>times</i> see START TIMES below.
/IntegrateStartAfterCapture		Defines that the integrate job should run after a capture job routes new data.
/IntegrateStartTimes	<i>times</i>	Defines that the integrate jobs should be triggered at the given times, rather than cycling continuously. For the format of <i>times</i> see START TIMES below.
/RefreshStartTimes	<i>times</i>	Defines that the refresh jobs should be triggered at the given times. By default they must be trigger manually. For the format of <i>times</i> see START TIMES below.

START TIMES

Argument *times* uses a format that closely resembles the format of Unix's **crontab** and is also used by scheduler attribute **trig_crono**. It is composed of five integer patterns separated by spaces. These integer patterns specify:

- ❖ minute (0-59)
- ❖ hour (0-23)
- ❖ day of the month (1-31)
- ❖ month of the year (1-12)
- ❖ day of the week (0-6 with 0=Sunday)

Each pattern can be either an asterisk (meaning all legal values) or a list of comma-separated elements. An element is either one number or two numbers separated by a hyphen (meaning an inclusive range). All dates and times are interpreted using the local-time. Note that the specification of days can be made by two fields (day of the month and day of the week): if both fields are restricted (i.e. are not *), the job will be started when either field matches the current time.

EXAMPLE

/CaptureStartTimes="0 * * * 1-5" specifies that capture jobs should be triggered at the start of each hour from Monday to Friday.

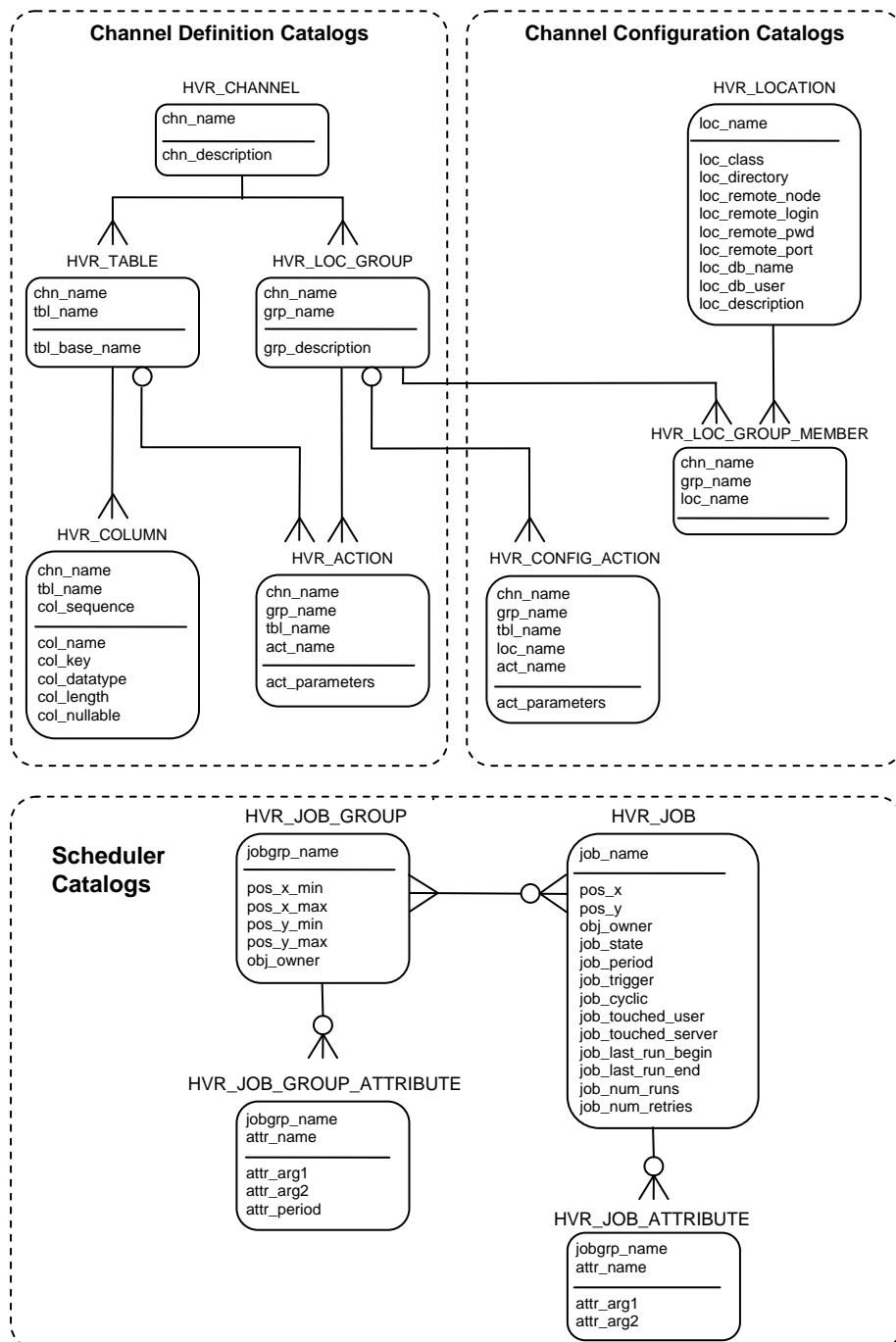
7. Objects Used Internally by HVR

This chapter describes the database objects HVR uses inside its [hub database](#) and also the [objects](#) it creates inside replicated databases.

7.1. Catalog Tables

The catalog tables are tables inside the hub database that contain a repository for information about what must be replicated. They are normally edited using the HVR GUI.

The HVR catalogs are divided into channel definition information (delivered by the developer) and location configuration information (maintained by the operator or the DBA). The HVR Scheduler catalogs hold the current state of scheduling; operators can control jobs by directly inserting, updating and deleting rows of these catalogs.



HVR CHANNEL

Column	Datatype	Optional?	Description
chn_name	String 12 characters	No	Unique name for channel. Used as the parameter by most HVR commands, and also as a component for naming jobs, database objects and files. For example an HVR capture job is named <i>chn-cap-loc</i> . Must be a lowercase identifier containing only alphanumerics and underscores. Because this value occurs so often in every logfile, program, database etc. it is recommended that this name be kept as small and concise as possible. Values hvr_* and system are reserved.
chn_description	String 200 characters	Yes	Description of channel.

HVR_TABLE

Column	Datatype	Optional?	Description
chn_name	string 12 characters	No	Name of channel to which this table belongs. Each table name therefore belongs to a single channel.
tbl_name	string 124 characters	No	Replication name for table. Typically this is the same as the name of the table in the database location, but it could differ. For example if the table's database name is too long or is not an identifier. It must be a lowercase identifier; an alphabetic followed by alphanumerics and underscores.
tbl_base_name	string 128 characters	Yes	Name of database table to which this replication table refers. If the table has different names in different databases then the specific value can also be set with action TableProperties /BaseName .

HVR_COLUMN

Column	Datatype	Optional?	Description						
chn_name	string 12 characters	No	Channel name.						
tbl_name	string 124 characters	No	Table name.						
col_sequence	number	No	Sequence of column in the table.						
col_name	string 128 characters	No	If the column has a different name in different databases, this value can be overridden with action ColumnProperties /BaseName .						
col_key	number	Yes	Is column part of the unique replication key? Value are 0 (column not in key) or 1 (column is in key). Key information is needed to replicate updates and deletes. The replication key does not have to match a physical unique index in the replicated table. If necessary, uniqueness can be achieved by defining all columns as part of the replication key. If this is still not a unique key then action TableProperties /DuplicateRows must be defined.						
col_datatype	string 38 characters	No	Datatype of column. Any database type can be used here, i.e. varchar , varchar2 , char , integer , integer4 , number or date .						
col_length	string 8 characters	Yes	The meaning of this column depends on the datatype: <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">String datatypes such as binary, byte, c, char, text, raw, varchar, varchar2</td> <td>Maximum length of string.</td> </tr> <tr> <td>Datatypes number and decimal</td> <td>Indicates scale and precision. Left of the decimal point is precision and right is scale. For example, value 3.2 indicates precision 3 and scale 2. Value -5.2 indicates precision 5 and scale -2.</td> </tr> <tr> <td>Other datatypes</td> <td>Unused.</td> </tr> </table>	String datatypes such as binary , byte , c , char , text , raw , varchar , varchar2	Maximum length of string.	Datatypes number and decimal	Indicates scale and precision. Left of the decimal point is precision and right is scale. For example, value 3.2 indicates precision 3 and scale 2. Value -5.2 indicates precision 5 and scale -2.	Other datatypes	Unused.
String datatypes such as binary , byte , c , char , text , raw , varchar , varchar2	Maximum length of string.								
Datatypes number and decimal	Indicates scale and precision. Left of the decimal point is precision and right is scale. For example, value 3.2 indicates precision 3 and scale 2. Value -5.2 indicates precision 5 and scale -2.								
Other datatypes	Unused.								
col_nullable	number	No	Is column datatype nullable? Values are 0 (not nullable) or 1 (nullable).						

HVR_LOC_GROUP

Column	Datatype	Optional?	Description
chn_name	string 12 characters	No	Name of channel to which this location group belongs.
grp_name	string 11 characters	No	Unique UPPERCASE identifiers used as name of location group. Should begin with an alphabetic and contain only alphanumerics and underscores.
grp_description	string 200 characters	Yes	Description of location group.

HVR_ACTION

Column	Datatype	Optional?	Description
chn_name	string 12 characters	No	Channel affected by this action. An asterisk "*" means all channels are affected.
grp_name	string 11 characters	No	Location group affected by this action. An asterisk "*" means all location groups are affected.
tbl_name	string 124 characters	No	Table affected by this action. An asterisk "*" means all tables are affected.
act_name	string 24 characters	No	Action name. See also section 6 Action Reference for available actions and their parameters.
act_parameters	string 1000 characters	Yes	Each action has a list of parameters which change that action's behavior. Each parameter must be preceded by a '/'. If an action takes an argument it is given in the form <code>/Param=arg</code> . Arguments that contain non-alphanumeric characters should be enclosed in double quotes (""). If an action needs multiple parameters they should be separated by a blank. For example action Restrict can have the following value for this column: <code>/CaptureCondition="{a} > 3"</code> .

HVR_LOCATION

Column	Datatype	Optional?	Description
loc_name	string 5 characters	No	A short name for each location. Used as a part of name of generated HVR objects as well as being used as an argument in various commands. A lowercase identifier composed of alphanumerics but may not contain underscores. Example: the location database in Amsterdam could be ams .
loc_class	string 10 characters	No	Class of location. Valid values are: ingres Ingres database. oracle Oracle database. sqlserver Microsoft SQL Server database. file File location.
loc_directory	string 200 characters	Yes	The meaning of this column depends on the contents of loc_class . ingres The value of \$II_SYSTEM ; it defaults to the setting on the hub machine. oracle The value of \$ORACLE_HOME ; it defaults to the setting on the hub machine. sqlserver Unused. file Top directory of file location. Pathnames can follow Unix or Windows conventions.
loc_remote_node	string 80 characters	Yes	Network name or IP address of the machine on which remote location resides. Only necessary for HVR remote connections.
loc_remote_login	string 24 characters	Yes	Login-name under which HVR slave process will run on remote machine. Only necessary for remote HVR connections.
loc_remote_pwd	string 128 characters	Yes	Password for login-name on remote machine. Only necessary for remote HVR connections. This column can be encrypted using command hvincryptdb .
loc_remote_port	number	Yes	TCP/IP port number for remote HVR connection. On Unix the inetd daemon must be configured to listen on this port. On Windows the HVR Remote Listener Service listens on this port itself. Only necessary for remote HVR connections.
loc_db_name	string 128 characters	Yes	The meaning of this column depends on the value of loc_class . ingres The Ingres database name. May contain an Ingres vnode (e.g. <code>vnode::db</code>), although a remote HVR connection should be more efficient over a network than Ingres/Net.

Column	Datatype	Optional?	Description
			<p>oracle The value of <code>\$ORACLE_SID</code>; it defaults to the value on the hub machine.</p> <p>sqlserver Value of the form <code>dbname</code> or <code>inst\dbname</code> where <code>dbname</code> is the name of the Microsoft SQL Server database and <code>inst</code> is the name of the SQL Server instance. May contain a node name (e.g. <code>node\inst\db</code>) although a remote HVR connection should be more efficient over a network than a SQL Server client/server connection.</p> <p>file Unused.</p>
loc_db_user	string 128 characters	Yes	<p>The meaning of this column depends on the value of <code>loc_class</code>. Passwords in this column can be encrypted using command <code>hvincryptdb</code>.</p> <p>ingres The DBA (owner) of the database. If specified, HVR connects to the location using Ingres <code>-u</code> flag and this value.</p> <p>oracle Value of the form the <code>[user]/[pwd]</code>, where <code>user</code> is the name of the Oracle user and <code>pwd</code> is the user's password. Both <code>user</code> and <code>pwd</code> are optional. If this column is empty, it defaults to the value used to connect to the hub database. May contain a TNS aliases (e.g. <code>user/pwd@node</code>) although a remote HVR connection should be more efficient over a network than SQL*Net.</p> <p>sqlserver Value of the form <code>[user]/[pwd]</code> where <code>user</code> is the name of the Microsoft SQL Server user and <code>pwd</code> is the user's password. This value can be encrypted using command <code>hvincryptdb</code>.</p> <p>file Unused.</p>
loc_description	string 200 characters	Yes	Description of location.

HVR_LOC_GROUP_MEMBER

Column	Datatype	Optional?	Description
chn_name	string 12 characters	No	Channel name for location group.
grp_name	string 11 characters	No	Name of location group defined in catalog <code>hvr_loc_group</code> .
loc_name	string 5 characters	No	Location belonging to this location group.

HVR_CONFIG_ACTION

Column	Datatype	Optional?	Description
chn_name	string 12 characters	No	Channel affected by this action. An asterisk '*' means all channels are affected.
grp_name	string 11 characters	No	Location group affected by this action. An asterisk '*' means all location groups are affected.
tbl_name	string 124 characters	No	Table affected by this action. An asterisk '*' means all tables are affected.
loc_name	string 5 characters	No	Location affected by this action. An asterisk '*' means all locations are affected.
act_name	string 24 characters	No	Action name. See also section 6 Action Reference for available actions and their parameters.
act_parameters	string 1000 characters	Yes	Each action has a list of parameters which change that action's behavior. Each parameter must be preceded by a '/'. If an action takes an argument it is given in the form <code>/Param=arg</code> . Arguments that contain non-alphanumeric characters should be enclosed in double quotes (""). If an action needs multiple parameters they should be separated by a blank. For example action Restrict can have the following value in this column: <code>/CaptureCondition="{a} > 3"</code> .

HVR_JOB

Column	Datatype	Optional?	Description
job_name	string 40 characters	No	Unique name of job. Case sensitive and conventionally composed of lowercase identifiers (alphanumerics and underscores) separated by hyphens. Examples: foo and foo-bar .
pos_x pos_y	number number	No No	X and Y coordinates of job in job space. The coordinates of a job determines within which job groups it is contained and therefore which attributes apply.
obj_owner	string 24 characters	No	Used for authorization: only the HVR Scheduler administrator and a job's owner may a change a jobs attributes or attributes.
job_state	string 10 characters	No	Valid values for cyclic jobs are PENDING , RUNNING , READY , HANGING , ERROR , ALERTING , FAILED , RETRY , CLEANING , SUSPEND and DISABLED are also allowed.
job_period	string 10 characters	No	Mandatory column indicating the period in which the job is currently operating. The job's period affects which job group attributes are effective. The typically value is normal .
job_trigger	number	Yes	0 indicates job is not triggered, 1 means it may run if successful, and 2 means it may run even if it is unsuccessful.
job_cyclic	number	Yes	0 indicates job is acyclic, and will disappear after running; 1 indicates job is cyclic.
job_touched_user	date	Yes	Last time user or hvrload (not hvrscheduler) changed job tuple.
job_touched_server	date	Yes	Last time hvrscheduler changed job tuple.
job_last_run_begin	date	Yes	Last time job was started.
job_last_run_end	date	Yes	Last time job finished running.
job_num_runs	number	Yes	Number of times job has successfully run.
job_num_retries	number	Yes	Number of retries job has performed since last time job successfully ran. Reset to zero after job runs successfully.

HVR_JOB_ATTRIBUTE

Column	Datatype	Optional?	Description
job_name	string 40 characters	No	Name of object on which attribute is defined.
attr_name	string 24 characters	No	Type of attribute. Case insensitive.
attr_arg1 attr_arg2	string 200 characters string 200 characters	Yes Yes	Some attribute types require one or more arguments, which are supplied in these columns.

HVR_JOB_GROUP

Column	Datatype	Optional?	Description
jobgrp_name	string 40 characters	No	Job group name. Case sensitive and conventionally composed of UPPERCASE identifiers (alphanumerics and underscores) separated by hyphens. Examples: FOO and FOO-BAR .
pos_x_min pos_x_max pos_y_min pos_y_max	number number number number	No No No No	These form coordinates of the job group's box in job space. Objects such as jobs, resources and other job groups whose coordinates fall within this box are contained by this job group and are affected by its attributes.
obj_owner	string 24 characters	Yes	Owner of a job group. Only a job group's owner and the HVR Scheduler administrator can make changes its coordinates or attributes.

HVR_JOB_GROUP_ATTRIBUTE

Column	Datatype	Optional?	Description
jobgrp_name	string 40 characters	No	Name of job group on which attribute is defined. These also affect objects contained in job group.
attr_name	string 24 characters	No	Type of attribute. Case insensitive.
attr_arg1 attr_arg2	string 200 characters string 200 characters	Yes Yes	Some attribute types require one or more arguments, which are supplied in these columns.
attr_period	string 10 characters	No	For which period does this attribute apply? Must be a lowercase identifier or an asterisks (*).

7.2. Naming of HVR Objects Inside Database Locations

The following table shows the database objects which HVR can create to support replication. The name of each database object either begins with an `hvr_` prefix or consists of a replicated table name followed by two underscores and a suffix.

Name	Description
<code>tbl__c0</code> <code>tbl__c1</code>	Capture tables (trigger-based capture only).
<code>tbl__ci</code> <code>tbl__cd</code> <code>tbl__cu</code>	Capture database rules or triggers.
<code>tbl__c</code> <code>tbl__c0</code> <code>tbl__c1</code>	Capture database procedures (trigger-based capture only).
<code>tbl__l</code> <code>tbl__li</code> <code>tbl__ld</code> <code>tbl__lu</code>	Database procedures, rules and triggers for capture of dynamic lookup table. Created for action <code>Restrict /DynamicHorizLookup</code> .
<code>hvr_togchn</code>	Capture toggle state table (trigger-based capture only).
<code>hvr_qtogchn</code>	Capture quick toggle state table (trigger-based capture only).
<code>hvr_lktogchn</code>	Capture toggle lock table (Ingres trigger-based capture only).
<code>hvr_seqchn</code>	Capture sequence number (Oracle trigger-based capture only).
<code>hvr_XXtbl</code>	Oracle supplemental logging group (Oracle log-based capture only).
<code>tbl__ii</code> <code>tbl__id</code> <code>tbl__iu</code>	Integrate database procedures. Created for action <code>DbIntegrate /DbProc</code> .
<code>tbl__f</code>	Integrate fail table. Created when needed, i.e. when an integrate error occurs.
<code>hvr_stinchn_loc</code>	Integrate receive timestamp table.
<code>hvr_stischn_loc</code>	Integrate commit frequency table.
<code>tbl__h</code>	Collision history table. Created if action <code>CollisionDetect</code> is defined.
<code>hvr_strrchn_loc</code>	Bulk refresh recovery state table.
<code>hvr_integrate</code>	Integrate role (Ingres only).

NOTES

- ❖ If a table has no non-key columns (i.e. the replication key consists of all columns) then some update objects (e.g. `tbl__iu`) may not exist.
- ❖ Capture objects are only created for trigger-based capture; log-based capture does not use any database objects.
- ❖ Action `DbObjectGeneration` can be used to inhibit or modify generation of these database objects.

7.3. Extra Columns for Capture, Fail and History Tables

Each capture, fail or history table created by HVR contains columns from the replicated table it serves, plus extra columns. These columns contain information about what the captured operation was and where it must be sent.

The following extra columns can appear in capture, fail or history tables:

Column	Datatype	Description
hvr_seq	Float or byte10 on Ingres, numeric on Oracle and timestamp on SQL Server.	Sequence in which capture triggers were fired. Operations are replayed on integrate databases in the same order, which is important for consistency.
hvr_tx_id	string	Transaction ID of captured change. This number is unique but may not be increasing.
hvr_tx_seq	string	Sequence number of transaction. For log-based capture this value can be mapped to the Ingres LSN or the Oracle SCN of the transaction's commit statement.
hvr_tx_countdown	number	Countdown of change within transaction, for example if a transaction contains three changes the first change would have countdown value 3, then 2, then 1. A value of zero indicates that commit information is missing for that change.
hvr_op	number	Operation type. Values are 0 (delete), 1 (insert), 2 (after update), 3 (before key update) or 4 (before non-key update). A key-update sometimes appears as a before update followed by an after update but is sometimes converted into a delete followed by an insert.
hvr_cap_loc	string	Name of location on which the change was captured.
hvr_cap_tstamp	date, or integer if action CollisionDetect / IntegerTimestamp is defined	Time stamp of capture change, for collision detection.
hvr_cap_user	string	Name of user who performed captured change.
hvr_address	string	Address of target location for change. Only set if action Restrict / HorizColumn is defined.
hvr_err_tstamp	date	Time at which integration error occurred. Written into fail table.
hvr_err_msg	long string	Integration error message written into fail table.
hvr_colval_mask	string	Mask showing which column values were missing or not updated. For example value ----m- could mean that log-based capture of an update is missing a value for the second last column of a table.

7.4. Integrate Receive Timestamp Table

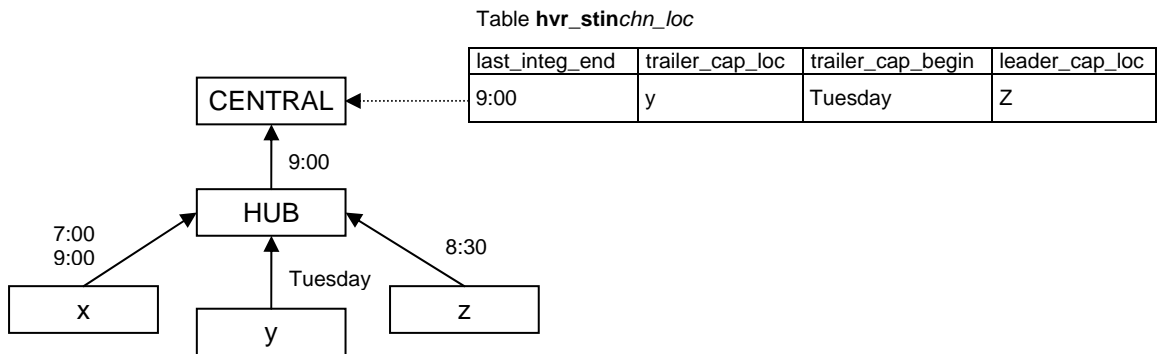
The receive stamp table in an integration database contains information about which captured changes have already been integrated. Because changes can be captured from more than one location the table contains information about the ‘leader’ and ‘trailer’ location. If there is only one capture location then that location is both the leader and the trailer.

The leader location is the capture location whose changes have arrived most recently. Column `recv_leader_code` contains the leaders’s location code and column `recv_leader` contains a time before which all changes captured on the leader are guaranteed to be already integrated. The trailer location is the location whose changes are oldest. Column `recv_trailer_code` contains the trailer’s location code and column `recv_trailer` contains a time before which all changes capture on the trailer are guaranteed to be already integrated. Receive timestamps are updated by HVR when the integrate jobs finishes running.

HVR accounts for the fact that changes have to be queued both in the capture database and then inside routing before being integrated. The receive stamp table is only updated if an arrival is guaranteed, so if a capture job was running at exactly the same time as an integrate job and the processes cannot detect whether a change ‘caught its bus’ then receive stamps are not reset. The receive stamp table is name `hvr_stinchn_loc`. It is created the first time the integrate jobs run. The table also contains columns containing the date timestamps as the number of seconds since 1970 1st Jan GMT.

EXAMPLE

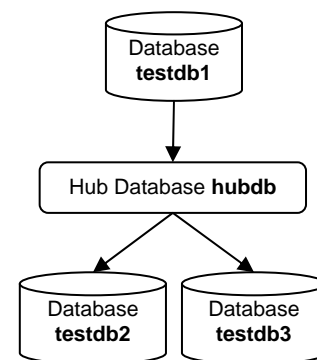
Data was last moved to the hub from location **x** at 7:00 and 9:00, from **y** on Tuesday, from **z** at 8:30 and from the hub to central at 9:00. For the central location the leader location is **z** and the trailer location is **y**. The contents of the integrate receive timestamp table is shown in the diagram below. Note that location **x** is not the leader because its job ran at the same time as the central job, so there is no guarantee that all data available at **x** has arrived.



Appendices

A: Quick Start for Ingres

This appendix shows how to set up an HVR channel (called [hvr_demo01](#)) to replicate between Ingres databases. The steps actually start by creating new databases and tables for HVR to replicate between. In real life these databases would already exist and be filled with the user tables, but for simplicity everything is created from scratch. Likewise these databases would normally be on different machines, but again for simplicity everything is just kept on the hub machine.



CREATE TEST DATABASES AND TABLES

Create three test databases, each containing two empty tables named [dm01_product](#) and [dm01_order](#). If replication is configured between existing databases and tables then this step should be skipped.

```

$ createdb testdb1
$ createdb testdb2
$ createdb testdb3

$ cd $HVR_HOME/demo/hvr_demo01/base/ingres
$ sql testdb1 < hvr_demo01.cre
$ sql testdb1 < hvr_demo01.mod
$ sql testdb2 < hvr_demo01.cre
$ sql testdb2 < hvr_demo01.mod
$ sql testdb3 < hvr_demo01.cre
$ sql testdb3 < hvr_demo01.mod

$ ckpdb +j testdb1
  
```

The last command (to checkpoint the source database) is needed so that if HVR log-based capture cannot find a change anymore in the Ingres log file it can always look into the Ingres journal files.

INSTALL HVR

First read section [1 Introduction](#) which explains the HVR's terminology and architecture. In particular this explains the importance of a hub database.

Then install the HVR software on the hub machine by following the installation steps in section [3.1](#) or [3.2](#). If the hub machine is a Unix machine then HVR must also be installed on a PC, so the HVR GUI can be run.

CREATE THE HUB DATABASE

Create the hub database, in which the HVR GUI will store the channel definition.

```

$ createdb hvrhub
  
```

CREATE THE CHANNEL

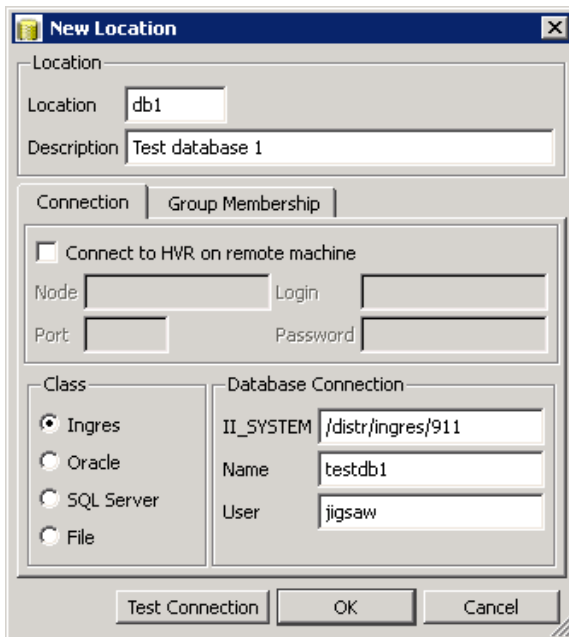
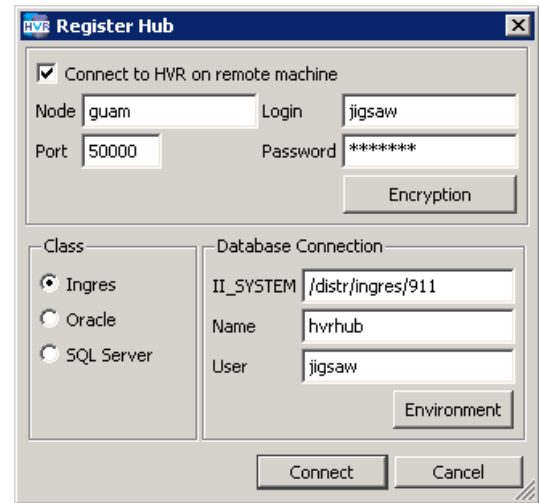
Start the HVR GUI on a PC by clicking on the HVR GUI icon (this is created the HVR Installer for Windows) or by running `hvrgui` on Linux.

First, Register the hub database: Right-click on **hub machines** > **Register hub**.

Enter connection details.

In this example the hub is a machine called **guam**, where an INET daemon is listening on port 50000. See section [3.1 New Installation on Unix or Linux](#) for how to do configure this.

For a new hub database a dialog will prompt “Do you wish to create the catalogs?”; answer **Yes**.



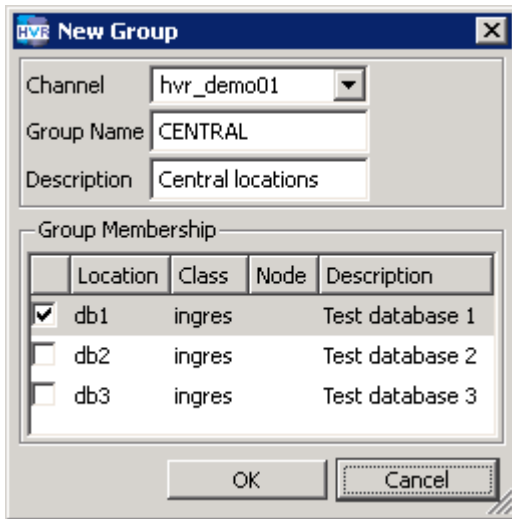
Next create three locations (one for each test database) using right-click on **Location Configuration** > **New Location**.

In this example there is no need to check **Connect to HVR on remote machine** because **testdb1** is on the same machine as the hub.

Ignore the **Group Membership** tab for now.

Make locations for **testdb2** and **testdb3** too.

Now define a channel using **Channel Definitions** > **New Channel**.



The channel needs two location groups. Under the new channel: right-click on **Location Groups** > **New Group**. Enter a group name (for instance CENTRAL).

Add location **db1** as a member of this group by checking the box for **db1**.

Then create a second location group, called **DECENTRAL** that has members **db2** and **db3**.

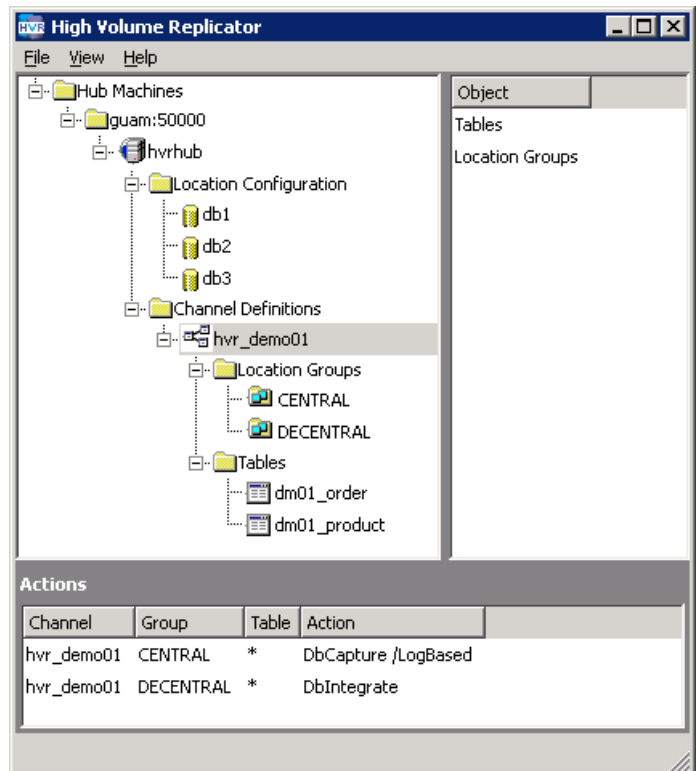
The new channel also needs a list of tables to replicate. This can be done as follows; Right-click on **Tables** > **Table Select**.

- Choose one of the three locations > **Connect**.
- In the **Table Selection window**, click on both tables and click **Add**.
- In new dialog **HVR Table Name** click **OK**.
- Close the **Table Selection** window.

Define Actions

Finally the new channel needs two actions to indicate the direction of replication.

- Right-click on group **CENTRAL** > **New Action** > **DbCapture**. Check **/LogBased** so that the channel will detect changes using the Ingres log file, instead of using Ingres database rules.
- Right-click on Group **DECENTRAL** > **New Action** > **DbIntegrate**.

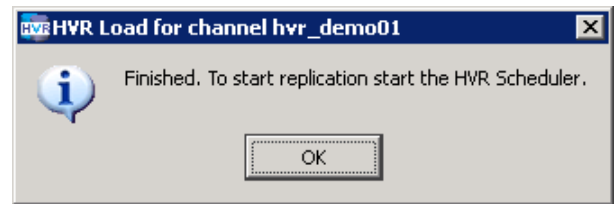


Note that the Actions pane only displays actions related the objects selected in the left hand pane. So click on channel **hvr_demo01** to see both actions.

ENABLE REPLICATION WITH HVR LOAD

Now that the channel definition is complete, create the runtime replication system.

Right-click on channel `hvr_demo01` > **HVR Load**. Choose **Create objects only** and click **HVR Load**.



From the moment that HVR Load is done all changes to database `testdb1` will be captured by HVR when its capture job looks inside the Ingres logging.

HVR Load also creates three replication jobs, which can be seen by querying the hub database;

```
$ sql hvrhub
* select job_name, job_state from hvr_job
* \g
```

job_name	job_state
hvr_demo01-cap-db1	PENDING
hvr_demo01-integ-db2	PENDING
hvr_demo01-integ-db3	PENDING

(3 rows)

START SCHEDULING OF REPLICATION JOBS

Start the Scheduler on the hub machine:

```
$ hvrscheduler hvrhub
```

Next, instruct the HVR Scheduler to un-suspend (activate) the replication jobs. Note that the channel name `HVR_DEMO01` is in uppercase.

```
$ hvrsuspend -u hvrhub HVR_DEM001
```

```
hvrsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEM001-CAP.
hvrsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEM001-INTEG.
hvrsuspend: Finished. (elapsed=0.73s)
```

The replication jobs inside the Scheduler each execute a script under `$HVR_CONFIG/job/hvrhub/hvr_demo01` that has the same name as the job. So job `hvr_demo01-cap-db1` detects changes on database `testdb1` and stores these as transaction files on the hub machine. The other two jobs (`hvr_demo01-integ-db2` and `hvr_demo2-integ-d3`) pick up these transaction files, and perform inserts, updates and deletes on the two target database.

TEST REPLICATION

To test replication, make a change in **testdb1**:

```
$ sql testdb1
* insert into dm01_product values (1, 19.99, 'DVD') \g
* commit \g
```

In the HVR log file you can see the output of the jobs:

```
$ tail $HVR_CONFIG/log/hvrhub/hvr.out

hvr_demo01-cap-db1[110]: Scanned 1 transaction containing 1 row (1 ins) for 1 table.
hvr_demo01-cap-db1[107]: Routed 215 bytes (compression=40.6%) from 'db1' into 2
locations.
hvr_demo01-cap-db1[146]: Capture cycle 3.
hvr_demo01-integ-db2[49]: Integrate cycle 2 for 1 transaction file (215 bytes).
hvr_demo01-integ-db2[56]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db2[56]: Integrate used 1 transaction and took 0.004 seconds.
hvr_demo01-integ-db3[49]: Integrate cycle 2 for 1 transaction file (215 bytes).
hvr_demo01-integ-db3[56]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db3[56]: Integrate used 1 transaction and took 0.013 seconds.
hvr_demo01-integ-db3[67]: Waiting...
```

This indicates that the jobs replicated the original change to **testdb2** and **testdb3**. A query on **testdb2** confirms this:

```
$ sql testdb2
* select * from dm01_product \g
Executing . . .
```

prod_id	prod_price	prod_descrip
1	19.99	DVD

(1 row)

FURTHER MONITORING

To keep track of the replication state, install HVR Monitor (see section [3.3](#), [3.4](#) or [3.5](#)), or install **hvrmaint** (see section [5.7](#)).

Stop replication:

```
$ hvrsuspend hvrhub HVR_DEM001
```

Re-activating a job that has state **FAILED**:

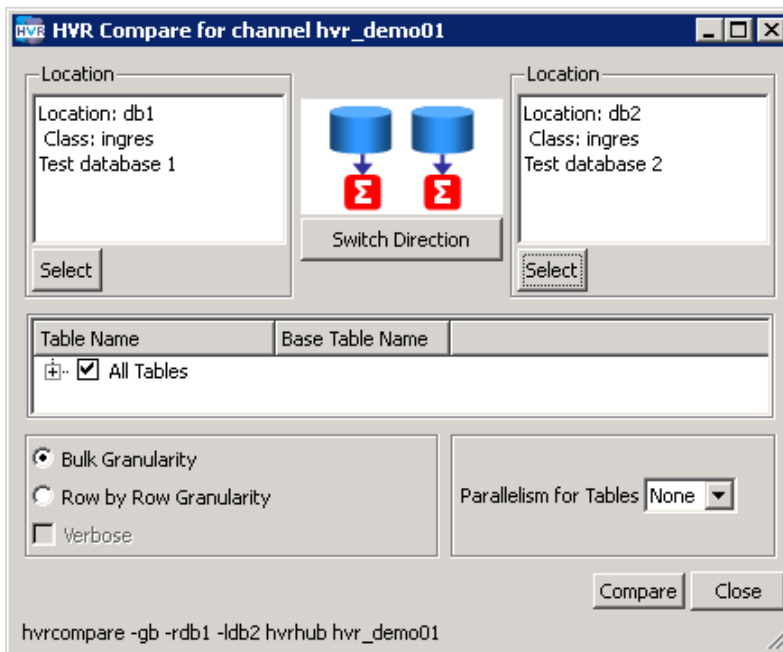
```
$ hvrtrigger -r hvrhub hvr_demo01-integ-db2
```

Stop the HVR Scheduler

```
$ hvrscheduler -k hvrhub
```

HVR COMPARE AND REFRESH

HVR Compare checks whether two locations have identical rows, and HVR Refresh copies the content of one location to second location. In the HVR GUI, right-click on a channel > **HVR Compare** (or **HVR Refresh**). Choose two locations by clicking on the **Select** buttons.

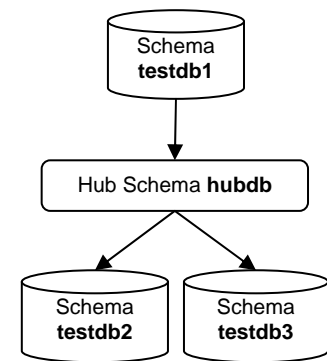


The outcome of the comparison is displayed in the pane beneath the [Actions](#) pane;

```
Compare of location 'db1' and 'db2' found 2 identical tables and \
0 different tables. (elapsed=0.28s)
```

B: Quick Start for Oracle

This appendix shows how to set up an HVR channel (called [hvr_demo01](#)) to replicate between Oracle databases. In real life HVR would usually replicate between Oracle instances on different machines. But for simplicity, in this example HVR will replicate between three schemas inside a single Oracle instance on the hub machine. The steps actually start by creating new users and tables for HVR to replicate between.



CREATE TEST SCHEMAS AND TABLES

Create three test schemas, each containing two empty tables named [dm01_product](#) and [dm01_order](#). If replication is configured between existing databases and tables then this step should be skipped.

```

$ sqlplus system/manager
SQL> create user testdb1 identified by hvr
  2 default tablespace users
  3 temporary tablespace temp
  4 quota unlimited on users;

SQL> create user testdb2 identified by hvr
  2 default tablespace users
  3 temporary tablespace temp
  4 quota unlimited on users;

SQL> create user testdb3 identified by hvr
  2 default tablespace users
  3 temporary tablespace temp
  4 quota unlimited on users;

SQL> grant create session to testdb1;
SQL> grant create table to testdb1;
SQL> grant create sequence to testdb1;
SQL> grant create procedure to testdb1;
SQL> grant create trigger to testdb1;
SQL> grant create view to testdb1;
SQL> grant execute any procedure to testdb1;
  
```

Give the same grants to [testdb2](#) and [testdb3](#).

Grant execution rights on `dbms_alert` to the schema that is the source location of replication (that is, [testdb1](#)). Login as oracle and perform the following extra statements:

```

$ sqlplus
Enter user-name: / as sysdba
SQL> grant execute on dbms_alert to testdb1;
SQL> exit;
  
```

Create the test tables.

```

$ cd $HVR_HOME/demo/hvr_demo01/base/oracle
$ sqlplus testdb1/hvr < hvr_demo01.cre
$ sqlplus testdb1/hvr < hvr_demo01.mod
$ sqlplus testdb2/hvr < hvr_demo01.cre
$ sqlplus testdb2/hvr < hvr_demo01.mod
$ sqlplus testdb3/hvr < hvr_demo01.cre
$ sqlplus testdb3/hvr < hvr_demo01.mod
  
```

INSTALL HVR

First read section [1 Introduction](#) which explains the HVR's terminology and architecture. In particular this explains the importance of a hub database.

Then install the HVR software on the hub machine by following the installation steps in section 3.1 or 3.2. If the hub machine is a Unix machine then HVR must also be installed on a PC, so the HVR GUI can be run.

Log-based capture requires that:

- On Unix and Linux the user name that HVR uses must be in Oracle's **dba** Unix group (file `/etc/group`).
- On Windows the database user name that HVR uses must have **sysdba** privilege. This is done with statement **grant sysdba to hvr**.
- The Oracle instance must have archiving enabled. This can be done by running the following statement as **sysdba** against a mounted but unopened database **alter database archive log**. The current state of archiving can be checked with query **select log_mode from v\$database**.

CREATE THE HUB DATABASE

Create the hub database, in which the HVR GUI will store the channel definition. This is actually another user/schema in the Oracle instance.

```
$ sqlplus system/manager
SQL> create user hvrhub identified by hvr
  2  default tablespace users
  3  temporary tablespace temp
  4  quota unlimited on users;

SQL> grant create session to hvrhub;
SQL> grant create table to hvrhub;
SQL> grant create sequence to hvrhub;
SQL> grant create procedure to hvrhub;
SQL> grant create trigger to hvrhub;
SQL> grant create view to hvrhub;
SQL> grant execute any procedure to hvrhub;

$ sqlplus
Enter user-name: / as sysdba
SQL> grant execute on dbms_alert to hvrhub;
SQL> exit;
```

CREATE THE CHANNEL

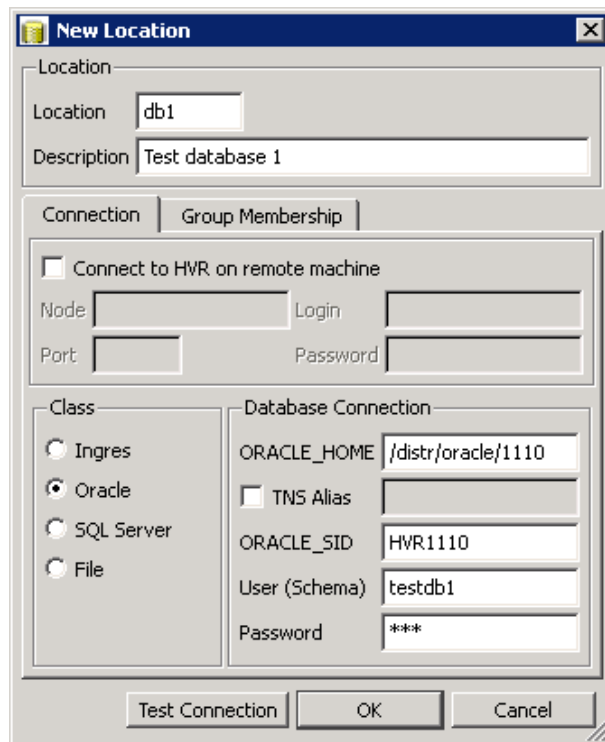
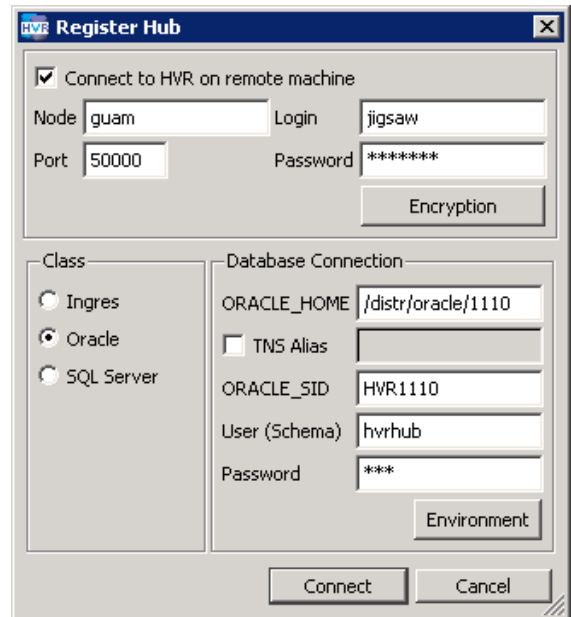
Start the HVR GUI on a PC by clicking on the HVR GUI icon (this is created the HVR Installer for Windows) or by running **hvrgui** on Linux.

First, Register the hub database: Right-click on **hub machines > Register hub**.

Enter connection details.

In this example the hub is a machine called **guam**, where an INET daemon is listening on port 50000. See section [3.1 New Installation on Unix or Linux](#) for how to do configure this.

For a new hub database a dialog will prompt “Do you wish to create the catalogs?”; answer **Yes**.



Next create three locations (one for each test database) using right-click on **Location Configuration > New Location**.

In this example there is no need to check **Connect to HVR on remote machine** because **testdb1** is on the same machine as the hub.

Ignore the **Group Membership** tab for now.

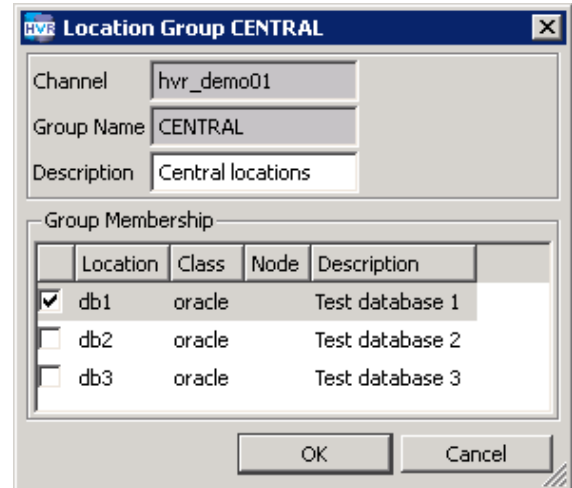
Make locations for **testdb2** and **testdb3** too.

Now define a channel using [Channel Definitions > New Channel](#).

The channel needs two location groups. Under the new channel: right-click on [Location Groups > New Group](#). Enter a group name (for instance [CENTRAL](#)).

Add location [db1](#) as a member of this group by checking the box for [db1](#).

Then create a second location group, called [DECENTRAL](#) that has members [db2](#) and [db3](#).



The new channel also needs a list of tables to replicate. This can be done as follows; Right-click on [Tables > Table Select](#).

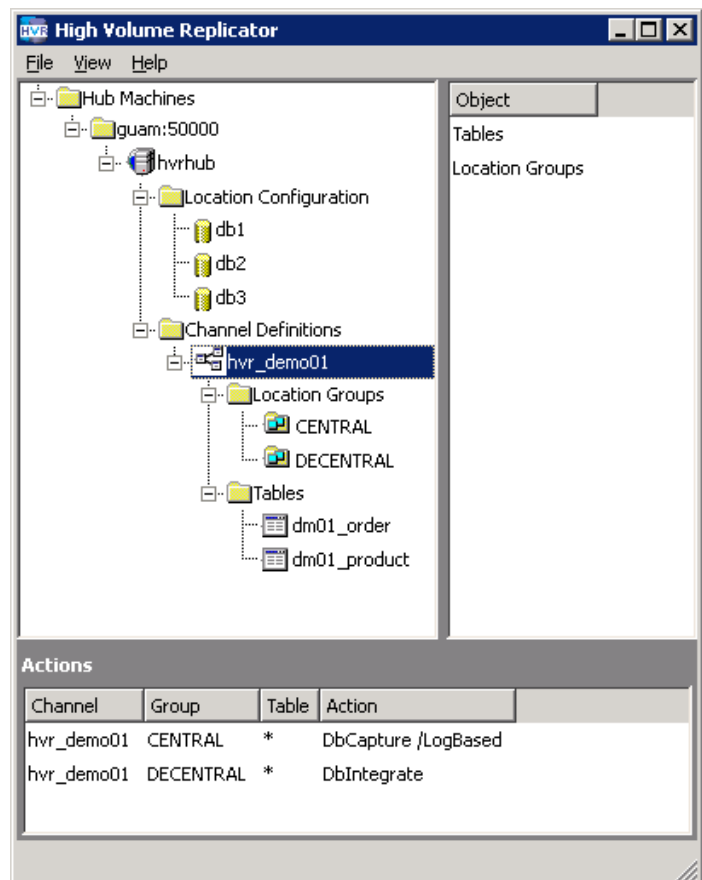
- Choose one of the three locations > Connect.
- In the Table Selection window, click on both tables and click Add.
- In new dialog HVR Table Name click OK.
- Close the Table Selection window.

DEFINE ACTIONS

Finally the new channel needs two actions to indicate the direction of replication.

- Right-click on group [CENTRAL > New Action > DbCapture](#). Check [/LogBased](#) so that the channel will detect changes using the Oracle log file, instead of using Oracle database triggers.
- Right-click on [Group DECENTRAL > New Action > DbIntegrate](#).

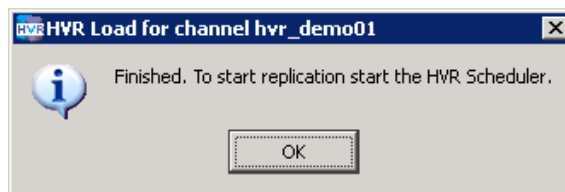
Note that the Actions pane only displays actions related the objects selected in the left hand pane. So click on channel [hvr_demo01](#) to see both actions.



ENABLE REPLICATION WITH HVR LOAD

Now that the channel definition is complete, create the runtime replication system.

Right-click on channel **hvr_demo01 > HVR Load**. Choose **Create objects only** and click **HVR Load**.



From the moment that HVR Load is done all changes to database **testdb1** will be captured by HVR when its capture job looks inside the Oracle logging.

HVR Load also creates three replication jobs, which can be seen by querying the hub database;

```
$ sqlplus hvrhub/hvr
SQL> select job_name, job_state from hvr_job;
```

JOB_NAME	JOB_STATE
hvr_demo01-cap-db1	PENDING
hvr_demo01-integ-db2	PENDING
hvr_demo01-integ-db3	PENDING

START SCHEDULING OF REPLICATION JOBS

Start the Scheduler on the hub machine:

```
$ hvrscheduler hvrhub/hvr
```

Next, instruct the HVR Scheduler to un-suspend (activate) the replication jobs. Note that the channel name **HVR_DEMO01** is in uppercase.

```
$ hvrsuspend -u hvrhub/hvr HVR_DEMO01

hvrsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEMO01-CAP.
hvrsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEMO01-INTEG.
hvrsuspend: Finished. (elapsed=0.73s)
```

The replication jobs inside the Scheduler each execute a script under **\$HVR_CONFIG/job/hvrhub/hvr_demo01** that has the same name as the job. So job **hvr_demo01-cap-db1** detects changes on database **testdb1** and stores these as transactions files on the hub machine. The other two jobs (**hvr_demo01-integ-db2** and **hvr_demo2-integ-d3**) pick up these transaction files, and perform inserts, updates and deletes on the two target databases.

TEST REPLICATION

To test replication, make a change in `testdb1`:

```
$ sqlplus testdb1/hvr
SQL> insert into dm01_product values (1, 19.99, 'DVD');
SQL> commit;
```

In the HVR log file you can see the output of the jobs:

```
$ tail $HVR_CONFIG/log/hvrhub/hvr.out

hvr_demo01-cap-db1[110]: Scanned 1 transaction containing 1 row (1 ins) for 1 table.
hvr_demo01-cap-db1[107]: Routed 215 bytes (compression=40.6%) from 'db1' into 2
locations.
hvr_demo01-cap-db1[146]: Capture cycle 3.
hvr_demo01-integ-db2[49]: Integrate cycle 2 for 1 transaction file (215 bytes).
hvr_demo01-integ-db2[56]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db2[56]: Integrate used 1 transaction and took 0.004 seconds.
hvr_demo01-integ-db3[49]: Integrate cycle 2 for 1 transaction file (215 bytes).
hvr_demo01-integ-db3[56]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db3[56]: Integrate used 1 transaction and took 0.013 seconds.
hvr_demo01-integ-db3[67]: Waiting...
```

This indicates that the jobs replicated the original change to `testdb2` and `testdb3`. A query on `testdb2` confirms this:

```
$ sqlplus testdb2/hvr
SQL> select * from dm01_product;

  PROD_ID  PROD_PRICE  PROD_DESCRIP
-----
         1         19.99  DVD
```

FURTHER MONITORING

To keep track of the replication state, install HVR Monitor (see section [3.3](#), [3.4](#) or [3.5](#)), or install `hvrmaint` (see section [5.7](#)).

Stop replication:

```
$ hvrsuspend hvrhub/hvr HVR_DEM001
```

Re-activating a job that has state FAILED:

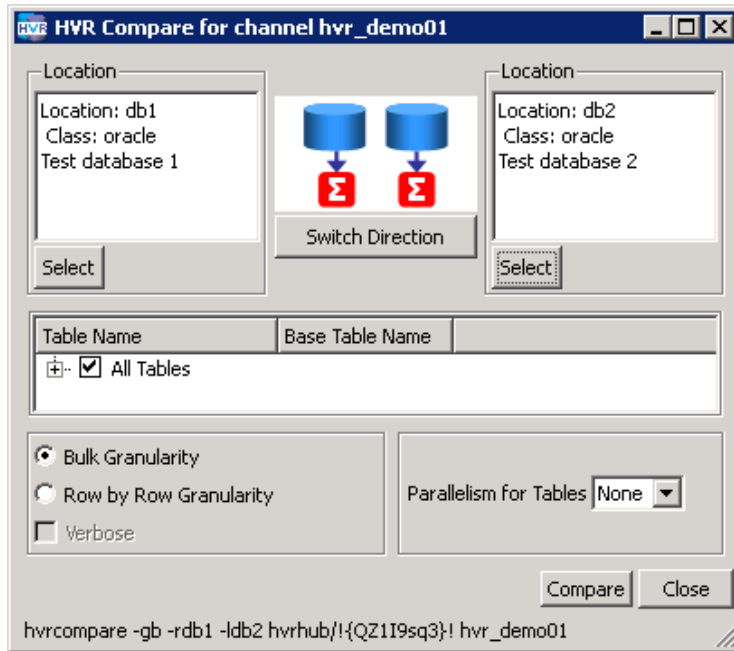
```
$ hvrtrigger -r hvrhub/hvr hvr_demo01-integ-db2
```

Stop the HVR Scheduler

```
$ hvrscheduler -k hvrhub/hvr
```

HVR COMPARE AND REFRESH

HVR Compare checks whether two locations have identical rows, and HVR Refresh copies the content of one location to second location. In the HVR GUI, right-click on a channel > **HVR Compare** (or **HVR Refresh**). Choose two locations by clicking on the **Select** buttons.

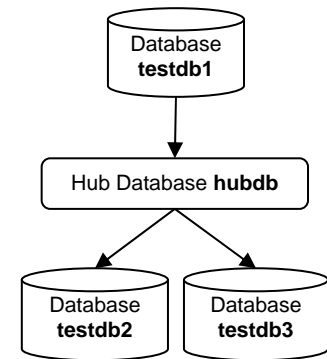


The outcome of the comparison is displayed in the pane beneath the **Actions** pane;

```
Compare of location 'db1' and 'db2' found 2 identical tables and \
0 different tables. (elapsed=0.28s)
```

C: Quick Start for SQL Server

This appendix shows how to set up an HVR channel (called [hvr_demo01](#)) to replicate between SQL Server databases. The steps actually start by creating new databases and tables for HVR to replicate between. In real life these databases would already exist and be filled with the user tables, but for simplicity everything is created from scratch. Likewise these databases would normally be on different machines, but again for simplicity everything is just kept on the hub machine.



CREATE TEST DATABASES AND TABLES

Create three test databases, each containing two empty tables named [dm01_product](#) and [dm01_order](#). If replication is configured between existing databases and tables then this step should be skipped.

In SQL Server Management Studio, create databases [testdb1](#), [testdb2](#) and [testdb3](#). Next, create the test tables;

```

C:\> cd %HVR_HOME%\demo\hvr_demo01\base\sqlserver
C:\> osql -U hvr -P hvr -d testdb1 < hvr_demo01.cre
C:\> osql -U hvr -P hvr -d testdb1 < hvr_demo01.mod
C:\> osql -U hvr -P hvr -d testdb2 < hvr_demo01.cre
C:\> osql -U hvr -P hvr -d testdb2 < hvr_demo01.mod
C:\> osql -U hvr -P hvr -d testdb3 < hvr_demo01.cre
C:\> osql -U hvr -P hvr -d testdb3 < hvr_demo01.mod
  
```

INSTALL HVR

First read section [1 Introduction](#) which explains the HVR's terminology and architecture. In particular this explains the importance of a hub database.

Then install the HVR software on the hub machine by following the installation steps in section [3.1](#) or [3.2](#).

An extended stored procedure called [hvrevent](#) must be created on the machines with a hub database or a capture database. This DLL must be registered in the master database with the username under which HVR will run with following SQL command:

```

$ osql -dmaster -Uuser -Ppwd
> exec sp_addextendedproc 'hvrevent', 'c:\hvr_home\bin\hvrevent.dll'
> go
  
```

Replace pathname [c:\hvr_home](#) with the correct value of [HVR_HOME](#) on that machine.

CREATE THE HUB DATABASE

In SQL Server Management Studio, create the hub (called hvrhub, for instance) in which the HVR GUI will store the channel definition.

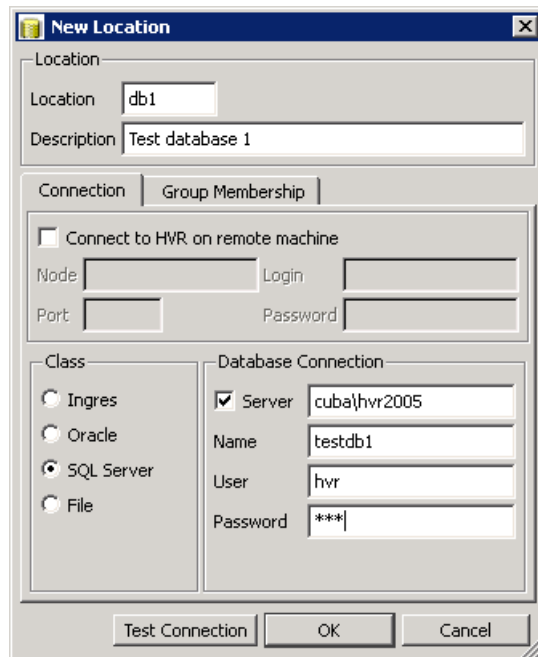
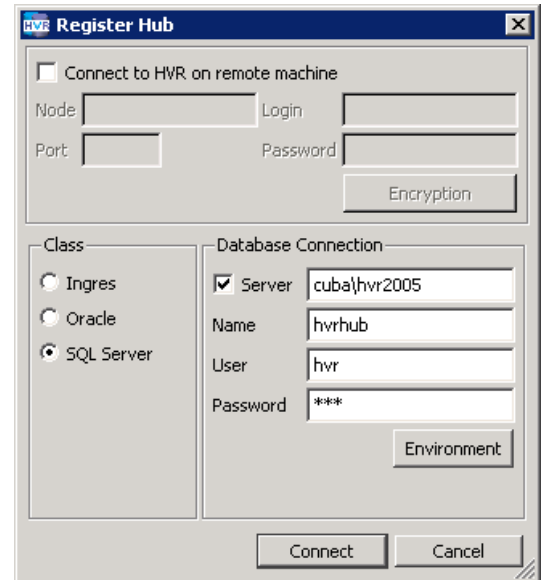
CREATE THE CHANNEL

Start the HVR GUI on the hub machine by clicking on the HVR GUI icon.

First, Register the hub database: Right-click on **hub machines > Register hub**.

Enter connection details.

For a new hub database a dialog will prompt “Do you wish to create the catalogs?”; answer **Yes**.



Next create three locations (one for each test database) using right-click on **Location Configuration > New Location**.

In this example there is no need to check **Connect to HVR on remote machine** because **testdb1** is on the same machine as the hub.

Ignore the **Group Membership** tab for now.

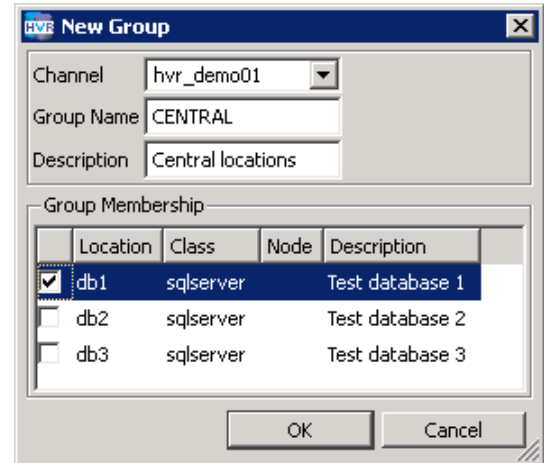
Make locations for **testdb2** and **testdb3** too.

Now define a channel using **Channel Definitions > New Channel**.

The channel needs two location groups. Under the new channel: right-click on **Location Groups > New Group**. Enter a group name (for instance **CENTRAL**).

Add location **db1** as a member of this group by checking the box for **db1**.

Then create a second location group, called **DECENTRAL** that has members **db2** and **db3**.



The new channel also needs a list of tables to replicate. This can be done as follows; Right-click on **Tables > Table Select**.

- Choose one of the three locations > **Connect**.
- In the **Table Selection** window, click on both tables and click **Add**.
- In new dialog **HVR Table Name** click OK.
- Close the **Table Selection** window.

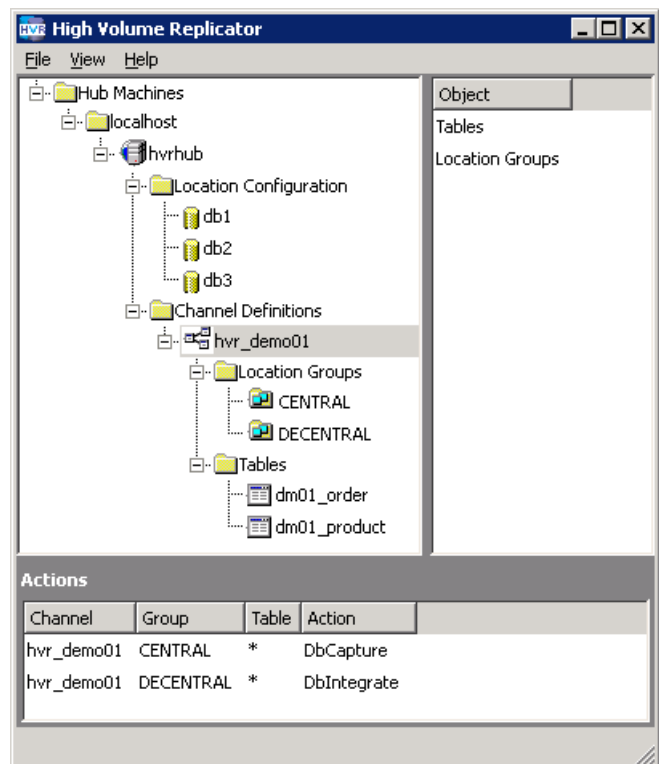
DEFINE ACTIONS

Finally the new channel needs two actions to indicate the direction of replication.

Right-click on group **CENTRAL > New Action > DbCapture**.

Right-click on Group **DECENTRAL > New Action > DbIntegrate**.

Note that the Actions pane only displays actions related the objects selected in the left hand pane. So click on channel **hvr_demo01** to see both actions.



ENABLE REPLICATION WITH HVR LOAD

Now that the channel definition is complete, create the runtime replication system.

Right-click on channel **hvr_demo01 > HVR Load**. Choose **Create objects only** and click **HVR Load**.

From the moment that HVR Load is done all changes to database **testdb1** will be captured by HVR triggers that write changes to capture tables.

HVR Load also creates three replication jobs, which can be seen by running the following query on hvrhub:

```
select job_name, job_state from hvr_job

hvr_demo01-cap-db1          PENDING
hvr_demo01-integ-db2       PENDING
hvr_demo01-integ-db3       PENDING
```

START SCHEDULING OF REPLICATION JOBS

Start the Scheduler on the hub machine:

```
$ hvrscheduler hvrhub
```

Next, instruct the HVR Scheduler to un-suspend (activate) the replication jobs. Note that the channel name HVR_DEMO01 is in uppercase.

```
$ hvrsuspend -u hvrhub HVR_DEMO01

hvsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEMO01-CAP.
hvsuspend: Deleted attribute [attr_name='suspend'] from job group HVR_DEMO01-INTEG.
hvsuspend: Finished. (elapsed=0.73s)
```

The replication jobs inside the Scheduler each execute a script under **\$HVR_CONFIG/job/hvrhub/hvr_demo01** that has the same name as the job. So job **hvr_demo01-cap-db1** detects changes on database testdb1 and stores these as transaction files on the hub machine. The other two jobs (**hvr_demo01-integ-db2** and **hvr_demo2-integ-d3**) pick up these transaction files, and perform inserts, updates and deletes on the two target databases.

TEST REPLICATION

To test replication, make a change in **testdb1**, table `dm01_product`:

```
insert into dm01_product values (1, 19.99, 'DVD')
go
commit
go
```

In the HVR log file you can see the output of the jobs:

```
C:\> notepad %HVR_CONFIG%/log/hvrhub/hvr.out

hvr_demo01-cap-db1[119]: Capture cycle 1.
hvr_demo01-cap-db1[24]: Selected 1 row from 'dm01_product__c0' (201 wide).
hvr_demo01-cap-db1[24]: Routed 212 bytes (compression=42.7%) from 'db1' into \
  2 locations.
hvr_demo01-cap-db1[153]: Finished. (elapsed=1.20s)
hvr_demo01-integ-db2[50]: Integrate cycle 1 for 1 transaction file (212 bytes).
hvr_demo01-integ-db2[57]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db2[57]: Integrate used 1 transaction and took 0.017 seconds.
hvr_demo01-integ-db2[66]: Finished. (elapsed=0.14s)
hvr_demo01-integ-db3[50]: Integrate cycle 1 for 1 transaction file (212 bytes).
hvr_demo01-integ-db3[57]: Integrated 1 change from 'dm01_product' (1 ins).
hvr_demo01-integ-db3[57]: Integrate used 1 transaction and took 0.02 seconds.
hvr_demo01-integ-db3[66]: Finished. (elapsed=0.15s)
```

This indicates that the jobs replicated the original change to **testdb2** and **testdb3**. A query on **testdb2** confirms this:

```
select * from dm01_product
go

  prod_id  prod_price  prod_descrip
-----
          1          19.99  DVD
(1 row)
```

FURTHER MONITORING

To keep track of the replication state, install HVR Monitor (see section [3.3](#), [3.4](#) or [3.5](#)), or install **hvrmaint** (see section [5.7](#)).

Stop replication:

```
C:\> hvrsuspend \hvrhub HVR_DEM001
```

Re-activating a job that has state FAILED:

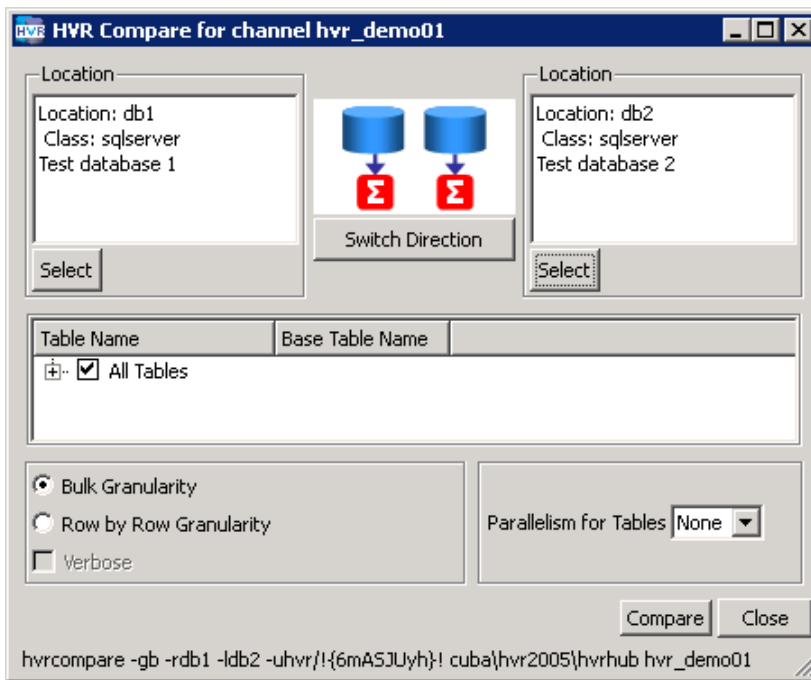
```
C:\> hvrtrigger -r \hvrhub hvr_demo01-integ-db2
```

Stop the HVR Scheduler

```
C:\> hvrscheduler -k \hvrhub
```

HVR COMPARE AND REFRESH

HVR Compare checks whether two locations have identical rows, and HVR Refresh copies the content of one location to second location. In the HVR GUI, right-click on a channel > **HVR Compare** (or **HVR Refresh**). Choose two locations by clicking on the **Select** buttons.

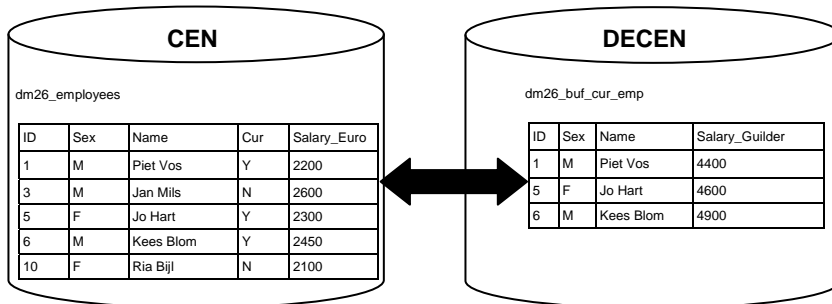
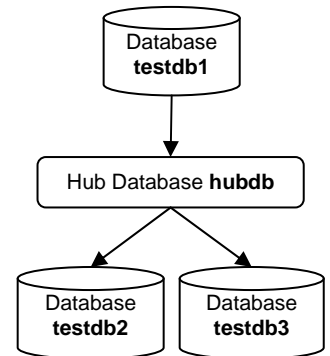


The outcome of the comparison is displayed in the pane beneath the **Actions** pane;

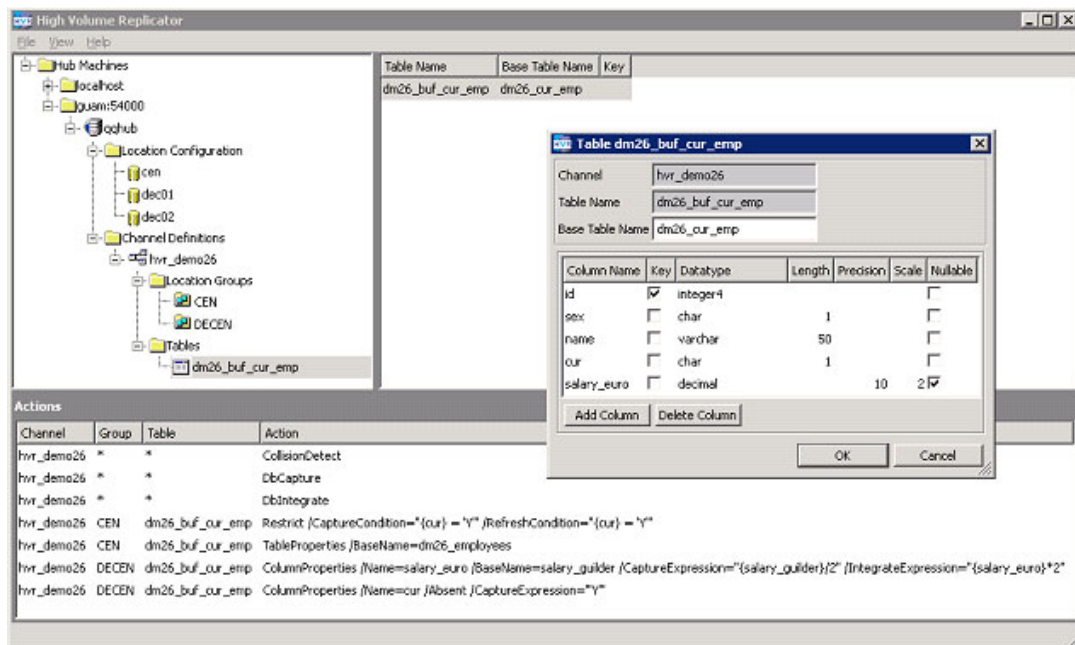
```
Compare of location 'db1' and 'db2' found 2 identical tables and \
0 different tables. (elapsed=0.28s)
```

D: Example Replication between Different Table Layouts

The employee table in a company's central database is called **dm26_employees** and contains both current employees and employees who have left the company; this is indicated with an extra column called **cur**. In the de-central database the employee table is called **dm26_buf_cur_emp** and only contains current employees. This version of table also contains an extra column for the employee's salary in the local currency.



The HVR channel is defined as follows: Action **DbCapture**, **DbIntegrate** and **CollisDetect** are set on both sides for bi-directional replication. A **Restrict** is defined on the central database to filter rows. The decentral database has a **TableProperties** action and two **ColumnProperties** actions which rename the table, hide one column and recalculate another. The following is a screenshot of the channel definition to replicate between the tables.



Data can be changed on either database and is replicated to the other database. Command **hvrrefresh** and **hvrcompare** can also be used for this channel; they will also add extra columns and filter out restricted rows as required. This channel can be found in directory **\$HVR_HOME/demo/hvr_demo26**.