

Hot-Swapping Reference Data for Address Verification (On- Premises)

Abstract

Hot-swapping in Informatica Address Verification (On-Premises) refers to the replacement of a set of reference data files without interrupting an address job or restarting the engine. This article provides instructions on how to perform hot-swapping with the Address Verification C API and by updating the configuration file.

Supported Versions

- Informatica Address Verification (On-Premises) 6.1 and later versions

Table of Contents

What is File Set Hot-Swapping?	2
Understanding the Reference Data Directory Structure.	2
How the Engine Chooses the File Set.	3
Enabling File Set Hot-Swapping.	3
Triggering a File Set Hot Swap.	3
Hot-Swap by API.	4
Hot-Swap by File Configuration.	4
During and After the Hot Swap.	4
Bulk Jobs and Hot-Swapping.	5

What is File Set Hot-Swapping?

Hot-swapping allows you to swap the complete set of reference data files that Address Verification (On-Premises) uses for another data set while the engine is active and running.

Hot-swapping is disabled by default. You enable hot-swapping in the *IDVEConfig.json* file before you initialize the engine.

Use hot-swapping to upgrade to a newer set of reference data files.

Understanding the Reference Data Directory Structure

The reference data files currently loaded by the engine are locked on the file system level and cannot be deleted or moved while the engine is running. Therefore, both the current files and the newer files must be present at the same time for the engine to make a transition from one to the other.

To achieve this, the engine works with two directories on disk, called *FileSetA* and *FileSetB*. The directories must be direct sub-directories in the directory that the *IDVEConfig.json* file specifies under `</Core/System/FileSetsDirectoryPath>`. Address Verification will only accept and look for reference data in the *FileSetA* and *FileSetB* directories at the specified location.

Place the reference data files directly into *FileSetA* or *FileSetB*.

Note: It is valid to have a single directory (either *FileSetA* or *FileSetB*) present on disk, which is typically the case if you do not intend to use hot-swapping.

How the Engine Chooses the File Set

Upon initialization, Address Verification scans the parent file set directory that *IDVEConfig.json* specifies and determines the file set directory to use in the following manner:

- If only *FileSetA* or *FileSetB* exists, it becomes the active file set.
- If both *FileSetA* and *FileSetB* exist, the directory that contains data files becomes the active file set.
If both directories contain data, or if neither directory contains data, initialization proceeds and Address Verification treats *FileSetA* as the active file set.
- If neither *FileSetA* or *FileSetB* exist, the initialization will abort.

Address Verification then creates a JSON file called *FileSetsInfo.json* within the parent directory on the same level as *FileSetA* and *FileSetB*. *FileSetsInfo.json* identifies the currently active file set. It is therefore necessary for Address Verification to have write access to the parent file set directory. If Address Verification cannot create or write to the JSON file, initialization will abort.

The JSON file contents may look like this:

```
{ "FileSetAState": "InUse", "FileSetBState": "Unused" }
```

In this example, *FileSetA* is currently active. *FileSetB* is currently unused.

The *FileSetsInfo.json* file persists on disk after Address Verification is deinitialized. If Address Verification finds this file during initialization it will base the decision on which file set to use on the file contents. If the JSON file has valid contents, this supersedes the rules outlined above. However, if the JSON file instructs Address Verification to use a file set for which no sub-directory is present, initialization will abort. Similarly, if the JSON file does not have the right format or has invalid content, initialization will abort.

Enabling File Set Hot-Swapping

By default, hot-swapping of file sets is disabled, and you cannot switch from one file set to another without deinitializing Address Verification.

To enable hot-swapping, set the `</Core/System/FileSetsHotSwappingEnabled>` value in *IDVEConfig.json* to true.

With hot-swapping enabled, you can switch from *FileSetA* to *FileSetB* or vice versa at any time while Address Verification is running. Such a switch always affects all of the data files in the directories. For example, when switching from *FileSetA* to *FileSetB*, all data files in *FileSetA* will be unloaded and replaced by all data files in *FileSetB*. It is not possible to switch a subset of the data files. If you want to keep certain data files available after the switch, the files must be present in both directories.

To enable hot-swapping, Address Verification reserves memory for twice the amount of function servers than specified in the configuration, excluding memory assigned for hot standby servers. Update the maximum memory settings accordingly, or the available memory for preloading will be reduced. Initialization might fail completely if sufficient memory is not available.

Triggering a File Set Hot Swap

To perform a hot swap, identify the file set that is not currently in use as ready to use. You can do this either via API calls to Address Verification or by updating the *FileSetsInfo.json* file.

A hot-swap procedure must fully finish before another one can be triggered.

Hot-Swap by API

Use the API method for hot-swap integration into a host application.

The Address Verification C API offers the following function to trigger a hot swap:

```
/// Trigger a file set hot swap
IDVE_EXPORTCALL1 IDVE_StatusCode IDVE_EXPORTCALL2 IDVE_SwapFileSet (
    const IDVE_Bool kbFileSetA , ///< [in] A
    bool value, if != 0 then file set A is ready to use, if 0 then file
    set B is ready to use
    const IDVE_Bool kbWait , ///< [in] A bool value, if != 0
    then the function waits for completion of the hot swap operation,
    if 0 then it returns immediatly
    char * const kpsExtStatusMsg ///< [out] Pointer to buffer of size
    IDVE_EXT_STATUS_MSG_BUFFER_SIZE for an optional extended status
    msg, may be NULL
);
```

The first argument `kbFileSetA` is a flag that identifies the file set to declare as ready to use. A value of `IDVE_TRUE` corresponds to *FileSetA* while `IDVE_FALSE` corresponds to *FileSetB*. The call fails if you trigger a hot swap to activate a file set that is already in use.

The second argument `kbWait` determines if the verification engine will block a call to the engine and wait for the completion of a hot swap. A value of `IDVE_TRUE` will cause the call to wait until the hot swap has been performed and all new data files are available. A value of `IDVE_FALSE` will perform the hot swap in a background thread while the calling thread returns immediately.

The third argument `kpsExtStatusMsg` enables an extended status message in the same manner as other C API calls.

Note: When you call `IDVE_SwapFileSet` synchronously, the status code and status message will contain any error that occurred during the attempt to swap. When you perform an asynchronous call, error information will only appear in Address Verification's initialization log.

Hot-Swap by File Configuration

To trigger a hot swap without interacting with the software, update the *FileSetsInfo.json* file.

The *FileSetsInfo.json* file may appear as follows after a regular initialization:

```
{ "FileSetAState": "InUse", "FileSetBState": "Unused" }
```

To trigger a hot swap, update the JSON file to set `FileSetBState` to `ReadyToUse`:

```
{ "FileSetAState": "InUse", "FileSetBState": "ReadyToUse" }
```

After you save the file, Address Verification will re-parse the file. If the file contents are valid, Address Verification starts a hot-swap procedure in a background thread in the same manner as an asynchronous `IDVE_SwapFileSet` call. Any error information will appear in the initialization log.

If the new contents of the JSON file are not valid or the requested hot swap cannot be performed, the JSON file is restored to its previous state.

During and After the Hot Swap

Address Verification performs the following steps during a hot swap:

Note: The following steps continue the example of swapping from *FileSetA* to *FileSetB*.

1. Address Verification updates the contents of *FileSetsInfo.json* to reflect that the new file set is being prepared:

```
{ "FileSetAState": "InUse", "FileSetBState": "PreparingForUse" }
```

2. Address Verification starts a new batch of function servers and preloads the reference data files in the *FileSetB* directory for the servers. During this time, the function servers that use *FileSetA* remain fully available for all jobs.
3. When all new function servers have started successfully, the previous function servers are shut down.
4. The contents of *FileSetsInfo.json* are updated to reflect that *FileSetB* is now in use and *FileSetA* has been unloaded:

```
{ "FileSetAState": "Unused", "FileSetBState": "InUse" }
```

As starting a new batch of function servers is akin to a regular initialization procedure, Address Verification writes messages logged during this sub-initialization to the initialization log. However, any error that prevents the hot swap from starting is written to the Address Verification error log.

After a successful hot swap, the reference data files that the operation unloaded are unlocked on the file system and can be moved or deleted again.

Bulk Jobs and Hot-Swapping

Address Verification supports bulk processing, where a single process call for a job instance can contain up to 1000 inputs.

It might happen that a file set hot swap replaces the data files in the middle of a bulk job, so that some inputs are processed with *FileSetA* while others are processed with *FileSetB*.

In many cases this does not pose an issue. However, there may be cases where you must process the complete set of inputs with exactly the same reference data, such as in certified runs of address verification.

To guarantee a consistent file set for a job, you can create the job with the following POST command:

```
IDVE_Post(" ... ",
"AV/v1/Jobs/createLockedFileSet",
...)
```

Any job created with this command rather than the regular `/create` command will be bound to the file set that was active when the command ran. A hot swap can still take place, and regular jobs will use the new data, but any `createLockedFileSet` job will use the old data files until the job is deleted.

Multiple jobs can exist at the same time. Address Verification releases the previously-used reference data files after you delete the `createLockedFileSet` jobs. The call to delete the last of the jobs will cause the previous batch of function servers to shut down and the data files to unlock.

Both batches of function servers (those using *FileSetA* and those using *FileSetB*) must stay active while a `createLockedFileSet` job runs, and there is a corresponding impact on resources. Delete any `createLockedFileSet` job as soon as possible.

While a job remains locked to a previous file set, the contents of *FileSetsInfo.json* will reflect that state accordingly:

```
{ "FileSetAState": "StillInUse", "FileSetBState": "InUse" }
```

The flag switches to `Unused` after the last locked job is deleted.

Authors

Torben Lutz - Software Engineer

Ester Nunes McGarry - Technical Writing Intern