

FileFormat

Contents

- [Description](#)
- [Parameters](#)
- [HVR's XML Format](#)
 - [Simple Example](#)
 - [Extended Example](#)
- [Capture and Integrate Converters](#)
 - [Environment](#)
 - [Examples](#)

Description

Action **FileFormat** can be used on file locations (including HDFS and S3) and on Kafka locations.

For file location, it controls how HVR's read and write files. The default format for file locations is [HVR's own XML format](#).

For Kafka, this action controls the format of each message. HVR's Kafka location sends messages to the Kafka broker in JSON format by default, unless the location property [Kafka_Schema_Registry](#) is defined, in which case each message uses Kafka Connect's compact Avro-based format. Note that this is not a true Avro because each message would not be a valid Avro file (e.g. no file header). Rather, each message is a 'micro Avro', containing fragments of data encoded using Avro's data type serialization format. Both JSON (using mode `SCHEMA_PAYLOAD`, see parameter [JsonMode](#) below) and the 'micro AVRO' format conform to Confluent's 'Kafka Connect' message format standard. The default Kafka message format can be overridden by parameter such as [Xml](#), [Csv](#), [Avro](#), [Json](#) or [Parquet](#).

A custom format can be defined by using parameters [CaptureConverter](#) or [IntegrateConverter](#). Many parameters only have effect if the channel contains table information; for a 'blob file channel' the jobs do not need to understand the file format.



- If this action is defined on a specific table, then it affects all tables in the same location.
- Defining more than one file format ([Xml](#), [Csv](#), [Avro](#), [Json](#) or [Parquet](#)) for the same file location using this action is not supported, i.e., defining different file formats for each table in the same location is not possible. For example, if one table has the file format defined as [Xml](#) then another table in the same location cannot have [Csv](#) file format defined.

Parameters

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

New Action: FileFormat ✕

*All Channels ▼
*All Locations ▼
*All Tables ▼

<input type="checkbox"/> Xml <input type="checkbox"/> Csv <input type="checkbox"/> Avro <input type="checkbox"/> Json <input type="checkbox"/> Parquet <input type="checkbox"/> Compact <input type="checkbox"/> Compress ▼ <input type="checkbox"/> Encoding ▼ <input type="checkbox"/> HeaderLine <input type="checkbox"/> FieldSeparator <input type="checkbox"/> LineSeparator <input type="checkbox"/> QuoteCharacter <input type="checkbox"/> EscapeCharacter <input type="checkbox"/> FileTerminator <input type="checkbox"/> NullRepresentation	<input type="checkbox"/> JsonMode <input type="checkbox"/> BlockCompress <input type="checkbox"/> AvroVersion <input type="checkbox"/> PageSize <input type="checkbox"/> RowGroupThreshold <input type="checkbox"/> ParquetVersion <input type="checkbox"/> BeforeUpdateColumns <input type="checkbox"/> BeforeUpdateColumnsWhenChanged <input type="checkbox"/> ConvertNewlinesTo ▼ <input type="checkbox"/> CaptureConverter <input type="checkbox"/> CaptureConverterArguments <input type="checkbox"/> IntegrateConverter <input type="checkbox"/> IntegrateConverterArguments <input type="checkbox"/> Context	<div style="border: 1px solid #ccc; height: 20px; margin-bottom: 5px;"></div> <div style="border: 1px solid #ccc; height: 20px; margin-bottom: 5px;"></div> <div style="border: 1px solid #ccc; height: 20px; margin-bottom: 5px;"></div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">PageSize</div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">RowGroupThreshold</div> <div style="border: 1px solid #ccc; height: 20px; margin-bottom: 5px;"></div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">BeforeUpdateColumns</div> <div style="border: 1px solid #ccc; height: 20px; margin-bottom: 5px;"></div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">CaptureConverter 📁</div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">CaptureConverterArguments</div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">IntegrateConverter 📁</div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">IntegrateConverterArguments</div> <div style="border: 1px solid #ccc; padding: 2px 10px; margin-bottom: 5px;">Context</div>
---	--	--

Regular
Cancel
Save

Parameter	Argument	Description
Xml		Read and write files as HVR's XML format . Default. This parameter is applicable only for the channels with table information; not a 'blob file'.
Csv		Read and write files as Comma Separated Values (CSV) format. This parameter is applicable only for the channels with table information; not a 'blob file'.

<p>Avro</p>		<p>Transforms the captured rows into Avro format during Integrate.</p> <p>An Avro file contains the schema defining data types in JSON and a compact binary representation of the data. See Apache Avro documentation for the detailed description of schema definition and data representation.</p> <p>Avro supports primitive and logical data types. The normal way to represent Avro file in human-readable format is converting it to JSON via Apache Avro tools.</p> <p>However, there is a drawback in representing decimal values using standard Avro-tools utilities.</p> <p>The decimal type in Avro is supported as a logical type and is defined in the Avro schema file as follows:</p> <pre> { "type": "bytes", "logicalType": "decimal", "precision": precision, "scale": scale } </pre> <ul style="list-style-type: none"> • Precision is the total number of digits in the number. • Scale is the number of digits after the decimal point. <p>The decimal logical type represents an arbitrary-precision signed decimal number of the form unscaled x 10^{-scale}. For example, value 1.01 with a precision of 3 and scale of 2, is represented as 101.</p> <p>Decimal values are encoded as a sequence of bytes in Avro. In their JSON representation, decimal values are displayed as an unreadable string instead of human-readable values.</p> <p>For example:</p> <p>A source table is defined as follows:</p> <pre> create table Dec (c1 number(10,2) , c2 number(10,4)); </pre> <p>wherein column c1 stores a decimal value with precision 10 and scale 2, and column c2 stores a decimal value with precision 10 and scale 4.</p> <p>If we insert values (1, 1) into Dec table and select them from the table, we expect to see (1, 1) as an output.</p> <p>But Avro format uses the specified scales and represents them in binary format as 100 (1.00) in column c1 and 10000 (1.0000) in column c2. According to the JSON specification, a binary array is encoded as a string.</p> <p>JSON will display these values as "d" (wherein "d" is 100 according to ASCII) and ""x10" (wherein 10000 is 0x2710, and 0x27 is ' according to the ASCII encoding).</p> <p>Formats like Parquet with ParquetVersion=v2 or v3, Json with JsonMode=SCHEMA_PAYLOAD uses the same rules to encode decimal data types.</p> <p>When using Hive (Hive external table) to read Avro files, the decimal data type is displayed properly.</p>
<p>Json</p>		<p>Transforms the captured rows into JSON format during Integrate. The content of the file depends on the value for parameter JsonMode.</p>
<p>Parquet</p>		<p>Transforms the captured rows into Parquet format during Integrate.</p>
<p>Compact</p>		<p>Write compact XML tags like <code><r></code> & <code><c></code> instead of <code><row></code> and <code><column></code>.</p> <p>This parameter can be used only if Xml is selected.</p>
<p>Compress</p>	<p><i>algorithm</i></p>	<p>HVR will compress files while writing them, and uncompress them while reading.</p> <p>Available options for <i>algorithm</i> are:</p> <ul style="list-style-type: none"> • GZIP • LZ4 <p>The file suffix is ignored but when integrated, a suffix can be added to the files by defining action Integrate with parameter RenameExpression="{hvr_cap_filename}.gz".</p> <p>This parameter is not supported for Avro and Parquet.</p>

Encoding	<i>encoding</i>	<p>Encoding for reading or writing files.</p> <p>Available options for <i>encoding</i> are:</p> <ul style="list-style-type: none"> • US-ASCII • ISO-8859-1 • ISO-8859-9 • WINDOWS-1251 • WINDOWS-1252 • UTF-8 • UTF-16LE • UTF-16BE
HeaderLine		<p>First line of CSV file contains column names.</p>
FieldSeparator	<i>str_esc</i>	<p>Field separator for CSV files.</p> <p>The default value for this parameter is comma (,).</p> <p>Note that only a single Unicode glyph is supported as a separator for this parameter.</p> <p>Examples: , \x1f or \t.</p> <p>This parameter can be used only if the parameter Csv is selected.</p>
LineSeparator	<i>str_esc</i>	<p>Line separator for CSV files.</p> <p>The default value for this parameter is newline (\n).</p> <p>Examples: ;\n or \r\n</p> <p>This parameter can be used only if the parameter Csv is selected.</p>
QuoteCharacter	<i>str_esc</i>	<p>Character to quote a field with, if the fields contains separators.</p> <p>The default value for this parameter is double quotes (").</p> <p>This parameter can be used only if the parameter Csv is selected.</p>
EscapeCharacter	<i>str_esc</i>	<p>Character to escape the quote character with.</p> <p>The default value for this parameter is double quotes (").</p> <p>This parameter can be used only if the parameter Csv is selected.</p>
FileTerminator	<i>str_esc</i>	<p>File termination at end-of-file.</p> <p>Example: EOF or \xff.</p> <p>This parameter can be used only if the parameter Csv is selected.</p>
NullRepresentation	<i>str_esc</i>	<p>String representation for column with NULL value.</p> <p>Note that Hive 'deserializers' recognize \N as NULL when reading a CSV file back as an SQL row, this can be configured by setting this parameter to \N".</p> <p>Example: \N</p> <p>This parameter can be used only if the parameter Csv is selected.</p>

<p>JsonMode</p>	<p><i>mode</i></p>	<p>Style used to write row into JSON format.</p> <p>This parameter can be used only if the parameter Json is selected.</p> <p>Available options for <i>mode</i> are:</p> <ul style="list-style-type: none"> <p>ROW_FRAGMENTS: This format is compatible with Hive and BigQuery deserializers. Note that this option produces an illegal JSON file as soon as there is more than one row in the file.</p> <p>Example:</p> <pre>{ "c1":44, "c2":55 } { "c1":66, "c2":77 }</pre> <p>ROW_ARRAY:</p> <p>Example:</p> <pre>[{ "c1":44, "c2":55 }, { "c1":66, "c2":77 }]</pre> <p>TABLE_OBJECT (<small>default</small> JSON mode for all location types except for Kafka):</p> <p>Example:</p> <pre>{ "tab1" : [{ "c1":44, "c2":55 }, { "c1":66, "c2":77 }] }</pre> <p>TABLE_OBJECT_BSON: This format is the same as TABLE_OBJECT, but in BSON format (binary). Note that a BSON file cannot be bigger than 2GB. This makes this format inapplicable for some tables (e.g. when LOB values are present).</p> <p>TABLE_ARRAY: This mode is useful if parameter RenameExpression in action Integrate does not contain a substitution which depends on the table name and when the location class is not Kafka.</p> <p>Example:</p> <pre>[{ "tab1" : [{ "c1":44, "c2":55 }, { "c1":66, "c2":77 }] }, { "tab2" : [{ "c1":88, "c2":99 }] }]</pre> <p>SCHEMA_PAYLOAD (<small>default</small> JSON mode for Kafka): This format is compatible with Apache Kafka Connect deserializers. Note that this option produces an illegal JSON file as soon as there is more than one row in the file.</p> <pre>{ "schema": {"type":"struct", "name":"tab1", "fields": [{"name":"c1", "type":"int"}, {"name":"c2", "type":"int"}]}, "payload": { "c1": 44, "c2":55 }} { "schema": {"type":"struct", "name":"tab1", "fields": [{"name":"c1", "type":"int"}, {"name":"c2", "type":"int"}]}, "payload": { "c1": 66, "c2":77 }} }</pre>
------------------------	--------------------	---

<p>BlockCompress</p>	<p><i>algorithm</i></p>	<p>Compression <i>algorithm</i> for Avro and Parquet. This parameter sets the native compression algorithm supported by these file formats.</p> <p>Available options for <i>algorithm</i> are:</p> <ul style="list-style-type: none"> • NONE This option may be used with Avro and Parquet. • DEFLATE This option may be used only with Avro. Also, this is the default compression method for Avro if BlockCompress is not defined. • GZIP This option may be used only with Parquet. • LZ4 This option may be used only with Parquet. Also, this is the default compression method for Parquet if BlockCompress is not defined. <p>This parameter can be used only if the parameter Avro or Parquet is selected.</p>
<p>AvroVersion</p>	<p><i>version</i></p>	<p>Version of Avro format. Available options for <i>version</i> are:</p> <ul style="list-style-type: none"> • v1_6: This version supports only the following basic types: Boolean, int (32-bit size), long (64-bit size), float, double, bytes, and string. • v1_7: This version supports only the following basic types: Boolean, int (32-bit size), long (64-bit size), float, double, bytes, and string. • v1_8 (default): This version supports the above mentioned basic types and the following logical types: decimal, date, time and timestamp (with micro and millisecond resolutions), and duration. <p>This parameter can be used only if the parameter Avro is selected.</p>
<p>PageSize</p>		<p>Parquet page size in bytes.</p> <p>Default value is 1MB.</p> <p>This parameter can be used only if the parameter Parquet is selected.</p>
<p>RowGroupThreshold</p>		<p>Maximum row group size in bytes for Parquet.</p> <p>This parameter can be used only if the parameter Parquet is selected.</p>
<p>ParquetVersion</p>	<p><i>version</i></p>	<p>Category of data types to represent complex data into Parquet format.</p> <ul style="list-style-type: none"> • v1 : Supports only the basic data types - boolean, int32, int64, int96, float, double, byte_array to represent any data. The logical data types decimal and date/time types are not supported. However, decimal is encoded as double, and date/time types are encoded as int96. • v2 (default): Supports all basic data types and one logical data type (decimal). The date/time types are encoded as int96. This is compatible with Hive, Impala, Spark, and Vertica. • v3 : Supports basic data types and logical data types - decimal, date, time_millis, time_micros, timestamp_millis, timestamp_micros. <p>For Hive, the date/time data type is encoded as int96. So, for Hive with date/time in source, only v1 or v2 can be used.</p> <p>For more information about parquet data types, refer to Parquet Documentation.</p> <p>This parameter can be used only if the parameter Parquet is selected.</p>
<p>BeforeUpdateColumns</p> <p>File / FTP/ SFTP</p> <p>Kafka</p>	<p><i>prefix</i></p>	<p>By default, the update operation is captured as 2 rows: 'before' and 'after' versions of a row. During the update operation, this option merges these two rows into one and adds user-defined <i>prefix</i> to all the columns of the 'before' version.</p> <p>For example:</p> <pre>insert into t values (1, 1, 1) update t set c2=2 where c1=1</pre> <p>The output will be as follows:</p> <pre>{"c1": 1, "c2": 2, "c3": 1, "old&c1": 1, "old&c2": 1, "old&c3": 1}</pre> <p>where old& is a user-defined <i>prefix</i>.</p> <p>For File / FTP/ SFTP, this parameter can be used only if the parameter Xml, Csv, Avro, Json, or Parquet is selected.</p>

<p>BeforeUpdateColumnsWhenChanged</p> <p>File / FTP/ SFTP</p> <p>Kafka</p>		<p>Adds the user-defined <i>prefix</i> (BeforeUpdateColumns) only to columns in which values were updated. This is supported only for JSON and XML formats.</p> <p>This option can be applied only when parameter BeforeUpdateColumns is selected.</p> <p>For example:</p> <pre>insert into t values (1, 1, 1) update t set c2 =2 where c1=1</pre> <p>The output will be as follows:</p> <pre>{"c1": 1, "c2": 2, "c3": 1, "old&c2": 1}</pre> <p>where old& is a <i>prefix</i> defined in the parameter BeforeUpdateColumns.</p> <p>For File / FTP/ SFTP, this parameter can be used only if the parameter Xml or Json is selected.</p>
<p>ConvertNewlinesTo</p>	<p><i>style</i></p>	<p>Write files with UNIX or DOS style newlines.</p>
<p>CaptureConverter</p>	<p><i>path</i></p>	<p>Run files through converter before reading. Value <i>path</i> 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. #!perl.</p> <p>A converter command should read from its stdin and write to stdout. Argument <i>path</i> can be an absolute or a relative pathname. If a relative pathname is supplied the command should be located in HVR_CONFIG/plugin/transform.</p> <p>This field is displayed only when the action definition is for a file location or if the channel contain a file location.</p>
<p>CaptureConverterArguments</p>	<p><i>userarg</i></p>	<p>Arguments to the capture converter.</p> <p>This field is displayed only when the action definition is for a file location or if the channel contain a file location.</p>
<p>IntegrateConverter</p>	<p><i>path</i></p>	<p>Run files through converter before writing. Value <i>path</i> 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. #!perl.</p> <p>A converter command should read from its stdin and write to stdout. Argument <i>path</i> can be an absolute or a relative pathname. If a relative pathname is supplied the command should be located in HVR_CONFIG/plugin/transform.</p>
<p>IntegrateConverterArguments</p>	<p><i>userarg</i></p>	<p>Arguments to the integrate converter program.</p>
<p>Context</p>	<p><i>context</i></p>	<p>Action only applies if Refresh or Compare context matches.</p>

HVR's XML Format

The XML schema used by HVR can be found in **HVR_HOME/etc/xml/hvr.dtd**.

Simple Example

Following is a simple example of an XML file containing changes which were replicated from a database location.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hvr version="1.0">
<table name="dm01_product">
<row>
<column name="prod_id">1</column>
<column name="prod_price">30</column>
<column name="prod_descrip">DVD</column>
</row>
<row>
<column name="prod_id">2</column>
<column name="prod_price">300</column>
<column name="prod_descrip" is_null="true"/>
</row>
</table>
</hvr>
```

Extended Example

Following is an extended example of HVR's XML.

1. Create tables in Oracle:

```
create table mytab (aa number not null, bb date, constraint mytab_pk primary key (aa));
create table tabx (a number not null, b varchar2(10) not null, c blob, constraint tabx_pk primary key (a, b));
```

2. Switch to a different user to create new table with same name.

```
create table tabx (c1 number, c2 char(5), constraint tabx_pk primary key (c1));
```

3. Defined an HVR channel with the following actions:

Group	Table	Action	Parameter(s)	Remarks
SOURCE	*	Capture		
TARGET	*	Integrate		
TARGET	*	ColumnProperties	Name=hvr_op_val IntegrateExpression={hvr_op} Extra	Causes an extra column named hvr_op_val to be shown, which indicates the operation type (0=delete, 1=insert, 2=update, 3=before key update, 4=before non-key update).
TARGET	*	ColumnProperties	Name=hvr_timekey IntegrateExpression={hvr_integ_key} Extra Time Key	This is needed if the target location is a file or Kafka location to replicate delete operations.

4. Apply changes to the source database using the following SQL statements:

```
insert into tabx (a,b,c) -- Note: column c contains binary/hex data
  values (1, 'hello', '746f206265206f72206e6f7420746f2062652c007468617420697320746865');
insert into tabx (a,b,c)
  values (2, '<world>', '71756573746966e');
insert into mytab (aa, bb) values (33, sysdate);
update tabx set c=null where a=1;
commit;
update mytab set aa=5555 where aa=33; -- Note: key update
delete from tabx; -- Note: deletes two rows
insert into user2.tabx (c1, c2) -- Note: different tables share same name
  values (77, 'seven');
commit;
```

The above SQL statements would be represented by the following XML output.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<hvr version="1.0">
  <table name="tabx">
    <row>
      <column name="hvr_op_val">1</column>
      <column name="hvr_timekey">5FFEF1E300000001</column>
      <column name="a">1</column>
      <column name="b">hello</column>
      <column name="c" format="hex">
        746f 2062 6520 6f72 206e 6f74 2074 6f20 7468 6174 2069 7320 7468 65
        # to be or not to be, .that is the
      </column>
    </row>
    <row>
      <column name="hvr_op_val">1</column>
      <column name="hvr_timekey">5FFEF1E300000002</column>
      <column name="a">2</column>
      <column name="b"><world></column>
      <column name="c" format="hex">
        7175 6573 7469 66e
        # question
      </column>
    </row>
  </table>
</hvr>
```



```

        </row>
    </table>
tag switches current table -->

    <table name="mytab">
        <row>
            <column name="hvr_op_val">1</column>
            <column name="hvr_timekey">5FFEF1E300000003</column>
            <column name="aa">33</column>
            <column name="bb">2012-09-17 17:32:27</column>
        </row>
    </table>
    <table name="tabx">
        <row>
            <column name="hvr_op_val">4</column>
        </row>
        <row>
            <column name="hvr_timekey">5FFEF1E300000004</column>
            <column name="a">1</column>
            <column name="b">hello</column>
        </row>
        <row>
            <column name="hvr_op_val">2</column>
            <column name="hvr_timekey">5FFEF1E300000005</column>
            <column name="a">1</column>
            <column name="b">hello</column>
            <column name="c" is_null="true"/>
        </row>
    </table>
    <table name="mytab">
        <row>
            <column name="hvr_op_val">3</column>
            <column name="hvr_timekey">5FFEF1E300000006</column>
            <column name="aa">33</column>
        </row>
        <row>
            <column name="hvr_op_val">2</column>
            <column name="hvr_timekey">5FFEF1E300000007</column>
            <column name="aa">5555</column>
        </row>
    </table>
    <table name="tabx">
        <row>
            <column name="hvr_op_val">0</column>
            <column name="hvr_timekey">5FFEF1E300000008</column>
            <column name="a">1</column>
            <column name="b">hello</column>
            <column name="c" is_null="true"/>
        </row>
        <row>
            <column name="hvr_op_val">0</column>
            <column name="hvr_timekey">5FFEF1E300000009</column>
            <column name="a">2</column>
            <column name="b"><world></column>
            <column name="c" format="hex">
                7175 6573 7469 6f6e
            </column>
        </row>
    </table>
    <table name="tabx1">
        <row>
            <column name="hvr_op">1</column>
            <column name="hvr_timekey">5FFEF1E300000010</column>
            <column name="c1">77</column>
        </row>
    </table>

```

<-- Note: Table

<-- Note: HVRs own date format -->

<-- Note: Hvr_op=4 means non-key update

before -->

Note: No table tag because no table switch -->

<-- Note: Hvr_op=2 means update-after -->

<-- Note: Nulls shown in this way -->

<-- Note: Hvr_op=4 means key update-before

-->

<-- Note: Hvr_op=0 means delete -->

rows -->

question

<-- Note: Name used here is channels name for table. -->

<-- Note: This may differ from actual table 'base name' -->

```

                <column name="c2">seven</column>
            </row>
        </table>
</hvr>
                <-- Note: No more changes in replication cycle -->

```

Capture and Integrate Converters

Environment

A command specified with parameter [CaptureConverter](#) or [IntegrateConverter](#) should read from its **stdin** and write the converted bytes to **stdout**. If the command encounters a problem, it should write an error to **stderr** and return with exit code **1**, which will cause the replication jobs to fail. The transform command is called with multiple arguments, which should be defined with [CaptureConverterArguments](#) or [IntegrateConverterArguments](#).

A converter command inherits the environment from its parent process. On the hub, the parent of the parent process is the HVR Scheduler. On a remote Unix machine, it is the **inetd** daemon. On a remote Windows machine it is the HVR Remote Listener service. Differences with the environment process are as follows:

- Environment variables **\$HVR_CHN_NAME** and **\$HVR_LOC_NAME** are set.
 - Environment variable **\$HVR_TRANSFORM_MODE** is set to either value **cap**, **integ**, **cmp**, **refr_read** or **refr_write**.
 - Environment variable **\$HVR_CONTEXTS** is defined with a comma-separated list of contexts defined with HVR Refresh or Compare (option **-Cctx**).
 - Environment variables **\$HVR_VAR_XXX** are defined for each context variable supplied to HVR Refresh or Compare (option **-Vxxx=va**).
 - For file locations variables **\$HVR_FILE_LOC** and **\$HVR_LOC_STATEDIR** are set to the file location's top and state directory respectively.
 - For an integrate converter for 'blob' file channel without table information and for all capture converters, environment variables **\$HVR_CAP_LOC**, **\$HVR_CAP_TSTAMP**, **\$HVR_CAP_FILENAME** and **\$HVR_CAP_SUBDIRS** are set with details about the current file.
 - Environment variable **\$HVR_FILE_PROPERTIES** contains a colon-separated *name=value* list of other file properties. This includes values set by 'named patterns' (see parameter [Pattern](#) in action [Capture](#)).
- If a channel contains tables: Environment variable **\$HVR_TBL_NAMES** is set to a colon-separated list of tables for which the job is replicating or refreshing (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 action [TableProperties](#) with parameter [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. Environment variables **\$HVR_TBL_KEYS** and **\$HVR_TBL_KEYS_BASE** are colon-separated lists of keys for each table specified in **\$HVR_TBL_NAMES** (e.g. **k1,k2:k:k3,k4**). The column list is specified in **\$HVR_COL_NAMES** and **\$HVR_COL_NAMES_BASE**. Any variable defined by action [Environment](#) is also set in the converter's environment.
- The current working directory is **HVR_TMP**, or **HVR_CONFIG/tmp** if this is not defined.
 - **stdin** is redirected to a socket (HVR writes the original file contents into this), whereas **stdout** and **stderr** are redirected to separate temporary files. HVR replaces the contents of the original file with the bytes that the converter writes to its **stdout**. Anything that the transform writes to its **stderr** is printed in the job's log file on the hub machine.

The output of a capture converter must conform the format implied by other parameters of this **FileFormat** action. Therefore if parameter **Csv** is not defined then the command should be XML.

Examples

A simple example is **FileFormat** defined with parameters [IntegrateConverter=perl](#) and [IntegrateConverterArguments="-e s/a/z/g"](#). This will replace all occurrences of letter **a** with **z**.

New Action: FileFormat ⓘ

hvr_demo * All Locations * All Tables

Compact
 Compress GZIP
 Encoding US-ASCII
 HeaderLine
 FieldSeparator
 LineSeparator
 QuoteCharacter
 EscapeCharacter
 FileTerminator

RowGroupThreshold
 ParquetVersion v1
 BeforeUpdateColumns
 BeforeUpdateColumnsWhenChanged
 ConvertNewlinesTo UNIX
 IntegrateConverter perl
 IntegrateConverterArguments -e s/a/z/g
 Context

Regular

Cancel Save

Directory **HVR_HOME/plugin_examples/transform** contains other examples of command transforms written in Perl. Converter **hvr_csv2xml.pl** maps CSV files (Comma Separated Values) to HVR's XML.

Converter **hvr_xml2csv.pl** maps HVR's XML back to CSV format. And **hvr_file2column.pl** maps the contents of a file into a HVR compatible XML file; the output is a single record/row.