Writing an AgentPlugin (connector)

If the AgentPlugin action is configured for a target group or location, HVR calls the program defined in the action four times:

* Before the refresh starts
* After the refresh completes
* Before the integrate cycle starts
* After an integrate cycle completes

The target platform should be providing either an API or a SQL interface. In all the AgentPlugins we’ve written so far, where the target is not a messaging service, there is some sort of data load API or SQL, and then some sort of MERGE-type SQL. The data load interface will be used to implement Refresh and the MERGE to implement CDC. If there is not a MERGE SQL, the connector may have to explicitly issue INSERT/UPDATE and DELETE statements.

Once the API has been identified, determine whether there is a Python package for that API. If the API is SQL then the Python package is pyodbc.

Depending on the target platform, and customer requirements, the connector may be expected to support a replication copy of the source table in the target, a replication copy with SoftDelete, and/or a Timekey target.

Support for a replication copy on the target will typically require an Extra column with IntegrateExpression set to {hvr\_op} as well as an Extra column with /SoftDelete set. These columns can be discarded by the connector when pushing the data into the target.

Support for MERGING the changes into the target table requires that the Integreate action has for following setting. The SORT\_COALESCE tells integrate to coalesce the changes before writing them to file so that there is only one change per row

/ReorderRows=SORT\_COALESCE

The target location will typically be a directory on the integrate machine or a cloud file store. A FileFormat action must be defined. The template scripts provided expect CSV files.

The file format depends upon what file formats are support by the target platorm API, and the customer’s requirements.

The connector typically removes the files written by Refresh or Integrate after it has merged the data into the target. If the target location is a cloud file store then this step requires a Python package for that cloud store.

The first step might be to create a channel with the ‘hvrskeletonagent.py’ and the following actions and initialize the channel, run Refresh and move data through Integrate.

skeleton SRC \* \* Capture

skeleton TRG \* \* AgentPlugin /Command=hvrskeletonagent.py

skeleton TRG \* \* FileFormat /Csv

skeleton TRG \* \* Integrate

The ‘hvrskeletonagent.py’ connector has tracing enabled by default. When run with refresh you should see somehting like:

skeleton-refr-orjo-csv[agent]: \*\*\*\*\*\*\*\*\* refr\_write\_begin \*\*\*\*\*\*\*\*\*

skeleton-refr-orjo-csv[agent]: ============================================

skeleton-refr-orjo-csv[agent]: HVR\_COL\_NAMES\_BASE = kc,c1,c2,c3

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_KEYS = kc

skeleton-refr-orjo-csv[agent]: HVR\_CONFIG = /usr/hvr/hvr\_config

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_KEYS\_BASE = kc

skeleton-refr-orjo-csv[agent]: HVR\_BASE\_NAMES = kc3col

skeleton-refr-orjo-csv[agent]: HVR\_TMP = /tmp

skeleton-refr-orjo-csv[agent]: HVR\_LOC\_STATEDIR = /usr/hvr/files/csv/\_hvr\_state

skeleton-refr-orjo-csv[agent]: HVR\_COL\_NAMES = kc,c1,c2,c3

skeleton-refr-orjo-csv[agent]: HVR\_HOME = /usr/hvr/hvr\_home

skeleton-refr-orjo-csv[agent]: HVR\_AGENT\_BEGIN\_TSTAMP = 1626291332

skeleton-refr-orjo-csv[agent]: HVR\_FILE\_LOC = /usr/hvr/files/csv

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_NAMES = kc3col

skeleton-refr-orjo-csv[agent]: ============================================

Refreshed 'kc3col' from location 'orjo' to location 'csv' (4005 rows 71 wide compression=99.6%). This bulk refresh took 0.014 seconds.

skeleton-refr-orjo-csv[agent]: \*\*\*\*\*\*\*\*\* refr\_write\_end \*\*\*\*\*\*\*\*\*

skeleton-refr-orjo-csv[agent]: ============================================

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_KEYS = kc

skeleton-refr-orjo-csv[agent]: HVR\_TMP = /tmp

skeleton-refr-orjo-csv[agent]: HVR\_FILE\_NROWS = 4005

skeleton-refr-orjo-csv[agent]: HVR\_FILE\_LOC = /usr/hvr/files/csv

skeleton-refr-orjo-csv[agent]: HVR\_COL\_NAMES\_BASE = kc,c1,c2,c3

skeleton-refr-orjo-csv[agent]: HVR\_BASE\_NAMES = kc3col

skeleton-refr-orjo-csv[agent]: HVR\_FILE\_NAMES = 20210714194354955-kc3col.csv

skeleton-refr-orjo-csv[agent]: HVR\_COL\_NAMES = kc,c1,c2,c3

skeleton-refr-orjo-csv[agent]: HVR\_AGENT\_BEGIN\_TSTAMP = 1626291332

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_NAMES = kc3col

skeleton-refr-orjo-csv[agent]: HVR\_AGENT\_END\_TSTAMP = 1626291333

skeleton-refr-orjo-csv[agent]: HVR\_LOC\_STATEDIR = /usr/hvr/files/csv/\_hvr\_state

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_NROWS = 4005

skeleton-refr-orjo-csv[agent]: HVR\_CONFIG = /usr/hvr/hvr\_config

skeleton-refr-orjo-csv[agent]: HVR\_TBL\_KEYS\_BASE = kc

skeleton-refr-orjo-csv[agent]: HVR\_HOME = /usr/hvr/hvr\_home

skeleton-refr-orjo-csv[agent]: ============================================

Make sure to run these tests with multiple tables in multiple transactions so that you can see the way that these environment variables are layed out.

The hvrdefaultagent.py script is a very complete starting point for a connector where:

* The API Is SQL
* The integrate location is a cloud file store
* The target table is a replication copy

As you build the connector, keep the following in mind:

* The table\_file\_name\_map method works as is written only if the FileFormat is CSV and there is no Integrate /RenameExpression.
* HVR expects an exit code of ‘0’ from the connector. If the connector fails, HVR will call it again with the same environment variables. Be sure that the connector handles this restart no matter where in the processing it might abnormally exit.