Capture

Contents

Description

Action **Capture** instructs HVR to capture changes from a location. Various parameters are available to modify the functionality and performance of capture.

For a database location, HVR gives you an option to capture changes using the log-based method (**/LogReadMethod)** or trigger-based method (**/TriggerBased**). HVR recommends using the log-based data capture because it has less impact on database resources as it reads data directly from its logs, without affecting transactions, manages large volumes of data and supports more data operations, such as truncates, as well as DDL capture. In contrast, the trigger-based data capture creates triggers on tables that require change data capture, so firing the triggers and storing row changes in a shadow table slow down transactions and introduces overhead.

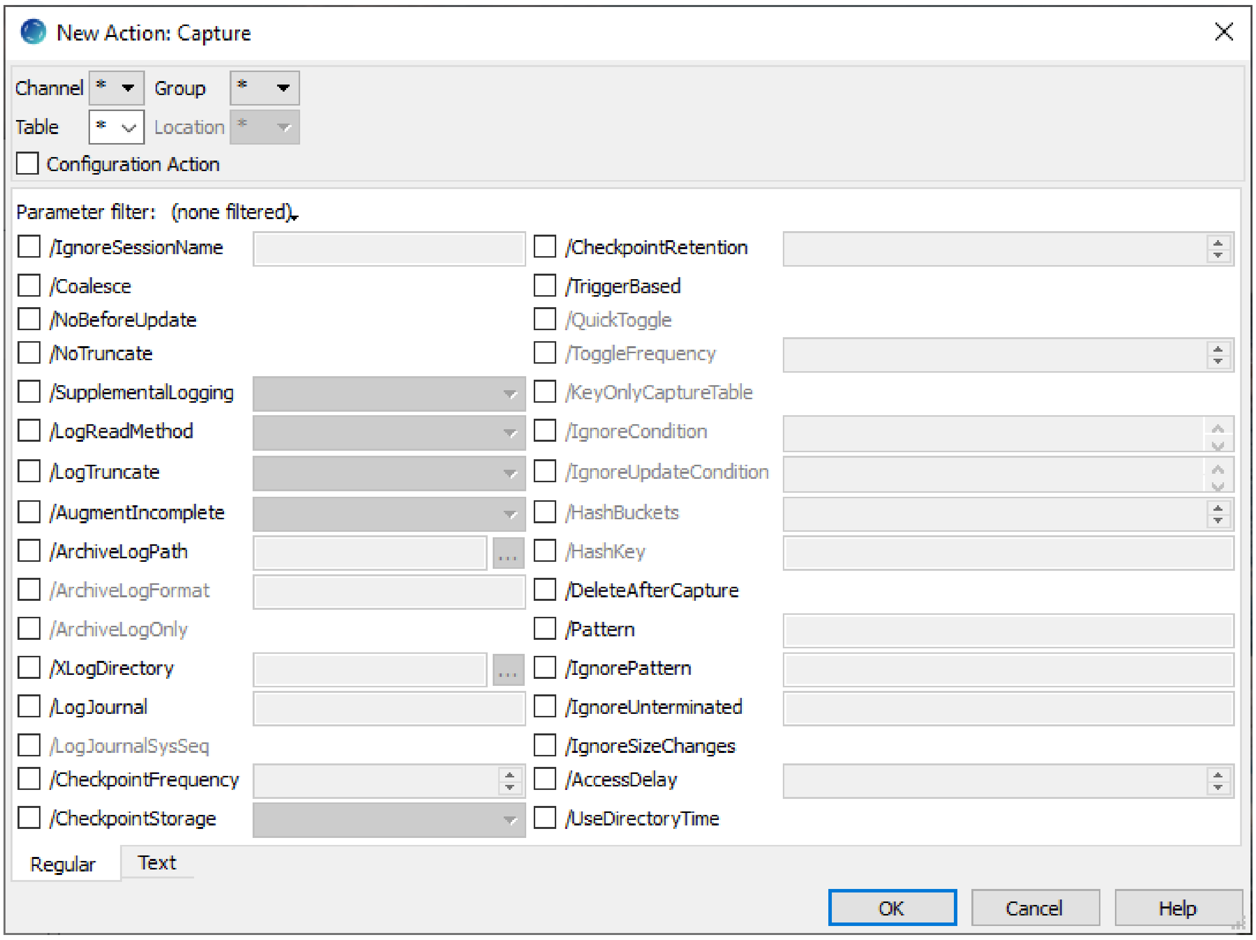
When defined on a file location this action instructs HVR to capture files from a file location's directory. Changes from a file location can be replicated both to a database location and to a file location if the channel contains table information. In this case any files captured are parsed (see action **FileFormat**).

If **Capture** is defined on a file location without table information then 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 **Capture** and **Integrate** (no parameters) then all files in the capture location's directory (including files in sub-directories) 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 sub-directories are preserved. New and changed files are replicated, but empty sub-directories and file deletions are not replicated.

Bidirectional replication (replication in both directions with changes happening in both file locations) is not currently supported for file locations. File deletion is not currently captured by HVR.
If **Capture** is defined on a file location without parameter **/DeleteAfterCapture** and action **LocationProperties /StateDirectory** is used to define a state directory outside of the file location's top directory, then HVR's file capture becomes read only; write permissions are not needed.

Parameters

This section describes the parameters available for action **Capture**. By default, only the supported parameters for the selected location class are displayed in the **Capture** window.



Parameter	Argument	Description
/IgnoreSession Name	*sess_name*	This action instructs the capture job to ignore changes performed by the specified session name. Multiple ignore session names can be defined for a job, either by defining **/IgnoreSessionName** multipl e times or by specifying a comma separated list of names as its value. Normally HVR's capture avoids recapturing changes made during HVR integration by ignoring any changes made by sessions named **hvr_integrate**. 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 **Integrate /SessionName**. For more information, see Managing Recapturing Using Session Names. If this parameter is defined for any table with log based capture, then it affects all tables captured from that location. For an example of using action **Capture** with parameter **/IgnoreSes sionName**, refer to section Examples below.

/Coalesce		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 **UPDATE**s 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. This parameter should not be used together with **Transform /SapXForm**.
/NoBeforeUpdate		Do not capture 'before row' for an update unless it is a key 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, because less data is transported. But that means that integrate will update all columns (normally HVR will only update the columns that were actually changed by the update statements and will leave the other columns unchanged). If this parameter is defined for any table with log based capture, then it affects all tables captured from that location.
/NoTruncate		Do not capture SQL truncate table statements such as **TRUNCATE** in Oracle and **modify** *mytbl* **to truncated** in Ingres. If this parameter is not defined, then these operations are replicated using **hvr_op** value **5**. For DB2 for z/OS, this parameter affects only **TRUNCATE IMMEDIATE**. HVR will always capture **TRUNCATE** if used without **IMMEDIATE** option (this will be replicated using **hvr_op** value **0**). This parameter is not supported for SQL Server.

/SupplementalLogging **SQL Server**	*method*	Specify what action should be performed to enable supplemental logging for tables. Supplemental logging should be enabled to make log-based capture of updates possible. Valid values for *method* are: • **CDCTAB** (**default** when source SQL Server supports CDC tables - SQL Server 2008 or higher (Enterprise edition) and SQL Server 2016 SP1 or higher (Enterprise and Standard editions)): Enable supplemental logging of updates by creating a Change Data Capture (CDC) instance for the source table. If users attempt some DDL such as **TRUNCATE TABLE** when a CDC instance exists for the table they will give an error message. If **/LogTruncate=NATIVE_DBMS_AGENT** is defined, creating a CDC instance may cause I/O overhead (SQL Server jobs copy each change to a CDC table, which no-one uses). This method is needed to capture updates to tables without a primary key. It is not available for SQL Server Standard and Express editions or for SQL Server 2005. This means that HVR on such databases can only capture changes to tables with primary keys. • **ARTICLE_OR_CDCTAB** (**default** when source SQL Server does not support CDC tables): Enable supplemental logging of updates by creating an SQL Server transactional replication article if the source table has Primary Key, or by creating a CDC table instance if the source table does not have Primary Key. If users attempt some DDL such as **DROP TABLE** or **TRUNCATE TABLE** when an article exists for the table they will give an error message. • **EXISTING**: Check that an article or a CDC table instance already exists for the source table. No new articles or CDC table instances are created. If an article or a CDC table instance does not exists the **Hvrinit** will give an error message. • **EXISTING_OR_CDCTAB**: Check if an article or a CDC table instance already exists for the source table. Enable supplemental logging of updates by creating a Change Data Capture (CDC) instance for the source table if no article or a CDC table instance exists.

/LogReadMethod	*method*	Select method of reading changes from the DBMS log file. This parameter is supported only for certain location classes. For the list of supported location class, see Log-based capture with **/LogReadMethod** parameter in Capabilities. Valid values for *method* are: • **DIRECT** (**default**): Read transaction log records directly from the DBMS log file using file I/O. This method is generally faster and more efficient than the SQL mode. The **DIRECT** log read method requires that HVR agent is installed on the source database machine. This method is supported only for certain location classes. For the list of supported location classes, see Direct access to logs on a file system in Capabilities. • **SQL**: Query transaction log records using a special SQL function. The advantage of this method is that it reads change data over an SQL connection and does not require an HVR agent to be installed on the source database machine. The disadvantages of the **SQL** method is that it is slower than the **DIRECT** method and exposes additional load on the source database. This method is supported only for certain location classes. For the list of supported location classes, see Access to logs using SQL interface in Capabilities. For MySQL, the **default** method of reading changes from the DBMS log file is **SQL**. For Oracle, the **SQL** method enables capture from LogMiner. For PostgreSQL, prior to HVR version 5.5, the **SQL** method does not support bidirectional replication because changes will be re-captured and replicated back. For SQL Server, the **DIRECT** method requires Windows Administrator privileges and reduced permission models are not supported. The **SQL** method supports reduced permission models but it may require incomplete row augmenting.
/LogTruncate SQL Server	*method*	Specify who advances SQL Server transaction log truncation point (truncates the log). Valid values for *method* are;

- **CAP_JOB** (**default**): is used to indicate that the capture job regularly calls **sp_repldone** to unconditionally release the hold of the truncation point for replication. When this option is selected and **HVR Initialize** is run, depending on the value of **/SupplementalLogging**, HVR will also drop/disable the SQL Server Agent jobs that are created when CDC tables are created through the CDC stored procedures, and the Agent jobs related to data distribution. As a result, the additional impact of auxiliary database objects to enable supplemental logging is minimized. For example, the CDC tables are not populated (and the space allocation increased) because the Agent job to do this is dropped. Multiple capture jobs can be set up on a single database with option **CAP_JOB** selected. However, note that if no capture job is running with the CDC tables and/or Articles in place, the transaction log will grow because the truncation point for replication is not released. Do not set this option if there is another data replication solution or the database uses CDC tables.
- **CAP_JOB_RETAIN**: This value must be used when capturing from a SQL Server database with the recovery mode set to Simple Recovery. The capture job moves the truncation point of the transaction log forward by calling the **sp_repldone** stored procedure at the end of each sub-cycle. Only part of the transaction log that has already been processed (captured) is marked for truncation (this is different from the **CAP_JOB** mode , where all records in the transaction log are marked for truncation, including those that have not been captured yet). This value is not compatible with multi-capture and does not allow for coexistence with a third party replication solution. This setting will also result in SQL Server's Agent jobs being dropped/disabled, so the TLog will grow when the capture job is not running and CDC tables and/or Articles are still in place. Do not set this option if another data replication solution is in place or CDC tables are used in the database.
- **LOGRELEASE_TASK**: should be set if a separate job/task is created to release the truncation point for replication. For example, schedule a separate SQL Server Agent job to unconditionally call **sp_repldone** at an interval. HVR also provides its own **Log Release** capability that can be accessed in the GUI through the pop-up menu on the SQL Server location. Choosing the option **LOGRELEASE_TASK** will also result in SQL Server's Agent jobs being dropped/disabled. However, as long as the scheduled log release task runs, the truncation point for replication is released, even if the capture job(s) is(are) not running. This option should only be used in conjunction with another replication or CDC solution if the log release task that is scheduled is aware of the requirements of the other solution.
- **NATIVE_DBMS_AGENT**: should be used if native replication and/or CDC tables are used on the database. With this option, HVR will not drop/disable the native SQL Server Agent jobs that are created when CDC tables and/or Articles are created. HVR will also not interfere with the release of the truncation point for replication. If CDC tables are used to enable supplemental logging (when **/SupplementalLogging** is defined with either **CDCTAB**, **ARTICLE_OR_CDCTAB**, or **EXISTING_OR_CDCTAB**), this may cause I/O overhead (SQL Server jobs copy each change to a CDC table, which no-one uses).

/AugmentIncomplete	*col_type*	During **capture**, HVR may receive partial/incomplete values for certain column types. Partial/incomplete values are the values that HVR cannot **capture** entirely due to technical limitations in the database interface. This parameter instructs HVR to perform additional steps to retrieve the full value from the source database, this is called augmenting. This parameter also augments the missing values for key updates. • ⚠ Defining this parameter can adversely affect the **capture** performance. • If this parameter is not defined and when a partial/incomplete value is received, the **capture** will fail with an error. Valid values for *col_type* are: • **NONE** (**default**): No extra augmenting is done. • **LOB**: Capture will augment partial/incomplete values for all columns of a table, if that table contains at least one lob column. For key-updates, missing values are augmented too. • **ALL**: Capture will augment partial/incomplete values for all columns of any table. For key-updates, missing values are augmented too. In certain situations, the default behavior changes and setting **/AugmentIncomplete** can only override the behavior with a 'stronger' value. For DB2 for Linux Unix and Windows, **LOB** should be selected to capture columns with **xml** data type. For DB2 for z/OS, the **default** *col_type* is **LOB** and can only be changed to **ALL**. For SQL Server, capture when **/LogReadMethod** is set to **SQL** and tables that contain non-key columns, the **default***col_type* is **ALL** and can not be changed. For Oracle, capture when **/LogReadMethod** is set to **SQL** the **default***col_type* is **LOB** and can only be changed to **ALL**.

/ArchiveLogPath	*dir*	Instruct HVR to search for the transaction log archives in the given directory. For Oracle, HVR will search for the log archives in the directory *dir* in addition to the 'primary' Oracle archive directory. If **/ArchiveLogOnly** parameter is enabled then HVR will search for the log archives in the directory *dir* only. Any process could be copying log archive files to this directory; the Oracle archiver (if another **LOG_ARCHIVE_DEST_***N* is defined), RMAN, **Hvrlogrelease** or a simple shell script. Whoever sets up copying of these files must also arrange that they are purged periodically, otherwise the directory will fill up. For SQL Server, HVR normally locates the transaction log backup files by querying the backup history table in the msdb database. Specifying this parameter tells HVR to search for the log backup files in the *dir* folder instead. When this parameter is defined, the **/ArchiveLogFormat** parameter must also be defined. For HANA, HVR will search for the log backups only in the directory *dir* instead of the default log backup location for the source database. Since HVR 5.7.5/20, HVR will search for the log backups in the directory *dir* in addition to the default log backup location for the source database.
/ArchiveLogFormat	*format*	

Describes the filename format (template) of the transaction log archive files stored in the directory specified by the **/ArchiveLogPath** parameter.

The list of supported format variables and the default *format* string are database-specific.

For Oracle, this parameter accepts the following format variables:

- **%d** - match numbers (zero or more decimal digits). Numbers matched using this variable are ignored by HVR.
- **%r** or **%R** - reset logs ID
- **%s** or **%S** - log sequence number
- **%t** or **%T** - thread number
- **%z** or **%Z** - match alphanumeric characters. Characters matched using this variable are ignored by HVR.
- Wildcard character ***** is not supported.

For more information about the format variables, refer to the article LOG_ARCHIVE_FORMAT in Oracle documentation.

For Oracle, when this parameter is not defined, then by **default** HVR will query the database for Oracle's initialization parameter - **LOG_ARCHIVE_FORMAT**.

For SQL Server, this parameter accepts the following format variables:

- **%d** - database name
- **%Y** - year (up to 4 digit decimal integer)
- **%M** - month (up to 2 digit decimal integer)
- **%D** - day (up to 2 digit decimal integer)
- **%h** - hours (up to 2 digit decimal integer)
- **%m** - minutes (up to 2 digit decimal integer)
- **%s** - seconds (up to 2 digit decimal integer)
- **%n** - file sequence number (up to 64 bit decimal integer)
- **%%** - matches %
- ***** - wildcard, matches zero or more characters

HVR uses the **%Y**, **%M**, **%D**, **%h**, **%m**, **%s** and **%n** values to sort and processes the log backup files in the correct (chronological) order. The combinations of the **%Y**, **%M**, **%D** and **%h**, **%m**, **%s** values are expected to form valid date and time values, however no validation is performed. Any value that is missing from the *format* string is considered to be 0. When sorting the files comparison is done in the following order: **%Y**, **%M**, **%D**, **%h**, **%m**, **%s**, **%n**.

For SQL Server, this parameter has no **default** and must be specified if **/ArchiveLogPath** parameter is defined.

For HANA, this parameter accepts the following format variables:

- **%v** - log volume ID
- **%p** - log partition ID
- **%s** - start sequence number
- **%e** - end sequence number
- **%t** - start timestamp (in milliseconds since UNIX epoch)
- **%%** - matches %
- ***** - wildcard, matches zero or more characters

The **%s**, **%e** and **%t** format variables are mandatory.

For HANA, this parameter is optional, the **default** *format* value is **log_backup_%v_%p_%s_%e.%t**.

/ArchiveLogOnly		Capture data from archived redo files in directory defined by **/ArchiveLogPath** only and do not read anything from online redo files or the 'primary' archive destination. This allows the HVR process to reside on a different machine than the Oracle DBMS or SQL Server and read changes from files that are sent to it by some remote file copy mechanism (e.g. **FTP**). The capture job still needs an SQL connection to the database for accessing dictionary tables, but this can be a regular connection. Replication in this mode can have longer delays in comparison with the 'online' mode. For Oracle, to control the delays it is possible to force Oracle to issue an archive once per predefined period of time. For Oracle RAC systems, delays are defined by the slowest or the least busy node. This is because archives from all threads have to be merged by SCNs in order to generate replicated data flow. For SQL Server, it is possible to control this delay by **HVR_MSSQL_ZREAD_ARCHIVE_SCAN_INTERVAL** environment variable.
/XLogDirectory	*dir*	Directory containing current PostgreSQL **xlog** files.
/LogJournal **Db2 for i**	*schema.journal*	Capture from specified DB2 for i journal. Both the schema (library) of the journal and the journal name should be specified (separated by a dot). This parameter is mandatory for DB2 for i. All tables in a channel should use the same journal. Use different channels for tables associated with different journals. If this parameter is defined for any table, then it affects all tables captured from that location.
/LogJournalSysSeq **Db2 for i**		Capture from journal using ***SYSSEQ**. This parameter requires **/LogJournal**.

/CheckpointFrequency **Since** v5.2.3/15	*secs*	Checkpointing frequency in seconds for long running transactions, so the capture job can recover quickly when it restarts. Value *secs* is the interval (in seconds) at which the capture job creates checkpoints. The **default** frequency (when **/CheckpointFrequency** is not defined) is **300** seconds (5 minutes). Value **0** means no checkpoints are written. Without checkpoints, capture jobs must rewind back to the start of the oldest open transaction, which can take a long time and may require access to many old DBMS log files (e.g. archive files). The checkpoints are written into directory **$HVR_CONFIG/capckp/***hub/chn*. If a transaction continues to make changes for a long period then successive checkpoints will not rewrite its same changes each time; instead the checkpoint will only write new changes for that transaction; for older changes it will reuse files written by earlier checkpoints. Only long-running transactions are saved in the checkpoint. For example if the checkpoint frequency is each 5 minutes but users always do an SQL commit within 4 minutes then checkpoints will never be written. If however some users keep transactions open for 10 minutes, then those transactions will be saved but shorter-lived ones in the same period will not. The frequency with which capture checkpoints are written is relative to the capture jobs own clock, but it decides whether a transaction has been running long enough to be checkpointed by comparing the timestamps in its DBMS logging records. As consequence, the maximum (worst-case) time that an interrupted capture job would need to recover (rewind back over all its open transactions) is its checkpoint frequency (default 5 minutes) plus the amount of time it takes to reread the amount of changes that the DBMS can write in that period of time. When a capture job is recovering it will only use checkpoints which were written before the 'capture cycle' was completed. This means that very frequent capture checkpointing (say every 10 seconds) is wasteful and will not speed up capture job recovery time. This parameter is supported only for certain location classes. For the list of supported location classes, see Log-based capture checkpointing in Capabilities.

/CheckpointStorage **Since** v5.2.3/15	*STOR*	Storage location of capture checkpoint files for quick capture recovery. Available options for *STOR* are: • **LOCATION** (**default**): Save checkpoints in a directory on capture location. • **HUB**: Save checkpoints in a directory on hub machine. Writing checkpoints on the hub is more expensive because extra data must be sent across the network. Checkpoints should be stored on the hub machine when capturing changes from an Oracle RAC, because the directory on the remote location where capture job would otherwise write checkpoints (**$HVR_CONFIG/capckp/**) may not be shared inside the RAC cluster, so it may not be available when the capture job restarts. Checkpoints are saved in directory **$HVR_CONFIG/capckp/** (this can be either on capture machine or hub). If capture job is restarted but it cannot find the most recent checkpoint files (perhaps the contents of that directory have been lost during a failover) then it will write a warning and then rewind back to the start of the oldest open transaction. On busy systems, it is recommended to change this parameter only when there are no existing capture checkpoints, otherwise, there can be performance costs and superfluous warnings when the capture job starts for the first time with new settings.
/CheckpointRetention **Since** v5.5.5/6	*period*	Retains capture checkpoint files up to the specified *period* (in seconds). The retained checkpoint files are saved in **$HVR_CONFIG/capckpretain/***hub***/***channel***/***location* (this can be either on capture machine or hub) based on the location defined in **/CheckpointStorage**.
/TriggerBased		Capture changes through DBMS triggers generated by HVR, instead of using log-based capture. This parameter is supported only for certain location class. For the list of supported location class, see Trigger-based capture in Capabilities.

/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 is 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. If this parameter is defined for any table, then it affects all tables captured from that location. This parameter requires **/TriggerBased**. For Ingres, variable **ING_SET** must be defined to force **readlock**=**nolock** on the quick toggle table. **Example**: `$ ingsetenv ING_SET 'set lockmode on hvr_qtogmychn where readlock=nolock'`
/ToggleFrequency	*secs*	Instruct HVR's trigger based capture jobs to wait for a fixed interval *secs* (in seconds) before toggling and reselecting capture tables. If this parameter is defined for any table then it affects all tables captured from that location. If this parameter is not selected, the trigger based capture job dynamically waits 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. This parameter requires **/TriggerBased**.
/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 data types. 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. This parameter requires **/TriggerBased**.

/IgnoreCondition	*sql_expr*	Ignore (do not capture) any changes that satisfy expression *sql_expr* (e.g. **Prod_id < 100**). This logic is added to the HVR capture rules/triggers and procedures. This parameter differs from the **Restrict /CaptureCondition** as follows: • 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 update conversion'. Restrict update 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. This parameter requires **/TriggerBased**.
/IgnoreUpdateCondition	*sql_expr*	Ignore (do not capture) any update changes that satisfy expression *sql_expr*. This logic is added to the HVR capture rules/triggers and procedures. This parameter requires **/TriggerBased**.
/HashBuckets **Ingres**	*int*	Identify the number *int* of hash buckets, with which the capture table is created. 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 sparser, 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. This parameter requires **/TriggerBased**.
/HashKey **Ingres**	*col_list*	Identify the list of columns *col_list*, the values of which are used to calculate the hash key value. 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. This parameter requires **/TriggerBased**.
/DeleteAfterCapture **File/FTP /Sharepoint**		Delete file after capture, instead of capturing recently changed files. If this parameter is defined, then the channel moves files from the location. Without it, the channel copies files if they are new or modified.
/Pattern **File/FTP /Sharepoint**	*pattern*	Only capture files whose names match pattern. The **default** pattern is **'**/*'** which means search all sub-directories and match all files. Possible patterns are:

- **'*.c'** – Wildcard, for files ending with **.c**. A single asterisk matches all or part of a file name or sub-directory name.
- **'**/*txt'** – Recursive Sub-directory Wildcard, to walk through the directory tree, matching files ending with **txt**. A double asterisk matches zero, one or more sub-directories but never matches a file name or part of a sub-directory name.
- **'*.lis'** Files ending with **.lis** or **.xml**
- **'a?b[d0 9]'** Files with first letter **a**, third letter **b** and fourth letter **d** or a digit. Note that [**a f**] matches characters, which are alphabetically between **a** and **f**. Ranges can be used to escape too; [*****] matches ***** only and [**[**] matches character **[** on ly.
- '***.csv|*.xml|*.pdf**' Multiple patterns may be specified. In this case, all csv files, all xml files, all pdf files will be captured.
- **{***hvr_tbl_name***}** is only used when data is replicated from structured files to a database with multiple tables. If there are multiple tables in your channel, the capture job needs to determine to which table a file should be replicated and will use the file name for this. In this case, the **Capture/Pattern** must be defined. The **Capture /Pattern** is not required for channels with only 1 table in them.

  Example: In your channel, you have a file **audit.csv** that needs to be replicated to a table called **file_a**, in which column names are the same as in csv file. To do this, the following actions should be defined on the source group **Cap ture /Pattern={file_a}.csv** and **FileFormat/Csv /HeaderLine.**
- **{***hvr_address***}** When a file is matched with this pattern, it is only replicated to integrate locations specified by the matching part of the file name. Locations can be specified as follows:
  - An integrate location name, such as **tgt1**.
  - A location group name containing integrate locations, such as **TGTGRP**.
  - 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 **src**, **tgt1**.
- **{***name***}** is only used for replicating files between directories ("blob file" or "flat file"). An example of the **{***name***}** pattern is **{abc}.txt**. The value inside the braces is an identifier. Although the 'named pattern' works the same as a wildcard (*****), but it also associates the captured file with a property named **{abc}**. This property can be used in the 'named substitution' (see **Integrate /RenameExpression**).

  Example 1: suppose a channel has capture pattern **{office}. txt** and rename expression **xx_{office}.data**. If file **paris.txt** is matched, then property **{office}** is assigned string value **p aris**. This means it is renamed to **xx_paris.data**.

  Example 2: suppose the **77-99.pdf** file on source needs to be renamed to **new-77-suff-99.pdf2** on target. In this case, the **Capture /Pattern** is **{a}-{b}.pdf**, and the **Integrate /Ren ameExpression** must be defined as **new-{a}-suff-{b}-pdf2**.

On Unix and Linux, file name matching is case sensitive (e.g. ***.lis** d oes not match file **FOO.LIS**), but on Windows and SharePoint it is case-insensitive. For FTP and SFTP the case sensitivity depends on the OS on which HVR is running, not the OS of the FTP/SFTP server.

/IgnorePattern File/FTP /Sharepoint	*pattern*	Ignore files whose names match *pattern*. For example, to ignore all files underneath sub-directory **qqq** specify ignore pattern **qqq/**/***. The rules and valid forms for **/IgnorePattern** are the same as for **/Pattern**, except that 'named patterns' are not allowed.
/IgnoreUnterminated File/FTP /Sharepoint	*pattern*	Ignore files whose last line does not match *pattern*. This ensures that incomplete files are not captured. This pattern matching is supported for UTF 8 files but not for UTF 16 file encoding.
/IgnoreSizeChanges File/FTP /Sharepoint		Changes in file size during capture is not considered an error when capturing from a file location.
/AccessDelay File/FTP /Sharepoint	*secs*	Delay reading file for *secs* seconds to ensure that writing is complete. HVR will ignore this file until its last create or modify timestamp is more than *secs* seconds old.
/UseDirectoryTime File/FTP /Sharepoint		When checking the timestamp of a file, check the modify timestamp of the parent directory (and its parent directories), as well as the file's own modify timestamp. This can be necessary on Windows when **/DeleteAfterCapture** is not defined to detect if a new file has been added by someone moving it into the file location's directory; on Windows file systems moving a file does not change its timestamp. It can also be necessary on Unix/Windows if a sub-directory containing files is moved into the file location directory. The disadvantage of this parameter is that when one file is moved into a directory, then all of the files in that directory will be captured again. This parameter cannot be defined with **/DeleteAfterCapture** (it is not necessary).

Writing Files while HVR is Capturing Files

It is often better to avoid having HVR capture from files while they are still be written. One reason is to prevent HVR replicating an incomplete version of the file to the integrate machine. Another problem is that if **/DeleteAfterCapture** is defined, then HVR will attempt to delete the file before it is even finished.

Capture of incomplete files can be avoided by defining **/AccessDelay** or **/IgnoreUnterminated**.

Another technique is to first write the data into a filename that HVR capture will not match (outside the file location directory or into a file matched with **/IgnorePattern**) and then move it when it is ready to a filename that HVR will match. On Windows this last technique only works if **/DeleteAfterCapture** is defined, because the file modify timestamp (that HVR capture would otherwise rely on) is not changed by a file move operation.

A group of files can be revealed to HVR capture together by first writing them in sub-directory and then moving the whole sub-directory into the file location's top directory together.

- If column **hvr_op** is not defined, then it default to **1** (insert). Value **0** means delete, and value **2** means update.
- Binary values can be given with the **format** attribute (see example above).
- If the **name** attribute is not supplied for the **<column>** tag, then HVR assumes that the order of the **<column>** tags inside the **<row>** matches the order in the HVR catalogs (column **col_sequence** of table **hvr_column**).

Examples

This section includes an example of using the **/IgnoreSessionName** parameter.

Using /IgnoreSessionName

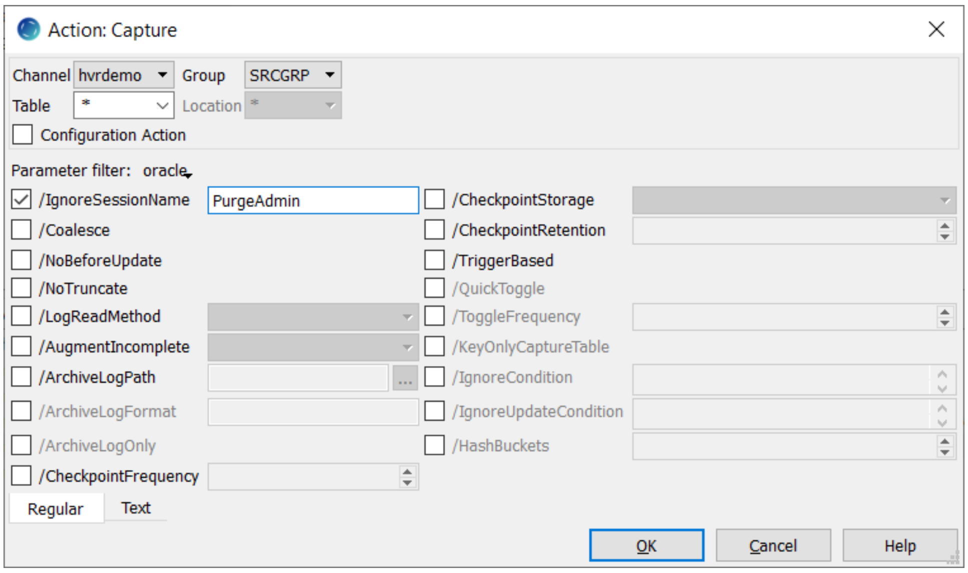
HVR allows to run a purge process on an Oracle source location without stopping active replication. Purging is deleting obsolete data from a database. To ensure that the deleted data does not replicate to a target location, the purge process must be started by a database user (e.g. **PurgeAdmin**) other than the user (e.g. **hvruser**) under which the replication process is running, and HVR must be configured to ignore the session name of the **PurgeAdmin**.

The steps for implementing this scenario are as follows:

1. In a source database, create a new user **PurgeAdmin** that will run a purge script against this database.
2. Grant the applicable permissions to user **PurgeAdmin**, e.g. a privilege to delete rows in another schema:

   ```
   grant delete any table to PurgeAdmin;
   ```

3. In the HVR GUI, update action **Capture** defined on the existing channel by adding parameter **/IgnoreSessionName**:
   a. Under the **Actions** pane, double-click a row with action **Capture**.
   b. In the **Action: Capture** dialog, select parameter **/IgnoreSessionName** and specify the user name 'PurgeAdmin'.
   c. Click **OK**.


Action: Capture

Channel hvrdemo Group SRCGRP
Table * Location *
☐ Configuration Action

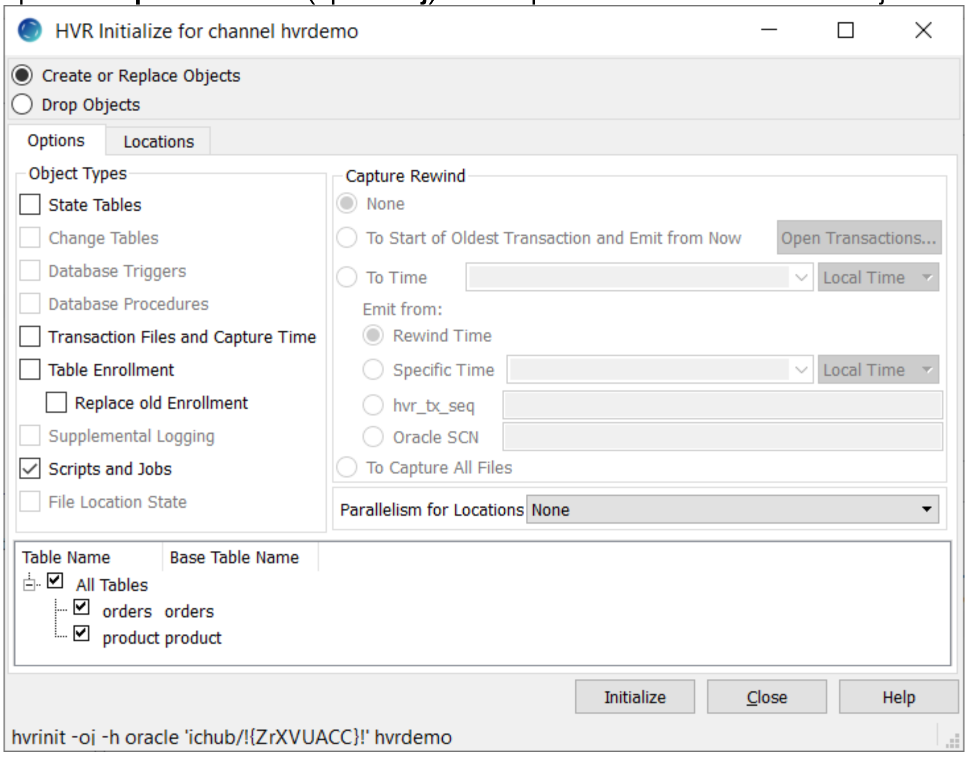
Parameter filter: oracle

☑ /IgnoreSessionName PurgeAdmin
☐ /Coalesce
☐ /NoBeforeUpdate
☐ /NoTruncate
☐ /LogReadMethod
☐ /AugmentIncomplete
☐ /ArchiveLogPath
☐ /ArchiveLogFormat
☐ /ArchiveLogOnly
☐ /CheckpointFrequency
☐ /CheckpointStorage
☐ /CheckpointRetention
☐ /TriggerBased
☐ /QuickToggle
☐ /ToggleFrequency
☐ /KeyOnlyCaptureTable
☐ /IgnoreCondition
☐ /IgnoreUpdateCondition
☐ /HashBuckets

4. Re-Initialize the capture job:

   a. In the navigation tree pane, right-click channel (e.g. **chn**) and click **HVR Initialize**.

b. In the **Options** pane, select **Script and Jobs** and click **Initialize**. Running **HVR Initialize** with option **Scripts and Jobs** (option **-oj**) will suspend and restart the affected jobs automatically.



HVR Initialize for channel hvrdemo

[● Create or Replace Objects]
○ Drop Objects

Options | Locations

Object Types
- ☐ State Tables
- ☐ Change Tables
- ☐ Database Triggers
- ☐ Database Procedures
- ☐ Transaction Files and Capture Time
- ☐ Table Enrollment
  - ☐ Replace old Enrollment
- ☐ Supplemental Logging
- ☑ Scripts and Jobs
- ☐ File Location State

Capture Rewind
- [● None]
- ○ To Start of Oldest Transaction and Emit from Now Open Transactions...
- ○ To Time Local Time
- Emit from:
  - [● Rewind Time]
  - ○ Specific Time Local Time
  - ○ hvr_tx_seq
  - ○ Oracle SCN
- ○ To Capture All Files

Parallelism for Locations: None

Table Name	Base Table Name
☑ All Tables	
☑ orders	orders
☑ product	product

Initialize | Close | Help

```
hvrinit -oj -h oracle 'ichub/!{ZrXVUACC}!' hvrdemo
```