

Oracle Fusion Cloud Human Resources

**Create and maintain data with HCM
Data Loader**



Oracle Fusion Cloud Human Resources
Create and maintain data with HCM Data Loader

G62147-01

Copyright © 2026, Oracle and/or its affiliates.

Author: Priyadarshini Gopinath

Contents

Get Help	i
3 Create and maintain data with hcm data loader	3
Get started with HCM Data Loader files	3
Build HCM Data Loader files	4
Model relationships and record changes	8
Control deletion and file processing	9
Load and validate data	12

Get Help

In addition to the resources available in Oracle Help Center, you can explore training, connect with the Oracle community, and share your feedback.

Get Training

Increase your knowledge of Oracle Cloud by taking courses at [Oracle University](#).

Join Our Community

Use [Cloud Customer Connect](#) to get information from industry experts at Oracle and in the partner community. You can join forums to connect with other customers, post questions, suggest [ideas](#) for product enhancements, and watch events.

Share Your Feedback

We welcome your feedback about Oracle Applications user assistance. If you need clarification, find an error, or just want to tell us what you found helpful, we'd like to hear from you.

You can email your feedback to oracle_fusion_applications_help_ww_grp@oracle.com.

Thanks for helping us improve our user assistance!

3 Create and maintain data with hcm data loader

Get started with HCM Data Loader files

HCM Data Loader (HDL) files provide a flexible way to create and maintain application data in bulk. Understanding how HDL files are structured and how their instructions are interpreted provides the foundation for building files that correctly identify and process business-object data.

About this playbook

Use this playbook to create HCM Data Loader files that:

- Create new application data;
- Update existing records;
- Add child records to existing business objects;
- Reference parent and foreign objects correctly;
- Delete individual records or complete logical objects;
- Provide reconciliation information for monitoring and reporting.

This playbook also explains the core HDL file concepts needed to perform those tasks, including file structure, METADATA and data-line instructions, source and user keys, source-system owners, parent and foreign references, partial updates, null values, reconciliation values, and ordinary record deletion. You don't need separate playbooks for these concepts.

HDL uses flexible, pipe-delimited `.dat` files. Include only the business-object components and attributes required for the task, rather than supplying a complete representation of every object.

Before creating a file, use HDL-01 — Understand HCM Data Loader business objects to identify the correct file name, component discriminators, attributes, supported actions, and available keys.

Understand the basic HDL file structure

An HDL load consists of one or more `.dat` files compressed into a `.zip` file.

Each `.dat` file represents a single business-object hierarchy. For example:

- **Worker.dat**
- **Job.dat**
- **Grade.dat**
- **ElementEntry.dat**

All components belonging to that hierarchy are supplied in the same `.dat` file. A **Job.dat** file can therefore contain Job, Job Valid Grade, Job Evaluation, and other Job child components.

A ZIP file can contain one or more business-object `.dat` files.

You can package related business objects together when that simplifies dependency processing. HDL imports the files and loads business objects in dependency order so referenced data can be available before the data that references it.

Use one business object per ZIP when testing a new file shape, so that import or load issues are easier to isolate.

During large data migrations, use one business object per ZIP when loading a very large file for that object. This keeps the data set focused and simplifies monitoring, error handling, and recovery.

For smaller or logically related loads, multiple business-object files can be included in the same ZIP where appropriate.

File construction rules

Follow these construction rules when creating and packaging HDL data files to ensure that the files can be recognized and processed correctly.

- Use the exact business-object file name identified in **View Business Objects**.
- Business-object file names are case-sensitive.
- Save files using UTF-8.
- Don't place `.dat` files inside folders in the ZIP.
- The default field delimiter is the pipe character (`|`).

Understand the HDL file instructions

HDL uses line instructions to define the structure of the data and specify how records are processed. The instruction you use determines whether a line defines metadata, creates or updates data, deletes data, changes processing behavior, or provides a comment.

Instruction	Purpose
METADATA	Defines the component and attributes supplied by subsequent data lines
MERGE	Creates a record or updates an existing record
DELETE	Removes an existing record
SET	Changes default processing for the file
COMMENT	Adds a comment

METADATA, MERGE, and DELETE lines also include a file discriminator, which identifies the component in the business-object hierarchy.

Build HCM Data Loader files

Building an HDL file involves defining the data to supply, specifying the required processing instruction, and uniquely identifying each record. Use the appropriate attributes and keys so that HDL can determine whether to create, update, or identify existing application data.

Define the attributes with METADATA

Every component that you supply in a file requires a METADATA line.

For example:

The values mean:

- **METADATA** — the instruction;
- **JobFamily** — the file discriminator;
- the remaining values — the component attributes and the order in which data will be supplied.

A corresponding data line must provide values in that same order:

You can define only one METADATA layout for each component within a file. Include only the attributes required for the scenario.

Use the **METADATA Attribute** values shown in **View Business Objects** rather than display labels.

Use MERGE for both creates and updates

HDL has one instruction for creating or updating application records:

You don't specify CREATE or UPDATE.

HDL identifies the record using the keys supplied on the line and then determines whether it must create a new record or update an existing one.

For example:

If the identified Job doesn't exist and the Job component supports Create, HDL creates it.

If it already exists and the component supports Update, HDL updates it.

This is distinct from the **Supported Actions** information described in Understand HCM Data Loader business objects. **Supported Actions** tells you whether the component permits Create, Update, and Delete. MERGE is the file instruction used for both Create and Update.

For updates supply only what changed

Keep update files focused on the values that need to change. For an existing record, provide:

- Attributes to identify the record
- Supply the attributes whose values you want to update

Attributes omitted from the file retain their existing values. Avoid including unchanged attributes unnecessarily.

Use NULL to Remove an Existing Value

A blank field doesn't explicitly set an existing attribute to null.

To clear an existing value, supply:

Don't confuse:

- attribute omitted from METADATA — leave the current value unchanged;

- blank value — doesn't mean “clear this value”;
- `#NULL` — explicitly set the attribute to null.

Choose how to identify each record

Every MERGE or DELETE record must be identified uniquely. Choose a supported key type based on whether you're creating a record, updating an existing record, and how the record is identified by the source or application.

HDL supports four key types:

Key type	Create	Update	Typical use
Source key	Yes	Yes	Recommended for integrations and migrations
User key	Yes	Usually	Business-visible natural key
GUID	No	Yes	Existing integration-enabled records
Surrogate ID	No	Yes	Existing records where internal ID is available

When you supply more than one supported key, HDL resolves them in this order:

- GUID
- source key
- surrogate ID
- user key

HDL doesn't cross-validate multiple supplied keys. If a GUID identifies one record and a supplied source key identifies another, the GUID wins and HDL doesn't raise an inconsistency error merely because the keys disagree.

For that reason, avoid supplying competing key types unless there's a specific need.

Prefer source keys for integrations

Source keys provide a stable way to identify records created or maintained by an external source system. Each source key combines the owner of the source data with an identifier that's unique within that source.

A source key consists of:

- `SourceSystemOwner`
- `SourceSystemId`

For example:

Oracle recommends source keys where possible, particularly for integrations where another system remains the system of record. Source keys are stable even when user-facing key values change.

Before using your own `SourceSystemOwner` value, add it to the `HRC_SOURCE_SYSTEM_OWNER` lookup.

User keys may still be required on create

Even when a source key identifies the record, some user-key attributes are still functionally required when creating the application record.

If you supply both source and user keys, the source key is the identifier HDL uses for matching.

Understand default source keys

If you create a record without supplying a source key, HDL generates a default source key:

- `SourceSystemOwner = FUSION`
- `SourceSystemId = the generated surrogate ID`

You can later use or replace this source key.

For planned integrations, however, don't rely on default source keys when you already have a stable source-system identifier. Supplying your own source key from the beginning normally makes later maintenance and cross-object references easier.

Update a source key when necessary

Don't change source-key values simply by changing `SourceSystemOwner` or `SourceSystemId` on the normal business-object MERGE line.

To replace an existing source key, use the Source Key business object and a **SourceKey.dat** file.

For example:

You can update both default source keys and source keys that you originally supplied for integration-enabled objects.

Use user keys when they're the appropriate identifier

A user key is made up of one or more business-visible values.

For example, Job may be identified by:

- Job Code; and
- Set Code.

All attributes that form the selected user key must be supplied when that key is being used to identify the record.

Be careful when user key values can change

Some user keys contain values such as names or codes that customers can change.

That makes them less suitable as long-term integration identifiers. Some objects can't reliably be updated using only a changed user key.

Where possible, use a source key for ongoing maintenance, particularly when:

- user-key values can change;
- values can be translated;
- historical data is involved;

- another system remains the system of record.

Model relationships and record changes

HDL files can represent complete business-object hierarchies as well as targeted changes to individual components. Correctly identify parent and related records so that HDL can assemble the supplied data into the intended logical objects and process changes without conflicting instructions.

Identify the parent of every child record

Child components are processed as part of their parent business-object hierarchy. Each child must therefore provide enough information to identify both the child record and the parent record to which it belongs.

A child component must:

- Identify itself uniquely and
- Identify its parent uniquely

For example, if JobGrade is a child of Job, the JobGrade line must identify which Job owns the valid-grade record.

You can use whichever parent-reference key type the component supports.

View Business Objects identifies:

- the **Parent Surrogate ID** attribute;
- supported integration keys;
- supported user-key combinations.

Reference related objects using supported foreign keys

A component can also reference another, independent business object.

For example, a Job Valid Grade needs to reference a Grade.

Where integration keys are supported, you can supply a source-system identifier through the foreign object's surrogate-ID attribute using an HDL hint.

For example:

Here:

- `scn_ic1` identifies the JobGrade record;
- `scn` identifies the parent Job by its `SourceSystemId`;
- `ic1` identifies the referenced Grade by its `SourceSystemId`.

If the referenced object uses a different `SourceSystemOwner`, also provide a `(SourceSystemOwner)` hint for that reference.

The referenced object must exist

An integration-key reference isn't usable until the referenced object has been created successfully.

If multiple related `.dat` files are supplied in the same ZIP, HDL loads business objects in dependency order. A reference will still fail if the referenced record itself failed to load.

Add or update child records without resupplying the whole object

You don't have to resupply an entire hierarchy to change one part of it.

For example, you can include the top-level component to identify the logical object and then supply only the child records that need to be created or changed.

This partial-object model is one of HDL's key strengths.

Avoid competing instructions for the same logical record

HDL groups related lines into logical objects; it doesn't simply process the file top-to-bottom as independent commands.

Don't rely on line order to resolve conflicting instructions.

In particular:

- don't use both MERGE and DELETE for the same logical record in one file;
- for a non-date-effective record, don't try to create and then separately update that same component in the same file as two processing stages.

You can, however, MERGE one component of a logical object while DELETEing another component where that combination is valid.

Control deletion and file processing

HDL provides instructions for deleting application data and controlling how individual files are processed. Before applying these controls, confirm that the business-object component supports the intended action and determine whether the load requires any processing overrides or special data-handling rules.

Delete an individual record

First use **View Business Objects** to confirm that the component supports the Delete action.

To delete a record, use the DELETE instruction and supply only the attributes required to identify it.

For example:

The DELETE line removes the specified Answer.

Including the parent with a MERGE line is recommended because it groups the records into the same logical object. Some object hierarchies require the top-level record when child records are changed or removed.

Remember:

Delete support is a component capability; DELETE is the HDL instruction used when that capability exists.

Delete a complete logical object

To remove a complete object, identify the top-level record with DELETE.

For example:

For many objects, deleting the top-level component also removes its children, so you don't need separate DELETE lines for every child.

This section covers ordinary HDL deletion using the DELETE instruction. Some business objects have additional deletion rules, prerequisites, or dedicated purge processes. Follow the object-specific playbook where one exists.

Find and use SET instructions

A SET instruction changes default HDL processing for the `.dat` file in which it appears.

SET instructions:

- are optional
- don't have a file discriminator
- can be repeated when more than one processing override is required
- must appear before the first METADATA line in the file

Find the available SET instructions

Review both enterprise-level configuration and business-object-specific parameters before using a SET instruction. The available sources provide different information about the processing options that can be overridden for a load.

There are two places to review the SET instructions relevant to a load.

Where	What you'll find
Configure HCM Data Loader	Common HDL processing parameters and their configured enterprise defaults
View Business Objects > Actions > View Parameters	Parameters that apply to the selected business object, including the default value and the SET instruction used to override it

Use **View Parameters** whenever it's available for the business object. This prevents you from relying on an instruction that doesn't apply to the object or overlooking object-specific processing options.

Understand defaults and file overrides

Many HDL processing behaviors have an enterprise default. Administrators can change supported defaults using **Configure HCM Data Loader**.

A SET instruction lets you override the applicable default for a particular load rather than changing the enterprise configuration. Oracle's current SET guidance explicitly distinguishes the configured enterprise default from the per-file SET override.

For example, if a processing option is normally disabled for the enterprise but one load requires it, use the corresponding SET instruction in that file rather than changing the default for every HDL load.

Don't copy SET instructions into every file routinely. Review what the instruction controls, its default, and whether the scenario genuinely requires an override.

Some SET instructions control specialist processing such as date-effective maintenance, postload processing, auditing, retention, or Oracle Search updates. Before using those instructions, follow the specialist guidance listed later in this playbook.

Follow the standard data formatting rules

HDL expects data values to use consistent formats unless the business-object attribute specifies otherwise. Apply these standard formats when preparing values for dates, numbers, lookups, and null values.

Data	Guideline
Date	YYYY/MM/DD
Date and time	YYYY/MM/DD HH24:MI:SS
Numeric values	Decimal notation only; don't include currency symbols or thousands separators
Lookup values	Prefer lookup codes over translated meanings
Nulling an existing value	#NULL

Lookup meanings can be language-dependent. Codes are normally safer for repeatable integrations.

For special characters, delimiters, embedded line breaks, and complex text, see [Handle special characters and complex text](#).

For translations, see [Load and maintain translated data](#).

Supply source references for reconciliation

Source references are not keys.

They are optional values held with HDL staging information to help you identify, filter, report on, and reconcile imported records.

You can define up to ten reference columns using `SourceRef001` through `SourceRef010`.

Use source references for values that help answer questions such as:

- Which source record produced this HDL object?
- Which assignment or transaction does this failure relate to?
- What source amount should be reconciled?
- Which batch or source file supplied this record?

The values aren't written to the target application record and aren't validated as business-object attributes.

Don't query HDL staging tables directly for application reconciliation.

Check whether your load needs specialist guidance

Before you submit the file, check whether the scenario has requirements that need more specific guidance than this general file-authoring playbook.

Requirement	See:
Date-effective corrections, updates, end dating, Replace/Retain behavior, or PURGE_FUTURE_CHANGES	Make date-effective changes with HCM Data Loader
Flexfield segments, images, or attachments	Load flexfield segments, images, and attachments with HCM Data Loader
Special characters, delimiters, embedded line breaks, or complex text	Handle special characters and complex text
Translated values	Load and maintain translated data
Automated postload processing or suppressing postprocesses for a file	Understand, configure, and control postload processing
Auditing HDL or HSDL processing	Configure auditing for HDL and HSDL
Data-set retention or extended staging retention	Maintain stage tables and data-set retention
Oracle Search index updates after bulk loading	Manage Oracle Search index updates after bulk loading

Load and validate data

After preparing and packaging the HDL files, submit the ZIP file for processing and monitor both the import and load stages. Once processing completes, verify the resulting application data and reconcile the outcome against the expected source data.

Import and load the file

Import the packaged ZIP file from the **Data Exchange** work area when performing an interactive load. Monitor the resulting data set until both import and load processing are complete.

For an interactive load:

- Navigate to **My Client Groups > Data Exchange**.
- Open **Import and Load Data**.
- Select **Import File**.
- Select or drag your ZIP file.
- Submit the load.
- Refresh the data set until import and load processing completes.

Monitor both stages separately:

- **Import Status** — whether the file has been parsed and imported into HDL staging.
- **Load Status** — whether the imported logical objects were written successfully to the application.

Also distinguish:

- **Total Lines** — physical data lines;
- **Total Objects** — logical business objects assembled from those lines;
- **Failed Objects** — logical objects that failed to load.

A complex Worker object, for example, can consist of many file lines but still represent one logical object.

For failed imports or loads, see Review, correct, and reload failed HCM Data Loader records.

Validate the application outcome

A successful HDL processing status confirms that HDL processed the submitted data successfully.

It doesn't always prove that the broader business process is complete.

After significant loads:

- validate representative records in the application;
- reconcile expected source counts or totals;
- use supplied source-reference information where appropriate;
- complete any required object-specific postload processes.

See:

- Validate and reconcile application data after a bulk load
- Understand, configure, and control postload processing

File Authoring Checklist

Before submitting a file, confirm:

- You're using the correct business-object `.dat` file name.
- Every component has exactly one appropriate METADATA definition.
- Every data line uses the correct discriminator.
- Each record is uniquely identified.
- Every child record identifies its parent.
- Foreign-object references use a supported key type.
- Source-system owner values are registered where source keys are used.
- You're using MERGE for both creates and updates.
- The object's **Supported Actions** permit the intended create, update, or delete.
- Only changed attributes are supplied for normal updates.

- **#NULL** is used where an existing value must be cleared.
- No logical record has conflicting MERGE and DELETE instructions.
- Any SET instructions appear before the first METADATA line.
- Dates, numbers, lookups, and translated values use the correct format.
- Source-reference values are included if they'll help with reconciliation.
- ZIP packaging matches the scenario: one business object per ZIP for new-file-shape testing or very large single-object migration loads; multiple related business-object files together where appropriate.