ColumnProperties

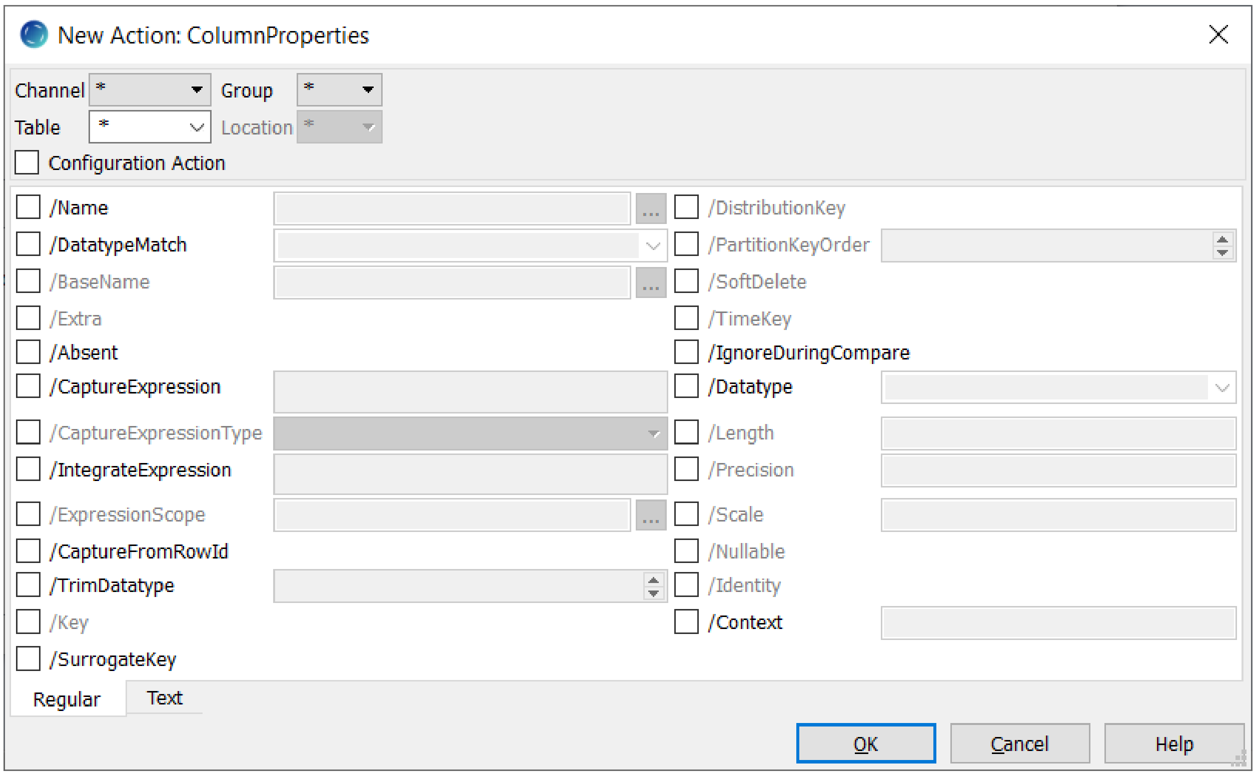
Contents

Description

Action **ColumnProperties** defines properties of a column. This column is matched either by using parameter **/Name** or **/DataType**. The action itself has no effect other than the effect of the other parameters used. This affects both replication (capture and integration) and HVR refresh and compare.

Parameters

This section describes the parameters available for action **ColumnProperties**.



New Action: ColumnProperties

Channel: * Group: *
Table: * Location: *
Configuration Action

/Name, /DatatypeMatch, /BaseName, /Extra, /Absent, /CaptureExpression, /CaptureExpressionType, /IntegrateExpression, /ExpressionScope, /CaptureFromRowId, /TrimDatatype, /Key, /SurrogateKey

/DistributionKey, /PartitionKeyOrder, /SoftDelete, /TimeKey, /IgnoreDuringCompare, /Datatype, /Length, /Precision, /Scale, /Nullable, /Identity, /Context

Parameter	Argument	Description
/Name	*col_name*	Name of column in **hvr_column** catalog. The *col_name* should not be same as the substitution defined in **/CaptureExpression** and **/IntegrateExpression**. HVR will not populate values for the column if *col_name* and substitution (*sql_expr*) are same. For example, when **/IntegrateExpression=hvr_op** is used then in that action definition **/Name=hvr_op** should not be used.

/DatatypeMatch	*datatypematch*	Data type used for matching a column, instead of **/Name**. **Since** v5.3.1/3 Value *datatypematch* can either be single data type name (such as **number**) or have form *datatype***[***condition***]**.*condition* has form *attrib ute operator value*. *attribute* can be **prec**, **scale**, **bytelen**, **charlen**, **encoding** or **null** *operator* can be **=**, **<>**, **!=**, **<**, **>**, **<=** or **>=** *value* is either an integer or a single quoted string. Multiple conditions can be supplied, which must be separated by &&. This parameter can be used to associate a **ColumnProperties** acti on with all columns which match the data type and the optional attribute conditions. Examples are: **/DatatypeMatch**=**"number"** **/DatatypeMatch**=**"number[prec>=19]"** **/DatatypeMatch**=**"varchar[bytelen>200]"** **/DatatypeMatch**=**"varchar[encoding='UTF-8' && null='true']"** **/DatatypeMatch**=**"number[prec=0 && scale=0]"** matches Oracle numbers without any explicit precision or scale.
/BaseName	*tbl_name*	This action defines the actual name of the column in the database location, as opposed to the column name that HVR has in the channel. This parameter is needed if the 'base name' of the column is different in the capture and integrate locations. In that case the column name in the HVR channel should have the same name as the 'base name' in the capture database and parameter **/BaseName** should be defined on the integrate side. An alternative is to define the **/BaseName** parameter on the capture database and have the name for the column in the HVR channel the same as the base name in the integrate database. The concept of the 'base name' in a location as opposed to the name in the HVR channel applies to both columns and tables, see **/ BaseName** in **TableProperties**. Parameter **/BaseName** can also be defined for file locations (to change the name of the column in XML tag) or for Salesforce locations (to match the Salesforce API name). Parameter **/BaseName** cannot be used together with **/Extra** and **/A bsent**.
/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.). Parameter **/Extra** cannot be used together with **/BaseName** and **/A bsent**. Parameters **/Extra** cannot be used on columns which are part of the replication key. Also, it cannot be defined in a given database on the same column, nor can either be combined on a column with parameter **/BaseName**.

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

/CaptureExpression	*sql_expr*	SQL expression for column value when capturing changes or reading rows. This value may be a constant value or an SQL expression. This parameter can be used to 'map' values data values between a source and a target table. An alternative way to map values is to define an SQL expression on the target side using **/IntegrateExpression**. Possible SQL expressions include **null**, **5** or '**hello**'. For many databases (e.g. Oracle and SQL Server) a subselect can be supplied, for example **select descrip from lookup where id={id}**. The following substitutions are allowed: • **{***colname* [*spec*]**}** is replaced/substituted with the value of current table's column *colname*. If the target column has a character based data type or if **/Datatype**=*character data type* then the default format is **%[localtime] %Y-%m-%d %H:%M:%S**, but this can be overridden using the timestamp substitution format specifier *spec*. For more information, see Timestamp Substitution Format Specifier. • **{hvr_cap_loc}** is replaced with the name of the source location where the change occurred. • **{hvr_cap_tstamp** [*spec*]**}** is replaced with the moment (time) that the change occurred in source location. If the target column has a character based data type or if **/Datatype**=*character data type* then the default format is **%[localtime] %Y-%m-%d %H:%M:%S**, but this can be overridden using the timestamp substitution format specifier *spec*. For more information, see Timestamp Substitution Format Specifier. • **{hvr_cap_user}** is replaced with the name of the user which made the change. • **{{hvr_col_name}}** is replaced with the value of the current column. • **{hvr_var_***xxx***}** is replaced with value of 'context variable' *xxx*. The value of a context variable can be supplied using option **–V** *xxx*=*val* in **hvrrefresh** or **hvrcompare**. • **{hvr_slice_num}**: is replaced with the current slice number if slicing is defined with **Count** (option **-S** *num*) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0 • **{hvr_slice_total}**: is replaced with the total number of slices if slicing is defined with **Count** (option **-S** *num*) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0 • **{hvr_slice_value}**: is replaced with the current slice value if slicing is defined with **Series** (option **-S** *val1*[**;***val2*]...) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0 It is recommended to define **/Context** when using the substitutions **{hvr_var_***xxx***}**, **{hvr_slice_num}**, **{hvr_slice_total}**, or **{hvr_slice_value}**, so that it can be easily disabled or enabled. **{hvr_slice_num}**, **{hvr_slice_total}**, **{hvr_slice_value}** cannot be used if the one of the old slicing substitutions **{hvr_var_slice_condition}**, **{hvr_var_slice_num}**, **{hvr_var_slice_total}**, or **{hvr_var_slice_value}** is defined in the channel/table involved in the compare/refresh.

/CaptureExpressionType Since v5.3.1/21	*expr_type*	Type of mechanism used by HVR's **Capture**, **Refresh** and **Compare** job to evaluate value in parameter **/CaptureExpression**. Available options: • **SQL_PER_CYCLE** (**default** for database locations if the capture expression matches a pattern in file **hvr_home/lib/constsqlexpr.pat)**: The capture job only evaluates the expression once per replication cycle, so every row captured by that cycle will get the same value. It requires less database 'round-trips' than **SQL_PER_ROW** and **SQL_WHERE_ROW**. For refresh and compare jobs the expression is just included in main select statement, so no extra database round-trips are used and the database could assign each row a different value. This type is is not supported for file locations. • **SQL_PER_ROW** (**default** for database locations if the capture expression does not match a pattern in file **hvr_home/lib/constsqlexpr.pat**): The capture job evaluates the expression for each change captured. This means every row captured by that cycle could get a different value but requires more database 'round-trips' than **SQL_PER_CYCLE**. For refresh and compare jobs the expression is just included in main **select** statement, so no extra database round-trips are used and the database could assign each row a different value. This type is is not supported for file locations. • **SQL_WHERE_ROW**: The capture job evaluates the expression for each change captured.but with an extra **where** clause containing the key value for the table on which the change occurred. This allows that expression to include expressions like **{colx}** which reference other columns of that table. Each row captured could get a different value but requires more database 'round-trips' than **SQL_PER_CYCLE**. For refresh and compare jobs the expression is just included in main **select** statement (without the extra **where** clause), so no extra database round-trips are used and the database could assign each row a different value. This type is not supported for file locations. • **INLINE**: String-based replacement by HVR itself. This type is only supported for capturing changes from file location.
/IntegrateExpression	*sql_expr*	Expression for column value when integrating changes or loading data into a target table. HVR may evaluate itself or use it as an SQL expression. This parameter can be used to 'map' values between a source and a target table. An alternative way to map values is to define an SQL expression on the source side using parameter **/CaptureExpression**. For many databases (e.g. Oracle and SQL Server) a subselect can be supplied, for example **select descrip from lookup where id={id}**. Possible expressions include **null**, **5** or '**hello**'. The following substitutions are allowed: • **{***colname* [*spec*]**}** is replaced/substituted with the value of current table's column *colname*. If the target column has a character based data type or if **/Datatype**=*character data type* then the default format is **%[localtime] %Y-%m-%d %H:%M:%S**, but this can be overridden using the timestamp substitution format specifier *spec*. For more information, see Timestamp Substitution Format Specifier.

- **{hvr_cap_loc}** is replaced with the name of the source location where the change occurred.
- **{hvr_cap_user}** is replaced with the name of the user who made the change.
- **{hvr_cap_tstamp** [*spec*]**}** is replaced with the moment (time) that the change occurred in source location. If the target column has a character based data type or if **/Datatype**=*character data type* then the default format is **%[localtime] %Y-%m-%d %H:%M:%S**, but this can be overridden using the timestamp substitution format specifier *spec*. For more information, see Timestamp Substitution Format Specifier.
- **{hvr_chn_name}** is replaced with the name of the channel.
- **{{hvr_col_name}}** is replaced with the value of the current column.
- **{hvr_integ_key}** is replaced with a 16 byte string value (hex characters) which is unique and continuously increasing for all rows integrated into the target location. The value is calculated using a high precision timestamp of the moment that the row is integrated. This means that if changes from the same source database are captured by different channels and delivered to the same target location then the order of this sequence will not reflect the original change order. This contrasts with substitution **{hvr_integ_seq}** where the order of the value matches the order of the change captured. Another consequence of using a (high precision) integrate timestamp is that if the same changes are sent again to the same target location (for example after option 'capture rewind' of **hvrinit**, or if a Kafka location's integrate job is restarted due to interruption) then the 're-sent' change will be assigned a new value. This means the target databases cannot rely on this value to detect 're-sent' data. This substitution is recommended for **/TimeKey** if the channel has multiple source locations. If the integrate location is a Kafka or file location, **/TimeKey** must be defined on that location to replicate delete operations.
- **{hvr_integ_seq}** is replaced with a 36 byte string value (hex characters) which is unique and continuously increasing for a specific source location. If the channel has multiple source locations then this substitution must be combined with **{hvr_cap_loc}** to give a unique value for the entire target location. The value is derived from source database's DBMS logging sequence, e.g. the Oracle System Change Number (SCN). This substitution is recommended for **/TimeKey** if the channel has a single source location. This substitution only has a value during **Integrate** or if **Select Moment - Specific** (option **-M***time*) was specified during **Refresh**.
- **{hvr_integ_tstamp** [*spec*]**}** is replaced with the moment (time) that the change was integrated into target location. If the target column has a character based data type or if **/Datatype**=*character data type* then the default format is **%Y-%m-%d %H:%M:%S[.SSS]**, but this can be overridden using the timestamp substitution format specifier spec. For more information, see Timestamp Substitution Format Specifier.
- **{{hvr_key_names** *sep***}}** is replaced with the values of table's key columns, concatenated together with separator *sep*.
- **{hvr_key_names** *sep***}** is replaced with the names of table's key columns, concatenated together with separator *sep*. **Since** v5.5.5/
- **{hvr_op}** is replaced with the HVR operation type. Values are **0** (delete), **1** (insert), **2** (after update), **3** (before key update), **4** (before non–key update) or **5** (truncate table). See also Extra Columns for Capture, Fail and History Tables. Note that this substitution cannot be used with parameter **/ExpressionScope**. If the integrate location is a Kafka or file location, this substitution should be used as extra column (**/Extra**).

- **{hvr_schema}** is replaced with the schema name of the table. This is only allowed if the channel is defined with tables and this can only be used when an explicit **TableProperties /Schema**=*my_schema* is defined for these tables on the target file location.
- **{hvr_tbl_name}** is replaced with the name of the current table.
- **{hvr_tbl_base_name}** is replaced with the base name of the current table
- **{hvr_tx_countdown}** is replaced with countdown of changes 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 **0** indicates that commit information is missing for that change. This substitution only has a value during **Integrate** or if **Select Moment - Specific** (option **-M***time*) was specified during **Refresh**.
- **{hvr_tx_scn}** is replaced with the source location's SCN (Oracle). This substitution can only be used if the source location database is Oracle. This substitution can only be used for ordering if the channel has a single source location. This substitution only has a value during **Integrate** or if **Select Moment - Specific** (option **-M***time*) was specified during **Refresh**.
- **{hvr_tx_seq}** is replaced with a hex representation of the sequence number of transaction. For capture from Oracle this value can be mapped back to the SCN of the transaction's commit statement. Value **[hvr_tx_seq, -hvr_tx_countdown]** is increasing and uniquely identifies each change. This substitution only has a value during **Integrate** or if **Select Moment - Specific** (option **-M***time*) was specified during **Refresh**.
- **{hvr_var_***xxx***}** is replaced with value of 'context variable' *xxx*. The value of a context variable can be supplied using option **–V** *xxx*=*val* to command **hvrrefresh** or **hvrcompare**.
- **{hvr_slice_num}**: is replaced with the current slice number if slicing is defined with **Count** (option **-S** *num*) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0
- **{hvr_slice_total}**: is replaced with the total number of slices if slicing is defined with **Count** (option **-S** *num*) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0
- **{hvr_slice_value}**: is replaced with the current slice value if slicing is defined with **Series** (option **-S** *val1*[**;***val2*]...) in **hvrrefresh** or **hvrcompare**. **Since** v5.6.5/0

It is recommended to define **/Context** when using the substitutions **{hvr_var_***xxx***}**, **{hvr_slice_num}**, **{hvr_slice_total}**, or **{hvr_slice_value}**, so that it can be easily disabled or enabled.

{hvr_slice_num}, **{hvr_slice_total}**, **{hvr_slice_value}** cannot be used if the one of the old slicing substitutions **{hvr_var_slice_condition}**, **{hvr_var_slice_num}**, **{hvr_var_slice_total}**, or **{hvr_var_slice_value}** is defined in the channel/table involved in the compare /refresh.

/ExpressionScope	*expr_scope*	Scope for which operations (e.g. insert or delete) an integrate expression (parameter **/IntegrateExpression**) should be used. Available options for *expr_scope* are: • **DELETE** • **INSERT** • **UPDATE_AFTER** • **UPDATE_BEFORE_KEY** • **UPDATE_BEFORE_NONKEY** • **TRUNCATE** When multiple *expr_scope* are defined, it should be a comma-separated list. Values **DELETE** and **TRUNCATE** can be used only if parameter **/SoftDelete** or **/TimeKey** is defined. This parameter can be used only when Integrate **/Burst** is defined. It is ignored for database targets if **/Burst** is not defined and for file-targets (such as HDFS or S3). This burst restriction means that no scopes exist yet or for 'update before' operations (such as **UPDATE_BEFORE_KEY** and **UPDATE_BEFORE_NONKEY**). Only bulk refresh obeys this parameter (it always uses scope **INSERT**); row-wise refresh ignores the expression scope. This value of the affected **/IntegrateExpression** parameter can contain its regular substitutions except for **{hvr_op}** which cannot be used. **Example 1:** To add a column **opcode** to a target table (defined with **/SoftDelete**) containing values **'I'**, **'U'** and **'D'** (for insert, update and delete respectively), define these actions; • **ColumnProperties /Name**=**opcode /Extra /IntegrateExpression**="**'I'**" **/ExpressionScope**=**INSERT /Datatype**=**varchar /Length**=**1 /Nullable** • **ColumnProperties /Name**=**opcode /Extra /IntegrateExpression**="**'U'**" **/ExpressionScope**=**UPDATE /Datatype**=**varchar /Length**=**1 /Nullable** • **ColumnProperties /Name**=**opcode /Extra /IntegrateExpression**="**'D'**" **/ExpressionScope**=**DELETE /Datatype**=**varchar /Length**=**1 /Nullable** **Example 2:** To add a column **insdate** (only filled when a row is inserted) and column **upddate** (filled on update and [soft]delete), define these actions; • **ColumnProperties /Name**=**insdate /Extra /IntegrateExpression**=**sysdate /ExpressionScope**=**INSERT /Datatype**=**timestamp** • **ColumnProperties /Name**=**upddate /Extra /IntegrateExpression**=**sysdate /ExpressionScope**=**DELETE, UPDATE_AFTER /Datatype**=**timestamp** Note that **HVR Refresh** can create the target tables with the **/Extra** columns, but if the same column has multiple actions for different scopes then these must specify the same data type (parameters **/Datatype** and **/Length**).

/CaptureFromRowId		Capture values from table's DBMS row-id (Oracle, HANA) or Relative Record Number (RRN in Db2 for i). Define on the capture location. This parameter is supported only for certain location classes. For the list of supported location class, see Log-based capture from hidden rowid/RRN column in Capabilities. This parameter is not supported for Oracle's Index Organized Tables (IOT).
/TrimDatatype	*int*	Reduce width of data type when selecting or capturing changes. This parameter affects string data types (such as **varchar**, **nvachar** and **clob**) and binary data types (such as **raw** and **blob**). The value is a limit in bytes; if this value is exceeded then the column's value is truncated (from the right) and a warning is written. An example of usage is **ColumnProperties /DatatypeMatch**=**clob /TrimDatatype**=**10 /Datatype**=**varchar /Length**=**30** which will replicate all columns with data type **clob** into a target table as strings. Note that parameter **/Datatype** and **/Length** ensures that HVR Refresh will create target tables with the smaller data type. Its length is smaller because **/Length** parameter is used. This parameter is supported only for certain location classes. For the list of supported location class, see Reduce width of datatype when selecting or capturing changes in Capabilities.
/Key		Add column to table's replication key.
/SurrogateKey		Use column instead of the regular key during replication. Define on the capture and integrate locations. Should be used in combination with **/CaptureFromRowId** to capture from HANA or Db2 for i or Oracle tables to reduce supplemental logging requirements. Integrating with **/SurrogateKey** is impossible if the **/SurrogateKey** column is captured from a **/CaptureFromRowId** that is reusable (Oracle).
/DistributionKey		Distribution key column. The distribution key is used for parallelizing changes within a table. It also controls the **distributed by** clause for a create table in distributed databases such as Teradata, Redshift and Greenplum.

/PartitionKeyOrder Since v5.5.5/0 Hive ACID	*int*	Define the column as a partition key and set partitioning order for the column. When more than one columns are used for partitioning then the order of partitions created is based on the value *int* (beginning with **0**) provided in this parameter. If this parameter is selected then it is mandatory to provide value *int*. Example: for a table **t** with columns **- col1**, **col2**, **col3**, 1. if **ColumnProperties /Name**= **col2** and **ColumnProperties /PartitionKeyOrder**=**0** then this is transformed into: **create table t (col1, col3) partitioned by (col2)** statement in Hive ACID. 2. if **ColumnProperties /Name**= **col3** and **ColumnProperties /PartitionKeyOrder**=**0** and **ColumnProperties /Name**= **col2** and **ColumnProperties /PartitionKeyOrder**=**0** then this is transformed into: **create table t (col1) partitioned by (*col2, col3*)** statement in Hive ACID. In this case the actual order of columns in a table is used for partitioning. 3. if **ColumnProperties /Name**= **col3** and **ColumnProperties /PartitionKeyOrder**=**0** and **ColumnProperties /Name**= **col2** and **ColumnProperties /PartitionKeyOrder**=**1** then this is transformed into: **create table t (col1) partitioned by (*col3, col2*)** statement in Hive ACID.
/SoftDelete		Convert **delete** operations in the source into **update** in the target. Defining this parameter avoids the actual deletion of rows in the target. Instead an extra column is added to indicate whether a row was deleted in the source. The initial value in this columns is **0**, indicating the row is not deleted. The value of this column is updated to **1** when a row is deleted in the source. In each integrate cycle the changes are coalesced and optimized away. For example, if an **insert** and **delete** operation performed on a row happens in the same integrate cycle, these changes will be coalesced and optimized. As a result, a soft-deleted row (with value **1**) will not be added in the target. If the same changes happen in two separate integrate cycles, in the first cycle a row will be inserted in the target and in the second cycle the row will be marked as deleted (value **1**) in the target.
/TimeKey		Convert all changes (**insert**, **update** and **delete**) in the source into **insert** in the target. Defining this parameter affects how all changes are delivered into the target table. This parameter is often used with **/IntegrateExpression**=**{hvr_integ_seq}**, which will populate a value. HVR uses the concept of TimeKey to indicate storing history. TimeKey is defined with an extra column on the target for every table uniquely storing the sequence in which changes came in to the channel. Action **ColumProperties /IntegrateExpression {hvr_integ_seq}** uniquely and deterministically defines the order in which changes were applied to the source system for the channel. For Kafka and File locations, this parameter must be defined to replicate **delete** operation.

/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. This parameter is ignored during row-wise compare/refresh if it is defined on a key column.
/Datatype	*data_type*	Data type in database if this differs from **hvr_column** catalog.
/Length	*attr_val*	String length in database if this differs from value defined in **hvr_column** catalog. When used together with **/Name** or **/DatatypeMatch**, keywords **bytelen** and **charlen** can be used and will be replaced by respective values of matched column. Additionally, basic arithmetic (+,-,*,/) can be used with **bytelen** and **charlen**, e.g., **/Length**=**"bytelen/3"** will be replaced with the byte length of the matched column divided by 3. Parameter **/Length** can only be used if **/Datatype** is defined.
/Precision	*attr_val*	Integer precision in database if this differs from value defined in **hvr_column** catalog. When used together with **/Name** or **/DatatypeMatch**, keywords **prec** can be used and will be replaced by respective values of matched column. Additionally, basic arithmetic (+,-,*,/) can be used with **prec**, e.g., **/Precision**=**"prec+5"** will be replaced with the precision of the matched column plus 5. Parameter **/Precision** can only be used if **/Datatype** is defined.
/Scale	*attr_val*	Integer scale in database if this differs from value defined in **hvr_column** catalog. When used together with **/Name** or **/DatatypeMatch**, keyword **scale** can be used and will be replaced by respective values of matched column. Additionally, basic arithmetic (+,-,*,/) can be used with **scale**, e.g., **/Scale**=**"scale*2"** will be replaced with the scale of the matched column times 2. Parameter **/Scale** can only be used if **/Datatype** is defined.
/Nullable		Nullability in database if this differs from value defined in **hvr_column** catalog. Parameter **/Nullable** can only be used if **/Datatype** is defined.
/Identity **SQL Server**		Column has SQL Server identity attribute. Only effective when using integrate database procedures (**Integrate /DbProc**). Parameter **/Identity** can only be used if **/Datatype** is defined.
/Context	*ctx*	Ignore action unless **Hvrrefresh** or **Hvrcompare** context *ctx* is enabled. The value must be the name of the context (a lowercase identifier). It can also be specified as **!***ctx*, which means that the action is effective unless context *ctx* is enabled. One or more contexts can be enabled for **Hvrcompare** or **Hvrrefresh** (on the command line with option **–C***ctx*). Defining an action which is only effective when a context is enabled can have different uses. For example, if action **ColumnProperties** with parameters **/IgnoreDuringCompare** and **/Context**=**qqq** is defined, then normally all data will be compared, but if context **qqq** is enabled (**-Cqqq**), then the values in one column will be ignored.

Columns Which Are Not Enrolled In Channel

Normally all columns in the location's table (the 'base table') are enrolled in the channel definition. But if there are extra columns in the base table (either in the capture or the integrate database) which are not mentioned in the table's column information of the channel, then these can be handled in two ways:

- They can be included in the channel definition by adding action **ColumnProperties /Extra** to the specific location. In this case, the SQL statements used by HVR integrate jobs will supply values for these columns; they will either use the **/IntegrateExpression** or if that is not defined, then a default value will be added for these columns (**NULL** for nullable data types, or **0** for numeric data types, or **''** for strings).
- These columns can just not be enrolled in the channel definition. The SQL that HVR uses for making changes will then not mention these 'unenrolled' columns. This means that they should be nullable or have a default defined; otherwise, when HVR does an insert it will cause an error. These 'unenrolled' extra columns are supported during HVR integration and HVR compare and refresh, but are not supported for HVR capture. If an 'unenrolled' column exists in the base table with a default clause, then this default clause will normally be respected by HVR, but it will be ignored during bulk refresh on Ingres, or SQL Server unless the column is a 'computed' column.

Substituting Column Values Into Expressions

HVR has different actions that allow column values to be used in SQL expressions, either to map column names or to do SQL restrictions. Column values can be used in these expressions by enclosing the column name embraces, for example a restriction "**{price} > 1000**" means only rows where the value in price is higher than 1000.

But in the following example it could be unclear which column name should be used in the braces;

> Imagine you are replicating a source base table with three columns (**A**, **B**, **C**) to a target base table with just two columns named (**E**, **F**). These columns will be mapped together using HVR actions such as **ColumnProperties /CaptureExpression** or **/IntegrateExpression**. If these mapping expressions are defined on the target side, then the table would be enrolled in the HVR channel with the source columns (**A**, **B**, **C**). But if the mapping expressions are put on the source side then the table would be enrolled with the target columns (**D**, **E**). Theoretically mapping expressions could be put on both the source and target, in which case the columns enrolled in the channel could be different from both, e.g. (**F**, **G**, **H**), but this is unlikely.
>
> But when an expression is being defined for this table, should the source column names be used for the brace substitution (e.g. **{A}** or **{B}**)? Or should the target parameter be used (e.g. **{D}** or **{E}**)? The answer is that this depends on which parameter is being used and it depends on whether the SQL expression is being put on the source or the target side.

For parameters **ColumnProperties /IntegrateExpression** and **Restrict /IntegrateCondition** the SQL expressions can only contain {} substitutions with the column names as they are enrolled in the channel definition (the "HVR Column names"), not the "base table's" column names (e.g. the list of column names in the target or source base table). So in the example above substitutions **{A}** **{B}** and **{C}** could be used if the table was enrolled with the columns of the source and with mappings on the target side, whereas substitutions **{E}** and **{F}** are available if the table was enrolled with the target columns and had mappings on the source.

But for **ColumnProperties /CaptureExpression**, **Restrict /CaptureCondition**, and **Restrict /RefreshCondition** the opposite applies: these expressions must use the "base table's" column names, not the "HVR column names". So in the example these parameters could use **{A}** **{B}** and **{C}** as substitutions in expressions on the source side, but substitutions **{E}** and **{F}** in expressions on the target.

Timestamp Substitution Format Specifier

Timestamp substitution format specifiers allows explicit control of the format applied when substituting a timestamp value. These specifiers can be used with **{hvr_cap_tstamp**[*spec*]**}**, **{hvr_integ_tstamp**[*spec*]**}**, and **{***colname* [*spec*]**}** if *colname* has timestamp data type. The components that can be used in a timestamp format specifier *spec* are:

Component	Description	Example
%a	Abbreviate weekday according to current locale.	Wed
%b	Abbreviate month name according to current locale.	Jan
%d	Day of month as a decimal number (01–31).	07
%H	Hour as number using a 24–hour clock (00–23).	17
%j	Day of year as a decimal number (001–366).	008
%m	Month as a decimal number (01 to 12).	04
%M	Minute as a decimal number (00 to 59).	58
%s	Seconds since epoch (1970–01–01 00:00:00 UTC).	1099928130
%S	Second (range 00 to 61).	40
%T	Time in 24–hour notation (%H:%M:%S).	17:58:40
%U	Week of year as decimal number, with Sunday as first day of week (00 – 53).	30
%V Linux	The ISO 8601 week number, range 01 to 53, where week 1 is the first week that has at least 4 days in the new year.	15
%w	Weekday as decimal number (0 – 6; Sunday is 0).	6
%W	Week of year as decimal number, with Monday as first day of week (00 – 53)	25
%y	Year without century.	14
%Y	Year including the century.	2014
%[localtime]	Perform timestamp substitution using machine local time (not UTC). This component should be at the start of the specifier (e.g. **{{hvr_cap_tstamp %[localtime]%H}}**).	
%[utc]	Perform timestamp substitution using UTC (not local time). This component should be at the start of the specifier (e.g. **{{hvr_cap_tstamp %[utc]%T}}**).	