AgentPlugin

Contents

Description

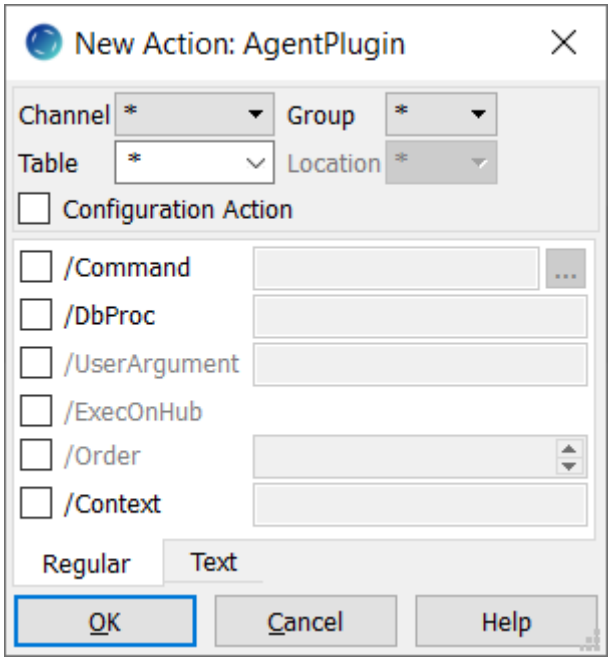
An agent plugin is a block of user–supplied logic which is executed by HVR during replication. An agent plugin can be an operating system command or a database procedure. Each time HVR executes an agent plugin it passes parameters to indicate what stage the job has reached (e.g. start of capture, end of integration etc.). If action **AgentPlugin** is defined on a specific table, then it affects the entire job including data from other tables for that location.

Since HVR 5.6.5/13, HVR will only execute binaries and scripts available inside **$HVR_HOME/lib/agent** or **$HVR_HOME/lib/transform**. So, it is recommended to save custom scripts/agent plugins in these directories.

HVR can also execute binaries and scripts available inside other directories if they are whitelisted. Directories can be whitelisted by defining the property **Allowed_Plugin_Paths** in file **$HVR_HOME/lib/hvraccess.conf**. For reference, the sample configuration file **hvraccess.conf_example** can be found in the same directory.

Parameters

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



Parameter	Argument	Description

/Command	*path*	Name of the agent plugin command. This can be a script or an executable. Scripts can be shell scripts on Unix and batch scripts on Windows or can be files beginning with a 'magic line' (shebang) containing the interpreter for the script e.g. **#!perl**. Argument *path* can be an absolute or a relative pathname. If a relative pathname is supplied the agents should be located in **$HVR_HOME/lib/agent** or in a directory specified with parameter **/Path**. **/Command** cannot be used with parameter **/DbProc**.
/DbProc	*dbproc*	Call database procedure *dbproc* during replication jobs. The database procedures are called in a new transaction; changes that do not commit themselves will be committed after agent plugin invocation by the HVR job. **/DbProc** cannot be used with parameters **/Command**, **/ExecOnHub**, and **/Path**.
/UserArgument	*userarg*	Pass extra argument *userarg* to each agent plugin execution.
/ExecOnHub		Execute agent plugin on hub machine instead of location's machine. **/ExecOnHub** cannot be used with parameter **/DbProc**.
/Order	*int*	Order of executing the agent plugin.
/Path	*dir*	Search directory *dir* for agent plugin. **/Path** cannot be used with parameter **/DbProc**. This parameter is not available since HVR version 5.6.5/13.
/Context	*context*	Action only applies if **Refresh**/**Compare** *context* matches

Agent Plugins in HVR Distribution

By default, HVR distribution contains the following agent plugins:

- Agent Plugin for BigQuery
- Agent Plugin for Cassandra
- Agent Plugin for Manifest Files
- Agent Plugin for MongoDB

Agent Plugin Arguments

If an agent plugin is defined, it is called several times at different points of the replication job. On execution, the first argument that is passed indicates the position in the job, for example **cap_begin** for when the agent plugin is called before capture.

Argument *mode* is either **cap_begin**, **cap_end**, **integ_begin**, **integ_end**, **refr_read_begin**, **refr_read_end**, **refr_write_begin** or **refr_write_end** depending on the position in the replication job where the agent plugin was called. Agent plugins are not called during **HVR Compare**.

Modes **cap_end** and **integ_end** are passed information about whether data was actually replicated.

Command agent plugins can use **$HVR_TBL_NAMES** or **$HVR_FILE_NAMES** and database procedure agent plugins can use parameter **hvr_changed_tables**. An exception if an integrate job is interrupted; the next time it runs it does not know anymore which tables were changed so it will set these variables to an empty string or **-1**.

Agent plugins specified with **/Command** are called as follows:

```
agent mode chn_name loc_name userarg
```

Database procedure agents (specified in **/DbProc**) are called as follows:

- In Ingres,

  ```
  execute procedure agent (hvr_agent_mode='mode', hvr_chn_name='chn',
  hvr_loc_name='locname', hvr_agent_arg='userarg', hvr_changed_tables=N);
  ```

- In Oracle,

  ```
  agent (hvr_agent_mode$=>'mode', hvr_chn_name$=>'chn', hvr_loc_name$=>'l
  ocname', hvr_agent_arg$=>'userarg', hvr_changed_tables$=N);
  ```

- In SQL Server,

  ```
  execute agent @hvr_agent_mode='mode', @hvr_chn_name='chn',
  @hvr_loc_name='locname', @hvr_agent_arg='userarg', @hvr_changed_tables=N
  ;
  ```

The parameter **hvr_changed_tables** specifies the number (*N*) of tables that were changed.

Agent Plugin Interpreter

If the agent plugin is a script, HVR will consider its shebang line to execute it with an interpreter. It is recommended that only the interpreter program name is specified here (for example, **#!perl** or **#!python**). It is not required to specify the absolute path in the shebang line. HVR will automatically determine the path for the specified interpreter using the environment variable **PATH**.

Agent Plugin Environment

An agent plugin inherits the environment of its parent process. On the hub, the parent of the agent plugin's 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 of the parent process are as follows:

- Environment variable **$HVR_TBL_NAMES** is set to a colon–separated list of tables for which the job is replicating (for example **HVR_TBL_NAMES=tbl1:tbl2:tbl3**). Also variable **$HVR_BASE_NAMES** is set to a colon–separated list of table 'base names', which are prefixed by a schema name if **TableProperties /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 **$HVR_COL_NAMES_BASE**.
- Environment variable **$HVR_CONTEXTS** is defined with a comma–separated list of contexts defined with HVR Refresh or Compare (option **–C***ctx*).
- Environment variables **$HVR_VAR_***XXX* are defined for each context variable supplied to HVR Refresh or Compare (option **–V***xxx*=*val*).
- For database locations, environment variable **$HVR_LOC_DB_NAME**, **$HVR_LOC_DB_USER** (unless no value is necessary).

- For Oracle locations, the environment variables **$HVR_LOC_DB_USER**, **$ORACLE_HOME** and **$ORACLE_SID** are set and **$ORACLE_HOME/bin** is added to the path.
- For Ingres locations the environment variable **$II_SYSTEM** is set and **$II_SYSTEM/ingres/bin** is added to the path.
- For SQL Server locations, the environment variables **$HVR_LOC_DB_SERVER**, **$HVR_LOC_DB_NAME**, **$HVR_LOC_DB_USER** and **$HVR_LOC_DB_PWD** are set (unless no value is necessary).
- For file locations variables **$HVR_FILE_LOC** and **$HVR_LOC_STATEDIR** are set to the file location's top and state directory respectively. For modes **cap_end** and **integ_end** variable **$HVR_FILE_NAMES** is set to a colon–separated list of replicated files, unless this information is not available because of recovery. For mode **integ_end**, the following environment variables are also set: **$HVR_FILE_NROWS** containing colon-separated list of number of rows per file for each file specified in **$HVR_FILE_NAMES** (for example **HVR_FILE_NROWS=1005:1053:1033**); **$HVR_TBL_NROWS** containing colon-separated list of number of rows per table for each table specified in **$HVR_TBL_NAMES**; **$HVR_TBL_CAP_TSTAMP** containing colon-separated list of first row's capture timestamp for each table specified in **$HVR_TBL_NAMES**; **$HVR_TBL_OPS** containing colon-separated list of comma-separated *hvr_op*=*count* pairs per table for each table specified in **$HVR_TBL_NAMES** (for example **HVR_TBL_OPS=1=50,2=52:1=75,2=26:1=256**). If the number of files or tables replicated are extremely large then these values are abbreviated and suffixed with "**...**". If the values are abbreviated, refer to **$HVR_LONG_ENVIRONMENT** for the actual values.
- Environment variables with too long values for operating system are abbreviated and suffixed with "**...**". If the values are abbreviated, HVR creates a temporary file containing original values of these environment variables. The format for this temporary file is a JSON map consisting of key value pairs and the absolute path of this file is set in **$HVR_LONG_ENVIRONMENT**.
- Any variable defined by action **Environment** is also set in the agent plugin's environment.
- The current working directory for local file locations (not FTP, SFTP, SharePoint/WebDAV, HDFS or S3) is the top directory of the file location. For other locations (e.g. database locations) it is **$HVR_TMP**, or **$HVR_CONFIG/tmp** if this is not defined.
- **stdin** is closed and **stdout** and **stderr** are redirected (via network pipes) to the job's logfiles.

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

Examples

This section lists few examples of agent plugin scripts:

Example 1: An agent plugin script (in Perl), which prints "hello world"

```
#!perl
# Exit codes: 0=success, 1=error, 2=ignore_mode
if($ARGV[0] eq "cap_begin") \{
    print "Hello World\n";
    exit 0;
\}
else \{
    exit 2;
\}
```

Example 2: An agent plugin script (in Perl), which prints out arguments and environment at the end of every integrate cycle

```
#!perl
require 5;
if ($ARGV[0] eq "integ_end")
\{
   print "DEMO INTEGRATE END AGENT (";
   foreach $arg (@ARGV) \{
       print "$arg ";
   \}
   print ")\n";
   # print current working directory
   use Cwd;
   printf("cwd=%s\n", cwd());
   # print (part of the) environment
   printf("HVR_FILE_NAMES=$ENV\{HVR_FILE_NAMES\}\n");
   printf("HVR_FILE_LOC=$ENV\{HVR_FILE_LOC\}\n");
   printf("HVR_LOC_STATEDIR=$ENV\{HVR_LOC_STATEDIR\}\n");
   printf("HVR_TBL_NAMES=$ENV\{HVR_TBL_NAMES\}\n");
   printf("HVR_BASE_NAMES=$ENV\{HVR_BASE_NAMES\}\n");
   printf("HVR_TBL_KEYS=$ENV\{HVR_TBL_KEYS\}\n");
   printf("HVR_TBL_KEYS_BASE=$ENV\{HVR_TBL_KEYS_BASE\}\n");
   printf("HVR_COL_NAMES=$ENV\{HVR_COL_NAMES\}\n");
   printf("HVR_COL_NAMES_BASE=$ENV\{HVR_COL_NAMES_BASE\}\n");
   printf("PATH=$ENV\{PATH\}\n");
   exit 0; # Success
\}
else
\{
   exit 2; # Ignore mode
\}
```

Example 3: An agent plugin script (in Python), which utilizes $HVR_LONG_ENVIRONMENT to print environment variables at the end of every integrate cycle

```
#!python

import os
import sys
import json

if __name__ == "__main__":
    if sys.argv[1] == 'integ_end':
        if 'HVR_LONG_ENVIRONMENT' in os.environ:
            with open(os.environ['HVR_LONG_ENVIRONMENT'], 'r') as f:
                long_environment= json.loads(f.read())
        else:
            long_environment= \{\}  # empty dict

        # print (part of the) environment
        if 'HVR_FILE_NAMES' in long_environment:
            print 'HVR_FILE_NAMES=\{0\}'.format(long_environment
['HVR_FILE_NAMES'])
        elif 'HVR_FILE_NAMES' in os.environ:
            print 'HVR_FILE_NAMES=\{0\}'.format(os.environ
['HVR_FILE_NAMES'])
        else:
            print 'HVR_FILE_NAMES=<not set>'
        if 'HVR_TBL_NAMES' in long_environment:
            print 'HVR_TBL_NAMES=\{0\}'.format(long_environment
['HVR_TBL_NAMES'])
        elif 'HVR_TBL_NAMES' in os.environ:
            print 'HVR_TBL_NAMES=\{0\}'.format(os.environ['HVR_TBL_NAMES'])
        else:
            print 'HVR_TBL_NAMES=<not set>'

        if 'HVR_BASE_NAMES' in long_environment:
            print 'HVR_BASE_NAMES=\{0\}'.format(long_environment
['HVR_BASE_NAMES'])
        elif 'HVR_BASE_NAMES' in os.environ:
            print 'HVR_BASE_NAMES=\{0\}'.format(os.environ
['HVR_BASE_NAMES'])
        else:
            print 'HVR_BASE_NAMES=<not set>'

        sys.exit(0)  # Success
    else:
        sys.exit(2)  # Ignore mode
```

Example 4: A database procedure agent plugin that populates table order_line after a refresh.

```
create procedure [dbo].[refr_agent] (
   @hvr_agent_mode varchar(20),
   @hvr_chn_name varchar(20),
   @hvr_loc_name varchar(20),
   @hvr_agent_arg varchar(20),
   @hvr_changed_tables numeric
) as
begin
   if @hvr_agent_mode = 'refr_write_begin'
   begin
       begin try
           delete from order_line
       end try
       begin catch
          -- ignore errors; nothing to delete
       end catch
   end
   else if @hvr_agent_mode = 'refr_write_end'
   begin
       insert into order_line
           SELECT i.item_id,
             i.item_no,
             i.description,
             i.attribute,
             i.item_type,
             p.id,
             p.date_set,
             p.price
             FROM items i, price p
   end
end
```