CHAPTER 08

Delphix (1): connecting, keys and profiling

What the Delphix masking engine is, connecting to databases and a VDB, rule sets and primary keys, and profiling to find sensitive data.

Delphix masking: what it is and how it is organised

You have now seen two other families of tools. Informatica TDM is a large suite that discovers, subsets, masks and generates data. GenRocket builds synthetic data from scratch. This part of the guide is about the third tool, **Delphix**, and more precisely its masking product, called **Delphix Continuous Compliance**. Its engine finds sensitive values in a database or a file and replaces them with realistic but fictitious ones, so that a copy of production can safely go to testers.

This section maps what the product is for, what its screens look like, which objects the engine keeps (an environment, a connector, a rule set, an inventory and so on), and in what order a masking project touches them. Tool-specific facts are from the Delphix Continuous Compliance documentation, version 2026.5 (documentation home). Delphix releases often, so screens may differ from the pictures here; treat them as a guide to layout.

8.1 Words to know first

These are the words you meet in the first five minutes. Each one gets a fuller treatment later in the chapter.

Continuous Compliance engine

The Delphix product that discovers sensitive data and replaces it with realistic fictitious data. It runs as a virtual appliance (a pre-built machine image), and people also call it the *masking engine* or the Compliance Engine.

Profiling

Scanning column names and, optionally, the data itself to decide which columns hold sensitive values such as names or Social Security numbers. Delphix documentation also calls this *sensitive data discovery*.

Masking

Replacing sensitive values with fictitious ones so the original cannot be recovered. It is one-way.

Tokenization

A reversible alternative: values are swapped for meaningless tokens, and the engine can later turn the tokens back into the originals (re-identification).

Application and environment

An *application* is a label for the system whose data you protect. An *environment* is the folder-like container, inside one application, that holds your connectors, rule sets and jobs.

Connector

The saved connection details (host, port, user and so on) that tell the engine how to reach one database or one set of files.

Rule set

The list of tables (or files) of one connector that you want to profile and mask.

Inventory

The detailed list of every column or field in a rule set, with its data classification and the masking algorithm chosen for it.

Domain

A named kind of sensitive data (for example "first name") paired with the default algorithm that should protect it.

Algorithm

The recipe that turns one real value into a masked one. A single recipe can be reused on many columns.

Job

A saved, runnable task: a profiling job, a masking job, a tokenization job or a re-identification job.

VDB

A virtual database: a lightweight, writable copy of a database that Delphix's virtualization product creates in minutes. The next chapter explains it from zero.

8.2 What Delphix masking is

8.2.1 The problem it answers

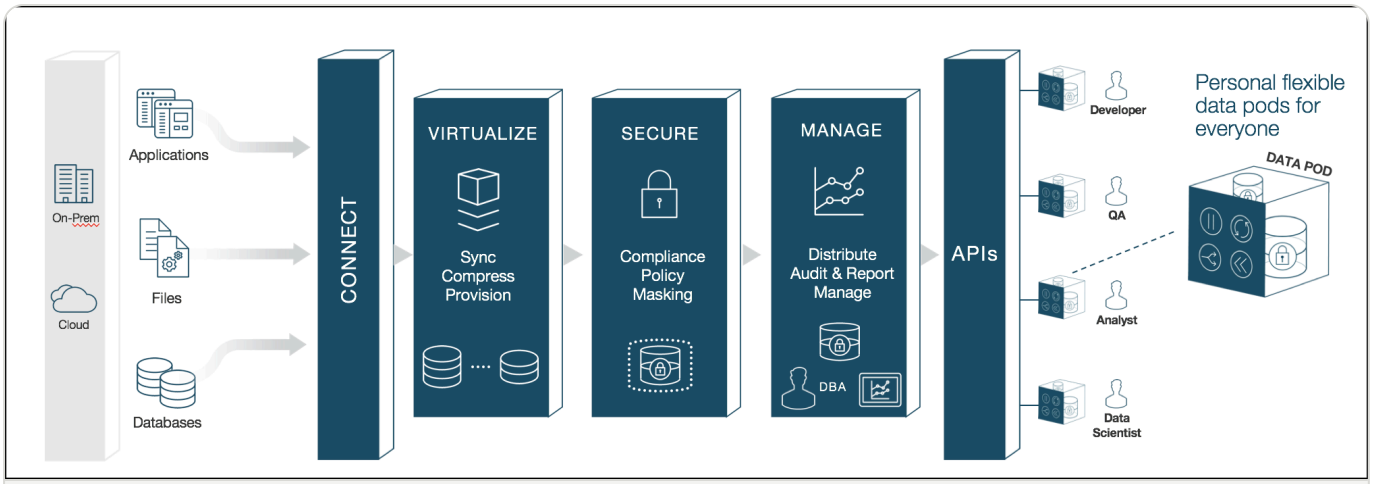
The story is covered in the chapter on sensitive data and regulations and the chapter on masking concepts. Production holds real customers. Testers and developers need data that behaves like production but must not leak real people. The Delphix introduction frames the same problem: breaches, pressure from regulators and the fact that teams also want to move faster. It says that encryption suits data in motion or at rest on disks, but is a poor fit for protecting non-production copies used for development, testing and reporting.

Its answer is masking: replace confidential values (the page names Social Security numbers, patient records and credit card numbers) with fictitious yet realistic data. It adds that masking, unlike encryption, protects data irreversibly in downstream environments, and that masking should stay *consistent* across different data sources while keeping *referential integrity*. Referential integrity means that a value which points to another table's row (a foreign key) still points to the right row after masking. That idea drives the rule set and keys chapter.

> **WHY THIS MATTERS FOR SHOPSPHERE**
>
> ShopSphere's testers need CUSTOMERS, ORDERS and ORDER_ITEMS that join correctly (every order belongs to a customer). If the tool blanked the names but broke the joins, test cases such as "show a customer's last five orders" would return nothing. Realistic and consistent is the whole point.

8.2.2 The platform around it

The masking capability is one part of the Delphix DevOps Data Platform. The introduction page describes four services: **virtualize** (compressed, lightweight, writable virtual copies), **identify and secure** (find sensitive values and mask them), **manage** (provision and audit copies from one point of control) and **self-service** (let developers and testers refresh, rewind, bookmark, branch and share their copies).



Official screenshot: The vendor's platform picture. Read it left to right: sources (applications, files, databases, on-premises or cloud) are connected, virtualized (sync, compress, provision), secured (compliance policy, masking), managed (distribute, audit and report) and then handed to people such as a developer, QA and an analyst. The masking engine of this section is the "Secure" box.

`Source page ↗`

Delphix sells more than one product, and the terminology page separates them. **Delphix Continuous Data** (the Data Engine) delivers data on demand and creates VDBs (next chapter). **Delphix Continuous Compliance** (the Compliance or masking engine) discovers sensitive data and replaces it, and is the subject of this whole part. **Data Control Tower** unifies governance across applications and clouds. The words "engine" and "Delphix" alone are ambiguous, so say which one you mean.

> **TWO NUMBERS THAT DISAGREE INSIDE THE DOCS**
>
> The introduction page says "eighteen algorithm frameworks" in one place and "seven out-of-the-box algorithm frameworks" in another. Both appear in the same page, so no count is quoted here. The algorithms chapter lists the frameworks that the algorithm pages actually describe.

8.3 What the engine does for you

Think of the engine as a three-step assembly line that you configure once and re-run on every refresh: *find* the sensitive columns, *decide* what replaces them, *apply* the replacement.

1. Connect
tell the engine where the data lives

→

2. Profile
find the sensitive columns

→

3. Assign
pick an algorithm per column

→

4. Mask or tokenize
run a job

8.3.1 Finding sensitive data: profiling

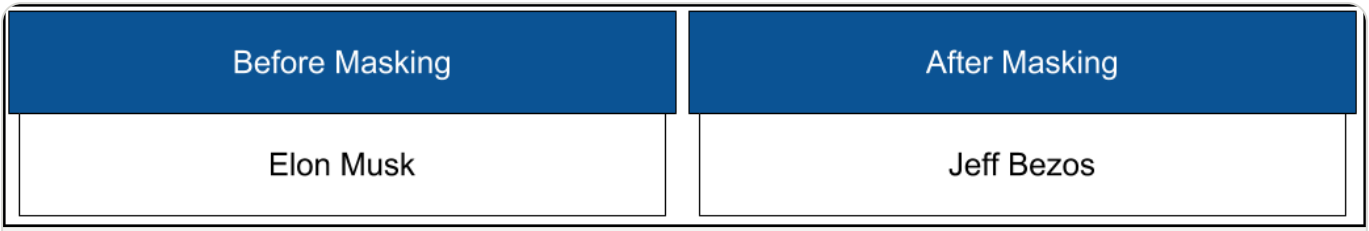
The introduction names two methods. **Column-level profiling** uses regular expressions to scan the *names* of columns (metadata) against dozens of pre-configured classifiers, for example a pattern that recognises a column called `ssn` or `last_name`. **Data-level profiling** also uses regular expressions but scans the *values*. You can write or import your own classifiers. When something is judged sensitive, the engine recommends and assigns a masking algorithm.

A *regular expression* (regex) is a compact pattern language for text. The pattern `[0-9]{3}-[0-9]{2}-[0-9]{4}` describes the shape of a US Social Security number, so a data-level classifier can ask "do the values in this column match that shape?".

The profiling chapter covers profile sets, domains and classifiers in practice. The object model section shows where they fit. For background on why discovery comes first, see discovery and subsetting concepts.

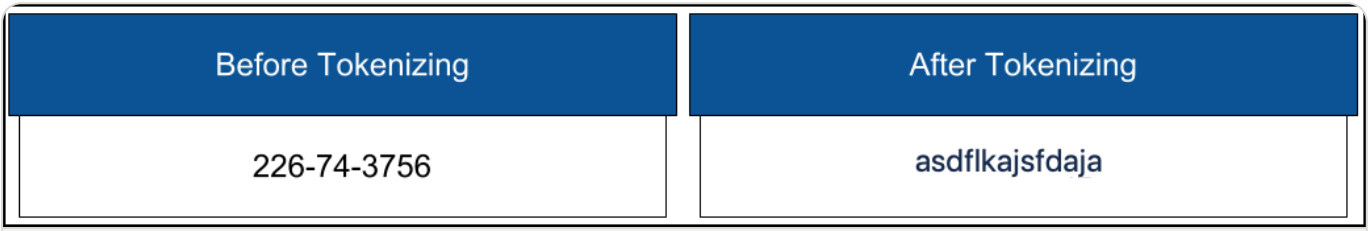
8.3.2 Replacing the values: masking and tokenization

The docs list two ways to secure data. **Masking** (also called anonymization on that page) replaces values with realistic, fictitious ones. **Tokenization** (also called pseudonymization) uses reversible algorithms so the data can be returned to its original state later. The page describes a token as something that conveys no meaning in appearance or format.



Official screenshot: The masking idea in one picture: one person's name goes in, a different but still name-shaped value comes out. The documentation chose famous names for the example; in ShopSphere you would see Asha Menon replaced by another believable name.

Source page ↗



Official screenshot: The tokenization idea: a number-shaped value is replaced by a meaningless string. Unlike masking, the engine keeps what it needs to reverse this on request.

Source page ↗

SHOPSPHERE EXAMPLE

Row 1001 of CUSTOMERS holds `Asha | Menon | asha.menon@example.com | 123-45-6789`. After masking, the test copy might hold `Divya | Pillai | email9482@example.com | 481-92-3057` (example values): same shape, different person, and nobody can work back to Asha. Tokenization would instead be chosen only if an outside vendor must analyse accounts and ShopSphere must later reverse the result.

MASKING IS NOT TOKENIZATION

Tokenization is the only reversible method: the engine stores both token and original. Do not use it on a test copy just to "be safe", because the originals stay recoverable by design.

8.3.3 In-place versus on-the-fly

A masking job has a **Masking Method**: *In-Place* or *On-The-Fly*. The job-creation page defines them in one sentence each.

In-Place

The job updates the data in the source environment with masked values. It reads rows, masks them, and writes the masked values back over the originals in the same database.

Consequence: run it on a *copy* of production, never on production itself. The unmasked originals are gone after the run.

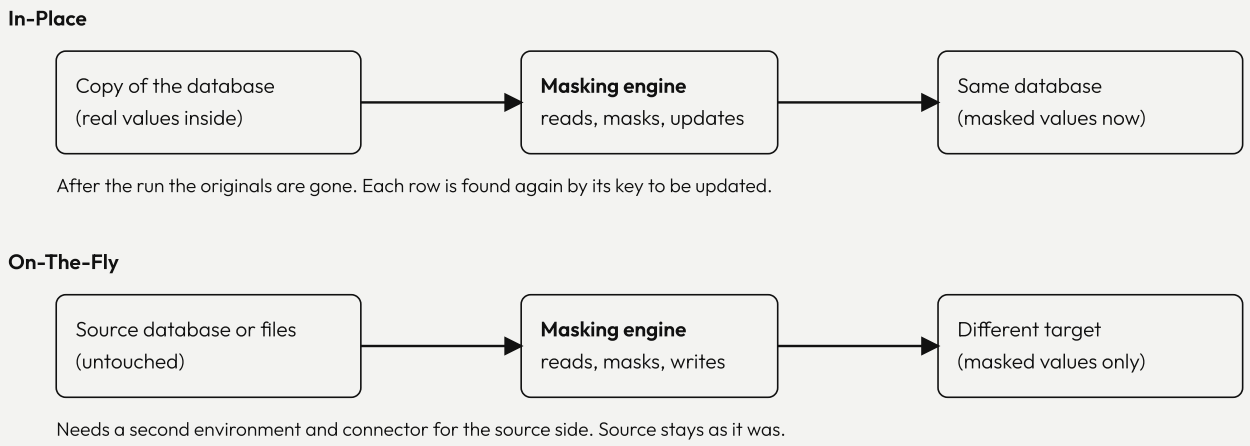
This is why the Delphix pages keep recommending a VDB or other copy as the thing you mask (see the VDB chapter).

On-The-Fly

The job reads unmasked data from a source and writes the masked result to a different target. You choose a source environment and source connector when you define the job.

Consequence: the source stays untouched. The supported pairings are limited: same-type to same-type (Oracle to Oracle, delimited file to delimited file), and database to delimited file.

It needs a second environment and connector for the source side.



Original diagram: the two masking methods. The one-sentence definitions come from the job-creation page; the "found again by its key" note comes from the database preparation pages and is explained in the keys chapter.

One more difference shows up in the keys discussion of the rule set chapter: because an in-place job has to find each row again in order to update it, it needs a key per table. The database preparation pages for SQL Server, Db2 and Sybase offer on-the-fly masking as the alternative when a table has no usable key.

8.3.4 Seeing profiling on ShopSphere

The discovery page names four classifier frameworks. Pairing them with ShopSphere's CUSTOMERS table shows why the engine can use both column names and values. The matches below show what each framework *would look at*, not output from a real profiling run.

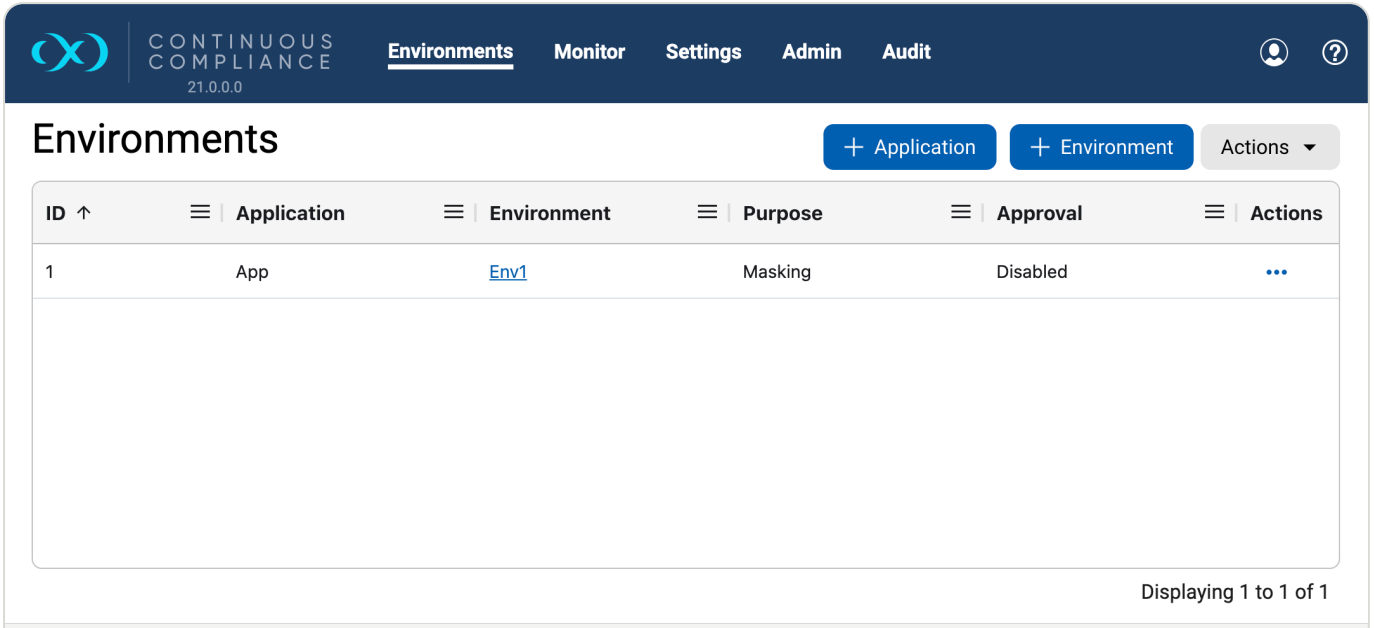
WHICH CLASSIFIER FRAMEWORK COULD HELP WITH EACH CUSTOMERS COLUMN (EXAMPLE DATA)

Column	PATH (column name)	TYPE (data type and length)	REGEX / LIST (the values)
first_name	name looks like "first name"	rules out a number or a date	LIST: is the value in a list of known first names?
ssn	name looks like "ssn"	a short string, not a date	REGEX: does the value look like three digits, two digits, four digits?
email	name contains "email"	a string of enough length	REGEX: does it have an "@" and a dot?
customer_id	no sensitive-sounding name	a number	nothing matches, so it is not flagged by profiling

The last row teaches something important. `customer_id` is not "sensitive" to a profiler, yet it is a key, and keys can break if you mask one side of a relationship. Profiling finds sensitive *values*; it does not understand your business relationships. You, or the data steward, must add key columns to the masking plan on purpose. That is the topic of the keys chapter.

8.4 The engine's web interface

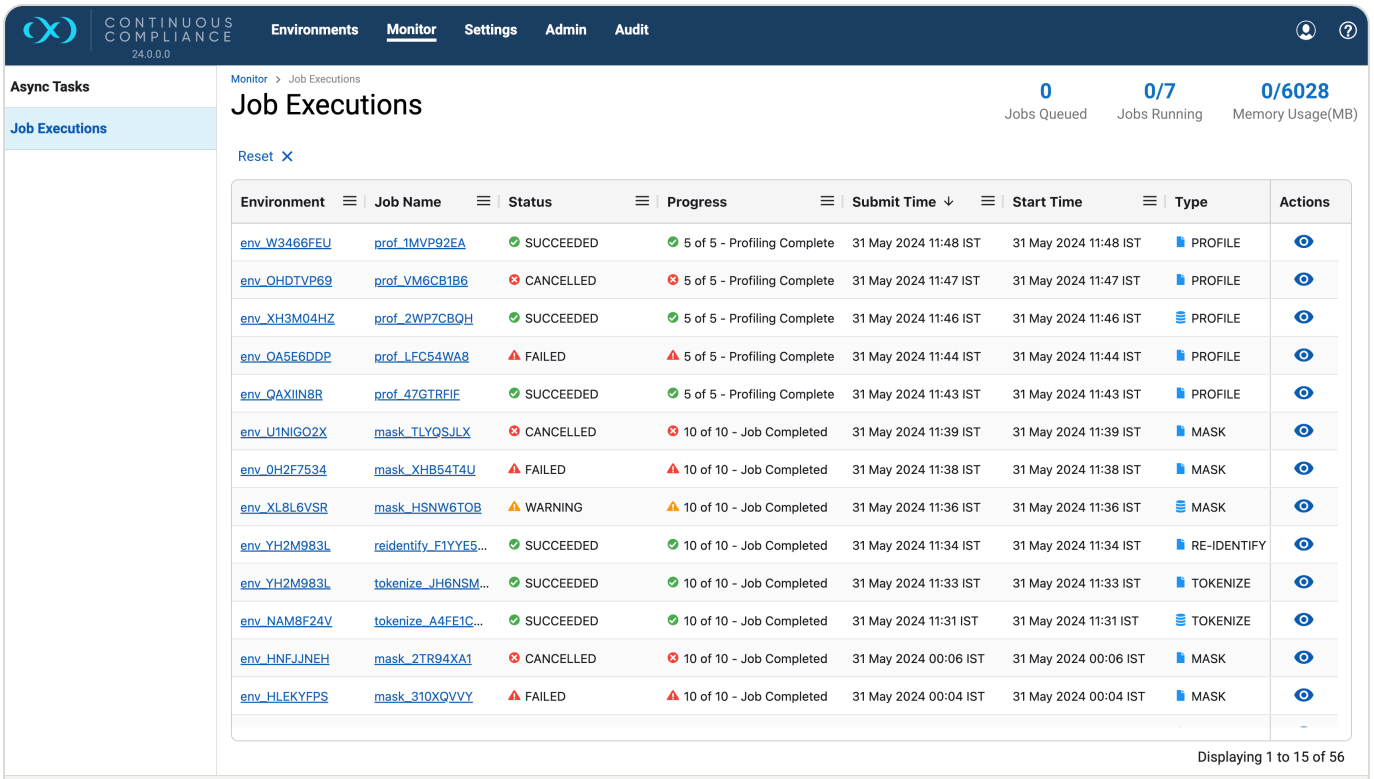
The introduction page calls Delphix Masking a multi-user, browser-based web application. You open a supported browser at the engine's address; the first-time setup page gives the login path as `http://masking-engine.example.com/masking`, where the host name is your own. Once signed in, you see the same strip of five menus on every screen.



Official screenshot: The first screen you see after signing in. Along the top: Environments (selected), Monitor, Settings, Admin and Audit. The grid lists ID, Application, Environment, Purpose, Approval and Actions. In this picture there is one row, application "App", environment "Env1", purpose Masking. The docs text lists no Approval column, but this picture shows one, which is a small sign that screens and text drift apart between releases.

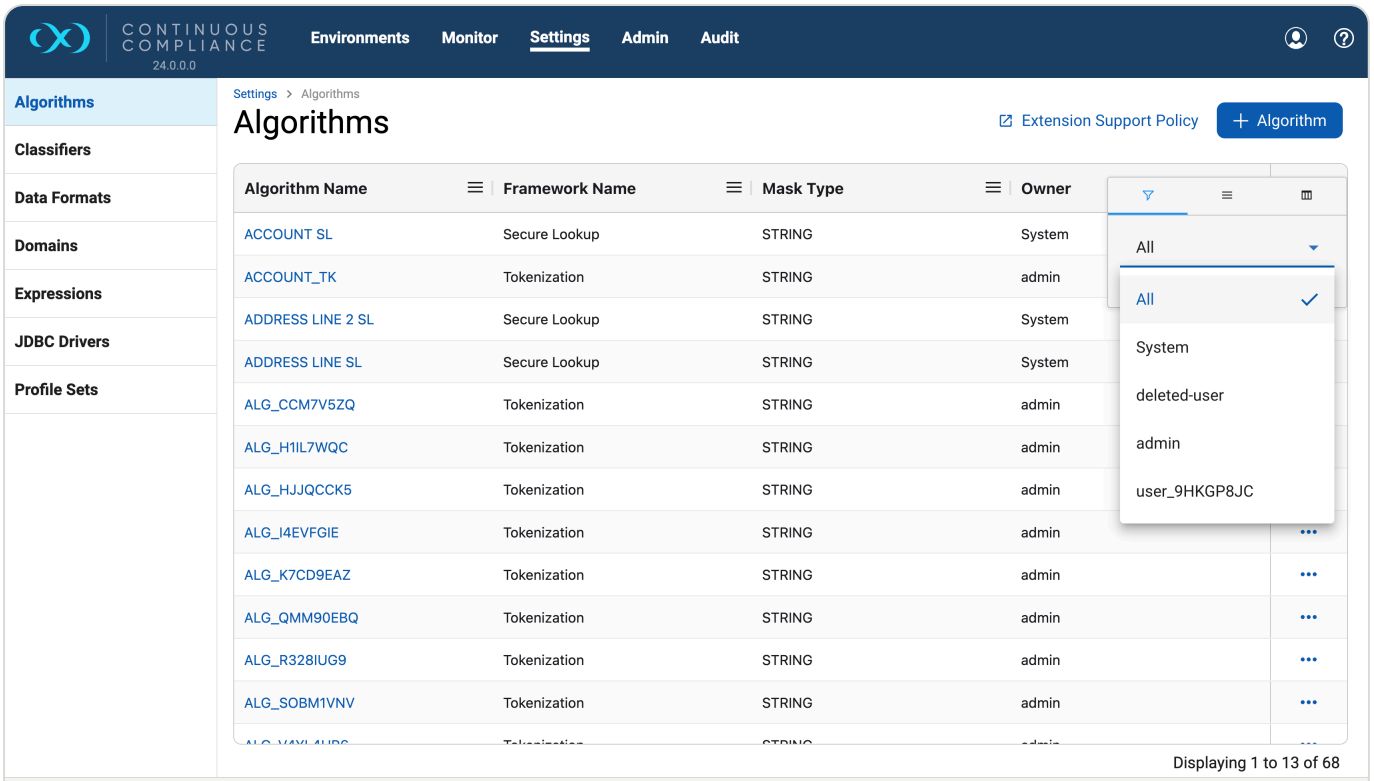
`Source page ↗`

Menu	What lives there	
Environments	Your work. Applications, environments and, inside each environment, three tabs: Jobs, Rule Sets, Connectors. This is the first screen after login.	
Monitor	Two pages in the screenshots: *Async Tasks* and *Job Executions*. Long operations (saving a rule set, exporting an environment) return an Async Task ID that you follow here.	
Settings	Objects shared across all environments. The screenshot's left-hand list shows Algorithms, Classifiers, Data Formats, Domains, Expressions, JDBC Drivers and Profile Sets.	
Admin	Administration. The roles screenshot shows About, Application Settings, Logs, Roles, Users and Utilization. Users and roles are created here.	(Admin > Roles, Admin > Users) and screenshot
Audit	Audit logs.	



Official screenshot: The Monitor menu, Job Executions page. Each row is one run of a job, with Environment, Job Name, Status (SUCCEEDED, FAILED, CANCELLED, WARNING), Progress, Submit Time, Start Time and Type (PROFILE or MASK, among others). Counters at top right show queued jobs, running jobs and memory usage. You will watch this page after every job you start.

`Source page ↗`



Official screenshot: Settings is where the shared building blocks live. The Algorithms grid shows Algorithm Name, Framework Name (for example Secure Lookup or Tokenization), Mask Type and Owner (System or a user name). Anything shown here can be used by any environment.

`Source page ↗`

8.4.1 The grid: the control you use everywhere

Domains, classifiers, profile sets, roles, inventories and rule sets all use one table component, which the GUI page calls the Grid UI. Learn its few controls once.

- **Sorting.** Hover a column name. If a star icon appears, the column can be sorted. Click to sort, click again to reverse. An arrow shows the direction. In a grouped grid, sorting happens inside each group.
- **Filtering.** Hover a header and click the three-lines icon, then the filter tab. Five filter kinds exist: text ("contains"), number (equals, greater than and so on), single-select list, multi-select list and date range. Applied filters appear as chips above the grid; the Reset button clears them. Filters persist in the browser cache until you reset them or clear the cache. A few screens (the page names Monitoring a Single Job) do not show chips or persist filters.
- **Pinning, sizing and extra columns.** The same header menu offers Pin Column, auto-size options and a last tab that reveals hidden columns.
- **Actions.** The "..." button at the end of a row lists what you can do to that row. An action you may not perform is shown disabled, not hidden.

8.4.2 Icons worth recognising

The icon reference page explains about thirty-eight icons. The ones you will meet here: Edit, Delete, Duplicate, View, Download, Test Connection, Database and File; five ruleset-refresh icons (success, in progress, failed, waiting, cancelled, and N/A for file rule sets); the job-status icons (success, created, failed, queued, running, skipped, warning); and, beside each table name in an inventory, three icons that update the Custom SQL, the Filter Condition and the **Logical Key** (shown as a key).

8.4.3 A rule you will trip over: object names

The naming requirements page lists the objects you may name yourself: algorithm, application, connector, domain, environment, file format, job, profiling group, record type, role and rule set. For all of them, leading and trailing spaces are not allowed, and a long list of special characters is rejected, including brackets, braces, parentheses, `~ ! @ # $ % ^ * " ? : ; , / \ ` + = < > ' |`. An attempt to create or rename an object with one of them fails with an error. The maximum name length varies by object; the page says to look it up in the engine's API client.

> **NAME THINGS SAFELY**
>
> Use letters, digits, underscores and hyphens. `SHOPSPHERE_DEV` and `shopsphere-rules-v1` are fine. `ShopSphere (Dev) #1` is not.

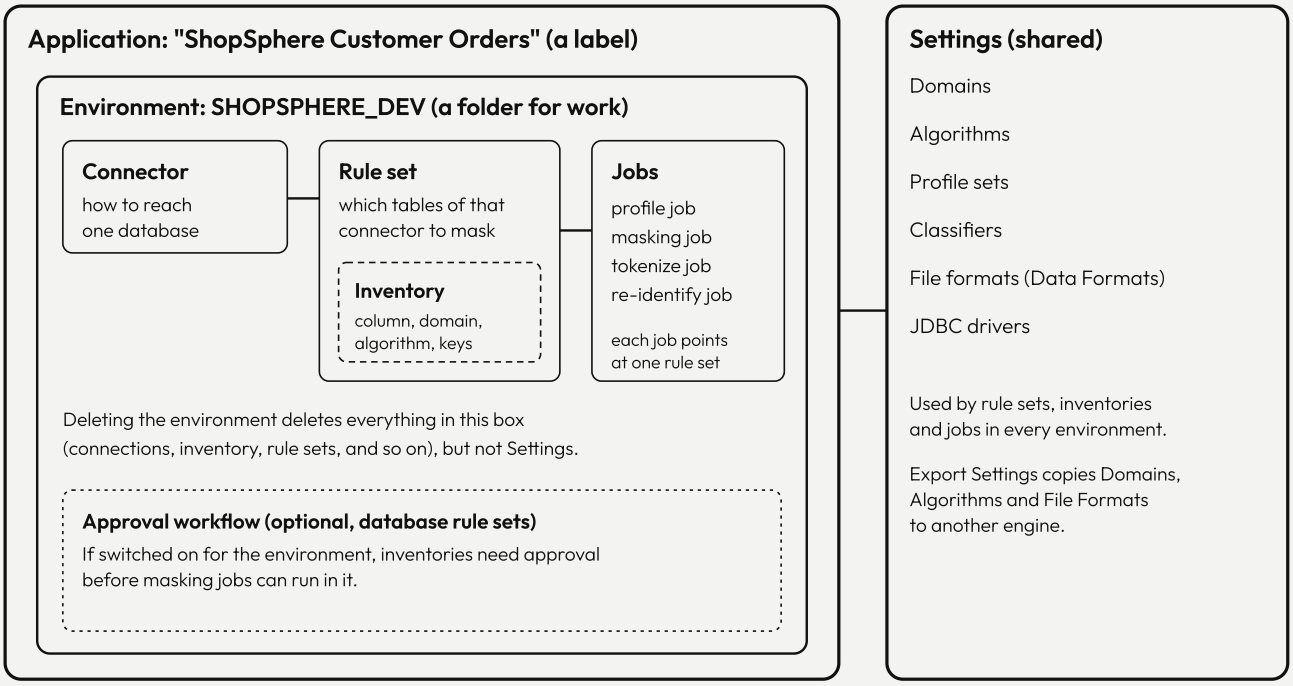
8.4.4 Monitor, Admin and Audit in brief

Long actions (saving a database rule set, exporting an environment) return an *Async Task* ID in a message at the bottom of the screen. Open **Monitor > Async Tasks**, search that ID, and download an export file or the log of a failed import from the row's Actions menu. **Monitor > Job Executions** is the history of job runs. Cancelling a RUNNING job leaves semi-masked data, and any indexes, constraints or triggers it dropped are not re-created, so cancel only in an emergency and expect to restore your copy.

Under **Admin** you find Roles, Users (later in this section), Application Settings (administrators only; engine-wide defaults such as the default update threads, streams and non-conformant data handling, grouped by area), Logs, Utilization (PDF reports of job executions and of database sizes) and About. The **Audit** tab, open to administrators only, records who did what (create, edit, delete, run, test, import, export, login and more) and can be exported to PDF. For a compliance officer it is the evidence that a masking job ran, and who ran it.

8.5 The object model in plain words

This is the most important picture in the chapter. The engine stores a handful of kinds of objects, and almost every confusion is "where does X live?". The short answer is that **work objects live inside an environment**, and **shared objects live in Settings**.



Original diagram: how the Delphix masking objects nest. The nesting comes from the environments page ("folders/containers where a specified user can create and manage connectors, rule sets, and jobs") and from its delete warning. Which Settings items exist comes from the Settings screenshot. A job always points at one rule set, which the job screens ask you to pick.

8.5.1 The objects, one at a time

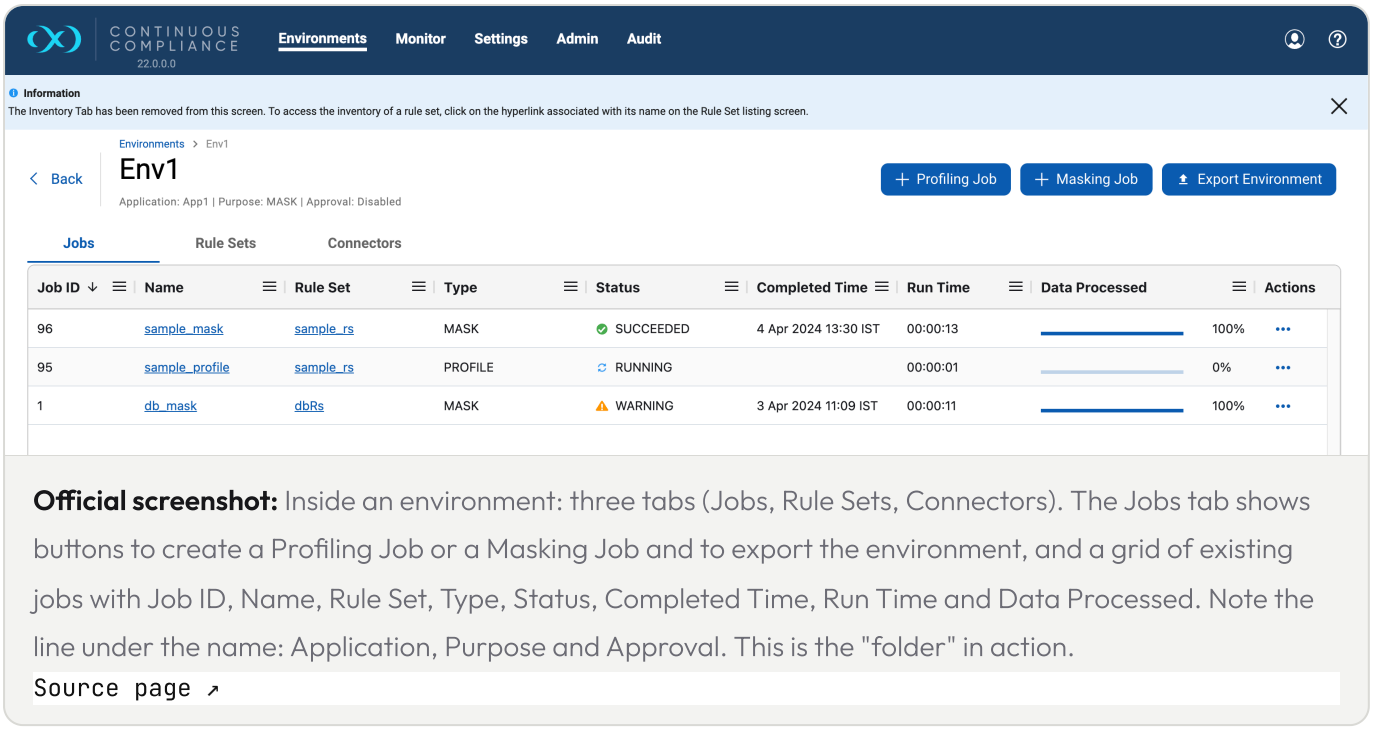
Here is each object, with the documentation's definition and what it is in ShopSphere terms.

Object	What the docs say	ShopSphere
Application	A tag assigned to one or more environments. The docs recommend naming it after the real application. An application must exist before you can create an environment.	"ShopSphere Customer Orders". (Example name; avoid special characters.)
Environment	A construct describing a collection of masking jobs associated with a group of data sources. In practice a folder for connectors, rule sets and jobs. Its *Purpose* is either Mask or Tokenize/Re-Identify and cannot be changed after creation.	SHOPSPHERE_DEV: one place for everything needed to mask a development copy.
Connector	The connection to one data source: a database, a set of files or a mainframe data set. Grouped by "Meta Data Source" on screen.	A connector to the VDB that holds the unmasked copy of SHOPSPHERE_PROD.
Rule set	A group of tables or flat files in one data source on which you may run profile, masking or tokenization jobs.	The four tables CUSTOMERS, ORDERS, ORDER_ITEMS, PRODUCTS.
Inventory	All the data in a rule set plus the method chosen to secure each piece: table or file name, column or field name, data classification and algorithm.	For CUSTOMERS: first_name assigned a first-name algorithm, ssn an SSN algorithm, customer_id a key-safe algorithm.
Domain	A correlation between a category of sensitive data (such as Social Security numbers) and the way it should be secured. Each domain has one default masking algorithm (and optionally a tokenization algorithm).	A "Social Security Number" domain whose default algorithm is the SSN algorithm.

Object	What the docs say	ShopSphere
Algorithm	An *algorithm framework* is a family (Secure Lookup, Segment Mapping and so on). An *algorithm instance* is a named framework-plus-settings that you assign to a column. Example from the docs: "FIRST NAME SL" is an instance of the Secure Lookup framework.	One instance reused on both CUSTOMERS.customer_id and ORDERS.customer_id (see the keys chapter).
Profile set	A named set of classifiers that decides what a profiling job can detect. Several built-in sets ship by default. (The terminology page describes it, in the profile job concepts, as a subset of expressions such as financial expressions.)	A standard profile set run over the four tables.
Classifier	One piece of recognition logic, tied to a domain. Frameworks are PATH, TYPE, REGEX and LIST; PATH and TYPE work on column names and types, REGEX and LIST on the data itself.	The pattern that decides the column called `ssn` is an SSN column.
Job	A saved task you run: profiling and masking in a Mask environment; profiling, tokenization and re-identification in a Tokenize environment.	"Mask SHOPSPHERE DEV rule set (in-place)".

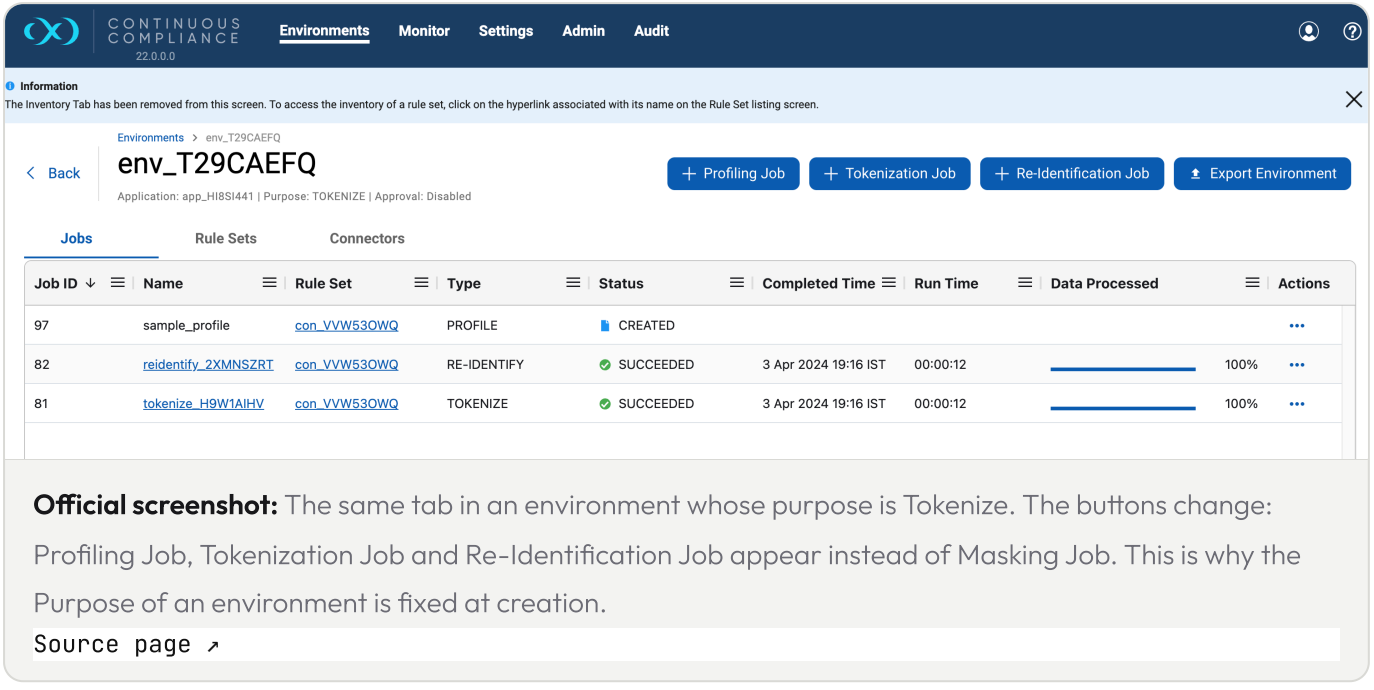
See Discovering your sensitive data and Managing domains for the classifier frameworks and domains. Because a domain has a single default algorithm, you need a distinct domain (with recognition logic) for each distinct algorithm if you want the profiler to assign that algorithm.

> **"CONNECTOR" MEANS TWO DIFFERENT THINGS IN DELPHIX**
>
> The terminology page defines *Connector* as the data-source connection mechanism and, separately, a *Delphix Connector* as a service that runs on Windows staging and target hosts to let them talk to the Continuous Data engine. It warns that the two should not be confused. Here, "connector" always means the first one, the data-source connection you create inside a masking environment. A related word with a licence flavour is also defined: Standard, Select and Premium connectors differ by who builds them and whether you need a separate licence (see the connectors chapter).



Official screenshot: Inside an environment: three tabs (Jobs, Rule Sets, Connectors). The Jobs tab shows buttons to create a Profiling Job or a Masking Job and to export the environment, and a grid of existing jobs with Job ID, Name, Rule Set, Type, Status, Completed Time, Run Time and Data Processed. Note the line under the name: Application, Purpose and Approval. This is the "folder" in action.

Source page ↗

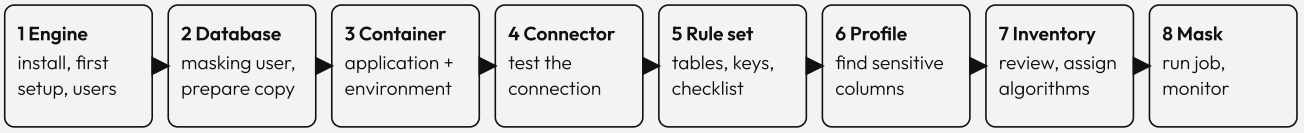


Official screenshot: The same tab in an environment whose purpose is Tokenize. The buttons change: Profiling Job, Tokenization Job and Re-Identification Job appear instead of Masking Job. This is why the Purpose of an environment is fixed at creation.

Source page ↗

8.6 How a masking project flows

With the objects named, the order of work follows from the nesting: you cannot create a rule set before a connector, and you cannot create a connector before an environment. The documentation's own table of contents for "Connecting data" (environments, connectors, extended connectors, file formats, rule sets, inventories) follows exactly this order, and the Continuous Compliance home page groups the rest as "Identify sensitive data", "Preparing data" and "Connecting data".



Steps 3 to 5 are the "Connecting data" part. Steps 6 and 7 are "Identify sensitive data". Step 8 is "Securing sensitive data".

Optional after step 8: if the masked copy is a VDB, the Virtualization Engine can run the masking job while provisioning it.

Steps 6 and 7 can loop: profile, review, adjust, run the profile again.

Original diagram: a typical order of work, summarised from how the documentation's sections depend on each other. Section titles in the notes are the documentation's own.

Where each step is taught: step 1 and the roles are in this section; steps 2 to 4 in the connectors and VDB chapter; step 5 in the rule sets and keys chapter; steps 6 and 7 in the profiling chapter; step 8 in the algorithms and jobs chapter and, for files, the file masking chapter. Profiling only *proposes* domains and algorithms; a human, usually the data steward, confirms them.

HOW A TDM TEAM SPLITS THIS

The "best practices for defining masking roles" page gives a sample RACI (who is Responsible, Accountable, Consulted, Informed). It makes IT Security accountable for deciding which data types are sensitive, the application owner accountable for inventory management, data masking and masked-data validation, and the data masking admin team responsible for masking administration (domains, classifiers, roles, users). In ShopSphere terms: the compliance officer decides that SSN, email and phone are sensitive, the data steward reviews the inventory, and the TDM engineer builds and runs the jobs. This is a sample from the vendor, not a rule.

8.7 The terminology page, term by term

The terminology page is a one-page glossary. The objects (application, environment, rule set, domain, connector) are covered above; this section covers the algorithm and job vocabulary.

8.7.1 Algorithm words

Algorithm framework

A type of masking algorithm. You can create one or more usable instances from it. "FIRST NAME SL" is an instance of the Secure Lookup framework.

Algorithm instance

A named combination of a framework and its settings. This is what you actually assign to a column in the inventory.

Built-in algorithm

An algorithm instance or framework shipped with the engine. Some built-ins have behaviour that matches no built-in framework.

Non-conformant data

A value that does not fit the format an algorithm expects (for example a 10-digit rule fed a 7-digit number). By default such values are *not* masked and a warning is recorded, shown as a yellow triangle beside the job execution. You can choose per job, or globally under Admin > Application Settings > Default Non-Conformant Data Handling, whether the job fails, leaves the value unmasked, or sets it to null.

Collision

Two different real values masked to the same fake value. "Amy" and "Jane" both becoming "Beth" is the docs' example.

> **NON-CONFORMANT DATA MEANS "SILENTLY LEFT REAL" BY DEFAULT**
>
> If ShopSphere's phone column contains `n/a` in some rows, an algorithm expecting ten digits does not fail the job. It leaves `n/a` unmasked and logs a warning. That is harmless for `n/a`, but a real SSN typed with spaces might slip through the same way. Always read the warning count after a job.

8.7.2 The algorithm families in one table

The terminology page describes eight families. The algorithms chapter shows how to build them.

Family	How it works (docs, in short)	Typical ShopSphere use
Secure Lookup	Replaces each value by one picked from an input file using a cryptographic hash of the original, so the same input always gets the same output. Most common. Collisions are possible.	first_name, last_name: a name list.
Segment Mapping	Permutes short numeric or alphanumeric values to others of the same format; guaranteed collision-free if the allowed output range is at least as large as the input set. Maximum 36 digits or characters in the masked value (as the terminology page states).	ssn, tax_id, and key columns that need unique values.
Mapping	You supply the replacement values. The engine records all input-to-output pairs in encrypted form so no two inputs share an output. You must know how many values you will mask. On engines before 6.0.10.0 you cannot mask more than one table at a time with it.	A small fixed list of codes, such as product_code, if you want to hand-pick substitutes.
Tokenization	The only reversible family. Creates a unique token per input and stores both token and original.	Only if an outside vendor must analyse accounts and ShopSphere must later reverse the result.
Binary Lookup, Min Max, Data Cleaning, Free Text Redaction	Replace images with a supplied image; move extreme values toward the middle; standardise spellings without masking; remove sensitive material from free-text columns.	Not needed for the core ShopSphere columns.

Two documented points matter for keys. First, Secure Lookup is deterministic on its input (same hash, same output), and Segment Mapping and Mapping are the collision-free choices, so these are the ones used on unique and key columns. Second, the terminology page's wording for Segment Mapping explicitly lists "Social Security Numbers, primary key columns, or foreign key columns" as its use.

8.7.3 Job options, in one look

The terminology page lists the options of both job types, and you will meet them on real screens in the profiling chapter and the algorithms and jobs chapter. A *profiling job* has a unique Job Name, a Rule Set, No. of Streams (tables processed in parallel), optional memory limits, Feedback Size, Profile Sets and Email. A *masking job* adds the Masking Method (In-Place or On-The-Fly), Update Threads (too many can deadlock T-SQL databases), Commit Size, Disable Trigger and Drop Index (both clear by default, so the job leaves indexes and triggers alone unless you turn them on), and Prescript and Postscript SQL files. Both have a Multi-Tenant option that lets one rule set serve identical schemas through different connectors chosen at run time.

8.8 Where the engine runs: deployment options at overview level

The documentation offers two deployment families.

Virtual machine image

A *virtual appliance*: a ready-made machine image imported into a hypervisor or cloud. The Installation page lists VMware, AWS EC2, Azure, Google Cloud Platform, IBM Cloud, Hyper-V and OCI. You set the network from a console, then finish in the browser-based **Setup Wizard**.

Containerized

The same engine as a pod of three containers on Kubernetes (tested on OpenShift, MicroK8s, AWS EKS and AWS ECS with Fargate), deployed directly or with Helm. FTP and FTPS file connectors are disabled in this form.

8.8.1 What the Setup Wizard asks

The wizard walks through Welcome (pick *Continuous Compliance* as the engine type), passwords and the three administrator accounts, time, network, certificates, storage, outbound connectivity (support proxy, usage analytics and the SMTP server used for job email), authentication (LDAP, SAML/SSO, OAuth2), Kerberos and password vaults, registration and a summary. Facts worth keeping: the documented first logins use default accounts that you must change; the **engine type cannot be changed** once chosen; three failed logins lock the account; the engine **works only in UTC**, so read job times in Monitor accordingly; and in non-containerized engines LDAP is configured in the setup application, not in the masking UI, a change from earlier releases.

8.8.2 Sizing, in one line

The best-practices page gives a dedicated Compliance Engine 8 vCPUs, at least 64 GB of RAM (128 GB or more recommended), a 127 GB system disk and at least 50 GB of data storage, plus rules of thumb of about one vCPU and 1 GB of memory per concurrent masking job. It also says cloning an engine through hypervisor features is not supported as a backup.

8.8.3 What the engine must reach

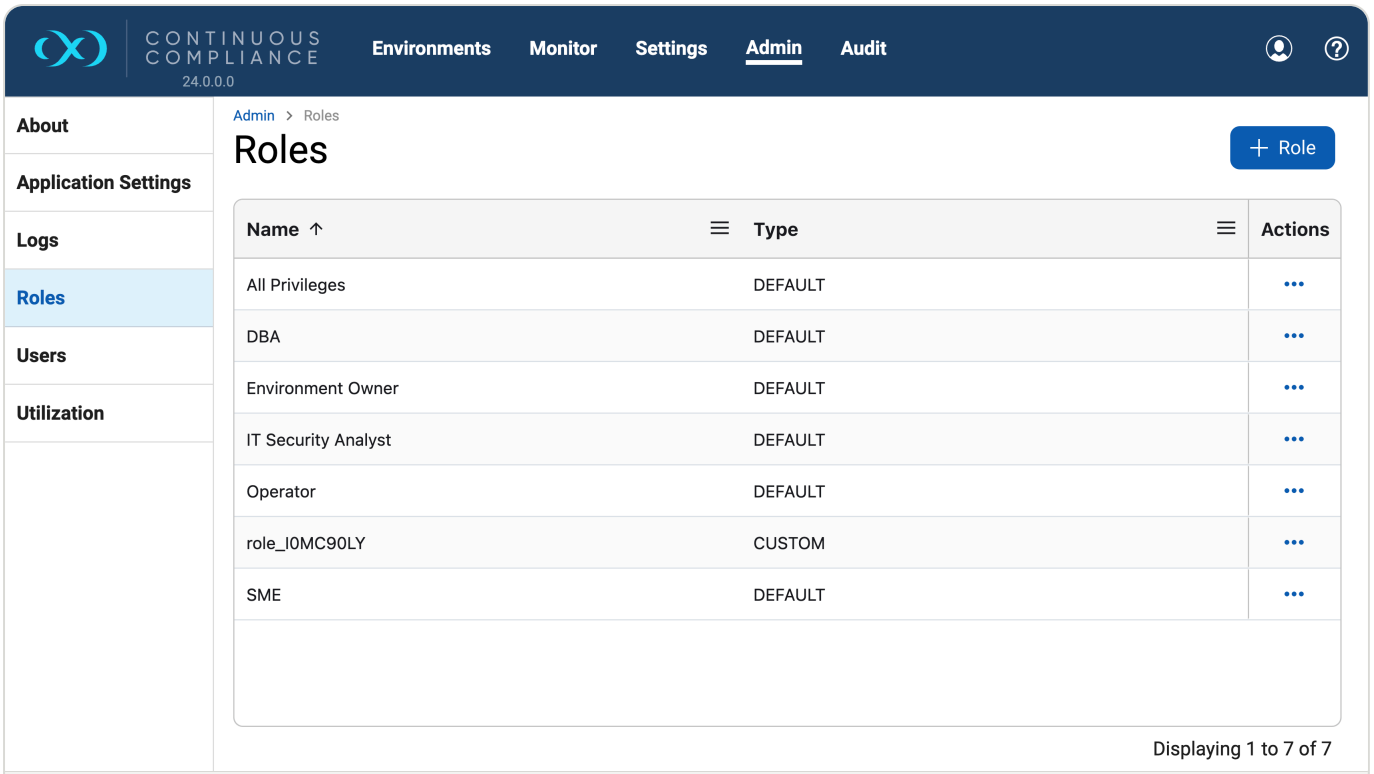
The network requirements page lists the traffic. Outbound, the engine needs SMTP (25), DNS (53), NTP (123, VM engine), HTTPS to Delphix Support and to an S3 endpoint if you use one (443), LDAPS (636) and, most important for you, **"various" ports to your own databases (JDBC) and file servers (FTP, SFTP, NFS, CIFS)**. Inbound, users reach the web interface on 443 (use HTTPS, not 80) and administrators use SSH on 22. The engine cannot be deployed behind a reverse proxy. Supported browsers are the latest and previous versions of Edge, Firefox and Chrome. When a connector test hangs, suspect a firewall on the database port before you suspect the password.

The containerized product is meant for *ephemeral* engines (spun up for a need, then discarded) and differs from the VM engine: no FTP file masking, no Kerberos, no SSO/OAuth, no upgrade validation and no Db2 z/OS or iSeries connector.

> **TWO ENGINES, NEVER ONE MACHINE**
>
> Both the masked-provisioning pages and the configuration page say that Delphix Masking and Delphix Virtualization must never run inside the same virtual machine. Use separate, dedicated engines. The best-practices page also says that Continuous Data and Continuous Compliance are delivered in one OVA, and you pick which one at first setup. So an engine *image* can become either product, but one running engine is only one of them.

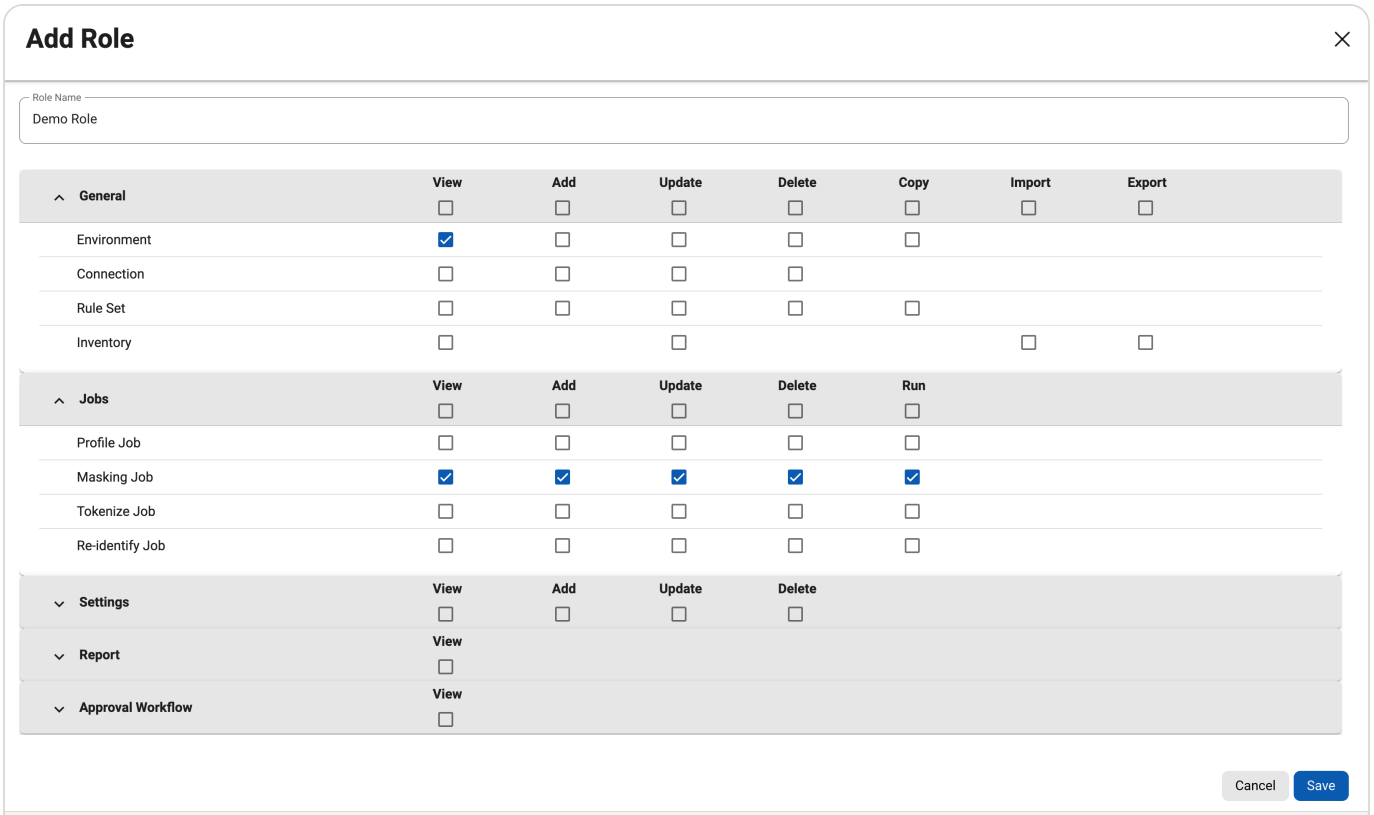
8.9 Users and roles overview

The masking engine uses role-based access control: a *role* is a set of allowed actions on certain kinds of objects, and a *user* is a person who has one role and access to certain environments. This is how ShopSphere keeps the data steward from editing connection passwords and keeps testers away from masking jobs.



Official screenshot: Admin > Roles. The left-hand list shows About, Application Settings, Logs, Roles, Users and Utilization. Built-in roles have type DEFAULT; one role created by a user has type CUSTOM. The + Role button adds a new one.

`Source page ↗`



Official screenshot: Adding a role. The matrix lists objects (Environment, Connection, Rule Set, Inventory, then Jobs such as Profile Job and Masking Job, then Settings, Report, Approval Workflow) against actions. In this picture the role gets View on Environment and View, Add, Update, Delete and Run on Masking Job. Some objects depend on others: the page notes that Masking Job actions depend on Environment - View.

`Source page ↗`

8.9.1 Roles, actions, objects

- **Actions** you can grant: View, Add, Update, Delete, Copy, Export and Import (not every action exists for every object).
- **Objects** you can grant them on include Environment, Connection, Ruleset, Inventory, Profile Job, Masking Job, Tokenize Job, Re-identify Job, Domains, Algorithms, Classifier, Profiler Set, File Format, JDBC Drivers, Plugins, Password Vault, User, Report, Approve Inventories and Diagnostic.
- **Role type**: DEFAULT roles come with the product and can be viewed and duplicated but not edited or deleted. CUSTOM roles can be added, edited, deleted and duplicated.
- **Five default roles** exist: IT Security Analyst, DBA, SME, Operator and Environment Owner, plus a default All Privileges role if one exists.
- Each user has **one role** and a list of environments. Access to an environment does not grant unlimited access: the role still limits what the user can do there.
- The **Administrator** setting on a user gives unrestricted access to everything, plus some admin-only powers (sync, audit page, user API-access settings, deleting any object, and others). With it enabled,

the role and environment fields disappear.

Role (docs)	One-line meaning
Administrator	Unrestricted. Manages the engine, domains, algorithms, roles, users.
IT Security Analyst	All settings functions and almost all application functions except creating, deleting or updating environments.
All Privileges	Everything inside the environments a user is given, but may delete only objects they created.
DBA	Manages connections, scripting and scheduling; no settings.
SME / Analyst / Developer	Manages inventories, creates and views jobs.
Operator	All job privileges.
Environment Owner	Approves workflow and inventories; view access to settings and environment.

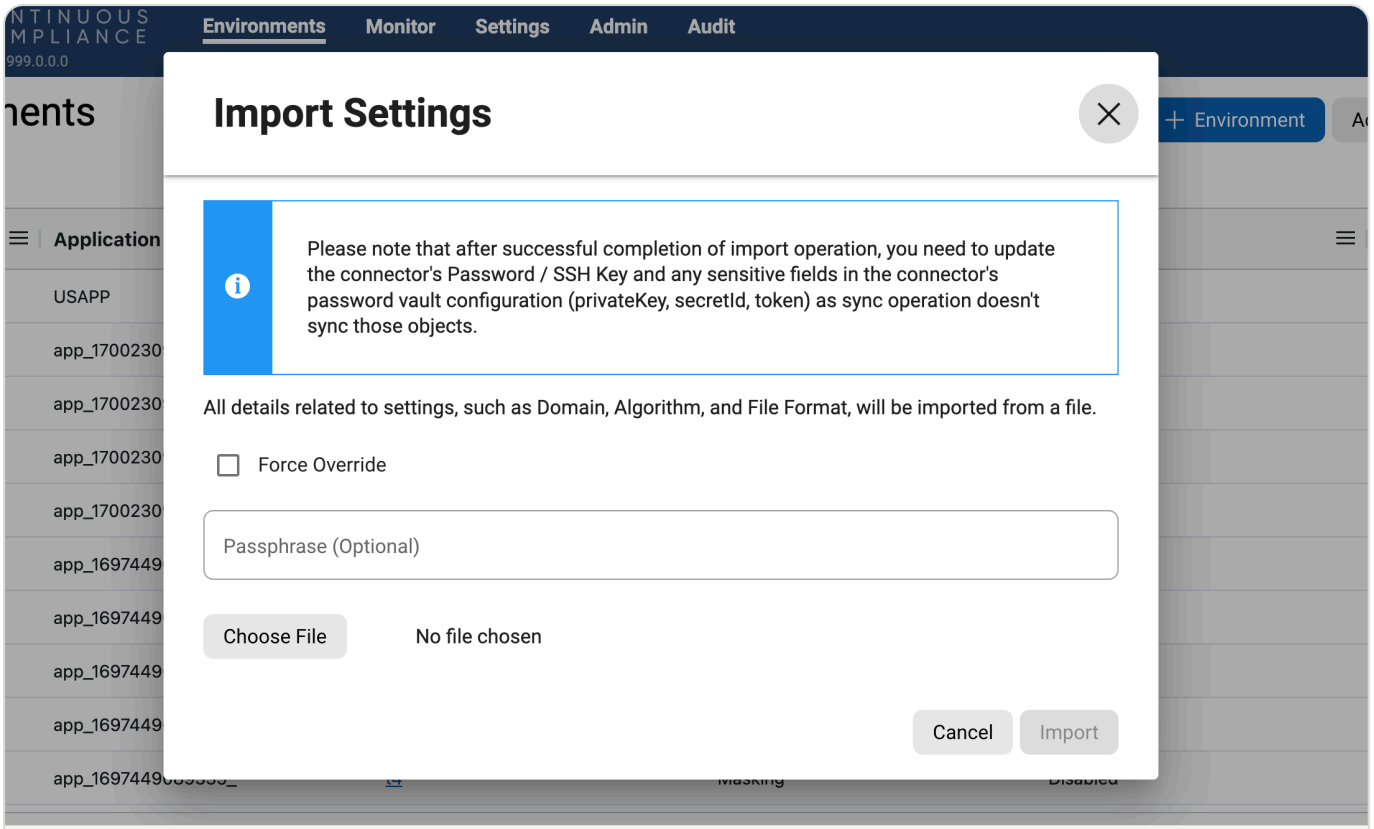
> **A ROLE THAT IS ALSO A SECURITY RISK**
>
> The Users and roles page warns that a user allowed to add or update JDBC drivers can upload code that runs inside the engine's JVM, which is effectively running code on the engine host. The built-in DBA role includes that permission by default; the page suggests a custom role or fewer DBA assignments if you separate database work from driver installation. Treat JDBC-driver permission as administrative.

Creating a user needs a user name, an email (job completion mail goes there), a password, a role and the environments the user may open, or the Administrator checkbox. An account locks after three invalid login attempts.

8.10 Moving work between engines: export, import and sync

Real organisations often run several engines: one per stage of the software life cycle (development, QA, production), or several production engines to share a large workload. Objects must then be copied

between them. **Export Settings / Import Settings** (on the Environments page) moves shared settings such as domains, algorithms and file formats. **Export / Import Environment** moves everything inside one environment (connectors, rule sets, inventory, jobs). An API-based *sync* does the same at scale; the docs advise moving objects in one direction only. Sync does not currently support the Application object or certain algorithms, and bundles from a newer engine may not import into an older one.



Official screenshot: The dialog you use to bring settings from another engine. The blue note matters: passwords are not carried across, so every imported connector needs its password set again before it can connect.

`Source page ↗`

8.11 ShopSphere in object terms

For ShopSphere the objects will be: application SHOPSPHERE_APP; environment SHOPSPHERE_DEV (purpose Mask); a connector to the VDB that holds an unmasked copy of SHOPSPHERE_PROD; a rule set with CUSTOMERS, ORDERS, ORDER_ITEMS and PRODUCTS; a profile job and an in-place masking job; and roles for the TDM engineer (All Privileges), data steward (SME) and DBA. Note that CUSTOMERS.customer_id and ORDERS.customer_id must get the *same* algorithm so every order still points to the right customer; the keys chapter is about exactly that.

Lab: find your way around an engine

Goal: confirm you can locate every object type in this section on a real engine (or on the screenshots, if you have no engine yet). Do not click Delete anywhere.

1. **Sign in and note the version.**
   Open the engine's address in a browser and sign in with the account your administrator gave you.

   Expected result: The Environments page appears, with menus Environments, Monitor, Settings, Admin and Audit. A version number sits next to the logo in the top bar.

2. **Add an application.**
   Click + Application, enter `SHOPSPHERE_APP` and Save. This needs a role that allows it.

   Expected result: You return to the Environments list. If the button is missing or disabled, your role lacks permission, which tells you how roles work.

3. **Add an environment.**
   Click + Environment. Choose the application, name it `SHOPSPHERE_DEV`, leave Purpose as Masking and leave the approval checkbox clear. Click Save And View.

   Expected result: The environment opens with tabs Jobs, Rule Sets and Connectors, all empty. The line under the name shows Application, Purpose and Approval.

4. **Visit each tab.**
   Open Connectors, then Rule Sets, then Jobs.

   Expected result: Each shows an empty grid. Connectors offers a connection button, Rule Sets an add button, Jobs the profiling and masking job buttons.

5. **Visit the shared Settings.**
   Open Settings and click Algorithms, then Domains.

   Expected result: A long list of built-in algorithms (Owner: System) and domains. Note the framework name column, for example Secure Lookup.

6. **Try the grid.**
   In the algorithms grid, hover the Algorithm Name header, open the menu, filter by "contains" with the text `NAME`, and read the chips above the grid. Then press Reset.

Expected result: Only matching rows remain while the filter chip is shown, and all rows return after Reset.

7 **Look at Monitor and Admin.**
Open Monitor (Async Tasks, Job Executions), then Admin > Roles.

Expected result: Monitor is probably empty on a new engine. Roles lists the default roles (All Privileges, DBA, Environment Owner, IT Security Analyst, Operator, SME) with type DEFAULT. If Admin is missing you are not an administrator.

▶ **Show a worked answer**

8.12 Frequent releases: expect your screen to differ

The documentation has a release-notes page for each of 2025.1 to 2026.5 (eleven releases in under two years), and it shows. Screenshots carry different version numbers in the corner (21.0.0.0, 22.0.0.0, 24.0.0.0, 26.0.0.0, 2025.6.0.0). The environments screenshot has an Approval column its text does not list. The connector wizard text says "Connection Mode: Basic or Advanced" while the screenshot says "Hostname/IP (Basic) or JDBC URL (advanced)". Release 2026.5 moves the host from JDK 8 to JDK 17, while older pages on extended connectors still mention Java 8. A rule set page calls one action both "Verify" and "Checklist".

> **A HABIT THAT SAVES HOURS**
> Before following any procedure on a real engine, find the version of your engine (the screenshots show it in the corner of the top bar; the Admin menu also has an About page) and compare it with the documentation page for that version. When screen and text disagree, trust the screen.

8.13 Common mistakes and gotchas

COMMON MISTAKES

- **Looking for algorithms inside the environment.** They live in Settings and survive an environment delete.
- **Assuming profiling masks anything.** It only proposes; data changes when a masking job runs.
- **Running an in-place job on production.** It overwrites the source values. Mask a copy.
- **Expecting to change an environment's Purpose.** It is fixed; create a new environment.
- **Ignoring warnings.** Non-conformant data is left unmasked by default and only shows a yellow warning.

8.14 Recap

RECAP

- Delphix Continuous Compliance is the masking engine: it profiles to find sensitive data, masks irreversibly, or tokenizes reversibly.
- Masking is *in-place* (overwrite the source, so use a copy) or *on-the-fly* (read one place, write another).
- The interface has five menus: Environments, Monitor, Settings, Admin, Audit. The same grid control (sort, filter, pin, actions) appears everywhere.
- Work objects (connector, rule set, inventory, jobs) live inside an environment inside an application. Shared objects (domains, algorithms, profile sets, classifiers, file formats, JDBC drivers) live in Settings.
- A project flows: engine, prepare the database, environment, connector, rule set, profile, review the inventory, mask.
- The engine is a VM image or a Kubernetes deployment, never in the same machine as the Virtualization Engine. It works in UTC.
- Roles are sets of actions on objects; each user has one role plus environments.
- The product changes often. Check your version and trust your screen over this text.

Delphix: connecting to data and to a VDB

A masking engine that cannot reach your data cannot mask it. This section covers the connection layer of Delphix Continuous Compliance (documentation version 2026.5): the application and environment that hold your work, the **connector** that tells the engine how to log in to a database or a file store, and the database-side preparation needed before the first job. It also answers what it means to "create a connector to a VDB from the Delphix masking engine".

The overview chapter introduces the vocabulary. The next object built on a connector is the rule set (rule sets and keys), and profiling and the hands-on labs use it.

> **DOCUMENTATION GAPS**
>
> The connector pages link to a "Data Source Support" page that was not available in the documentation copy used here, so the version-by-version support matrix for each database is not given; check it for your engine version. The documentation has *no page called "create a connector to a VDB"*. The VDB section builds the procedure from the documented connector steps and the masked-provisioning pages, and marks every step the documents do not spell out.

8.15 Words to know first

Application

A tag, assigned to one or more environments, naming the system whose data you manage. You must create one before you can create an environment.

Environment

The folder-like container that holds the connectors, rule sets and jobs for one group of data sources.

Connector

A saved definition of how the engine reaches one data source: type, host, port, credentials and options.

Meta Data Source

The grouping label on the connector screen: Database, File or Mainframe.

JDBC

Java Database Connectivity: the standard way a Java program talks to a database. The engine uses a *JDBC driver* (a library) for each kind of database and a *JDBC URL* (a connection string such as `jdbc:vendor://host:port/name`) to say where to connect.

Extended connector

A connector built on a JDBC driver that *you* upload, for databases the engine does not support natively.

Masking user

A dedicated database account that the connector logs in as, with just the permissions masking needs.

dSource

In Delphix virtualization, the engine's compressed copy of a source database's data, from which VDBs are made.

VDB (virtual database)

A full read-and-write copy of source data, provisioned from a dSource or from another VDB and managed by the Continuous Data engine.

Masked VDB

A VDB whose data was masked by a masking job during provisioning.

SDD (Selective Data Distribution)

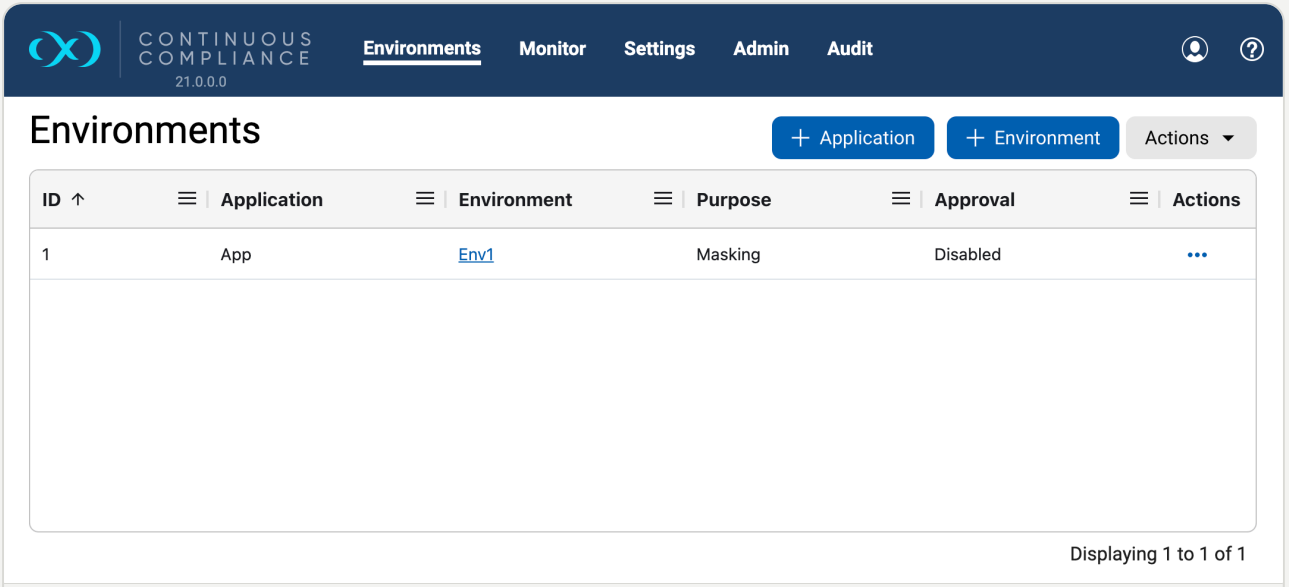
A way to replicate a masked VDB to another Delphix Engine in non-production without sending the original unmasked data.

8.16 Applications and environments

8.16.1 What they are and why they exist

Think of a shared drive at work. You do not dump every project's files in one folder; you create a folder per project and put sub-folders inside. An environment is that project folder. The environments page describes it as a place where "a specified user can create and manage connectors, rule sets, and jobs". The *application* is a label above it that groups related environments: for ShopSphere, one application "SHOPSPHERE_APP" and environments such as SHOPSPHERE_DEV and SHOPSPHERE_QA. Analogy limit: unlike a shared-drive folder, an environment also carries rules (its Purpose and whether approvals are required).

The main Environment screen is the first page every user sees after signing in. It lists only the environments that user may access. Its columns are ID (the number used to refer to it from the Masking API), Application, Environment, Purpose, and Actions (Edit, Export, Duplicate, Delete). The screenshots below show it. Sorting a column also sorts the Environment column ascending.



Official screenshot: The Environments list. The three buttons at the top right are the three places you will start: + Application, + Environment, and an Actions menu (which holds Export Settings, Import Settings and Import Environment). The environment name is a link that opens the environment.

`Source page ↗`

8.16.2 Adding an application

An environment cannot be created without an application, so start here.

1. **Click + Application** near the top right of the Environments page.

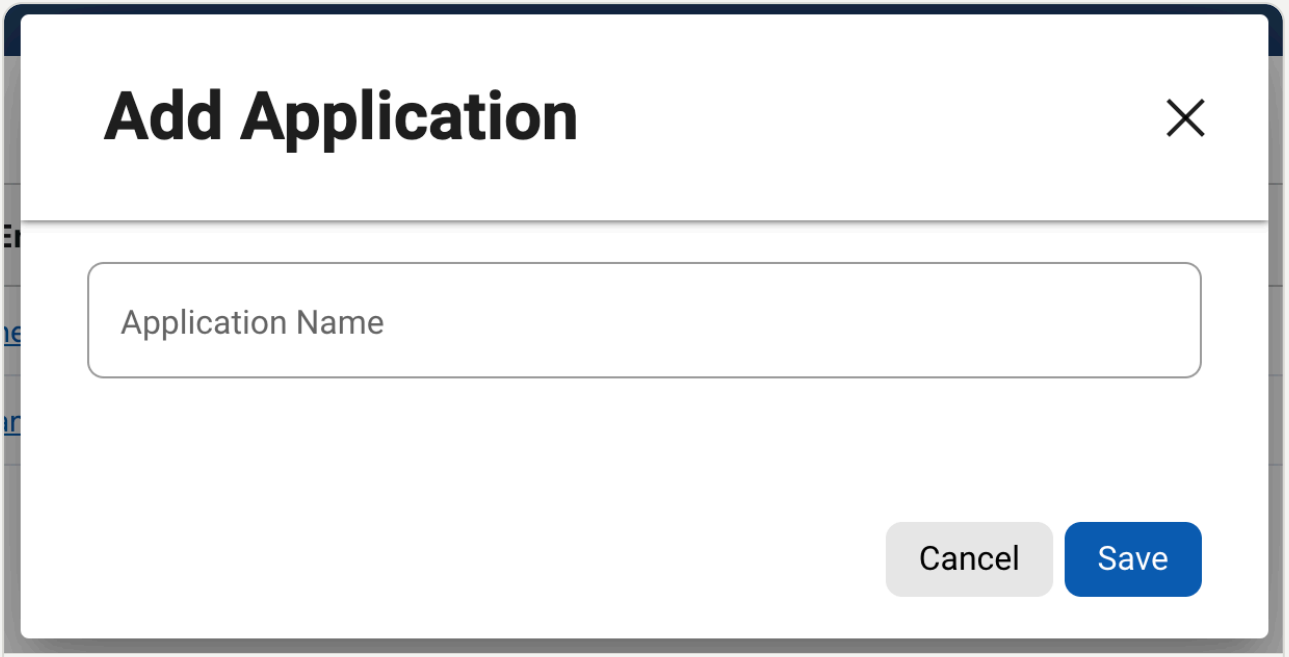
The engine wants a label first so that every environment is tagged with the system it belongs to.

Expected result: A small "Add Application" dialog with one box, Application Name, and Cancel and Save buttons.

2 **Type a name and Save.**

Use only characters the naming rules allow (see the overview chapter). Example for ShopSphere: `SHOPSPHERE_APP`.

Expected result: You return to the Environments list.



Official screenshot: The whole Add Application dialog. It really is just one field.
`Source page ↗`

8.16.3 Creating an environment

1 **Click + Environment.**

It opens the Add Environment dialog.

Expected result: A dialog with Select Application, Environment Name, Purpose and an approval checkbox.

2 **Choose the application.**

The dropdown shows existing applications. It is "for informative purposes", the page says: it labels the environment and does not change behaviour.

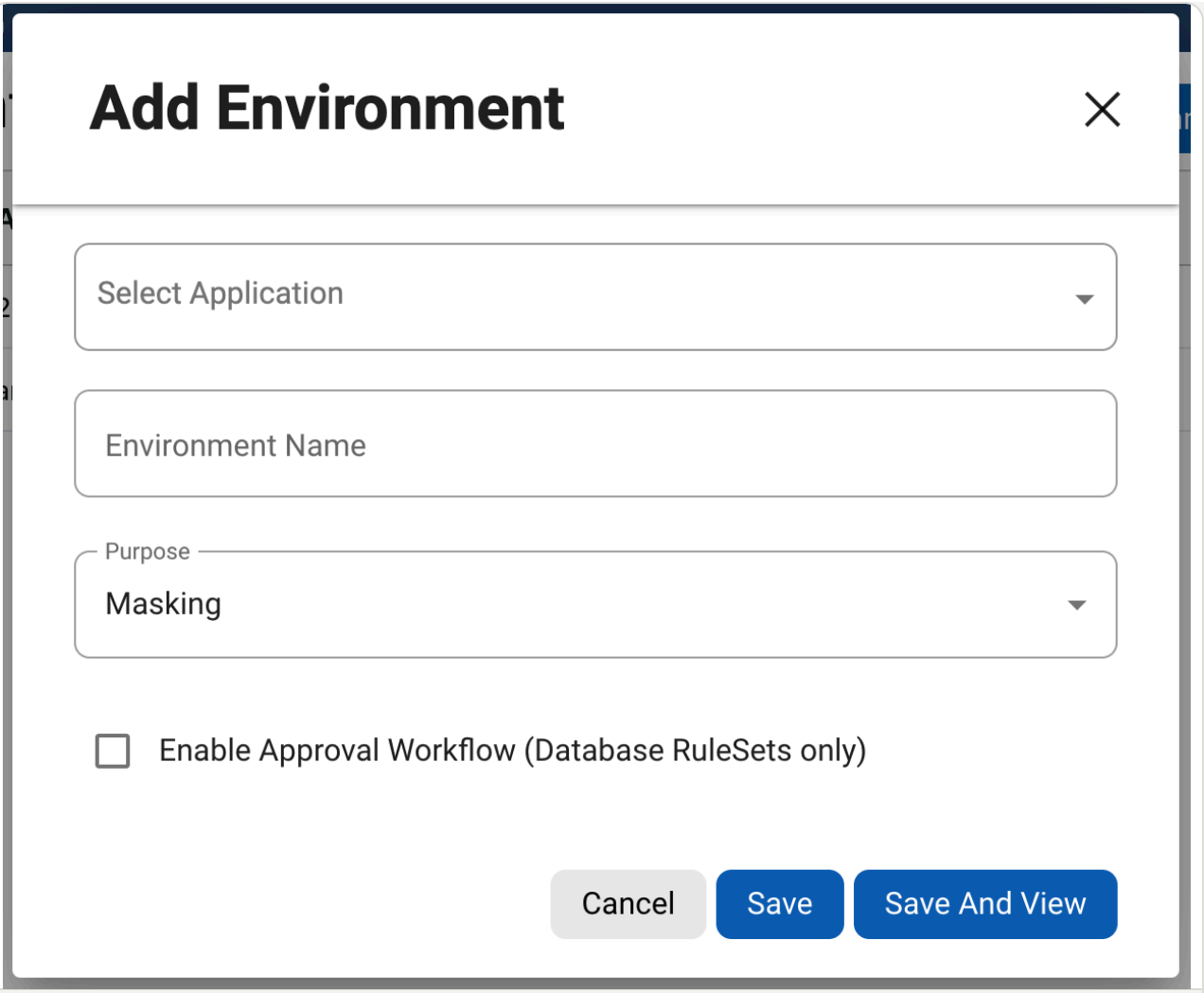
3 **Type an Environment Name.**
For example `SHOPSPHERE_DEV`.

4 **Choose the Purpose.**
Mask or *Tokenize/Re-Identify*. This decides which job types the environment offers. The screenshot shows the default as "Masking". Purpose becomes read-only after creation, so choose carefully.

5 **Decide on "Enable Approval Workflow (Database rulesets only)".**
If ticked, inventories must be approved before masking jobs can run in this environment. It applies only to database rule sets. For a first lab leave it clear.

6 **Click Save, or Save And View.**
Save returns to the list; Save And View opens the environment's Jobs screen.

Expected result: The new environment is in the grid. Opening it shows tabs Jobs, Rule Sets and Connectors.



Official screenshot: The dialog behind the steps above. Purpose defaults to Masking in this picture, and the approval checkbox is clear.

`Source page ↗`

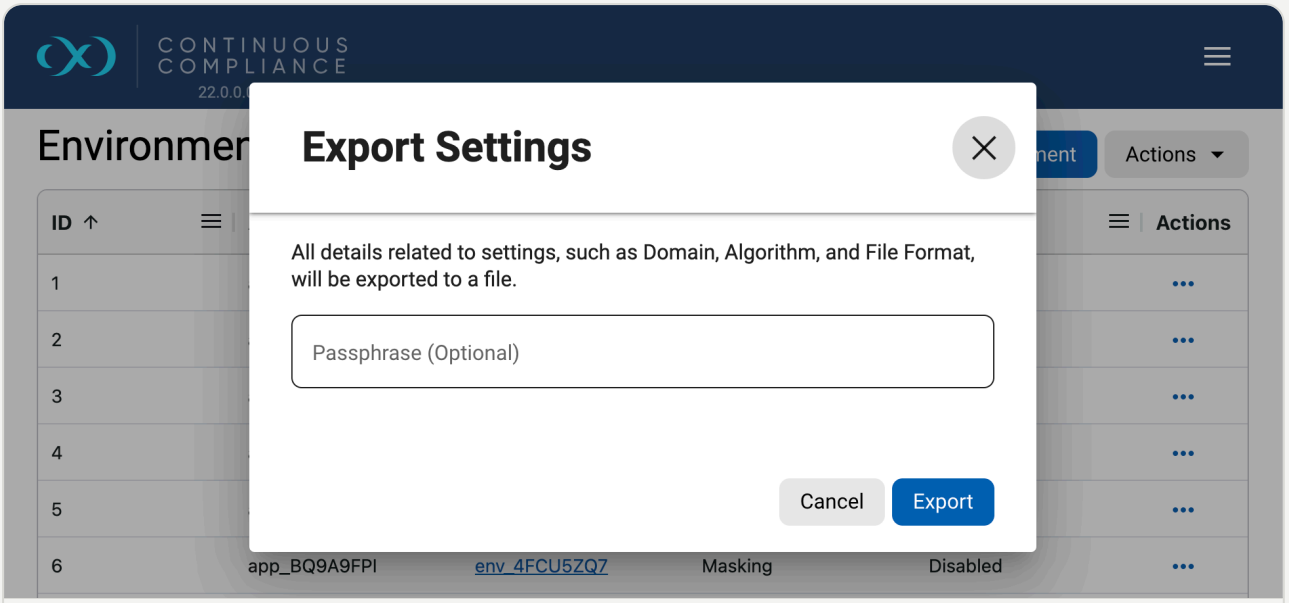
8.16.4 Editing, copying, exporting and deleting

All of these are in the Actions (...) menu beside each environment.

- **Edit.** You may change the application and the name and the approval workflow setting. Purpose is read-only.
- **Duplicate.** Makes an exact copy (the page calls this "a very powerful feature" when you want several similar environments). You give a new name; the copy has the same purpose. Choose Duplicate or Duplicate And View.
- **Export.** Writes everything in the environment (connectors, rule sets, inventory, jobs and so on) to a file, with an optional passphrase. The engine returns an Async Task ID; follow it under Monitor, and when it succeeds, use Download File in that task's Actions menu. The main reason to export is moving environments between masking engines.

- **Delete.** After a confirmation, deletes *everything* for that environment: connections, inventory, rule sets and so on. It does not delete universal settings such as algorithms and domains.

Two more actions live in the page-level Actions menu: **Export Settings / Import Settings** (Domain, Algorithm, File Format and similar; the import has a Force Overwrite checkbox and must use the same passphrase as the export) and **Import Environment** (with options to import settings too, to overwrite, to choose the target application and environment, and an OTF Environment checkbox for importing on-the-fly connectors). Details are in the overview chapter's section on moving work between engines.



Official screenshot: Export Settings. Notice the text: settings such as Domain, Algorithm and File Format go into the file. A passphrase is optional, but if you set one you need the same passphrase to import.

`Source page ↗`

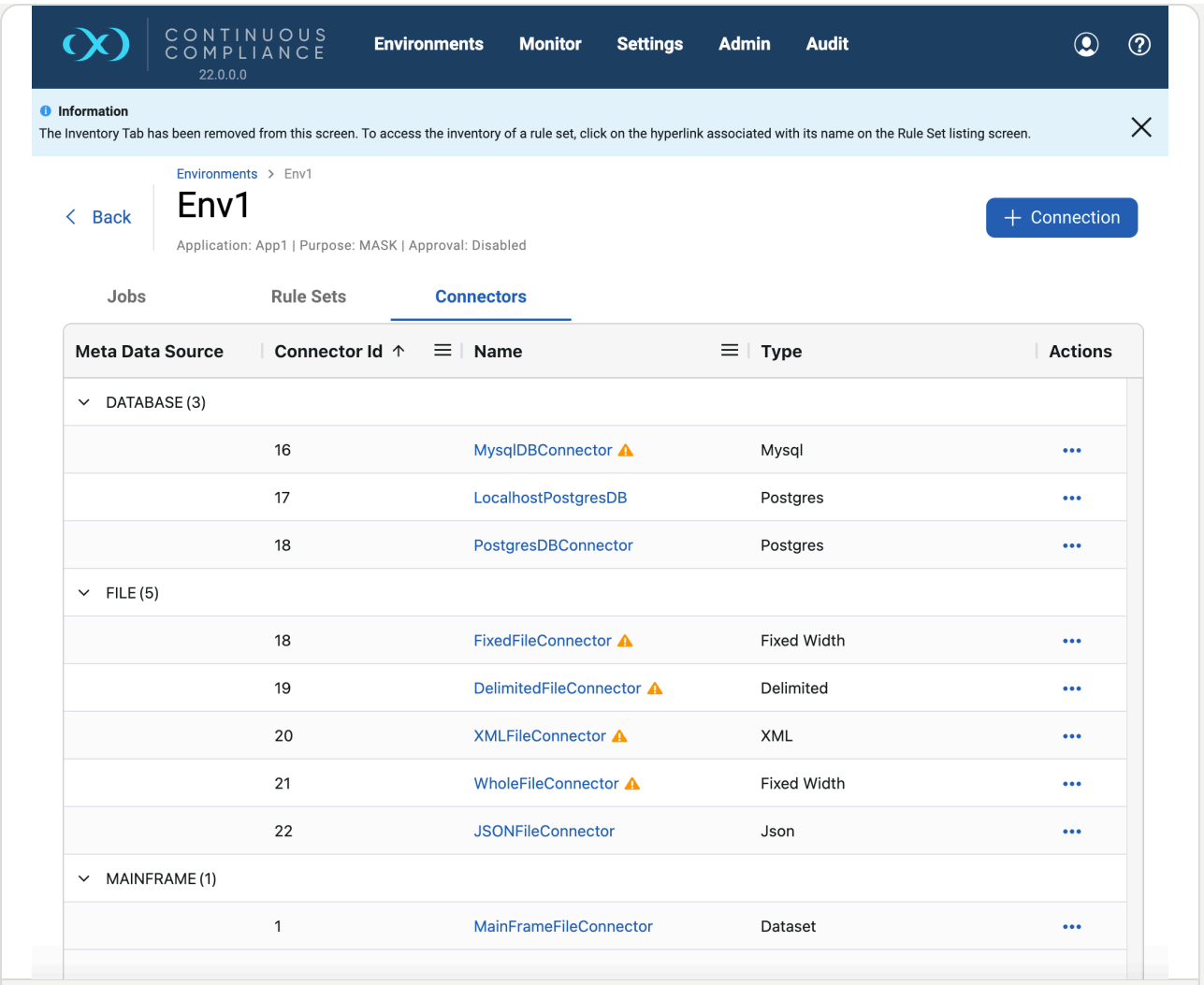
> **DELETE IS BIGGER THAN IT LOOKS**
>
> Deleting a *connector* also deletes its rule sets and inventory data (the connector page says so). Deleting an *environment* deletes all its connectors, rule sets, inventories and jobs. Neither can be undone from the screen. Export the environment first if you might want it back.

8.17 Connectors: what they are

A **connector** is a saved answer to the question "how does the engine log in to this data?". Without it, the engine has nowhere to read tables from. Connectors are grouped within an environment: open the environment and click its **Connectors** tab.

There are three families, shown as groups on the Connectors screen (the page calls the grouping field "Meta Data Source").

Meta Data Source	What it points at	Examples of Type (from the docs and screenshots)
Database	A relational database reached over JDBC.	Oracle, MS SQL, Sybase, Mysql, Postgres, Db2 and Extended (see below).
File	Files in a directory or storage location.	Delimited, Fixed Width, XML, JSON, Parquet.
Mainframe	Mainframe data sets, usually encoded in EBCDIC (the mainframe character encoding).	Dataset.



Official screenshot: The Connectors tab. Rows are grouped by Meta Data Source. Some names have an orange warning triangle: the page says that icon means authentication is missing, and hovering it tells you whether the password, SSH key, credential path or access keys are absent. The button at top right reads "+ Connection" in this release (other screenshots in the docs read "+ Connector").

`Source page ↗`

The columns are: **Connector ID** (used by the API), **Name**, **Type** and **Actions** (Edit, Test Connection, Delete). You can filter and sort by ID, Name and (filter only) Type inside each group.

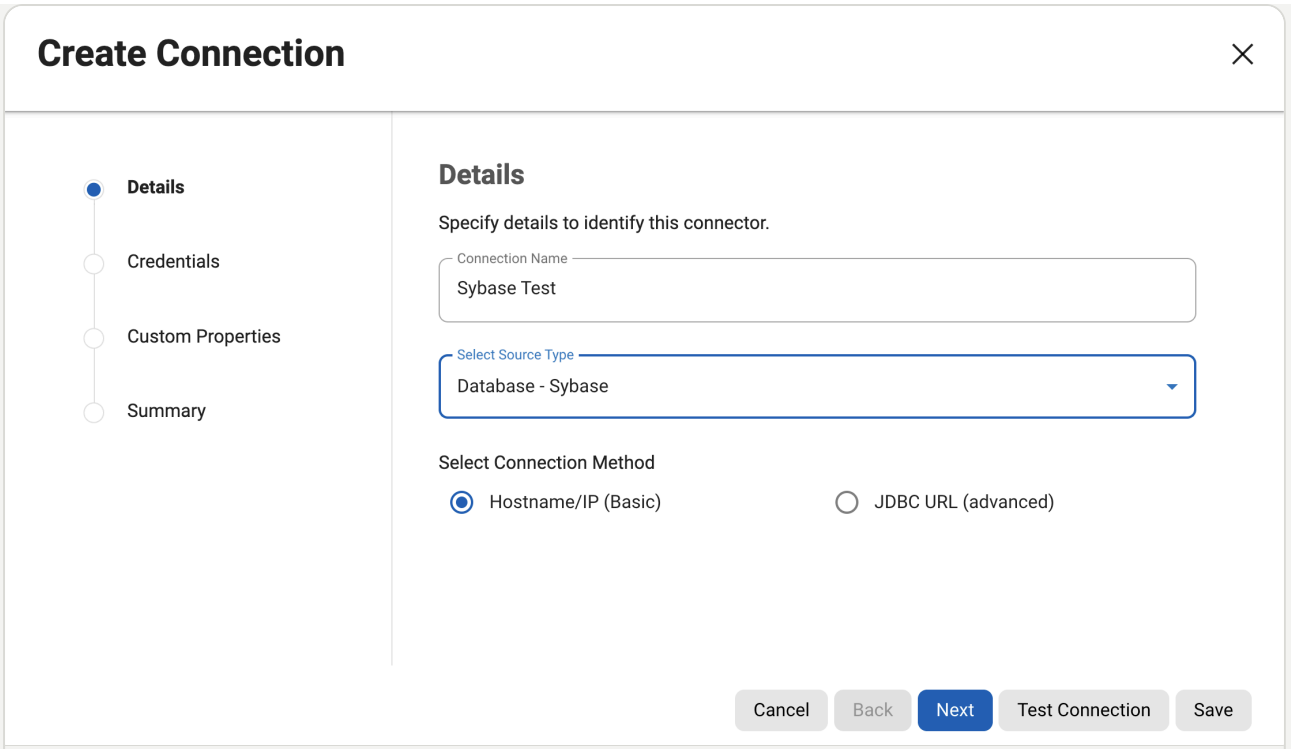
> **ENCRYPTION IS NOT ON BY DEFAULT**
>
> The connectors page states that connections are not encrypted by default, but you can enable TLS/SSL. The per-database pages (for example Oracle and SQL Server) explain how, typically by importing the database's certificate into the engine's truststore through the setup application, restarting the engine, and using a JDBC URL that asks for TLS. For Oracle the page adds that SSL with the BASIC Oracle connector is not supported, so you must use an advanced (JDBC URL) connector for TLS. For SQL Server the page says the needed JDBC properties depend on your instance and your DBA must supply them.

8.18 Creating a connector: the wizard, field by field

Click **+ Connection** (or "+ Connector") at the top right of the Connectors tab. A wizard opens with steps down the left side: **Details**, **Credentials**, **Custom Properties** (database connectors only; file connectors skip it) and **Summary**. Every step has Cancel, Back, Next, **Test Connection** and Save buttons at the bottom, so you can test at any step.

8.18.1 Step 1: Details

- **Connector Name.** A name of your choice. In the screenshots the label reads "Connection Name" or "Connector Name" depending on release. The same naming restrictions apply as for other objects. For ShopSphere: `SHOPSPHERE_VDB_CONN`.
- **Source Type.** The kind of data source. Each entry in the dropdown is prefixed with its Meta Data Source, for example "Database - Sybase", "File - Delimited" or "Database - Extended".
- **Connection Mode** (Oracle, MS SQL Server and Sybase only): *Basic* asks for host, port and so on; *Advanced* asks for the full JDBC connect string including any database parameters. In the screenshot of this step, the same choice is labelled "Select Connection Method" with the options "Hostname/IP (Basic)" and "JDBC URL (advanced)". The docs say Basic and Advanced are not available for all source types.



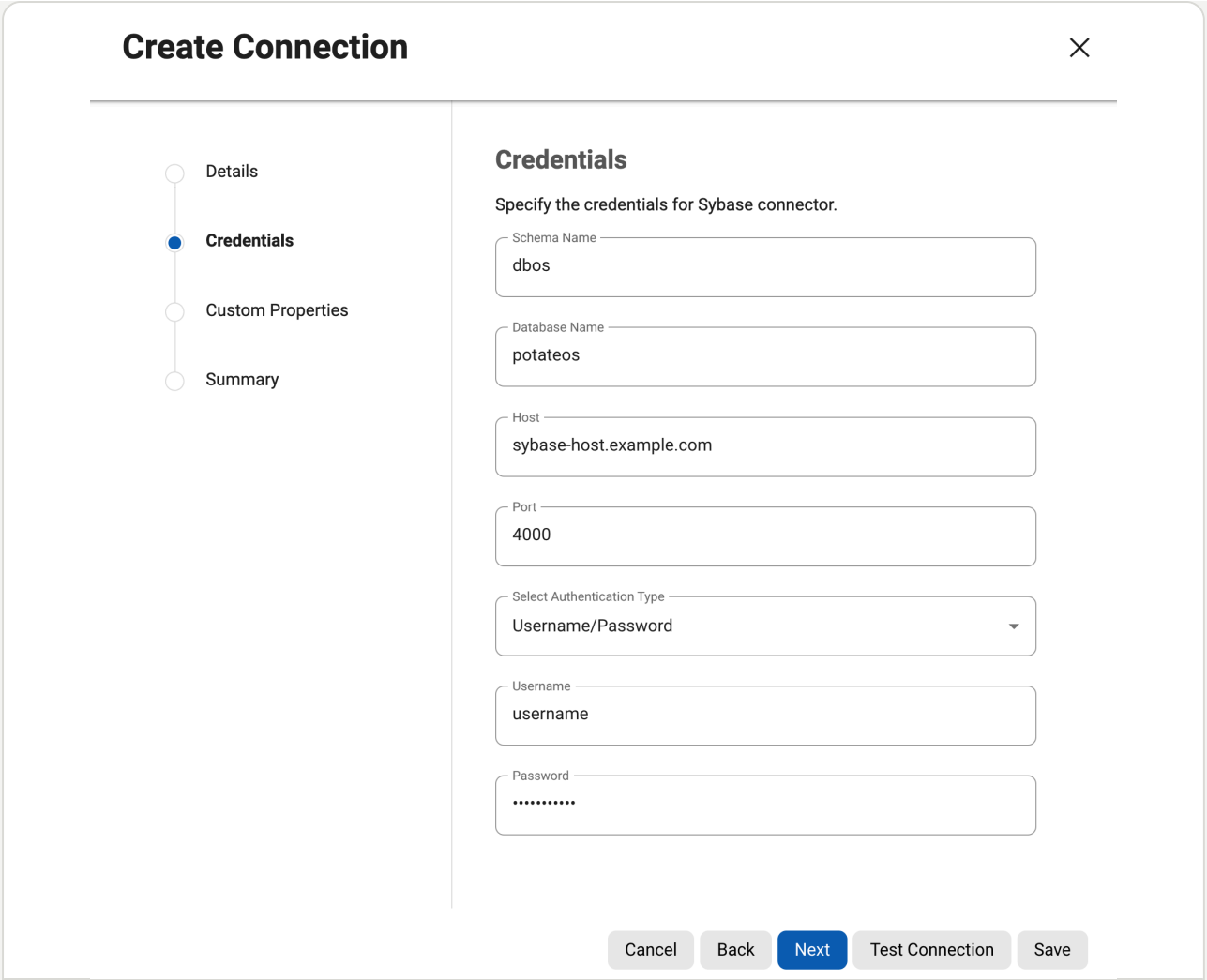
Official screenshot: Step 1, Details. The documentation's own example uses a Sybase connector. The Basic radio button is selected. Footer buttons: Cancel, Back (greyed on the first step), Next, Test Connection, Save.

`Source page ↗`

8.18.2 Step 2: Credentials for a database connector

The fields that appear depend on the source type, and the docs say to ask your DBA if you are unsure of a value. This is the full documented list of database fields.

Field	Meaning (from the docs)	Watch out for
Schema Name	The schema containing the tables this connector will access.	A *schema* is a named group of tables inside a database. The rule set can only list tables in this schema.
Database Name	The database you connect to.	**Case-sensitive.** It must match exactly the name as the instance knows it. The 2026.5 release notes even fix a case where leading or trailing spaces here passed the connection test but made jobs fail.
Host Name/IP	Network name or address of the database server.	The engine itself must be able to resolve and reach it.
Port	TCP port of the server.	For MS SQL Server optional: defaults to 1433 and need not be specified if the server uses it.
SID	(Oracle only) the Oracle System ID.	
Instance Name	(MS SQL Server only, optional) the instance name.	If given, the connector ignores Port and contacts the SQL Server Browser Service on port 1434 to learn the instance's port. So the firewall must allow 1434 and the instance's port.
JDBC URL	(Advanced mode for Oracle, MS SQL Server and Sybase) the custom URL, typically with host and port.	Also used by Generic and Extended connectors.
Authentication type, Username, Password	The login the connector uses.	The password is stored encrypted. Options below.



Official screenshot: Step 2, Credentials, for the same Sybase example. Fields in this release: Schema Name, Database Name, Host, Port, an authentication-type dropdown, Username and Password. The values are the documentation's sample values, not anything to copy.

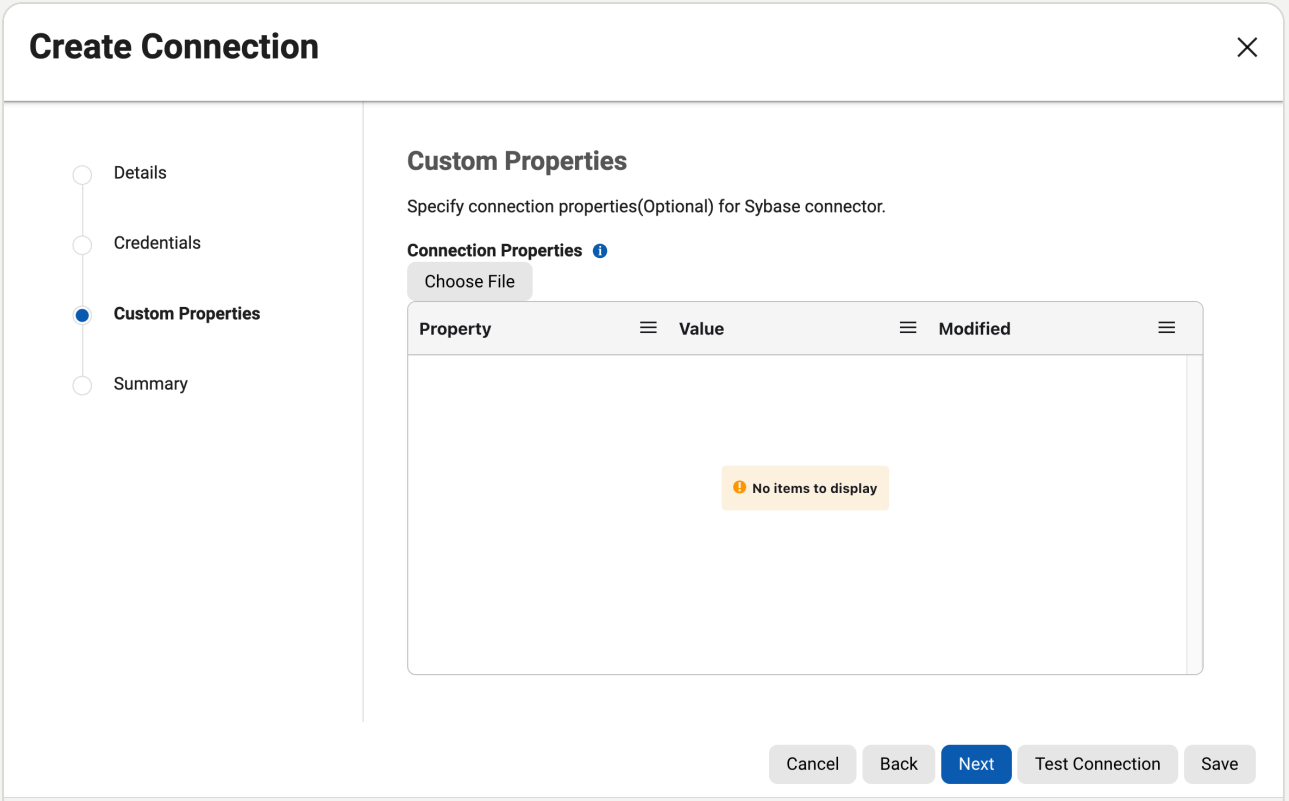
`Source page ↗`

The **authentication type** dropdown is offered according to the source type. The authentication-types page describes three kinds.

- **Username/Password.** The simplest. The password is stored encrypted.
- **Kerberos.** The engine authenticates to the Kerberos server first, then uses its Kerberos credentials instead of a password. The Login ID is then treated as the Kerberos user principal name, and extra fields (Principal Name, and for Sybase a Service Principal) appear. The appliance must be configured for Kerberos beforehand, and Kerberos is disabled in containerized masking.
- **Password Vault.** The engine asks an external vault for the credentials using a *Credential Path* you enter. The vault must be configured first.

8.18.3 Step 3: Custom Properties

Optional. Use it to upload a *connection properties file*, normally a Java properties file (lines of `key=value`), that adds settings to the JDBC connection. The rules, all from the managing-connectors page: the file must have the extension `.properties` and follow Java properties syntax; it is read as ISO 8859-1 text; any property you list is passed to the driver even if the driver does not officially support it, and every property loaded is logged.



Official screenshot: Step 3, Custom Properties. The table stays empty until you upload a file. After upload, or when editing an existing connector, it lists each property with its Value and whether it was Modified from the driver default.

`Source page ↗`

When the same property can be set in several places, the engine applies a fixed order. This matters when a connector "ignores" something you typed.

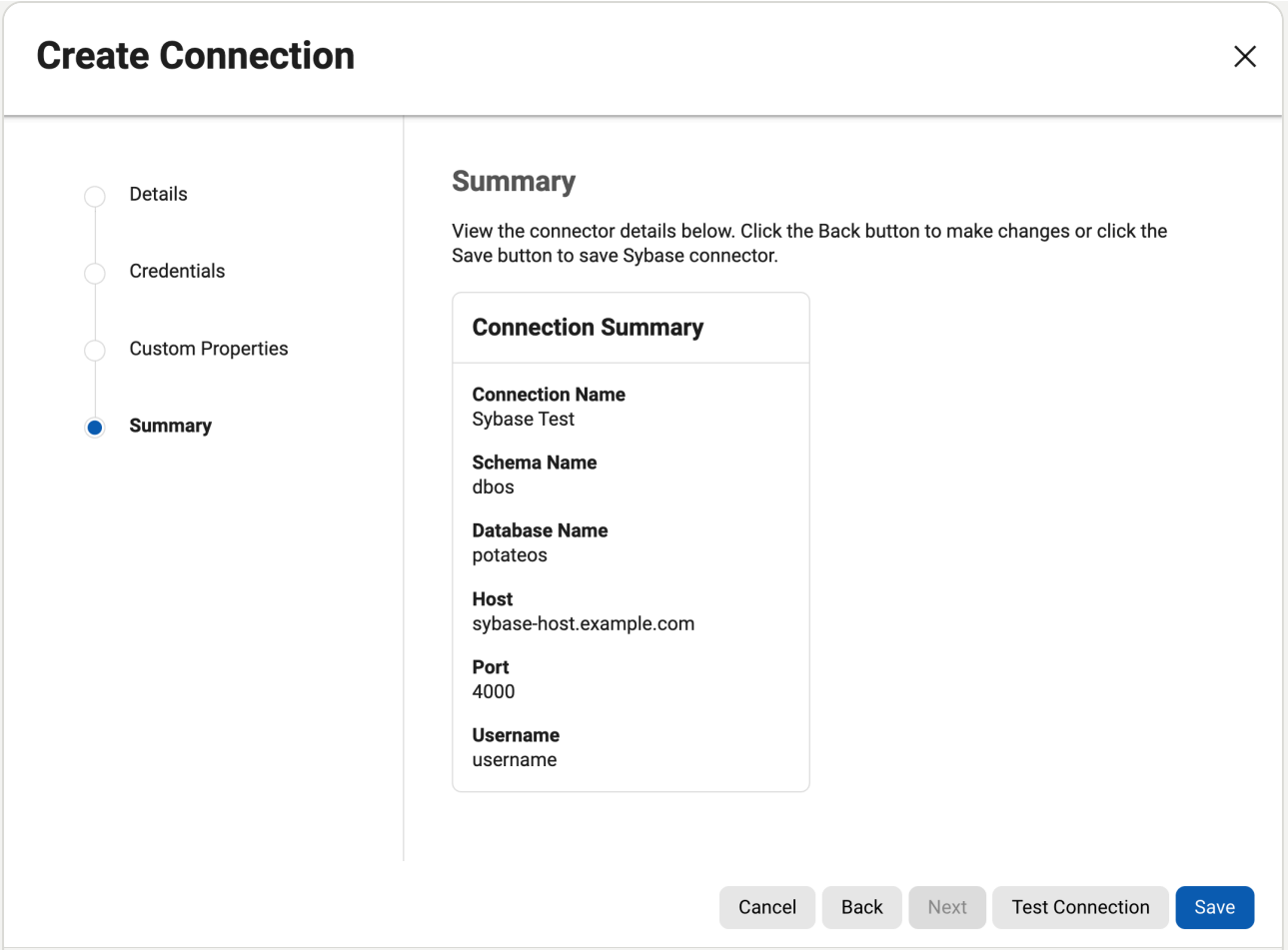
Property	Winner, then runner-up
Username, Password, Schema	The **form field** always wins.
Everything else (database name, host, port, SID, instance name, other driver properties)	1. The properties file, 2. the form fields, 3. the JDBC URL.

The page adds that the JDBC driver itself decides what happens with duplicates, and that a connector form shows *either* the Basic fields *or* the Advanced URL, never both, so those two cannot clash. It also notes a driver quirk: for Caché DB the schema cannot be set through the URL, so it has to come from the form field or a properties file. Customising the JDBC URL is limited to Advanced, Generic and Extended connectors, while a properties file works for every database connector.

> **SECRETS IN PROPERTIES FILES ARE VISIBLE**
>
> Anything in a properties file, including a password, is shown in plain text to every user who has the View Connector privilege. The page recommends putting sensitive values in the form fields (obfuscated everywhere) or, second best, in the JDBC URL (plain text on screen but redacted in support bundles).

8.18.4 Step 4: Summary and Save

The last step lists everything you entered (without the password) so you can check before saving. Click **Save**. If anything is wrong, Back returns you to the step to fix it.



Official screenshot: Step 4, Summary. Read it like a boarding pass before you leave: name, schema, database, host, port, username. Next is disabled because this is the last step.

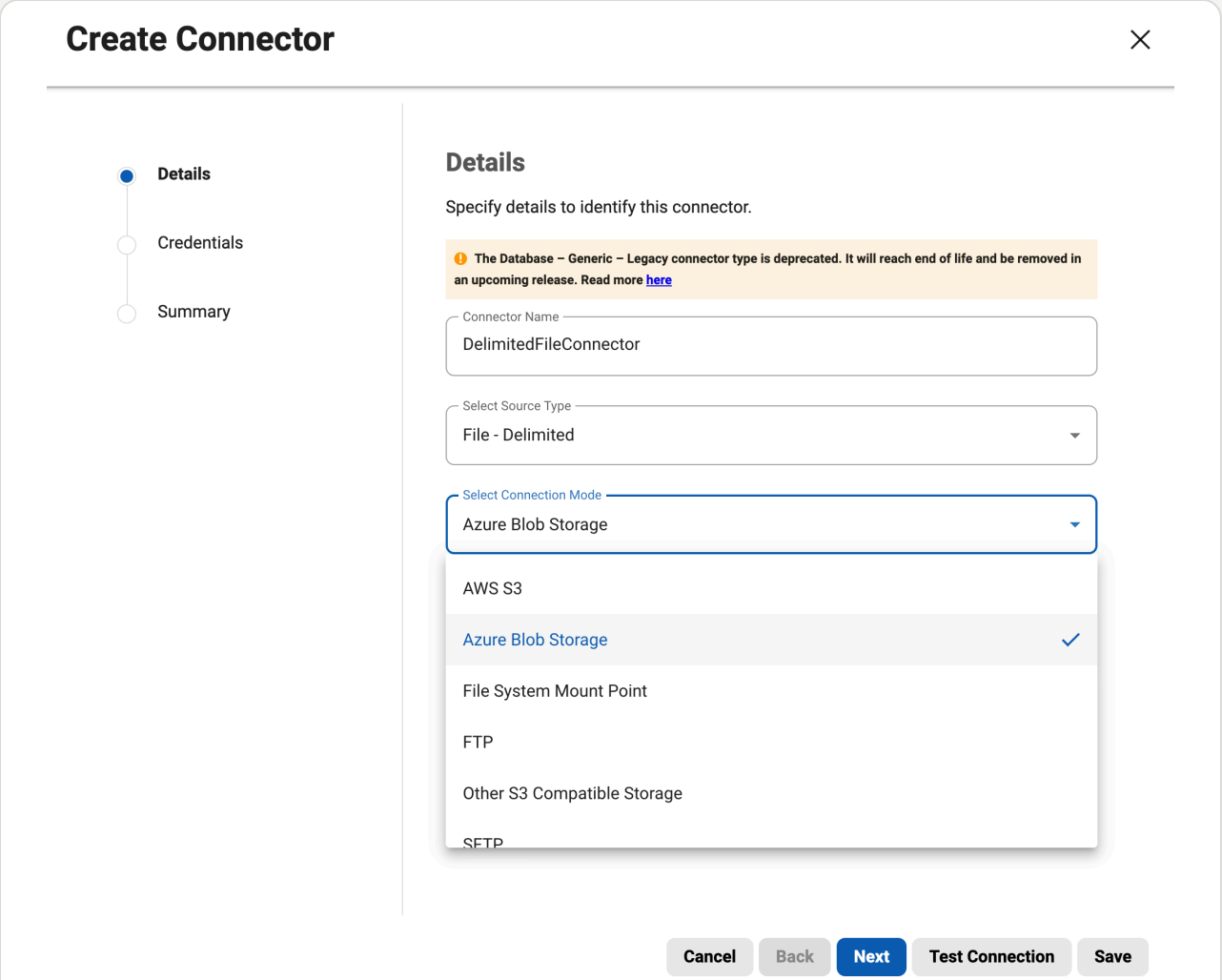
`Source page ↗`

8.18.5 The same thing through the API

Underneath, a connector is a small record. The documentation's sample script for a database connector posts `connectorName`, `databaseType` (for example `ORACLE`), `environmentId`, `host`, `port`, `schemaName`, `sid`, `username` and `password` to the engine's `/database-connectors` endpoint. Notice `environmentId`: a connector always belongs to one environment.

8.19 File and mainframe connectors

File connectors are created with the same wizard, but the credential step asks for different things. The Source Type is a file format (for example "File - Delimited"), and then you choose a **Connection Mode**. The list in the documentation is: Filesystem Mount Point, SFTP, FTP, FTPS (only for mainframe data sets), AWS S3, Other S3 Compatible Storage and Azure Blob Storage. The screenshot below shows that dropdown. Details of each mode live on separate pages (the S3, Azure Blob and remote mount pages). The file masking chapter picks this up.

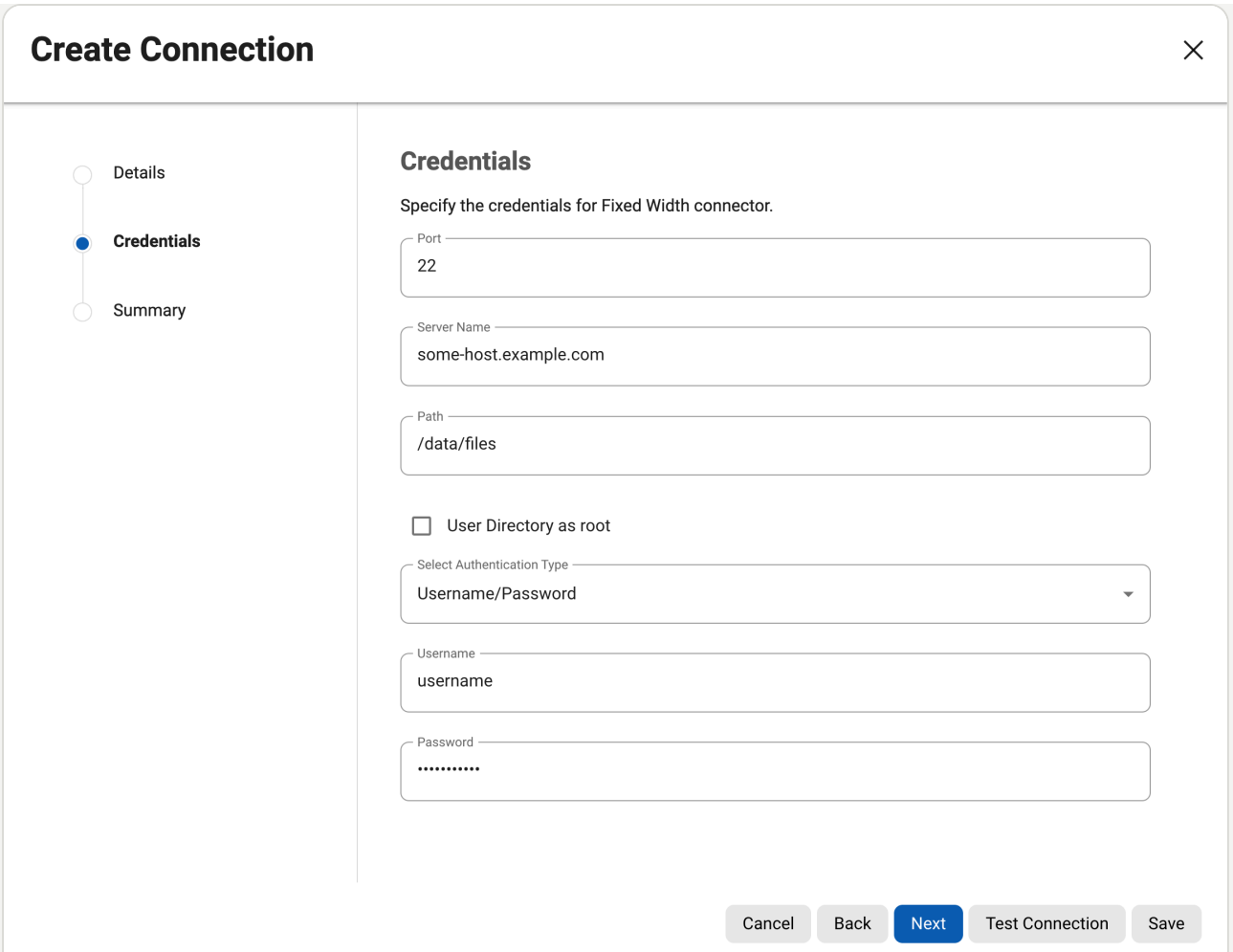


Official screenshot: The connection modes of a file connector. A banner in this picture says the "Database - Generic - Legacy" connector type is deprecated and will be removed in a later release, which is a good reminder that connector types change.

`Source page ↗`

For SFTP, FTP and FTPS the credential step asks for the following. The screenshots show the SFTP version.

- **Path**: the directory where the files are.
- **Server Name** and **Port**: where the file server listens (22 in the screenshot, the usual SFTP port).
- **User Name** and **Password** (not needed with public key authentication).
- **Public Key Authentication**: optional, SFTP only, chosen from the authentication-type dropdown; the page points to a knowledge-base article for the steps.
- **User Directory as root**: since version 6.0.9.0, tick this when the Path is relative to the user's home directory on the SFTP server. In the API this flag is `userDirIsRoot`.



Official screenshot: The SFTP credential step. The fields match the list above, plus the "User Directory as root" checkbox.

`Source page ↗`

ON-THE-FLY FILE MASKING NEEDS TWO CONNECTORS

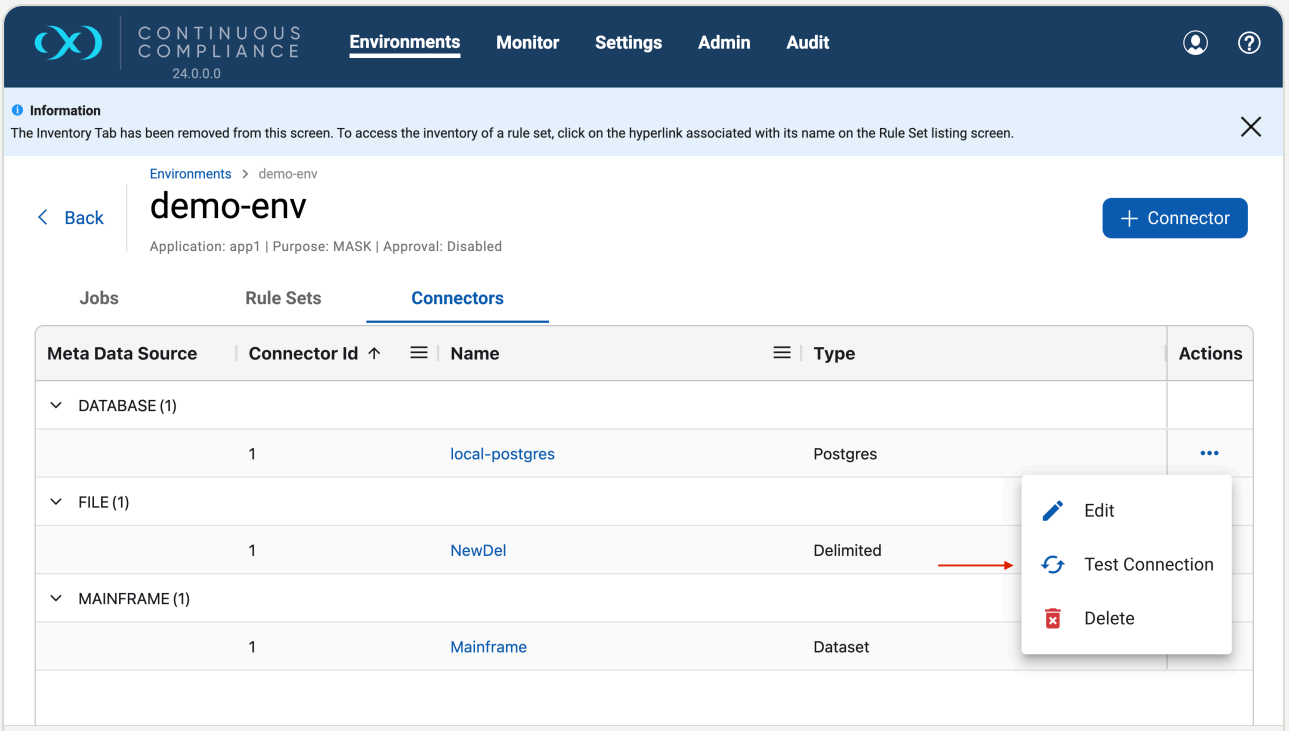
The connector page explains that for on-the-fly file masking you create a separate environment and connector for the source files. The masked files land in the directory the *target* connector points at, and the target rule set's file path must point to an existing file there: it does not have to be a copy, just an entry with the same name, which will be replaced. Mainframe data sets are covered by their own connector page: they must be EBCDIC encoded.

Two limits from the page worth knowing: FTP and FTPS are disabled in containerized deployments, and S3 role-based connectivity works only for engines hosted on AWS EC2. Supported file formats for the Files connector are Fixed Width, Delimited, XML, JSON and Parquet (ASCII or UTF-8; Parquet UTF-8).

8.20 Testing, editing and deleting a connector

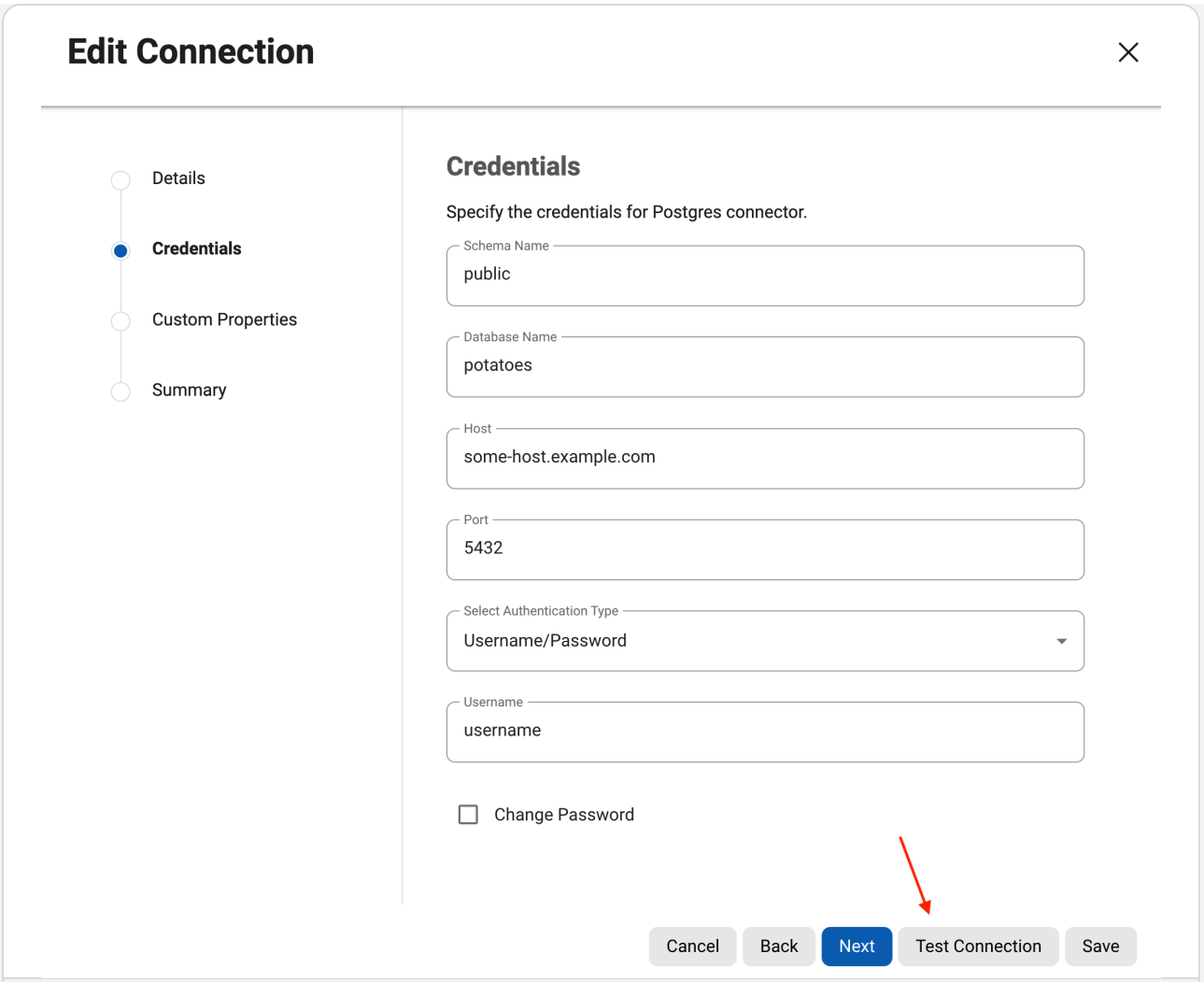
8.20.1 Testing

Test a connector as soon as you create it. You can do it in two places: with the **Test Connection** button inside the wizard at any step, and with **Test Connection** in the connector's Actions (...) menu on the grid. The engine takes the information on the screen, tries to open a database connection, and shows a success or failure message.



Official screenshot: Test from the grid: open the "..." menu on a connector row and choose Test Connection.

`Source page ↗`



Official screenshot: Test from inside the wizard: the Test Connection button is in the footer of every step.

`Source page ↗`

WHY A FAILED TEST IS USEFUL INFORMATION

A failure tells you which layer to blame. Wrong password, wrong schema or wrong database name usually return a login or "not found" message. A hang or timeout points at the network: a firewall, or a port that is not reachable from the engine (the overview chapter lists the ports). A "driver" message points at an unsupported version or a missing JDBC driver. The documentation says only that a message shows "success or failure"; the interpretations above are general guidance.

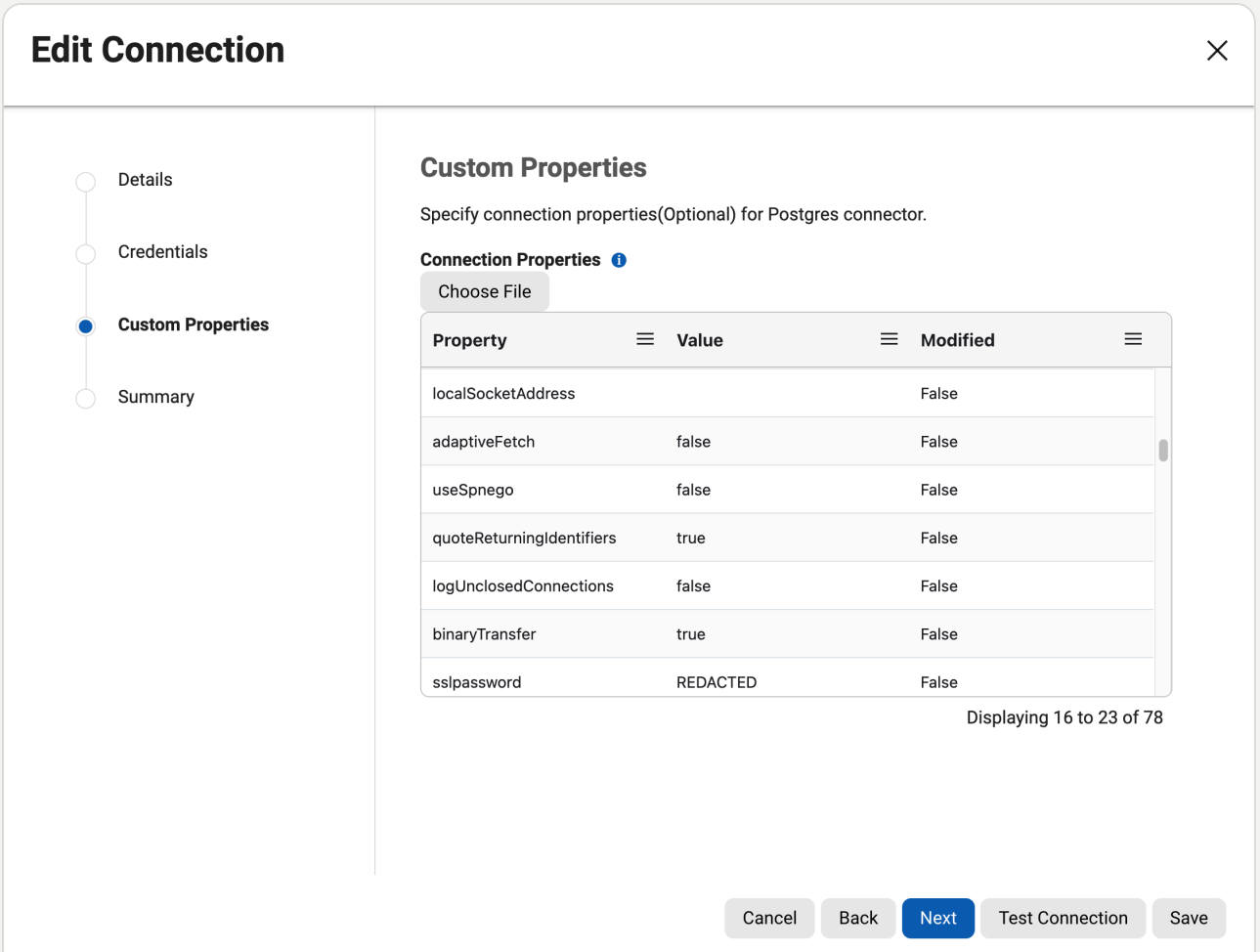
8.20.2 Editing

1. **Open the connector's edit wizard.**
   Click the connector name, or choose Edit in its Actions menu.

Expected result: The same wizard titled "Edit Connection", with values filled in.

2. **Change what you need across the steps.**
   To change the password, tick **Change Password** (the stored password is never shown). To change or replace properties, use **Choose File** on the Custom Properties step.

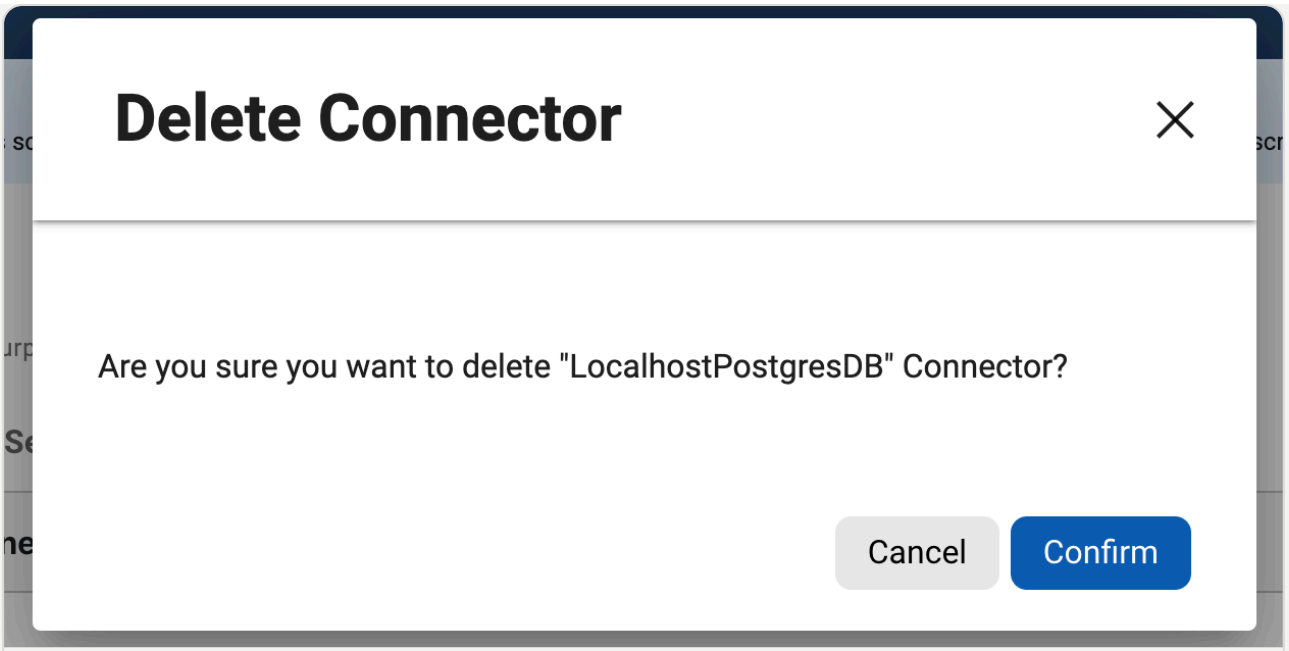
3. **Click Save**, or review the Summary and then Save.



Official screenshot: When a connector already exists, the Custom Properties table lists the default properties set by the driver. "Modified" tells you whether a property differs from the driver default because of an uploaded file. The password-like entry is shown as REDACTED.
`Source page ↗`

8.20.3 Deleting

Choose Delete in the Actions menu, then Confirm. Remember: deleting a connector also deletes the rule sets and inventory data that depend on it.



Official screenshot: The confirmation. Read the name in the question before clicking Confirm.
`Source page ↗`

8.21 Which data sources are supported, and extended connectors

The supported-connectors page splits what the engine can profile, mask and tokenize into two kinds. **Delphix connectors** are built in and come in three licensing flavours: *Standard* (built and supported by Delphix, included with the licence), *Select* (built and supported by Delphix but needing a separate licence) and *Premium* (built by a third party, often with a separate licence). **FEML** (File Extract, Mask and Load) is a technique for sources with no dedicated connector: extract data to files through the source's APIs, mask the files, load the masked data back.

Kind	Sources named on the page
Standard, databases	Db2 LUW, Oracle, MS SQL, MySQL, SAP ASE (Sybase), PostgreSQL, MariaDB
Standard, files	Fixed Width, Delimited, XML, JSON, Parquet
Select and Premium	Salesforce, CockroachDB, SAP HANA 2.0; mainframe and midrange: Db2 z/OS, Db2 iSeries, mainframe data sets

Two rules from that page apply. Database masking applies only to **physical tables**: views, materialized views, federated systems and proxy objects are not supported. And JSON or XML stored inside database columns can be masked with a separate "Document Store Type" feature. Pricing is usage-based, per terabyte managed. The per-database pages give versions and feature

tables (for example Oracle: TLS, password vault, Kerberos, in-place and on-the-fly options and "identity column support" all available), but the version matrix page was unavailable, so check the live Data Source Support page for your version.

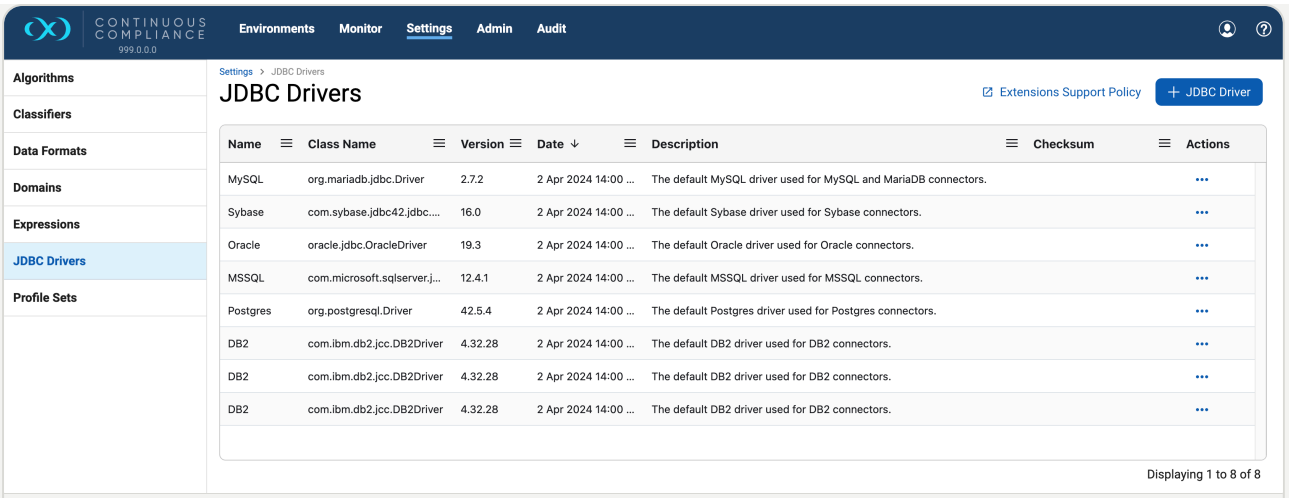
8.21.1 Extended connectors: bring your own JDBC driver

If your database is not natively supported, an **extended connector** lets you upload a JDBC driver and use it. The rules, from the extended-connectors page: the driver must be a *type 4* (pure Java) `.jar`, packaged inside a `.zip` even if it is a single file; anything needing compilation or an install script on the engine is not supported; drivers needing a Java version above 8 are listed as unsupported (a line that looks out of date, since release 2026.5 runs on Java 17); and extended connectors lack some native features, though driver support plugins can add Disable Constraints, Disable Triggers and Drop Indexes. Delphix supports them under its support policy.

1. **Get the driver from the database vendor's official site** and, where a checksum is published, verify it.
   The page warns that uploaded drivers run inside the engine's JVM, so a tampered driver could run arbitrary code on the engine host.

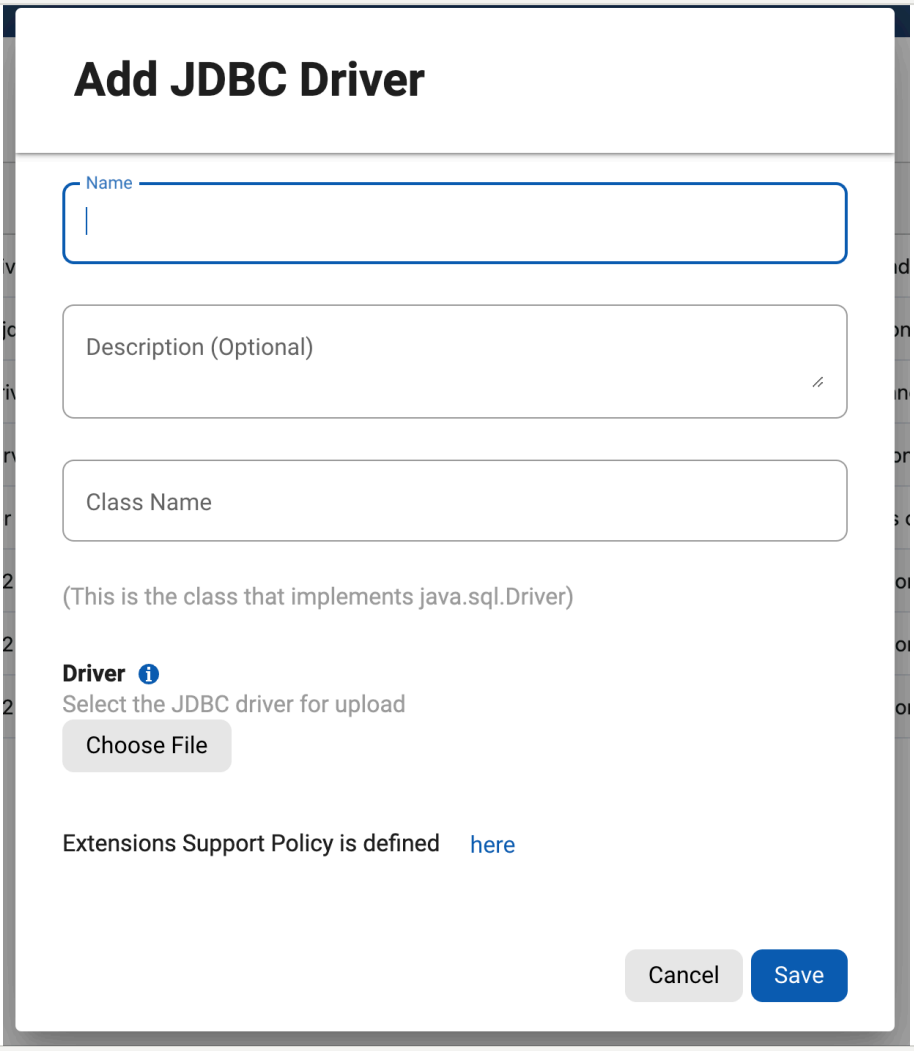
2. **Zip the driver files together** and open **Settings > JDBC Drivers**, then click **+ JDBC Driver**.
   Expected result: The Add JDBC Driver dialog: Name, optional Description, Class Name (the class that implements `java.sql.Driver`) and a file chooser.

3. **Create a connector** with Source Type *Database - Extended*.
   Expected result: The Credentials step asks for the JDBC driver, a Schema Name, a JDBC URL (avoid putting the username and password in it, because URLs may be logged), an Enable Logger checkbox (slows things down; for debugging only), Username and Password.



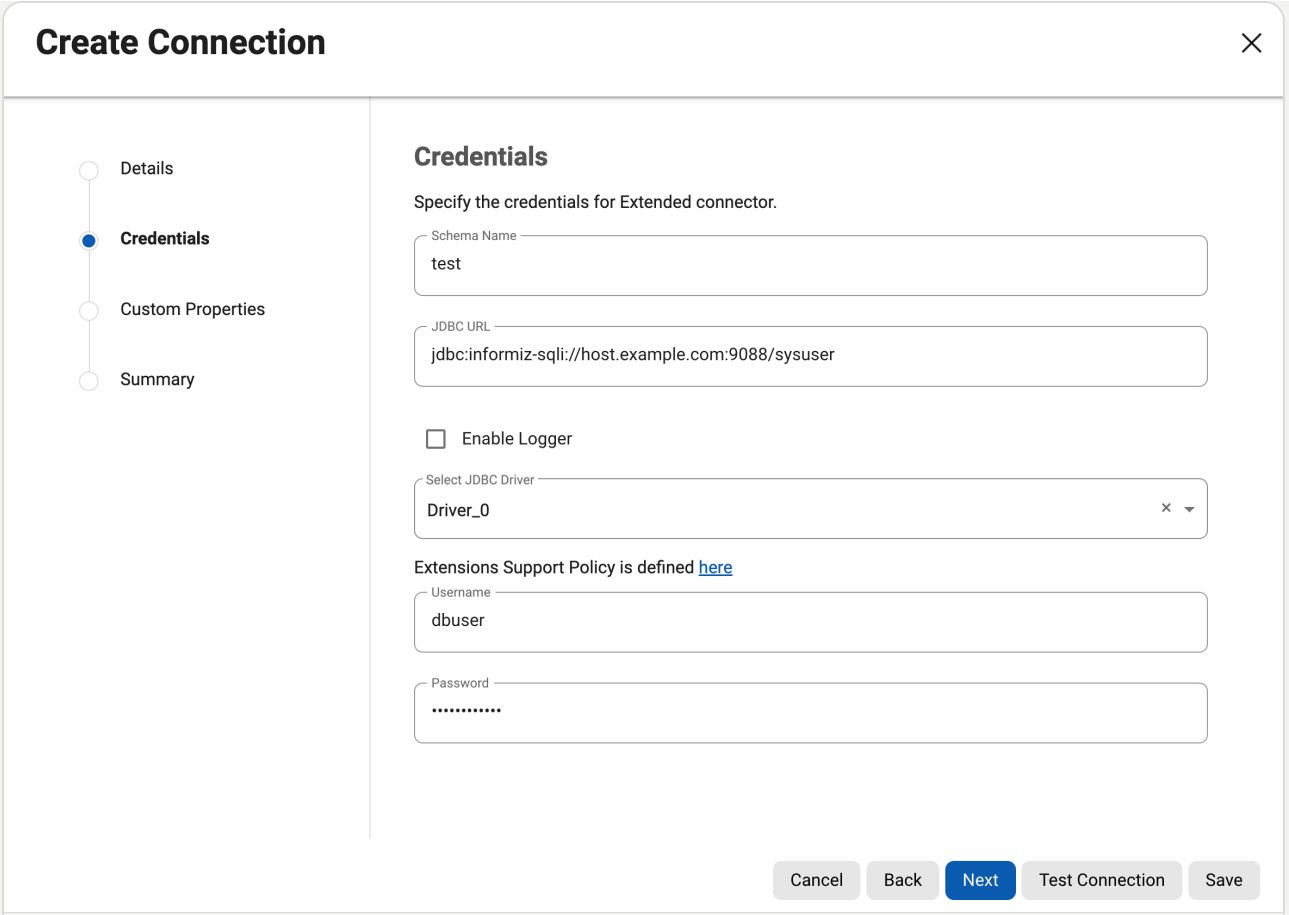
Official screenshot: Settings > JDBC Drivers. Each native database already has a default driver listed here with its version, and your uploaded drivers appear in the same list.

`Source page ↗`



Official screenshot: The upload dialog. The grey hint under Class Name says it is the class that implements java.sql.Driver.

`Source page ↗`



Official screenshot: Credentials for an extended connector. The extra field compared with a native connector is Select JDBC Driver.

`Source page ↗`

8.22 Database permissions and preparing the database

8.22.1 A dedicated masking user

The permissions page recommends creating a *separate* database user for the engine, in non-production, rather than reusing a person's account. Reasons given: it is easy to replicate with its privileges, it can be a central directory user used in many places, and access audits on the database host become simpler. For **masking** jobs that user needs to read, write and update table data; create, drop, disable and enable indexes, triggers and constraints; and add or delete columns (the engine may add a temporary column, as you will see in the keys chapter). For **profiling** jobs it needs only to see object definitions (the schema) and read table data.

8.22.2 Preparing each database

The "Preparing data" pages say, first of all, that using a VDB is recommended: it is also the best security practice because the database runs in a sandbox, and the page adds that Selective Data Distribution is advised for further security. The per-database pages share one logic. In-place

masking changes *every row of every masked table*, which creates huge amounts of log and undo data, so the database should be tuned for a heavy write load, and log files can hold unmasked copies of the data and must be cleared afterwards.

Database	What the page asks for (overview)
Oracle	Switch off archive logging for the copy (`NOARCHIVELOG`), enough memory (automatic SGA management), larger undo space and retention for big in-place updates (about 2 to 3 times the largest table) to avoid "snapshot too old" (ORA-01555), well-sized redo logs, PCTFREE 40 to 50, and use ROWID as the logical key. Grant SELECT on several V$ views, and `SELECT_CATALOG_ROLE` if jobs drop or disable indexes, constraints or triggers.
SQL Server	Use the SIMPLE recovery model (transaction logs may contain unmasked data), tune memory, prefer a clustered primary key, and turn on the drop indexes, constraints and triggers options. Add the masking user to `db_datareader` , `db_datawriter` and `db_ddladmin` . Identity columns cannot be masked without special steps. When provisioning a masked VDB, this user must exist on the source the VDB is made from.
Db2	For z/OS, `DESCSTAT` must be YES or rule set creation fails. Size the journal (about 3 times table size for in-place, 1.5 times for on-the-fly) and purge it after masking, because it holds unmasked data.
Sybase (SAP ASE)	Size memory and the transaction log (larger than all masked tables combined; a full log makes the job appear to hang). Create a masking user with grants such as SELECT, INSERT, UPDATE, DELETE and ALTER on the tables.

All four pages also explain how the engine finds a row to update: with a primary key, logical key or identity column. That topic is the heart of the next chapter.

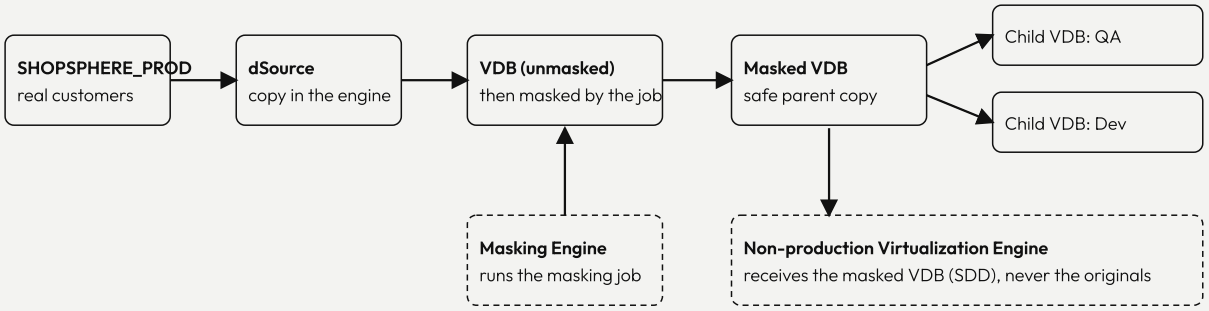
> **NEVER PREPARE PRODUCTION LIKE THIS**
> Switching off archive logging, dropping constraints or giving a user schema-altering rights are fine on a disposable copy and unacceptable on production. That is one more reason the docs steer you to a VDB or another non-production copy.

8.23 What a VDB is, from zero

Suppose ShopSphere's production database is 500 GB. Testers need a copy, and so do developers, and the compliance team will not allow the unmasked copy anywhere near them. Making four physical 500 GB copies takes hours and terabytes, and refreshing them is a chore. Delphix's **virtualization** product solves the copying half of the problem. Its pages describe it this way.

- The Continuous Data engine compresses the data it gathers, often to a third of the original size or better, and from that creates lightweight **virtual copies** that are fully readable and writable, independent of one another and created or removed in minutes. The introduction claims ten virtual copies fit in the space of one physical copy.
- A **dSource** is the engine's copy of a source database's persistent data layer, from which VDBs are made and refreshed. A **snapshot** is the state of a dataset at one moment, and a **TimeFlow** is the timeline of those states.
- A **VDB (virtual database)** is a full read-and-write copy of the source data, provisioned from a dSource or from another VDB, and managed by the Continuous Data engine. You can refresh it (pull newer data), rewind it (go back in time), disable it and enable it.

Analogy, with its limit: think of a VDB as your own editable version of a shared master file. Everyone edits their own version, and the big file is stored once. The documentation does not explain the storage mechanism, so do not push the analogy further than "cheap, fast, independent copies". To the application a VDB looks like an ordinary database with a host, a port and a name, which is what makes the connector idea below work.



Original diagram of the masked-VDB flow described on the provision-masked-VDBs page. Masking happens during provisioning.

WHY MASKING A VDB BEATS MASKING A COPY BY HAND

An in-place job overwrites its source (see the overview chapter). On a VDB that is safe: production is untouched, the VDB can be thrown away, and a refresh brings fresh data and re-runs the masking job. Child VDBs of a masked VDB contain only masked data, so you can hand out many independent masked copies cheaply.

8.23.1 How a masked VDB is provisioned

These steps happen mostly in the *Virtualization* engine's interface, not the masking engine's, so they are given at overview level. The masking engine's part is the masking job and its connector.

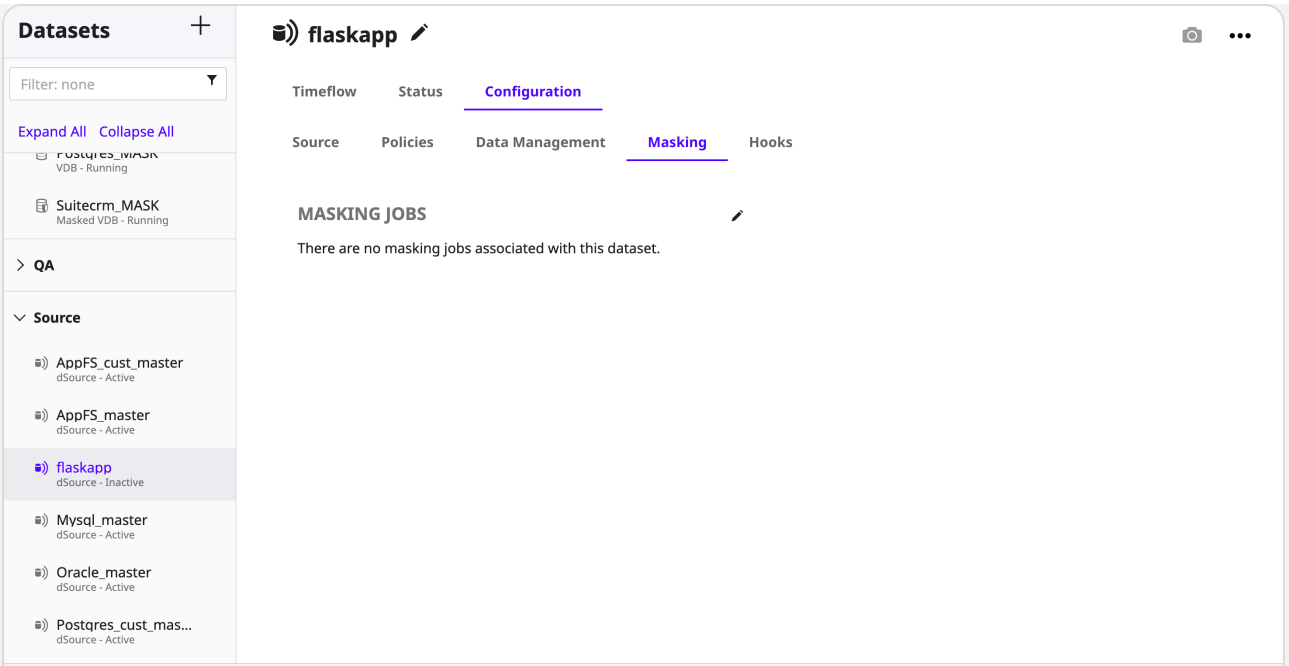
1. **Create a masking job on the masking engine** for the rule set, as usual, with in-place masking and the *Multi-Tenant* option ticked.
   The Multi-Tenant option lets one rule set serve identical schemas through different connectors chosen at run time, so the same job can be pointed at each new VDB. The job page says Selective Data Distribution must also be enabled for a job to be eligible for SDD.

2. **Associate the job with the dSource.** In the Virtualization Engine open the dSource, then Configuration > Masking, click the pencil, tick the job and confirm.
   This tells the engine the job is complete and applicable to that database. Jobs already associated with another dSource are not offered.

   Expected result: The dSource's Masking tab lists the job. Before you do this it says no masking jobs are associated.

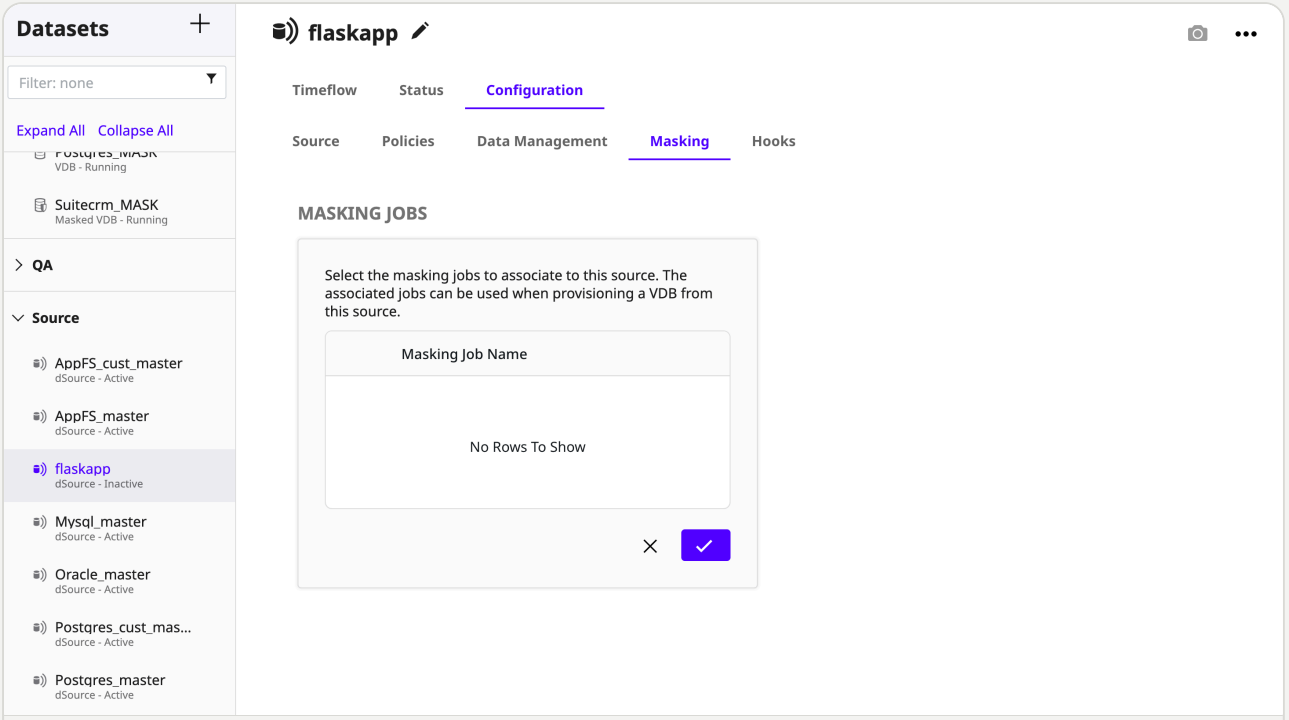
3. **Provision the VDB.** Select the dSource, open the TimeFlow tab, click Provision VDB, and step through the wizard (target environment, configuration, snapshot policy, *Masking*, hooks, summary).
   At the Masking step tick **Mask this VDB** and either pick an existing masking job (the engine fetches the jobs from the masking engine; only jobs associated with the parent dSource appear) or choose "Mask using script(s)". Choose snapshot policy **None**: the page says a policy together with SDD can leak sensitive data.

   Expected result: Submitting starts provisioning, and the Actions list shows masking running. Follow it on the masking engine under Monitor > Job Executions.



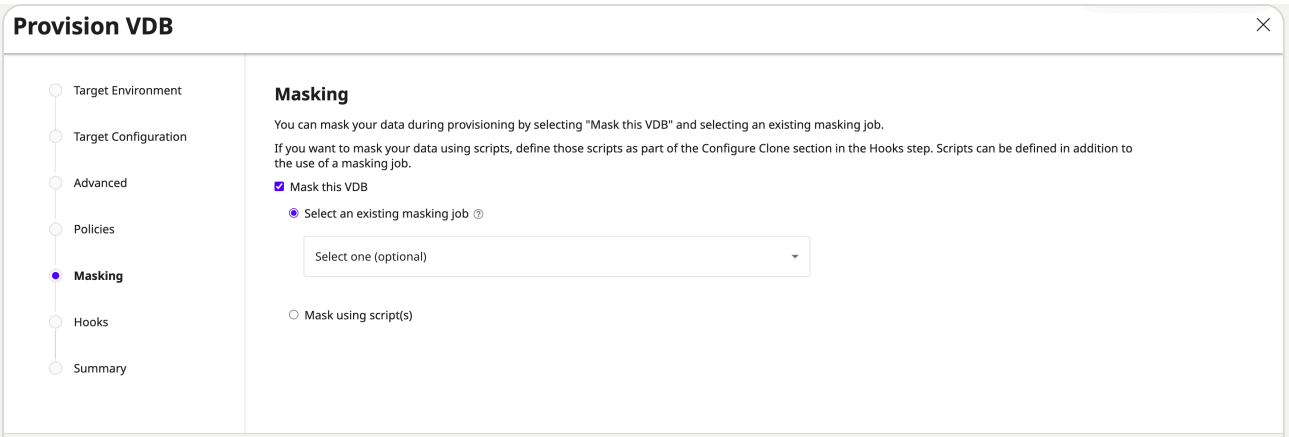
Official screenshot: Virtualization Engine, dSource > Configuration > Masking before any job is associated. The pencil icon at the right of MASKING JOBS starts the association.

`Source page ↗`



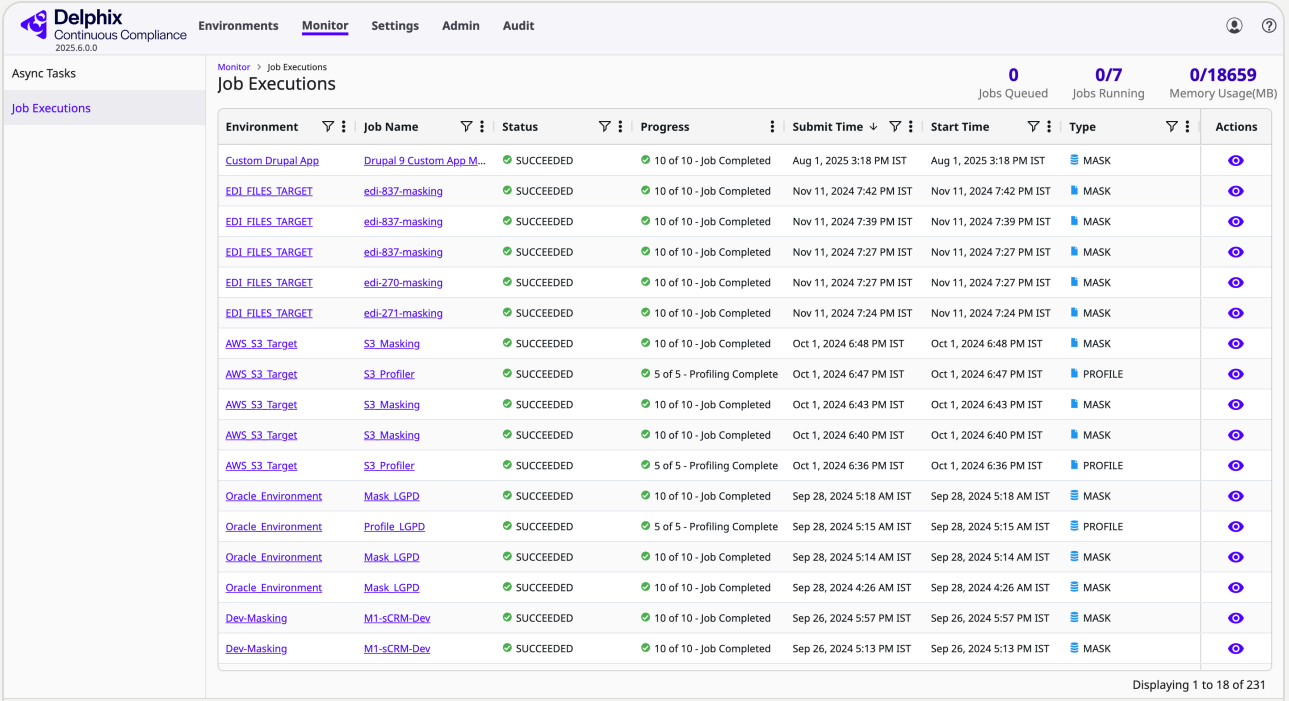
Official screenshot: The association panel. In this screenshot the table is empty because no masking jobs were listed for this dSource. A job appears here only after it has been created on the masking engine.

`Source page ↗`



Official screenshot: The Masking step of the provisioning wizard. The wizard's other steps run down the left: Target Environment, Target Configuration, Advanced, Policies, Masking, Hooks, Summary.

`Source page ↗`



Official screenshot: The masking engine's Job Executions page, where you watch the masking job that provisioning started. The page text says to monitor progress here.

`Source page ↗`

8.23.2 Rules and restrictions you must know

- Provisioning or refreshing **masked VDBs is supported only for Oracle, MS SQL Server and Sybase**; Db2, for example, is not supported.
- **One job per VDB.** A masking job cannot be assigned to several VDBs at once; if you reuse a rule set on many VDBs, create a unique job for each. **Only in-place jobs** can be selected.
- Masking happens **during provisioning**. A VDB that was provisioned without a masking job can never be masked later, and a masked VDB cannot be unmasked, changed to another job, or

given extra masking jobs. You cannot make an unmasked snapshot of it.

- The connector used for masked provisioning must be a **"basic" connector** (host and port), unless you mask an Oracle pluggable database, which needs an "advanced" connector.
- Masking and Virtualization must **never run in one virtual machine**, and engine versions must be compatible (a compatibility matrix is on the page: for example 5.3.0.0 and later pairs with 5.3.0.0 and later, and minor versions need not match).
- Refreshing a masked VDB (Refresh VDB, choose a snapshot or point in time) re-runs the masking job.

To let the Virtualization Engine reach the masking engine, an administrator can change the host, port and credentials it uses through the engine's CLI (`maskingjob` , then `serviceconfig` , then `update` , set values and commit). It lists jobs automatically at each login, and `maskingjob fetch` refreshes the list on demand. The page notes that Delphix does not check network availability between the two engines.

8.24 ”Create a connector to a VDB from the masking engine”

"Create a connector to a VDB from the masking engine" can mean two different things, and the documents describe only one of them in detail.

	A. Your own connector pointing at a VDB	B. Masked provisioning
Who starts it	You, on the masking engine, with the normal + Connection wizard.	The Virtualization Engine, while provisioning.
What the connector is	An ordinary database connector whose host, port, database and schema are those of the VDB.	A "basic" connector chosen at job run time (multi-tenant job).
Documented as	The generic connector steps, plus the advice to mask a VDB.	The masked provisioning pages.

The reasoning for A: a VDB is a normal, running database. The "Preparing data" page recommends using a vDB on the Virtualization Engine; the SQL Server page says the masking user must exist on the source from which the VDB is created. So "a connector to a VDB" is simply the connector from the wizard section above, filled in with the VDB's address. The documents contain **no page titled "connector to a VDB"** and do not say where to read a VDB's host, port or database name in the Virtualization Engine's interface. The lab flags that gap.

Lab: create a connector to the ShopSphere VDB

Goal: have a tested connector named `SHOPSPHERE_VDB_CONN` in the environment `SHOPSPHERE_DEV`. Assumptions: a VDB of the ShopSphere database already exists (someone provisioned it; the documents do not cover creating a plain VDB), it runs on PostgreSQL or another supported type, and you have the masking engine open. Labels: = documented step; = the documents do not say; general advice is given.

1. **Collect the VDB's connection details.**
   You need host or IP, port, database name and schema. Ask the person who runs the Virtualization Engine; the documents do not show where the Virtualization Engine displays them.
   Expected result: Four values written down, for example host `vdb-host.example.com` (example).

2. **Make sure a masking user exists on the VDB's database** with the permissions described in the permissions section.
   Use a dedicated user, not a person's account. On SQL Server the page says it must exist on the source from which the VDB is created.

3. **Create the application and environment** (see the environments section) if they do not exist: `SHOPSPHERE_APP` and `SHOPSPHERE_DEV`, purpose Mask.
   Expected result: The environment opens with tabs Jobs, Rule Sets, Connectors.

4. **Open the Connectors tab and click + Connection.** Enter the Connector Name and choose the Source Type (for example Database - Postgres), then Next.
   If the type offers a connection method, choose Basic (hostname/IP): masked provisioning requires basic unless it is an Oracle PDB.

5. **Fill in Credentials:** Schema Name, Database Name (case-sensitive), Host, Port, authentication type Username/Password, the masking user and its password.
   Expected result: All fields filled; no password shown in clear.

6. **Click Test Connection.**
   This proves the engine can reach the VDB and log in before you build anything on it.

Expected result: A success message. On failure, re-check the database name's case and then the network path to the VDB's port.

7 **Skip Custom Properties, check the Summary and Save.**
Expected result: The connector appears under DATABASE in the Connectors grid, without a warning triangle (the triangle means missing authentication).

▶ **Show a worked answer**

8.25 Common mistakes and gotchas

CONNECTOR AND VDB PITFALLS

- **Wrong case in Database Name.** It must match the database exactly.
- **Testing the wrong thing.** A passing Test Connection proves login, not that the user may drop indexes or alter tables; a job can still fail later for missing privileges.
- **Passwords in a properties file or JDBC URL.** Both can be seen by other users; use the form field.
- **Deleting a connector to "tidy up".** Its rule sets and inventory go with it.
- **Masking the real database.** Always mask a VDB or other copy, with an in-place job.
- **Trying to mask an existing VDB later.** Masking must be part of provisioning; otherwise you cannot mask that VDB afterwards.
- **Imported connectors without passwords.** Import and sync do not carry passwords; re-enter them.

8.26 Recap

RECAP

- Applications label systems; environments hold connectors, rule sets and jobs. Purpose is fixed; deleting an environment deletes its contents but not shared settings.
- A connector (database, file or mainframe) is created in four steps (Details, Credentials, Custom Properties, Summary), can be tested at any time and, when deleted, takes its rule sets with it.
- Form fields override the JDBC URL and properties file for username, password and schema; secrets in properties files are visible to others.
- Extended connectors add your own type 4 JDBC driver, uploaded as a zip under Settings > JDBC Drivers.
- Use a dedicated masking user, and prepare a non-production copy: tune logs and undo, and clear logs that may hold unmasked data.
- A VDB is a cheap, writable, independent copy provisioned by the Virtualization Engine. Masked VDBs are masked during provisioning, with a unique in-place multi-tenant job per VDB, for Oracle, SQL Server and Sybase.
- "A connector to a VDB" is an ordinary connector with the VDB's address. The docs do not give a dedicated procedure.

Delphix: rule sets, primary keys and data integrity

You now have an environment and a connector (the connectors and VDB chapter). The next object in the chain is the **rule set**: the list of tables, or files, that the engine will profile and mask. Building one is mostly point and click. The part that needs real understanding is *keys*. Delphix, like most masking engines, wants to know for every table which column or columns identify one row, and it expects masked values in related tables to stay matched. This section covers both.

Keys are covered in general in the data basics chapter; here they are explained briefly, with the masking twist. Tool-specific facts are from the Delphix Continuous Compliance documentation, version 2026.5.

DOCUMENTATION GAPS

The introduction says the engine scans metadata and data "to identify and preserve" primary/foreign key relationships, but the rule set pages describe no automatic foreign key handling, only the rule that the same deterministic algorithm must be used on both sides. The order in which the engine picks a key also differs between the database pages.

8.27 Words to know first

Rule set

A group of tables or flat files, within one data source, on which you may run profile, masking or tokenization jobs.

Inventory

The detailed list of every column or field in a rule set with its domain and algorithm. You open it by clicking the rule set's name.

Primary key (PK)

A column (or columns) that uniquely identifies each row of a table and is never empty.

Foreign key (FK)

A column in one table that holds the primary key value of a row in another table, linking them.

Referential integrity

The promise that every foreign key value matches a real row in the table it points to.

Logical key

A unique, non-null column (or comma-separated columns) *you tell Delphix to treat as the key*. It changes nothing in the database; it is only information for Delphix.

Identity column

A database column that numbers rows automatically. Delphix can use an existing one as the key, or create a temporary one when a table has no key.

ROWID

In Oracle, a built-in pseudocolumn holding each row's address. The Oracle preparation page recommends it as the logical key for speed.

Deterministic (consistent) algorithm

One that always turns the same input into the same output (for a given algorithm key). This is what keeps matching values matching.

Checklist

An action on a rule set (and on a job) that checks every table has a primary key, an indexed identity column or a logical key.

8.28 Primary and foreign keys from zero, with ShopSphere

A database table is like a spreadsheet, but databases promise things a spreadsheet does not. One promise is that each row can be found unambiguously. Two customers may share a name, so the name cannot find a row. A *primary key* is the column designed to do it. In ShopSphere, `CUSTOMERS.customer_id` is the primary key: 1001 is Asha Menon, 1002 is Ravi Kumar, 1003 is Meera Nair, and there will never be two 1001s or an empty one.

Tables connect through *foreign keys*. `ORDERS` has a column `customer_id` that stores which customer placed the order. Each value there must exist in `CUSTOMERS.customer_id`.

```
ORIGINAL DATA (FAKE). CUSTOMERS.CUSTOMER_ID IS THE PRIMARY KEY;
ORDERS.CUSTOMER_ID IS A FOREIGN KEY TO IT.
```

CUSTOMERS.customer_id	first_name	last_name	ssn
1001	Asha	Menon	123-45-6789
1002	Ravi	Kumar	234-56-7890
1003	Meera	Nair	345-67-8901

ORDERS (FAKE). WHICH CUSTOMER MADE EACH ORDER?

order_id (PK)	customer_id (FK)	order_date	total_amount
5001	1001	2026-03-02	1499.00
5002	1001	2026-03-19	299.00
5003	1003	2026-04-07	849.50

The query "show every order with its customer's name" joins the two tables on `customer_id`. That join is what the application does all day. It is also what a careless masking job can break, as the integrity section below shows.

> **"PRIMARY KEY DISCOVERY" IN DELPHIX**
>
> The documentation has no separate feature called primary key discovery. What it does show: the database inventory marks each column that is a Primary Key, Foreign Key or Index (written beneath the column name, as in the screenshot later in this section), and the Checklist reports tables that lack a usable key. So "discovery" here means *reading the key information from the database and checking it*.

8.29 Why a masking engine needs a key on every table

In-place masking works in three moves: read a row, compute the masked values, write them back *to that same row*. To write to the right row, the engine must be able to name it. The preparation pages for SQL Server, Db2 and Sybase say so plainly: a key is a unique, non-null value that identifies a row, and for in-place masking "this key looks up the row to update the masked record". For speed the key should be indexed (a clustered index is best, the pages say).

Analogy: a librarian who must correct the address on 100,000 library cards. If each card has a printed number, she can pull card 48213 directly. If cards have no numbers she must compare every card to find the right one, and with duplicate names she might correct the wrong person. The key is the card number. The analogy stops at speed: the engine updates in batches, not card by card.

The pages then list what the engine does with a table, in this order of preference.

Situation	What the engine uses
You defined a **logical key**	That key. It "can override the key used". One or several columns, but always unique and never null.
The table has a **primary key**	The primary key.
No primary key, but an **identity column**	The identity column. If it is not indexed, the engine adds an index (named with `i_MASKING_GENERATED_IDENTITY_TMP` on SQL Server).
No key at all	The engine adds a temporary identity column called `MASKING_GENERATED_IDENTITY_TMP` and removes it when the job ends. This needs a database user allowed to alter the table, and slows the job. If identity support is unavailable the job may fail.

A detail the pages disagree on: the SQL Server page lists the order as logical key, primary key, identity column, none. The Db2 and Sybase pages list primary key, identity, logical key, none, but still describe the logical key as the override. The practical rule is the same: **define a logical key when you want to control the choice**. The documentation recommends on-the-fly masking as the alternative when a table has no key, since that method writes new rows rather than updating old ones.

> **WHY THE TEMPORARY COLUMN IS NOT A FREE PASS**
>
> The 2025.5 and 2025.6 release notes list known issues: failed jobs on heap tables (tables without a clustered index) may leave the generated temporary identity column and index behind. So "the engine will add one for me" is a fallback with a cost. Give the table a real key.

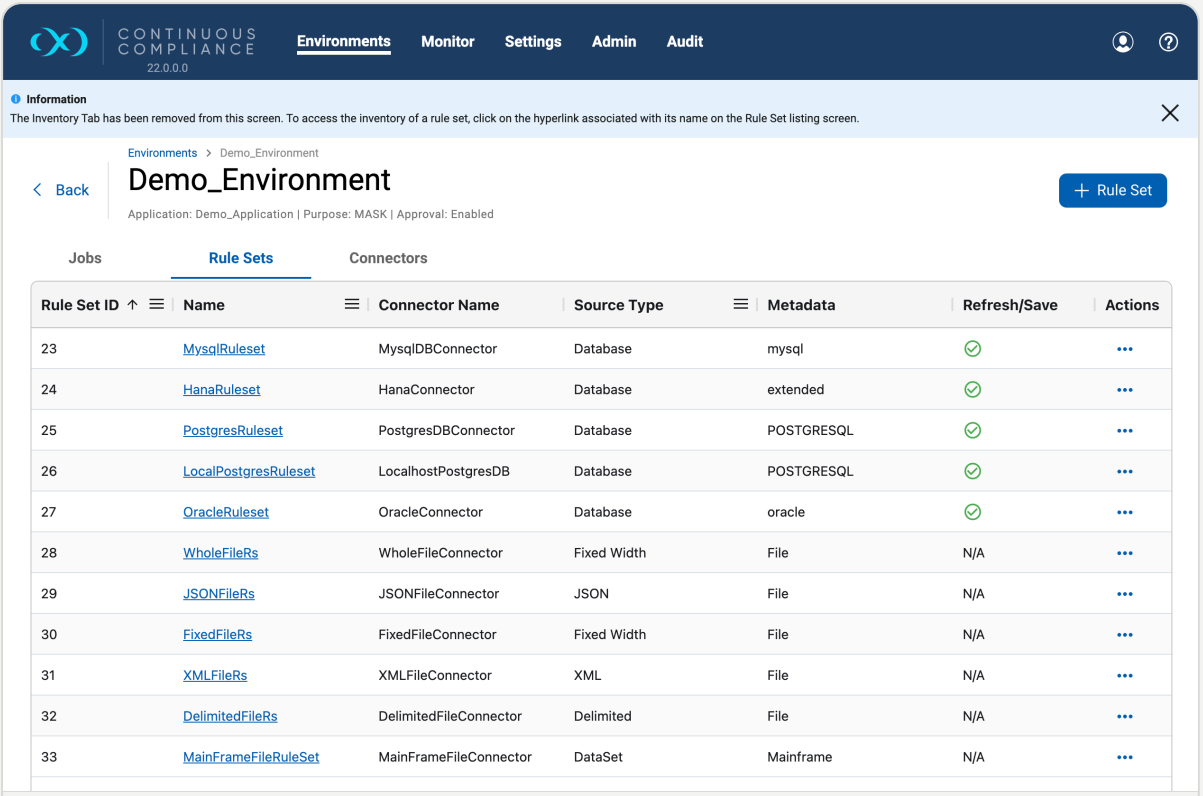
8.29.1 Oracle and ROWID

Oracle tables have the built-in `ROWID`. The Oracle preparation page says that adding ROWID as the logical key on each table through the GUI is generally the fastest approach, because Oracle manages the updates by the row address. A fixed issue in the 2025.2 release notes confirms that ROWID could not be added as a logical key from the Edit Ruleset screen before that, which is one reason to check your version.

8.30 Rule sets: what they are and the four types

A rule set answers "which tables or files, of this connector, are in scope?". Profiling and masking jobs run *only* on what is in the rule set. A table added to the database later is not discovered or masked until you add it. Each rule set belongs to one connector, and the inventory (the per-column plan) hangs off the rule set.

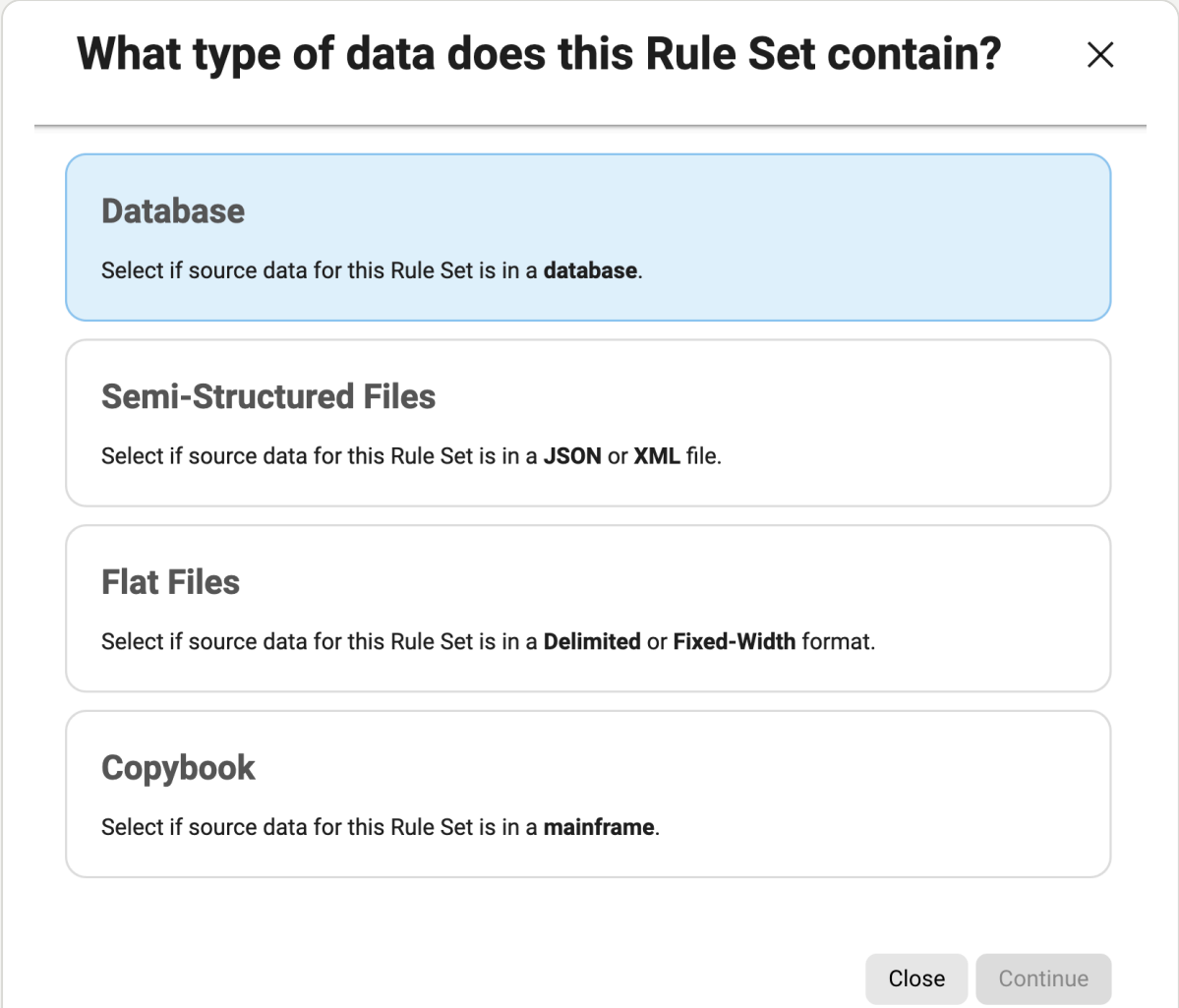
Open an environment and click the **Rule Sets** tab. The grid shows: Rule Set ID (for the API), Name, Connector Name, Source Type (Database, Fixed Width, Delimited, JSON, XML or DataSet), Metadata (the database or file type), a Refresh/Save status icon (database rule sets only; file rule sets show N/A) and Actions. You need View permission on Inventory and File Format to open a rule set's inventory.



Official screenshot: The Rule Sets tab. Database rule sets show a green tick in Refresh/Save; file rule sets show N/A because refresh does not apply to them. The "+ Rule Set" button sits at the top right.

`Source page ↗`

Click **+ Rule Set** and the engine first asks what kind of data the rule set holds. There are four choices.



Official screenshot: The four rule set types. Each card says what it is for. Database is highlighted in this picture.

`Source page ↗`

Type	For	Taught in
Database	Tables in a relational database. Has keys, filters and custom SQL.	This section.
Semi-Structured Files	JSON or XML files.	File masking chapter
Flat Files	Delimited or Fixed-Width files (for example .txt and .dat).	File masking chapter
Copybook	Mainframe data sets described by a copybook.	Not covered here.

The connector dropdown in the next step shows only connectors that match the type you chose here. If you have not created a connector yet, the dropdown is empty and you cannot continue.

8.31 Creating a database rule set

The Add Rule Set wizard has three steps: Details, Data Tables, Summary.

1. **Details: name the rule set and pick the connector.**
   The name follows the usual naming rules (no special characters). The connector dropdown can be filtered by typing. When you choose one, the screen shows its details (database type, database name, host, port, schema, user), so you can confirm you picked the right database before going on.

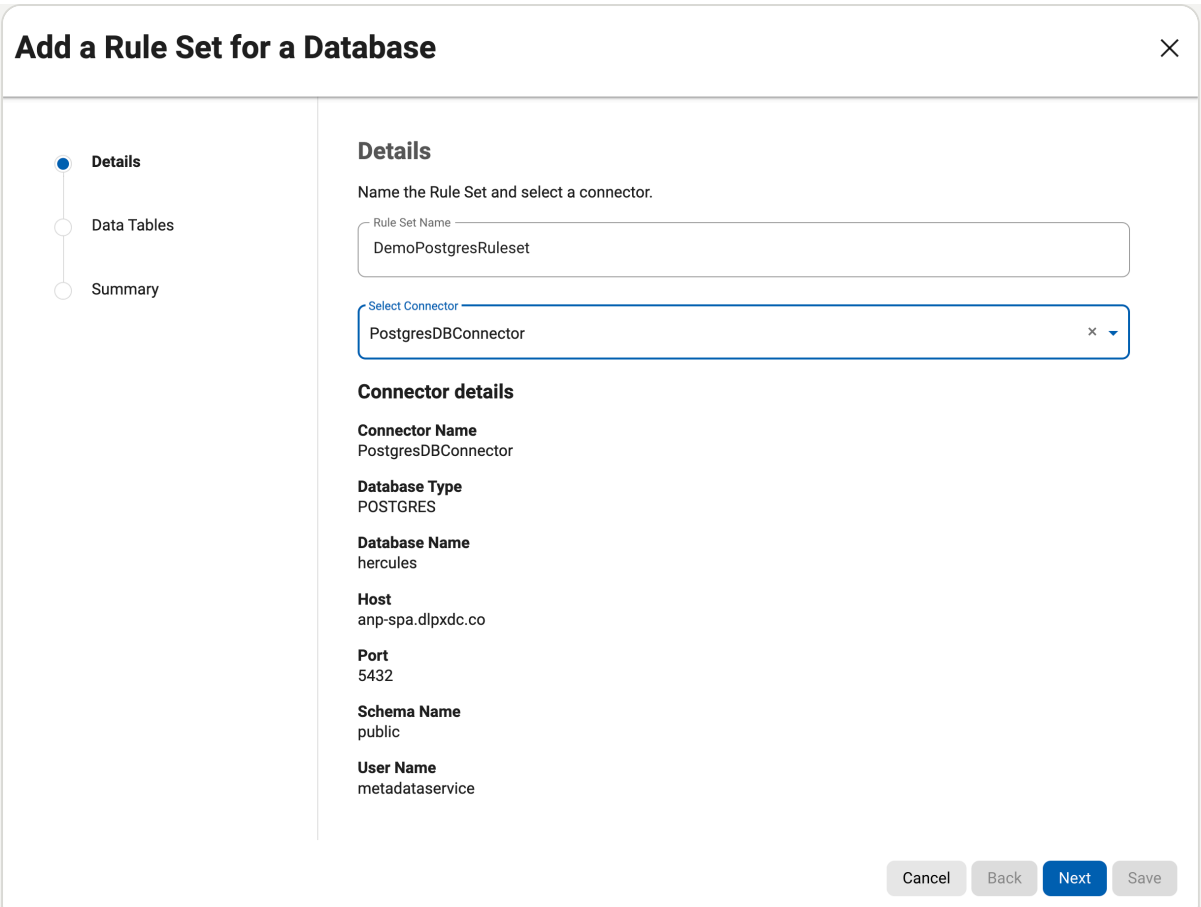
   Expected result: Connector details appear under the dropdown. Next is enabled.

2. **Data Tables: add the tables.**
   Click **Add Tables**. A dialog lists the tables the connector can see (tables only, not views, indexes or stored procedures). Filter by substring or starts-with, tick the tables you want, or tick the header box for all, then Save.

   Expected result: The selected tables appear in the grid with empty Logical Key, Filter and Custom SQL columns.

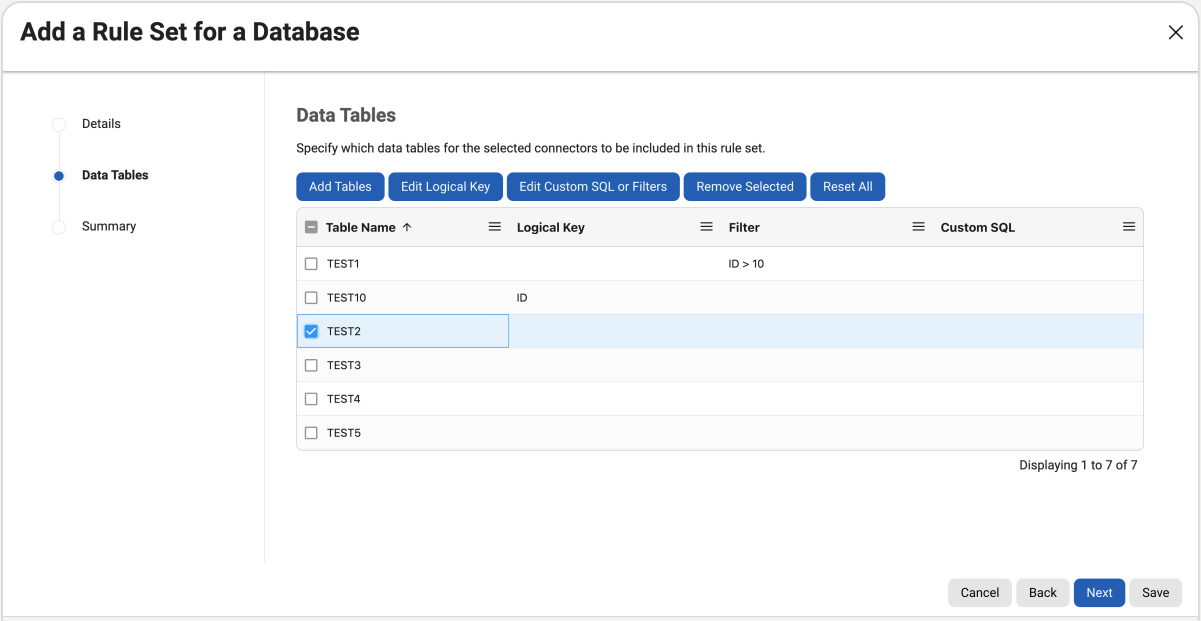
3. **Summary: review and Save.**
   Check the configuration; Back fixes errors. Save stays greyed out if the rule set has no tables. Changing the connector resets everything you added.

   Expected result: A bottom-sheet message with an Async Task ID. Follow it under Monitor > Async Tasks. When it finishes, the rule set shows in the grid.



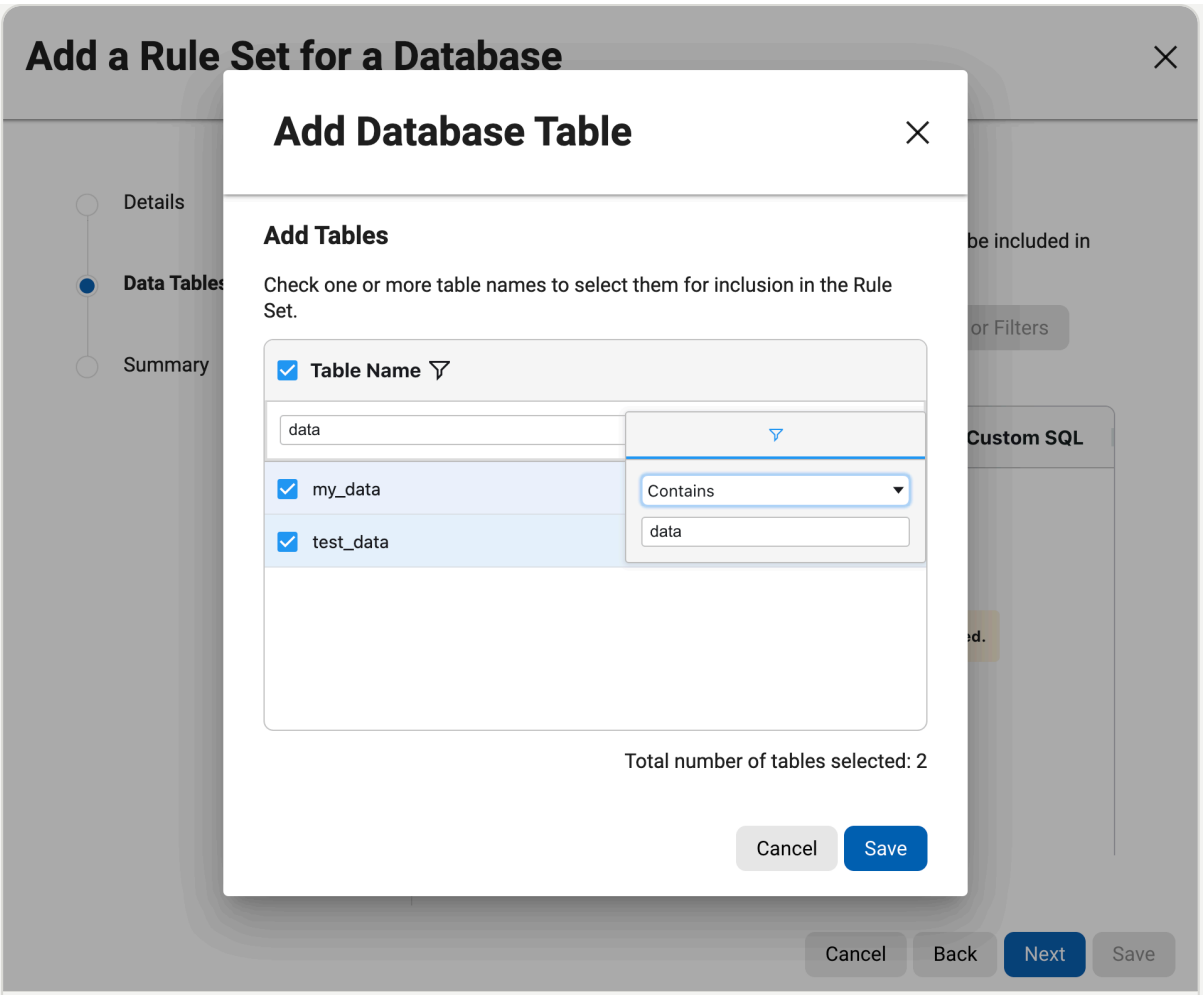
Official screenshot: Step 1. The Connector details panel below the dropdown is your last chance to catch a wrong connector.

`Source page ↗`



Official screenshot: Step 2. The grid is where keys, filters and custom SQL are set. TEST10 shows the logical key ID and TEST1 shows a filter on ID. The buttons above the grid act on the ticked rows.

`Source page ↗`



Official screenshot: The Add Tables dialog with a "contains" filter: typing "data" leaves two tables and the header box ticks them both. The footer shows how many tables are selected.

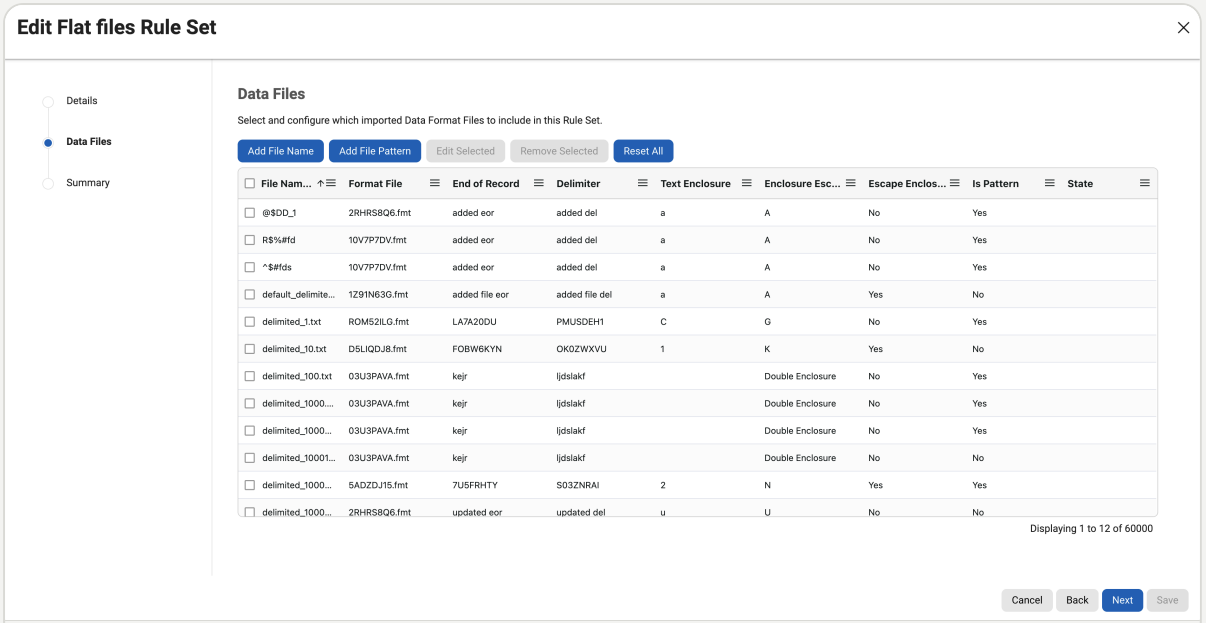
`Source page ↗`

> **ADD EVERY RELATED TABLE**
>
> The MS SQL driver page warns that any table with a foreign key linking to a primary key must be in the rule set. Add all four ShopSphere tables together (CUSTOMERS, ORDERS, ORDER_ITEMS, PRODUCTS). If a child table is left out and the parent's key is masked, the database will reject the change or the child rows will point at nothing.

8.31.1 Flat, semi-structured and copybook rule sets in one paragraph

For files, step 2 is called Data Files, and you click **Add File Name** (pick files the connector can see) or **Add File Pattern** (a Java regular expression, such as `.*\.txt`, is case sensitive). You then set properties that depend on the type: a format file (the description of the file's layout, created under Settings > Data Formats), end of record (Windows CR+LF or Linux LF, or a custom value), delimiter and text enclosure for delimited files, "Mask Whole File" for fixed-

width files, and "Variable Length" for mainframe data sets. File name patterns work only for file-to-file jobs, not database-to-file jobs. Control characters can be entered as `$[hex]`, for example `$[01]`. These are the topics of the file masking chapter.

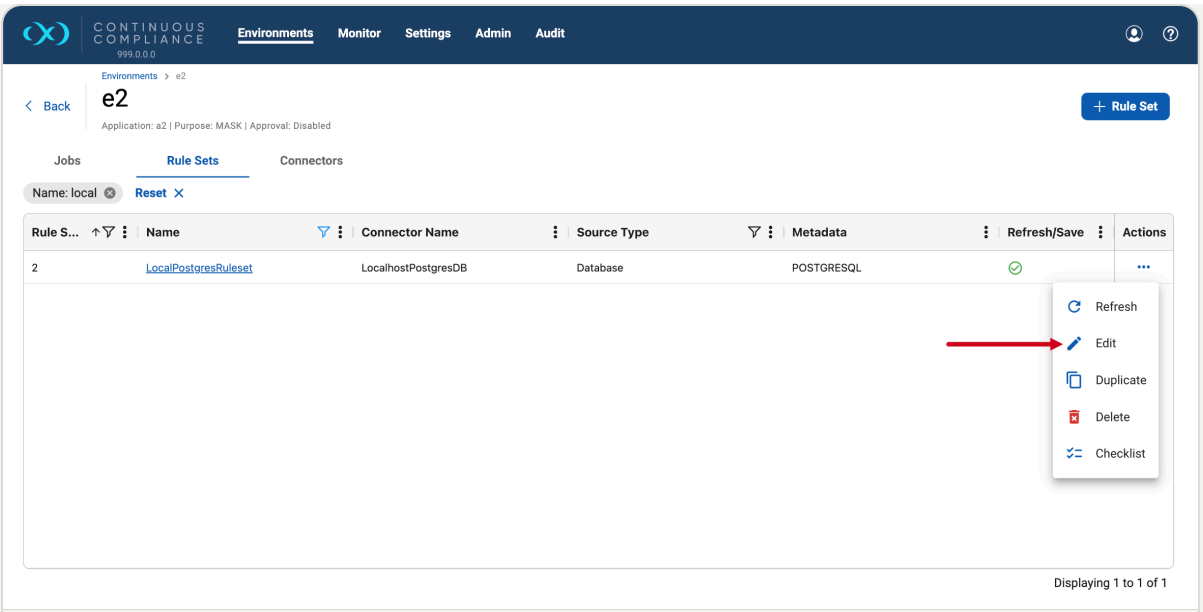


Official screenshot: The Data Files step of a flat file rule set. Compare it with the Data Tables step: the grid columns change with the type, but the pattern (add, edit selected, remove selected, reset) is the same.

Source page ↗

8.32 Editing, refreshing, copying, deleting and verifying

All of these are in the Actions (...) menu of a rule set. The screenshot below shows the menu: Refresh, Edit, Duplicate, Delete and Checklist.



Official screenshot: The rule set Actions menu. Note the last item, Checklist: the page text calls this "Verify" in its action list and "Checklist" in its own section, so expect either word.

`Source page ↗`

- **Edit.** The same wizard, pre-filled. Step 1 is read-only (name and connector cannot change). In Data Tables a read-only **State** column shows Added, Modified or Removed for rows you changed. Large rule sets load in batches and the grid becomes usable before the last batch arrives; "select all" before loading finishes affects only the rows loaded so far.
- **Remove tables.** Tick them, click Remove Selected, then Confirm. A table already saved shows State "Removed" until you click Next and Save: nothing is deleted until the wizard's own Save.
- **Refresh.** Rescans all tables and columns and refreshes the inventory, *keeping existing algorithm assignments*. The Refresh/Save icon turns into an hourglass while it runs and back to the circular arrow when done. Whether tables dropped from the source disappear depends on an API setting (`refreshDropsTables`). Refresh after the database schema changes, for example a new column.
- **Duplicate.** Copies the rule set *with* its inventory and its filters. Give a new name and Save, then modify.
- **Delete.** Deletes the rule set, its inventory and its filters.

TWO "SAVE" BUTTONS

Many dialogs in this wizard (Add Tables, Edit Logical Key, filters) have their own Save. That Save only stages the change. Nothing is stored until you click the *wizard's* Save on the last step. If you close without it, your key is gone.

8.33 Logical keys, filters and custom SQL

8.33.1 What a logical key is, and when you must add one

A logical key tells Delphix "treat these columns as the row identifier for this table". The page defines it as a unique, non-null value that identifies a row. It does **not change the target database**; it is information for Delphix only. The rule from the page: if a table has no primary key defined in the database and you use in-place masking, you must name an existing column or columns as a logical key. For several columns, separate them with commas. If you do not, and there is no primary key, the engine creates an identity column, which needs a database user allowed to modify the schema, otherwise the job may fail.

Limits worth knowing, all from the pages: a logical key cannot exceed 1,024 characters; the column must exist (the dialog warns that masking jobs fail if the key column does not exist); a fixed-issues entry says null values in logical key columns are now rejected at create or update; and if some tables get an invalid key, the other changes are still saved, the old key stays, and the failures are listed in the Async Tasks page under "Exception Detail".

WHICH SHOPSPHERE TABLES NEED ONE?

All four core ShopSphere tables have a declared primary key (customer_id, order_id, item_id, product_code), so none *needs* a logical key. A staging table with no declared key is the typical case. The lab adds one on an example table. A composite key is also possible: ORDER_ITEMS could use the logical key `order_id,product_code` if that pair is unique, though its own primary key is the better choice.

8.33.2 Adding a logical key in the rule set wizard

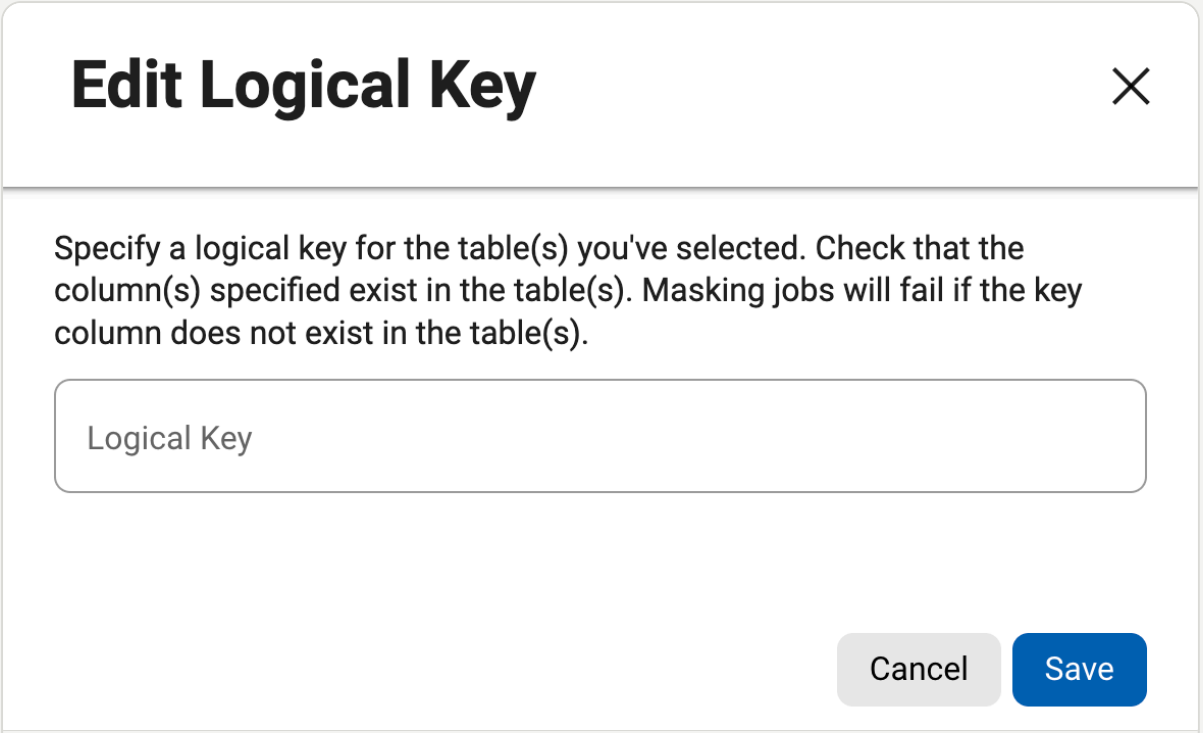
1. **Open the rule set for editing** (Actions, Edit) and go to the Data Tables step. Logical keys are set per table, at this step.

2. **Tick one or more tables** and click **Edit Logical Key**.

Expected result: A dialog asks you to specify a logical key and warns to check that the columns exist in the tables.

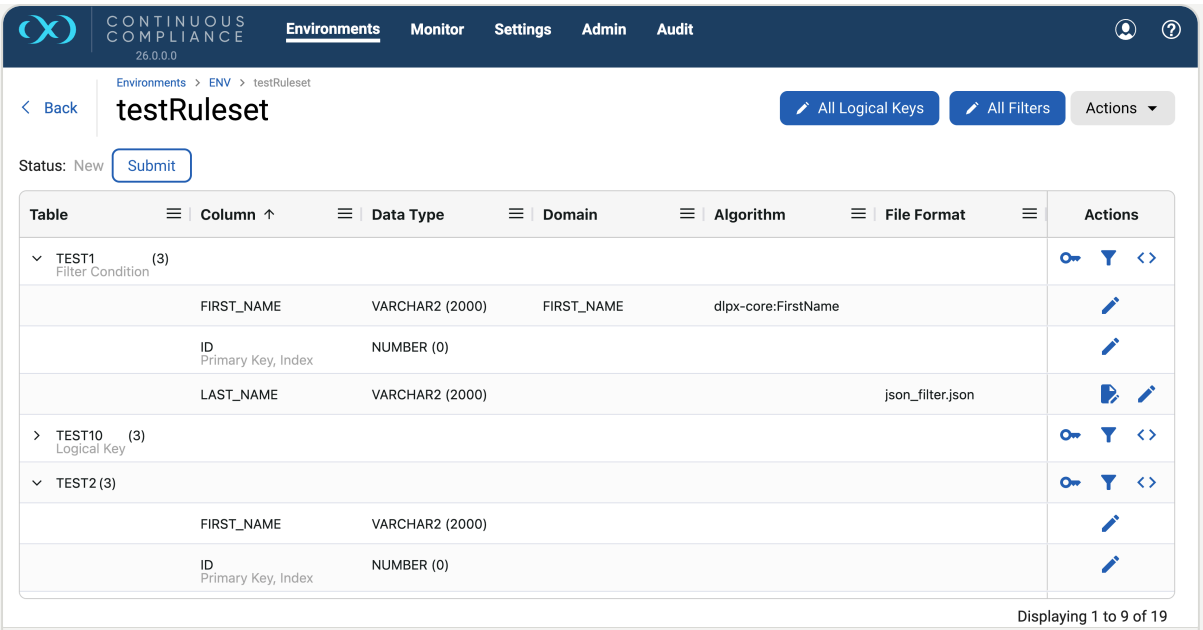
3 **Type the column name(s)** and click Save in the dialog. Then Next and **Save in the wizard**.
The dialog's Save only stages the change; the wizard's Save stores it and starts an Async Task.

Expected result: The Logical Key column of the grid shows your value; after the wizard Save the task completes in Monitor > Async Tasks.



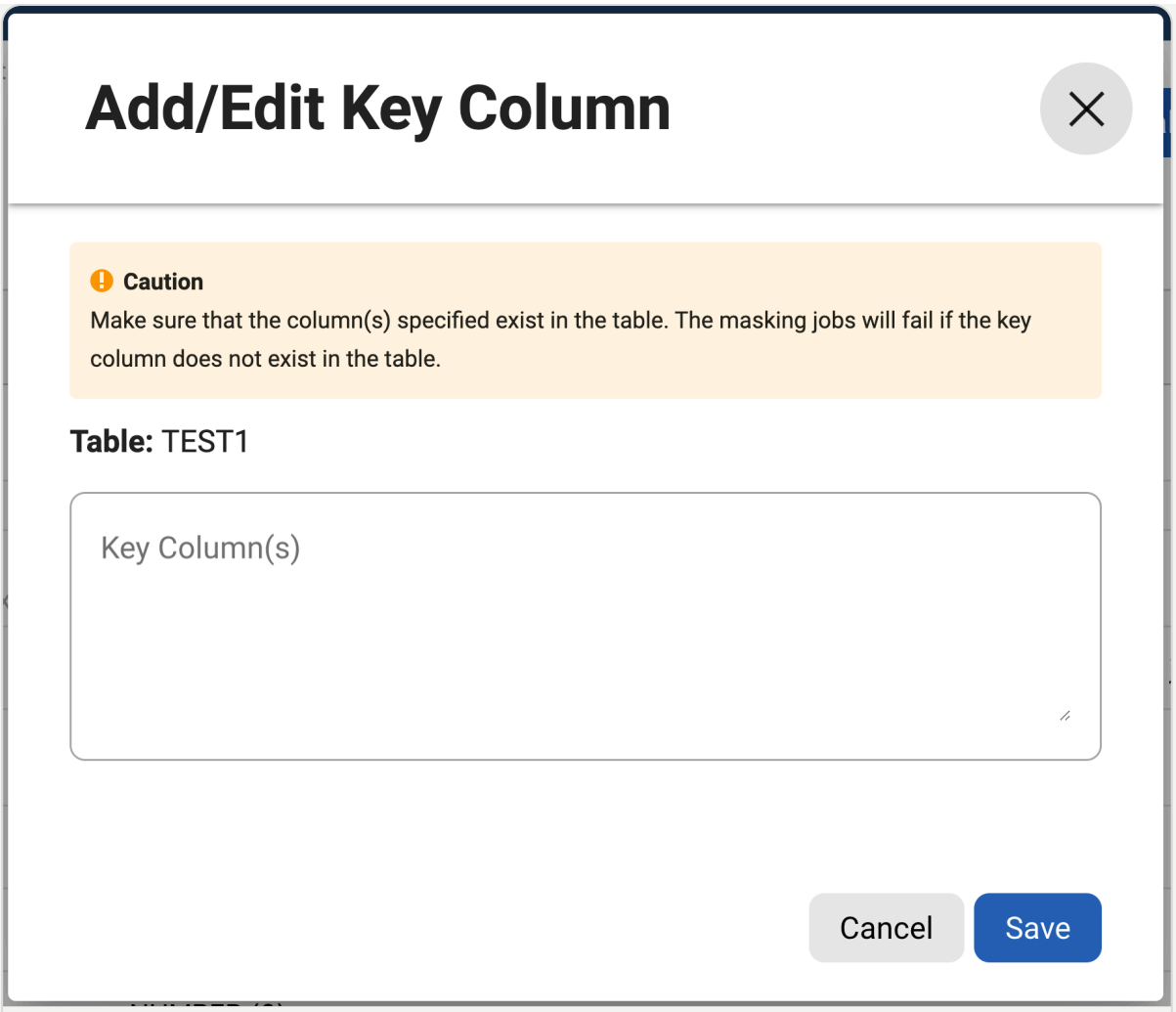
Official screenshot: The Edit Logical Key dialog. The warning in its text is the practical risk: a typo in a column name is accepted at this stage and shows up later.
`Source page ↗`

You can also set it from the **inventory**. Click the rule set's name. Each table row has three icons: a key (logical key), a funnel (filter) and a code symbol (custom SQL). A button at the top right, **All Logical Keys**, edits every table at once. Beneath each column name the inventory writes whether the column is a Primary Key, Foreign Key or Index.



Official screenshot: An inventory. Under the column ID the grid writes "Primary Key, Index"; table TEST10 is labelled Logical Key and TEST1 Filter Condition. The three icons on each table row (key, filter, code) edit the logical key, filter and custom SQL. A column that has a domain and algorithm, such as FIRST_NAME here, will be masked.

`Source page ↗`

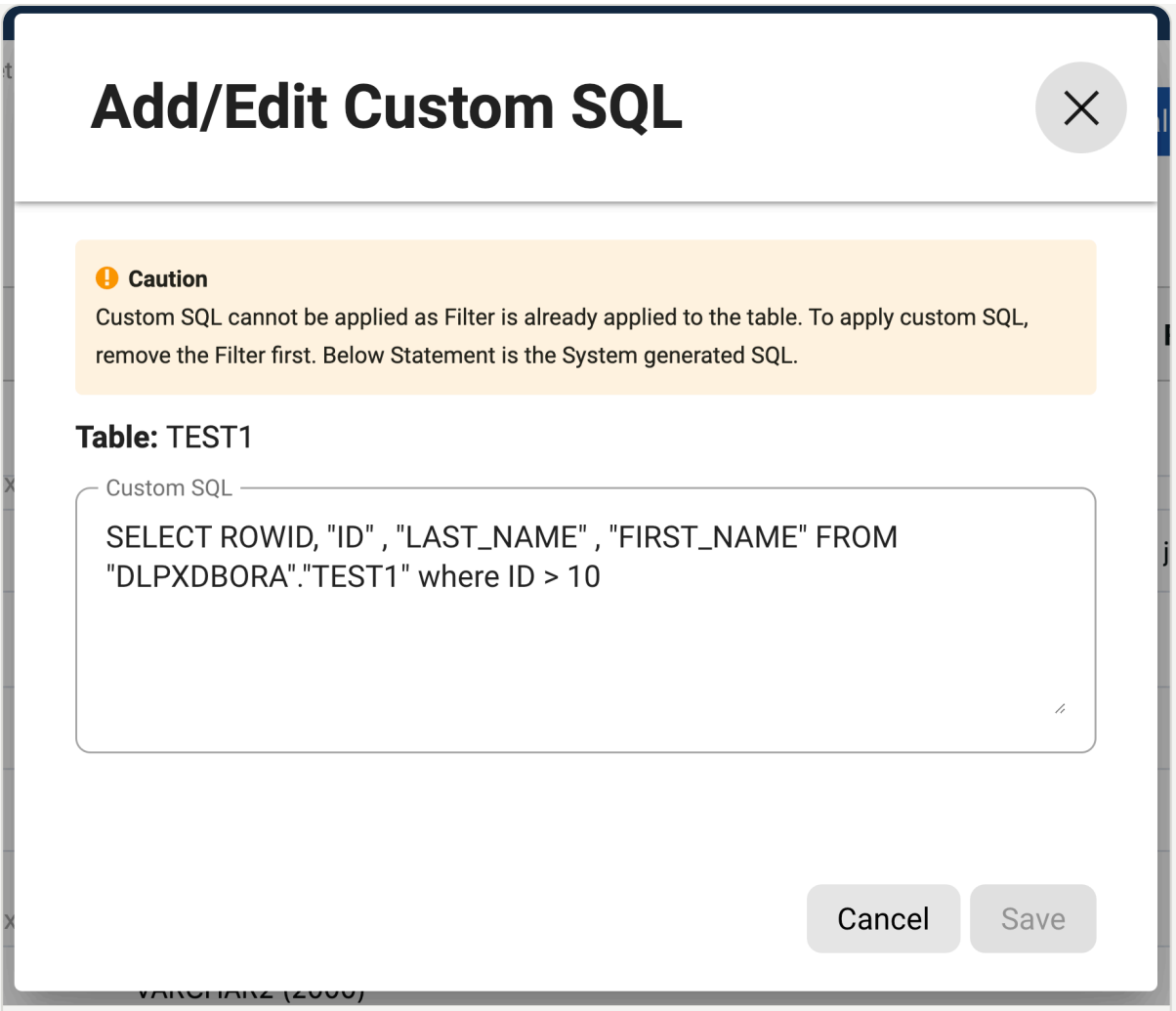


Official screenshot: The inventory's version of the same dialog, titled Add/Edit Key Column. Same idea, different wording: key columns for one table.

`Source page ↗`

8.33.3 Filter and custom SQL (briefly)

Two other table properties live beside the key. A **filter** is a WHERE condition that restricts which rows the job loads (the dialog's hint: prefix the column with the table name, as in `customer.cust_id < 1000`). **Custom SQL** replaces the engine's own SELECT. A table can have a filter or custom SQL, never both; adding one removes the other. The page's rules for custom SQL: it must include the primary key column(s) and every column you will mask; it must include the `ROWID` column (the engine uses it to identify and update each row, and omitting it may fail the job or leave masking incomplete); and you can edit only one table at a time. The "Generate Custom SQL" link suggests a statement for you. Filters are covered with subsetting ideas in the discovery and subsetting chapter.



Official screenshot: System-generated custom SQL for a sample table. Notice that ROWID is the first selected column, which matches the page's rule that custom SQL must include it.

`Source page ↗`

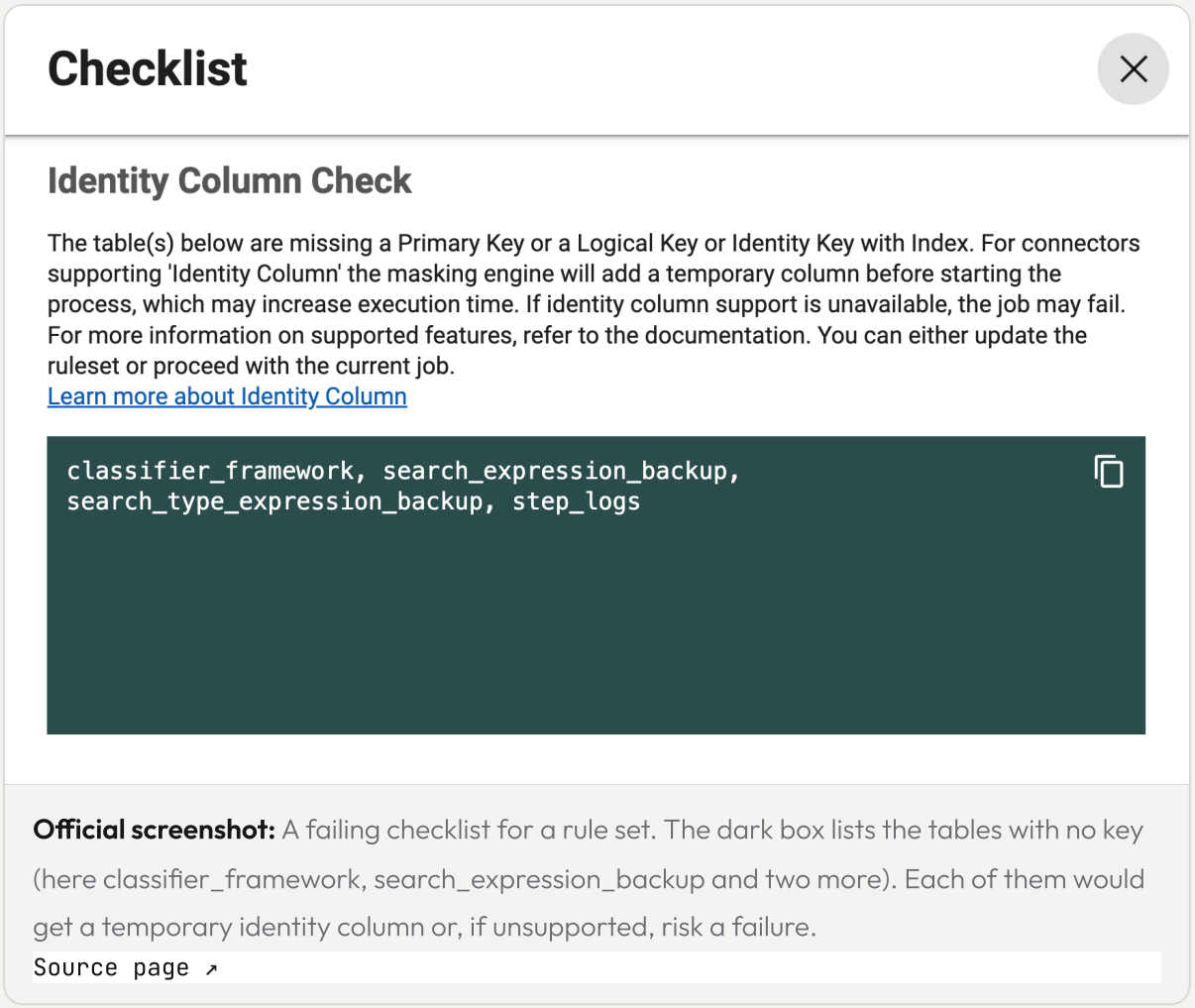
TWO STATEMENTS TO RECONCILE

The custom SQL text says it must contain the primary key column(s) and the masked columns, and also that it must include ROWID. A fixed-issues entry for Oracle also says a table with a primary key failed when the key was missing from custom SQL. Include both the key and the ROWID if your database supports it, and test with a small run. The documents do not say whether ROWID applies to non-Oracle databases.

8.34 The Checklist: does every table have a usable key?

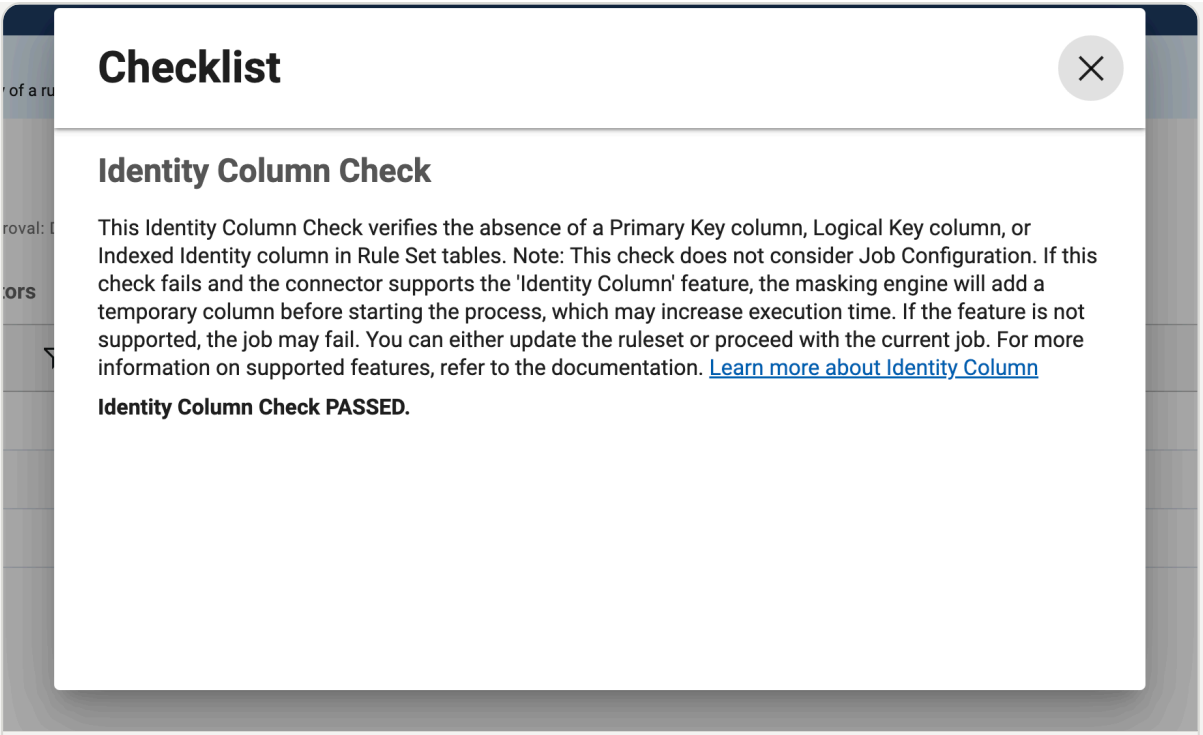
The engine can check your keys for you. Choosing **Checklist** in the Actions menu of a database rule set (or of a job) runs an **Identity Column Check**. It confirms that every table in the rule set has a primary key, an indexed identity column or a logical key. The new-features page describes it as helping you understand the side effects of a missing key before the job runs.

The result is a dialog. If tables lack a key, it lists their names and explains that, for connectors that support identity columns, the engine will add a temporary column before the job starts, which may lengthen the run, and that where identity support is not available the job may fail. You can either update the rule set or go ahead with the job. If everything has a key, the dialog says the check **passed**.



Official screenshot: A failing checklist for a rule set. The dark box lists the tables with no key (here classifier_framework, search_expression_backup and two more). Each of them would get a temporary identity column or, if unsupported, risk a failure.

Source page ↗



Official screenshot: The job-level version of the check, passing. Its text adds that the check does not consider job configuration.

`Source page ↗`

> **A PASSING CHECKLIST IS NARROWER THAN IT SOUNDS**
>
> The job page says the check "relies solely on ruleset configuration". Even if a job disables a primary key constraint, the check still passes. It proves tables *have* a key, not that the key will be usable when the job runs, and not that your masking will keep foreign keys consistent. That second question is the next section.

8.35 Data integrity in primary and foreign key columns

8.35.1 What breaks if you mask the two sides differently

Suppose ShopSphere's profiling flagged only the sensitive columns, and a well-meaning engineer also masked `customer_id`, but gave CUSTOMERS and ORDERS *different* algorithms, or a random one on each side. Each value is replaced independently.

AFTER MASKING CUSTOMER_ID DIFFERENTLY ON EACH SIDE (EXAMPLE DATA). THE JOIN I LONGER WORKS.

CUSTOMERS.customer_id	first_name (masked)	ORDERS.order_id	ORDERS.customer_
7342	Divya	5001	2199
9051	Karthik	5002	2199
3318	Lakshmi	5003	6670

No order's customer_id (2199, 6670) exists in CUSTOMERS (7342, 9051, 3318). The orders are *orphans*. Worse, the two orders that belonged to one customer (5001, 5002) happen to match each other here only because the same input gave the same output in ORDERS. Test cases such as "show the last five orders of customer Divya" now return nothing, and a database that enforces its foreign key will refuse the change outright (a constraint violation).

The fix is to give both columns the **same algorithm**, one that is *deterministic* (same input gives same output) and that produces unique outputs.

AFTER MASKING BOTH CUSTOMER_ID COLUMNS WITH THE SAME DETERMINISTIC, UNIQUE ALGORITHM (EXAMPLE DATA). THE JOIN WORKS.

CUSTOMERS.customer_id	first_name (masked)	ORDERS.order_id	ORDERS.customer_
7342	Divya	5001	7342
9051	Karthik	5002	7342
3318	Lakshmi	5003	3318

1001 became 7342 in both tables, and 1003 became 3318 in both. Every order still points at its customer. The IDs no longer reveal the real ones. The same logic applies to ORDER_ITEMS.order_id against ORDERS.order_id.

> **OFTEN THE BEST CHOICE IS NOT TO MASK THE KEY**
>
> A profiler marks `customer_id` as non-sensitive, because it is a meaningless number. If nobody can identify Asha from 1001, leaving key columns unmasked is the simplest and safest way to keep integrity. Mask a key only when the key itself is sensitive (an SSN used as a key, for example). This is general practice, not a statement from the documentation.

8.35.2 How Delphix keeps matching values matching

The documentation's promise appears in the introduction: Delphix masks consistently, so data is masked the same way across tables and databases. The pages that explain *how* give four ingredients.

1. **Use the same deterministic algorithm on columns that reference each other.** The Oracle, PostgreSQL and Db2 driver pages all say this almost word for word: use the same deterministic algorithms on primary key fields that reference each other so referential integrity is kept when constraints are re-enabled. The PostgreSQL and Db2 pages add that if a foreign key column and the key it references do not share one deterministic algorithm, the job fails with an integrity constraint violation. The SQL Server page says it too, with a stronger warning: the failure is near certain.
2. **Use an algorithm whose output is unique.** Secure Lookup picks a value from a list by hashing the input, so it is consistent, but different inputs can collide on the same output, which is fine for names and wrong for keys. The Segment Mapping page says to use it for columns with unique values such as SSNs, primary keys and foreign keys, and that for keys you must use the *same* Segment Mapping algorithm on each side. The Character Mapping page says the same input always gives the same output (until you rekey it) and no two different text inputs share an output, "useful for masking primary and foreign key columns". The SQL Server page's examples of unique algorithms are Dlpx-Core CM Alpha-Numeric, and the CM Numeric page describes a numeric one that keeps the number of significant digits. The algorithms chapter covers them.
3. **Do not use algorithms that are random or unsuitable.** The Secure Lookup page says its RANDOMIZE hash method picks a random value every time and "should never be used where referential integrity is required". The Gender SL page says the same of itself. The Date Shift Discrete page says it is not suitable for columns with unique constraints, unique indexes, primary keys or foreign keys. The terminology page notes the Mapping algorithm can mask only one table at a time (the Mapping page says this

limit ended with engine 6.0.10.0), so on older engines tables must be masked one after another.

4. **Let the database allow the update.** A primary key referenced by foreign keys cannot normally be changed. So a job that masks a key must enable **Disable Constraints** and also **Drop Indexes**. The known-issues page records a job failure when primary keys are masked without Drop Indexes together with Disable Constraints, and the Oracle plugin page says the same. The constraints are re-enabled after masking, which is when mismatched values would be caught.

> **WHERE THE DOMAIN HELPS**
>
> The discovery page says the domain mechanism "helps to ensure that the same masking algorithm is applied consistently across rule sets whenever a particular type of sensitive data is discovered". If ShopSphere defines one domain for customer identifiers and gives it one algorithm, every profiling run that finds such a column assigns the same algorithm. That is a good way to keep both sides of a key aligned, though the documentation does not claim it handles key relationships by itself.

8.35.3 Algorithm syncability: the same output on two engines

Many teams run a masking engine for each stage (see the overview chapter's note on multiple engines). If the dev engine and the QA engine mask 1001 differently, test data from the two will not join. The Algorithm syncability page is about this. It says algorithms can be synced between engines through an API, and that syncing out-of-the-box instances updates their individual *keys*. An algorithm key is the secret value that decides the output; two engines produce the same masked value only if they have the same algorithm and the same key.

The page also separates two kinds. **Non-deterministic** algorithms cannot guarantee the same output for the same input across jobs, so they "cannot be used for syncable masking": it lists DATE SHIFT(VARIABLE), the Gender SL and SexualOrientation SL instances and SECURE SHUFFLE, and says all Secure Lookup (Random) algorithms are non-deterministic by design. **Fixed** algorithms, such as DATE SHIFT(FIXED) and ZIP+4, always give the same result and need no syncing. Everything else is deterministic and depends on its key. The page does not mention primary or foreign keys at all: its subject is consistency *between engines*. Within one engine, consistency comes from using the same algorithm on both columns.

8.36 Hands-on lab: a rule set for the SHOPSPHERE tables

Lab: create a rule set, run the checklist, add a logical key

Goal: a rule set `SHOPSPHERE_RS` over the ShopSphere tables, a passing Identity Column Check, and a logical key on a table that has no primary key. Prerequisite: the environment and tested connector from the connectors and VDB chapter. The table `CUSTOMER_STAGING` is an added example, because the four core tables all have primary keys and a keyless table is needed to practise the logical key. Ask your DBA to create it in the lab database, or read steps 6 and 7 without running them:

```
CREATE TABLE CUSTOMER_STAGING (
  staging_row_id INTEGER NOT NULL,
  email          VARCHAR(100),
  first_name     VARCHAR(50),
  last_name      VARCHAR(50),
  ssn            VARCHAR(11)
);  -- no PRIMARY KEY declared (example)
```

1. **Start the wizard.** Open `SHOPSPHERE_DEV`, click the Rule Sets tab, then **+ Rule Set**, choose *Database* and click Continue.
   Expected result: The "Add a Rule Set for a Database" wizard opens on the Details step.

2. **Details.** Name it `SHOPSPHERE_RS` and select the connector `SHOPSPHERE_VDB_CONN`. Check the connector details shown below the dropdown, then Next.
   Confirming the database name and host now saves a confusing failure later.

3. **Add the tables.** Click Add Tables and tick CUSTOMERS, ORDERS, ORDER_ITEMS, PRODUCTS (and CUSTOMER_STAGING if it exists). Save the dialog.
   All related tables go in together so parent and child keys can be masked consistently.

   Expected result: Five (or four) rows in the grid with empty Logical Key, Filter and Custom SQL.

4 **Save the rule set.** Next, check the Summary, click Save, and note the Async Task ID. Watch Monitor > Async Tasks.
Expected result: The task completes and SHOPSPHERE_RS appears in the Rule Sets grid with a green Refresh/Save tick.

5 **Run the Checklist.** In the rule set's Actions menu choose **Checklist**.
It tests every table for a primary key, indexed identity column or logical key before any job exists.

Expected result: With the staging table present: a dialog titled Identity Column Check listing `CUSTOMER_STAGING` as missing a key. Without it: a passing result.

6 **Add the logical key.** Actions > Edit, Next to Data Tables, tick CUSTOMER_STAGING, click **Edit Logical Key**, type `staging_row_id` , Save the dialog.
staging_row_id is unique and never empty, which is what a logical key requires. It changes nothing in the database.

Expected result: The Logical Key column of that row shows `staging_row_id` .

7 **Save the wizard.** Click Next, then Save in the wizard, and wait for the Async Task.
The dialog Save only staged the change.

8 **Check again.** Run Checklist again.
Expected result: The check passes (the job version of the dialog reads "Identity Column Check PASSED").

9 **Read the keys in the inventory.** Click the rule set name to open the inventory and look under the key columns.
Expected result: Key columns carry the words "Primary Key" (and "Index" if indexed); a foreign key column such as ORDERS.customer_id is marked "Foreign Key" if the database declares it; the staging table is labelled "Logical Key".

▶ **Show a worked answer: the integrity plan**

8.37 Common mistakes and gotchas

KEY AND RULE SET PITFALLS

- **Believing a passing Checklist means integrity is safe.** It only proves each table has a key.
- **Masking the primary key with a non-unique algorithm.** Duplicates break the key. Use a unique, deterministic algorithm, and the same one on the foreign key side.
- **Masking a key without Disable Constraints and Drop Indexes.** The job fails on the key constraint.
- **Using a random algorithm on a key.** RANDOMIZE and the Gender SL type never preserve integrity.
- **Forgetting the wizard's Save.** The logical key, filter or table list is lost.
- **Typing a logical key column that does not exist.** The job fails, or the table is listed under Exception Detail in Async Tasks.
- **Leaving out a child table.** A foreign key table linked to a masked key must be in the rule set.
- **Assuming refresh wipes your masking choices.** It keeps existing algorithm assignments and updates the table list and columns.

8.38 Recap

RECAP

- A rule set is the list of tables or files of one connector that jobs run on. There are four types: Database, Semi-Structured Files, Flat Files and Copybook.
- Creating one takes three steps (Details, Data Tables, Summary); the wizard's Save stores everything and starts an Async Task. Edit, Refresh, Duplicate, Delete and Checklist are in the Actions menu.
- In-place masking finds each row by a key: a logical key if you set one, otherwise the primary key, otherwise an identity column, otherwise a temporary one the engine creates.
- A logical key is a unique, non-null column (or comma-separated columns) given to Delphix only; it changes nothing in the database. On Oracle, ROWID is the recommended one.
- The Checklist's Identity Column Check reports tables that have no usable key; it does not test integrity.
- If you mask a primary key, mask the foreign key with the *same* deterministic, unique algorithm, and enable Disable Constraints and Drop Indexes on the job.
- Algorithm syncability concerns producing the same output on several engines; non-deterministic algorithms cannot be synced.

Delphix: profiling to find sensitive data

You cannot protect data you have not found. A real application database has hundreds of tables and thousands of columns, and nobody remembers which of them hold a customer's phone number, a tax id, or a note typed in by a support agent. *Profiling* is Delphix's answer: a job that scans your data source, decides which columns look sensitive, and writes its decisions into a list that you review before any masking happens.

It covers what a profile set, a classifier, a domain and an inventory are, how they fit together, how to create and run a profiling job, how to read the results, and how to

correct them by hand. It ends with a lab on the ShopSphere sample database. Tool-specific facts are from the Delphix Continuous Compliance documentation 2026.5; screenshots come from engine versions 22.0.0.0 to 26.0.0.0, so your screen may differ. The connections and rule sets the lab needs are set up in the Delphix connectors chapter and the rule sets and keys chapter; the algorithms that profiling assigns are explained in the algorithms and masking jobs chapter. If you want the Informatica view of the same idea, see the Informatica discovery and profiling chapter.

8.39 Words to know first

Sensitive data discovery / profiling

The process of scanning a data source to find columns or file fields that hold sensitive information. The docs use both names for the same thing.

Profiling job

A saved set of instructions that says "scan this rule set using this profile set". You create it once and can run it many times.

Rule set

The list of tables (or files) that a job is allowed to look at. A profiling job covers only what is in the rule set; new tables are not discovered until you add them.

Profile set

A bundle of recognition rules (classifiers). Choosing a profile set decides which kinds of sensitive data the job can detect.

Classifier

One recognition rule, tied to one domain. Example: "a column whose name looks like *ssn*" or "a value that looks like an email address".

Domain

A named kind of sensitive data, such as FIRST_NAME, SSN or EMAIL, together with the default masking algorithm for it.

Algorithm

The recipe that actually replaces a real value with a safe one. Profiling only *chooses* an algorithm; the masking job *runs* it.

Inventory

The table of every column or field in a rule set with its domain and algorithm. It is the profiler's output and the masking job's input.

Confidence

The number (shown 0 to 100 in the UI) that says how sure the profiler is about a match.

Assignment threshold

The minimum confidence needed before the profiler writes a domain and algorithm into the inventory.

ASDD profiler

Automated Sensitive Data Discovery, the current profiler. It uses classifiers. The older "legacy" profiler uses search and type expressions.

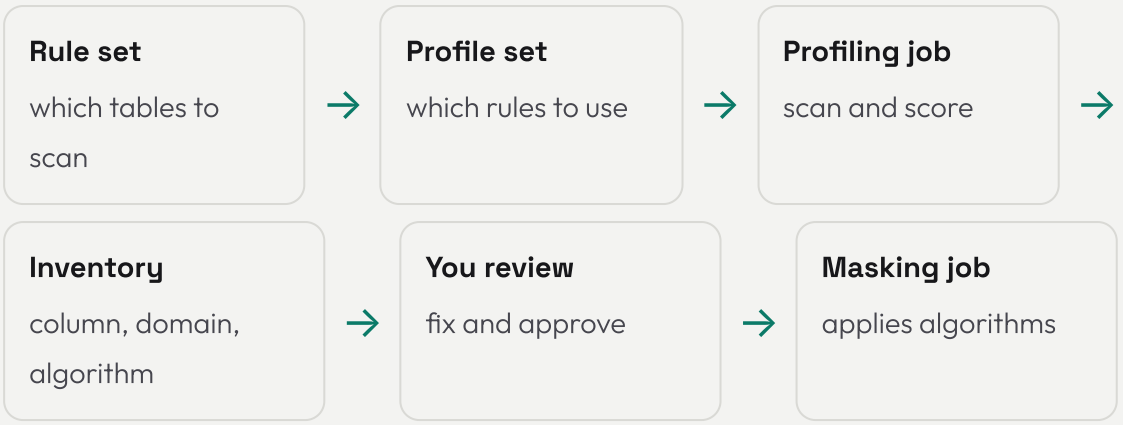
8.40 Why profile at all?

Picture the ShopSphere developers asking for a copy of `SHOPSPHERE_PROD` to test the Customer Orders app. A TDM engineer must hide every real name, phone number and tax id before handing it over. With four tables and nine customer columns you could do that by reading the table definitions. With a real database of 3,000 columns you cannot, and one forgotten column is a privacy incident.

A profiling job automates the first pass. The documentation describes it as examining metadata (column names and types) and potentially the data itself, then assigning the matching domain and its algorithm to each column it judges sensitive . A person then reviews the result. That last step matters: the profiler is a fast assistant, not an auditor.

> **WHY IT MATTERS**
>
> Profiling is where masking projects are won or lost. Masking jobs only protect what is in the inventory. A column the profiler missed, and nobody caught, goes to the test environment untouched.

8.40.1 The whole pipeline on one line



Analogy: an airport scanner flags bags using a rulebook (the profile set). A security officer (you) opens the flagged bags, waves through the harmless ones, and sometimes stops a bag the scanner missed. The analogy stops being accurate in one place: the scanner has no memory of the officer's decisions unless you tell it, which is what the "automatic updates" checkbox, described in the inventory review part of this section, is for.

8.41 Profile, domain, algorithm assignment and inventory: how they differ

These four words are often used as if they meant one thing. They are four different layers, and each answers one question.

Layer	Question it answers	Where you manage it	ShopSphere example
Profile set (made of classifiers)	How do we recognise sensitive data?	Settings, Profile Sets and Classifiers	"ASDD Standard"
Domain	What kind of sensitive data is this, and what is its default algorithm?	Settings, Domains	FIRST_NAME, SSN, TELEPHONE_NO
Algorithm assignment	Which recipe masks this particular column?	Inventory screen of a rule set	`CUSTOMERS.first_name` gets `dlpx-core:FirstName`
Inventory	For this rule set, what is every column's domain and algorithm?	Click a rule set name	The whole list for `SHOPSPHERE_PROD`

Two consequences follow, both stated in the docs :

- Because the profiler assigns an algorithm through the domain, the same kind of data is masked the same way everywhere. Every FIRST_NAME column in every rule set gets the same default algorithm, which keeps results consistent.
- Each domain has exactly one default algorithm, so if you want the profiler to pick a different algorithm for some data, a separate domain (with its own recognition logic) has to exist for it.

> **CHANGING A DOMAIN LATER**
> Editing a domain's default algorithm affects only future profiling runs. It does not change assignments already sitting in existing inventories . If you change a default and expect old inventories to follow, they will not.

8.42 How the profiler decides: classifiers, strengths and thresholds

The profiler looks at each column and asks a panel of four judges. Each judge looks at a different kind of evidence and answers with a score. The scores are combined into one confidence per domain. If the best confidence reaches the assignment threshold, that domain (and its algorithm) is written into the inventory.

8.42.1 The four classifier frameworks

A *classifier framework* is a kind of judge; a *classifier instance* is one configured judge for one domain. The docs list four frameworks and say you cannot install more .

Framework	Looks at	Level	Plain-English job
PATH	The name (for databases: table name and column name)	Column	"The column is called *first_name*, so it is probably a first name."
TYPE	Data type and length	Column	Rejects impossible matches, e.g. a yes/no flag cannot be an SSN.
REGEX	The actual values, using patterns	Data	"The values look like `123-45-6789`."
LIST	The actual values, checked against a list	Data	"The values are all on the list of known first names."

PATH and TYPE are cheap, because they only need the table definition. REGEX and LIST must read sample rows, so they are what lets the profiler find a sensitive column even when its name is meaningless, such as `COL_17`. The built-in value lists carry English values, so data-level detection is limited to English .

8.42.2 Match strength, reject strength and how scores combine

Inside the engine every classifier reports a number from -1.0 to 1.0. The UI shows the same idea as -100 to 100, and in practice you only see 0 to 100 because non-matches

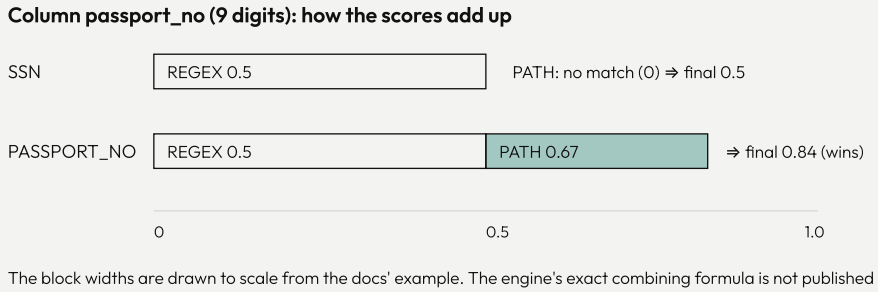
are not displayed .

- **Match strength**: how sure the classifier is that this data belongs only to its domain. 0.01 means "maybe, but it could be many things". 1.0 means "certainly this and nothing else".
- **Reject strength**: how strongly a *non-match* counts against the domain. -1.0 eliminates the domain at once.
- **Combining**: if results disagree in sign, the larger absolute value wins; if they agree, the match gets stronger. Only the strongest column-level result and the strongest data-level result are combined, and the docs warn that the exact formula may change.

TWO WORKED EXAMPLES FROM THE DOCS

Rejection. A column named `ssn_present` matches the SSN PATH classifier (0.67). But it is boolean, the SSN TYPE classifier returns -1.0, and the verdict is -1.0, so SSN is not assigned.

Competition. A column named `passport_no` holds 9-digit numbers. Both the SSN and the PASSPORT_NO REGEX classifiers give 0.5. Only the PASSPORT_NO PATH classifier also matches (0.67). Final confidence: PASSPORT_NO 0.84, SSN 0.5, so PASSPORT_NO wins.



Original diagram: the "passport_no" example from the classifier documentation. Both domains get the same data-level score, and only PASSPORT_NO also matches by column name.

The built-in PATH classifiers usually carry 0.67. A REGEX or LIST classifier has to beat that to override a name match, which is why the docs advise a low match strength (0.1 to 0.5) for patterns that are not unique. A nine-digit pattern could be an SSN, an

account number or a row id, so it gets a small score. A dashed pattern like `001-23-4567` is distinctive, so it gets a high one.

8.42.3 What the SSN classifiers actually look like

The ASDD Standard profile set is published as JSON. For the SSN domain it holds three classifiers (listing as of the 21.0.0.0 release), summarised here:

Classifier	Framework	What it checks	Strength
Social Security Number - Path	PATH	Name ends in *ssn* or contains *social security* plus number, code or id	match 0.67
Social Security Number - Regex	REGEX	Dashed or spaced pattern, plain nine digits	dashed forms 0.7, plain nine digits 0.1, reject 0.1
Social Security Number - Type	TYPE	Number or String, at least 9 long	no match strength, rejects other types

Notice the design: the name gives a good clue, the value pattern confirms it, and the type check rules out nonsense. The same three-part shape repeats for most domains.

8.42.4 Which domains can be found by value, and which only by name?

Not every domain has all three kinds of evidence. Reading the ASDD Standard listing (21.0.0.0) shows the pattern, and it matters when column names are cryptic. The domains are grouped by the classifier frameworks that the listing gives them:

Evidence available	Domains (examples from the listing)	What it means for you
Name, type and value list	FIRST_NAME, LAST_NAME, FULL_NAME, CITY, COUNTRY, COMPANY_NAME, GENDER, JOB_TITLE, US_STATE	Can be found even in a column called `COL_17`, as long as values are in the English lists
Name, type and value pattern	SSN, EMAIL, TELEPHONE_NO, ZIP, CREDIT CARD, IBAN, IP ADDRESS, US_PASSPORT, PAYMENT_AMOUNT, ADDRESS	Value patterns can rescue a badly named column, if the values fit the pattern (US-shaped for ZIP and phone)
Name and type only	DOB, TAX_ID, ACCOUNT_NO, CUSTOMER_NO, USER_ID, TEXT, PASSWORD, SERIAL_NO	No value check: if the column is not named sensibly, the profiler will not find it

Two ShopSphere consequences follow. `date_of_birth` and `tax_id` are found only because their names are clear, so a legacy table with a column called `DT1` would slip through. The grouping is read from the JSON listing, so check the live Classifiers screen of the engine for the exact picture.

8.42.5 Assignment threshold

The default minimum confidence for ASDD is 1, so "any positive match, no matter how weak, triggers an assignment" . That is very permissive, and the docs contrast it with the legacy profiler, which by default wanted an 80% match for data-level expressions. You can raise the threshold per profile set (integer 1 to 100) or globally with the application setting `ASDD/DefaultAssignmentThreshold`. A higher threshold means fewer false alarms and more missed columns. Because a missed column is worse than a false alarm, many teams keep the threshold low and review the inventory.

8.42.6 How much data the profiler reads

For data-level classifiers the ASDD profiler samples rows instead of reading whole tables. These documented settings steer it; the values are the defaults in the 2026.5 Masking API client page .

Application setting (ASDD group)	Default	What it controls
`DefaultTableSampleRows`	1000	Rows sampled per table
`DefaultAssignmentThreshold`	1	Minimum confidence to assign
`DefaultNullFilterThreshold`	75	Percent of null or empty values that triggers an extra query for more values
`DefaultMultipleProfilerExpressionAlgorithm`	NullValueLookup	Algorithm used when the "multiple profiler expression" option finds a tie

Statistical sampling (so the sample reflects the whole table) is supported for Oracle, SQL Server and SAP ASE. When sampling is used, the sample percentage is always 1%, and if that yields too few rows the query is repeated without sampling . The page does not say what happens on other databases beyond this; check your version.

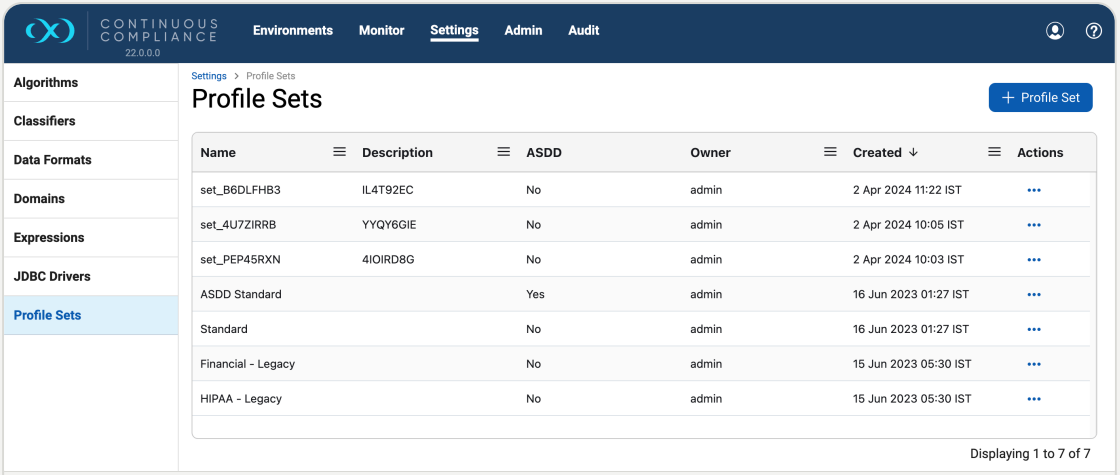
8.42.7 The legacy profiler, briefly

Before ASDD, profiling used *search expressions* (patterns on column names or data) and *type expressions* (data type and minimum length). The legacy profile sets are kept so that old jobs keep giving the same results, and they contain no type classifiers to restrict matches by column type . The legacy classifier tables list patterns such as `ssn(?!\w*ID)` for SSN and `(?>((tele?)?phone)|...)` for telephone. The legacy version of the Add Profile Set dialog (not the ASDD one pictured below) carries a banner saying legacy profile sets are deprecated and will be removed in a future release. Use ASDD for new work; the profileTypeExpressions page only shows API scripts for legacy type expressions.

ASDD currently supports databases, delimited files, fixed-width files, JSON files, XML files and mainframe data . Its stated limits: XML or JSON documents stored inside database columns are not profiled, and the Informix database is a known problem for discovery because of the column-truncation SQL.

8.43 Profile sets: choosing and configuring the rulebook

Open **Settings > Profile Sets** to see the available sets. The table can be sorted by Name, Description and Created, and filtered by Name, Description, Owner and Created .



Official screenshot: Settings > Profile Sets: ASDD Standard (ASDD column "Yes"), Standard, Financial - Legacy and HIPAA - Legacy, plus a few user-made sets.
`Source page ↗`

Built-in profile set	What it holds	When to pick it
ASDD Standard	Path, type, regex and list classifiers; widest coverage of domains, including FIRST_NAME and LAST_NAME detection by value	The recommended set for every source the ASDD profiler supports
Standard	Path and type classifiers	Column-level discovery only (names and types, no value checks)
Financial - Legacy, HIPAA - Legacy	Legacy path classifiers, no type restrictions	Only to keep old jobs giving the same answers

The table follows the Out of the box profiling settings page, which says the product ships logic for "over 30 types" of sensitive data. The Domains screenshot later in this section shows 69 domains on one test engine, so treat "over 30" as a floor, not a count.

8.43.1 Create your own profile set (ASDD)

You would build a custom set when, for example, a project wants only a handful of domains scanned, or a different threshold. The steps below are for the ASDD profiler .

1. **Click *+ Profile Set* above the grid.**
   This opens the Add Profile Set dialog.

2. **Type a unique Set Name; add a Description if you like.**
   Names must be unique across all profile sets.

3. **Leave *ASDD Support* ticked (the default).**
   This tells the engine the set will hold classifiers rather than legacy expressions. You cannot flip this later on an existing set; you would have to create a new one.

4. **Optionally enter an Assignment Threshold, 1 to 100.**
   If you leave it empty, the application setting `DefaultAssignmentThreshold` applies.

5. **Tick the classifiers you want.**
   Each classifier is tied to a domain, so the ticked list decides which domains this set can ever detect.

6. **Click *Save*.**
   Expected result: The new set appears in the grid and in the profile set drop-down of the profiling job wizard.

Official screenshot: The Add Profile Set dialog with ASDD Support ticked. On screen: it shows Set Name, Description, an optional Assignment Threshold field, a single "Classifiers" tab, and a checklist such as "Account Number - Path" and "Address Line 1 - Regex"; a few boxes are ticked.

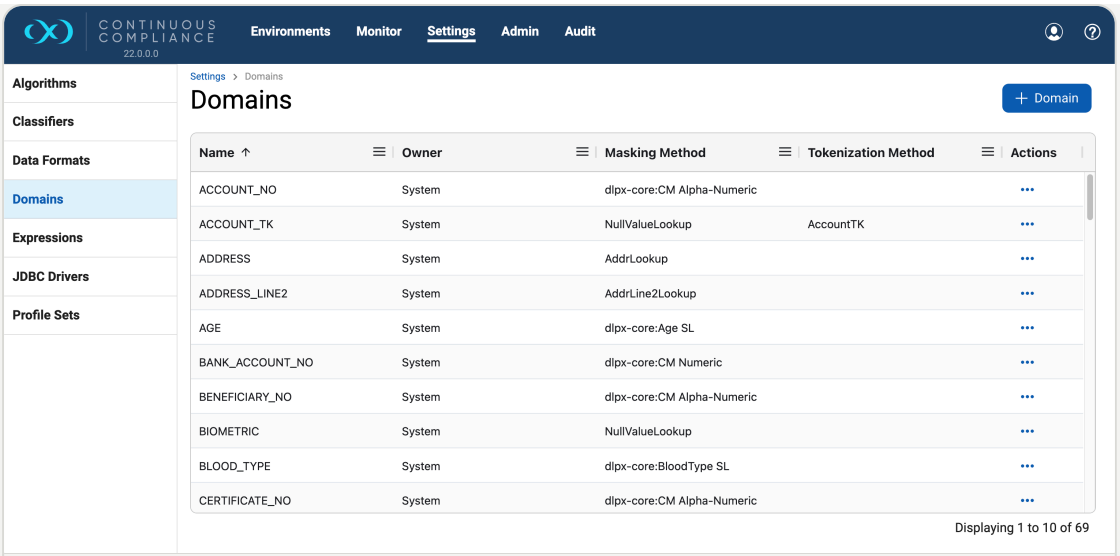
`Source page ↗`

Other actions in the row menu are **View** (everything read-only), **Edit** (tick or untick classifiers), **Duplicate** (copy with a new name) and **Delete**. You cannot delete a profile set that is assigned to any job . Profile sets can also be created through the API with two calls (see the ASDD profile set import and export page).

8.44 Domains and classifiers on the Settings screens

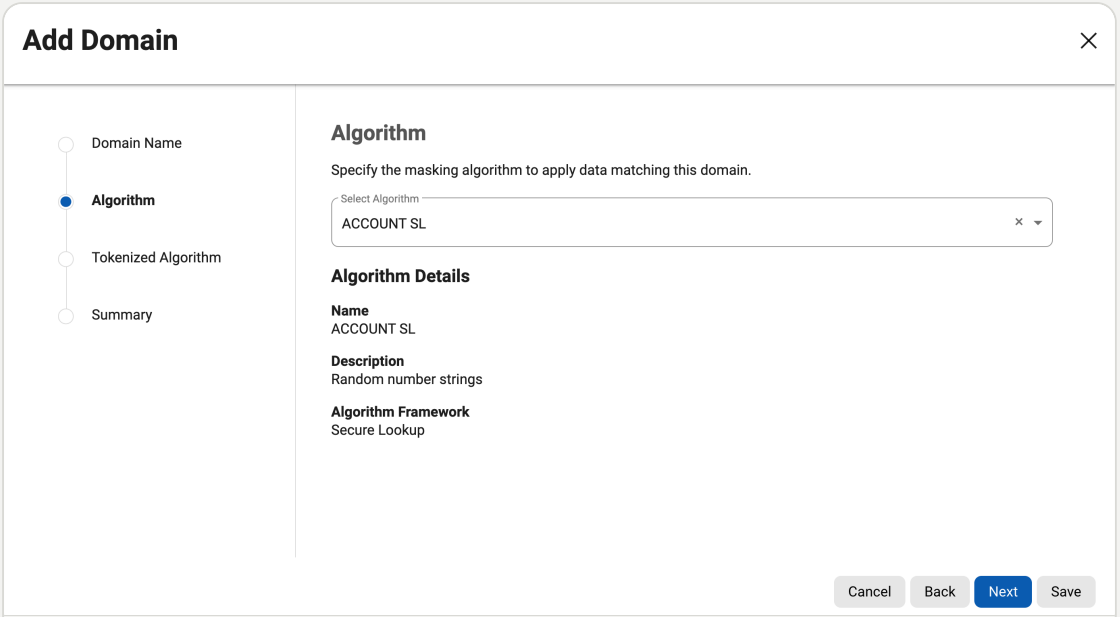
8.44.1 Domains

Settings > Domains lists each domain with its default *Masking Method* (algorithm) and, if defined, a *Tokenization Method*. The list can be sorted by Name, Masking Method and Tokenization Method .



Official screenshot: The Domains screen (engine 22.0.0.0, 69 domains on that test engine). On screen: for example ACCOUNT_NO maps to dlpx-core:CM Alpha-Numeric, ADDRESS to AddrLookup, AGE to dlpx-core:Age SL, BANK_ACCOUNT_NO to dlpx-core:CM Numeric and BLOOD_TYPE to dlpx-core:BloodType SL. These default pairings are what the profiler copies into an inventory.

`Source page ↗`



Official screenshot: The Add Domain wizard, Algorithm step. On screen: the left rail shows Domain Name, Algorithm, Tokenized Algorithm and Summary; "Select Algorithm" is set to ACCOUNT SL, with details panel (description "Random number strings", framework Secure Lookup). Whatever you pick here is what the profiler will copy into inventories for this domain.

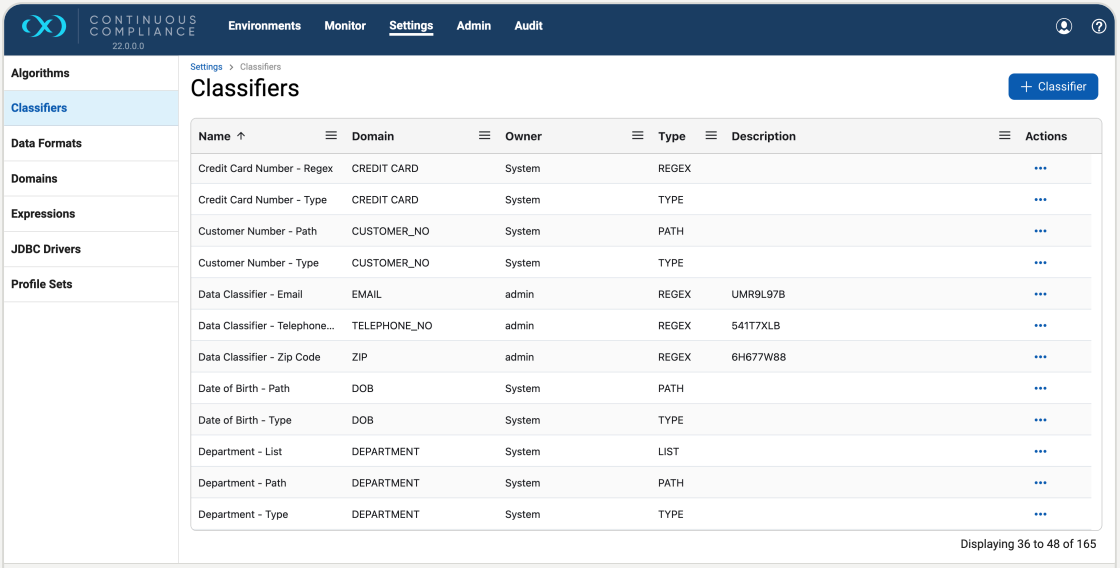
`Source page ↗`

To add a domain, click *+ Domain*, give it a unique name (this name will appear in the Domain drop-down on the inventory screen), pick a default masking algorithm, optionally a default tokenization algorithm, and save. If the environment's purpose is Tokenization/Re-Identify, the profiler assigns the domain's tokenization algorithm instead of its masking algorithm, so every domain used in such an environment should have one defined.

> **WHEN YOU WOULD ADD A DOMAIN**
>
> Suppose ShopSphere's compliance officer says "loyalty card numbers must be masked with a different algorithm than bank account numbers". The profiler can only assign one algorithm per domain, so you would create a LOYALTY_CARD domain with its own default algorithm and add a classifier that recognises such columns.

8.44.2 Classifiers

Settings > Classifiers lists every classifier instance with its Domain, Owner (System or a user), Type and Description. Sortable fields are Name, Domain and Description .

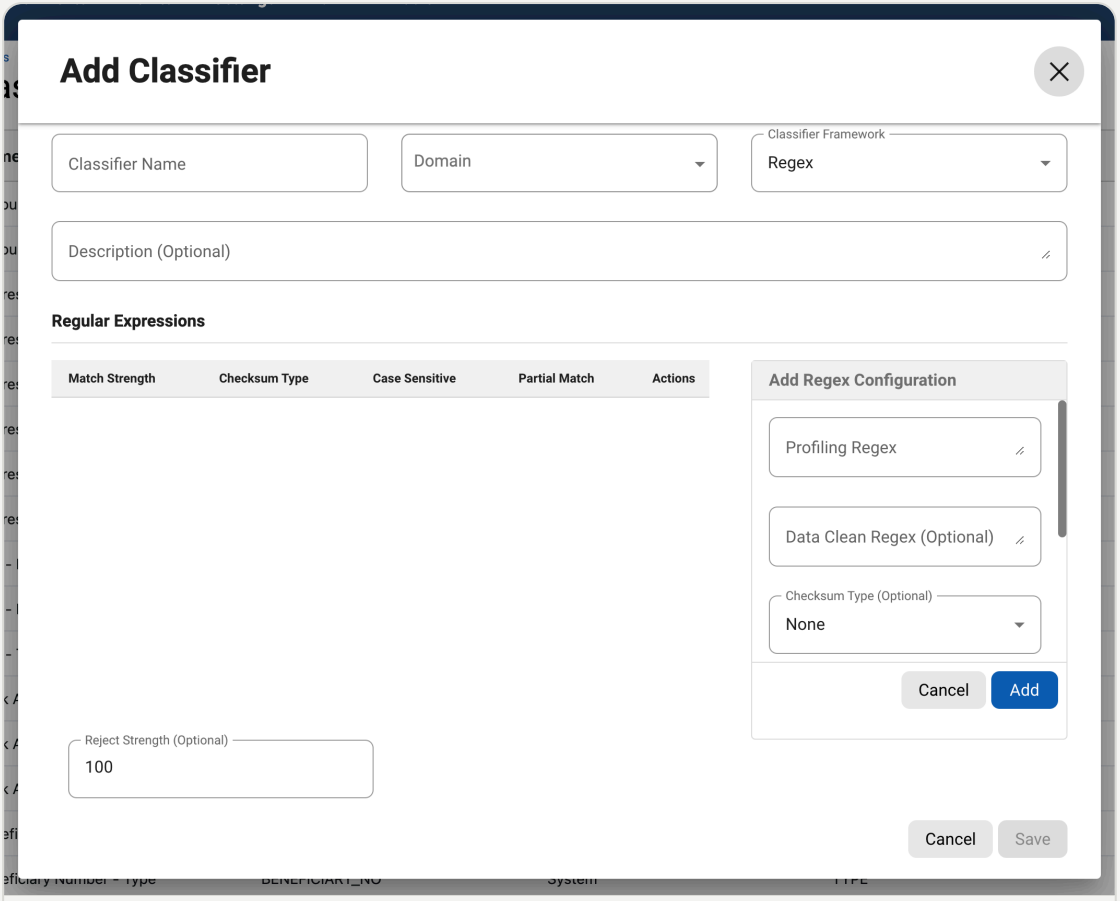


Official screenshot: The Classifiers screen (engine 22.0.0.0, "Displaying 36 to 48 of 165"). On screen: rows such as "Date of Birth - Path" (domain DOB, type PATH) and "Date of Birth - Type" (TYPE) owned by System, and "Data Classifier - Email" (REGEX) owned by admin.

`Source page ↗`

To add a classifier you choose a Domain, enter a name and description, choose a framework (the docs list Data Type, Regex, Path or List), then fill the one mandatory

field for that framework and click *Add*: a Profiling Regex for Regex, a Data Type for the type framework, an uploaded File for List, a Field Value for Path. Other fields take defaults. Name and framework cannot be changed after creation. Use the row menu to View, Edit or Delete.



Official screenshot: The Add Classifier dialog for the Regex framework. On screen: Classifier Name, Domain and Classifier Framework (Regex) across the top; a table with Match Strength, Checksum Type, Case Sensitive and Partial Match columns; an "Add Regex Configuration" panel with Profiling Regex, Data Clean Regex (optional) and Checksum Type (optional); and a Reject Strength (optional) box showing 100. Save stays greyed out until at least one regex row has been added.
`Source page ↗`

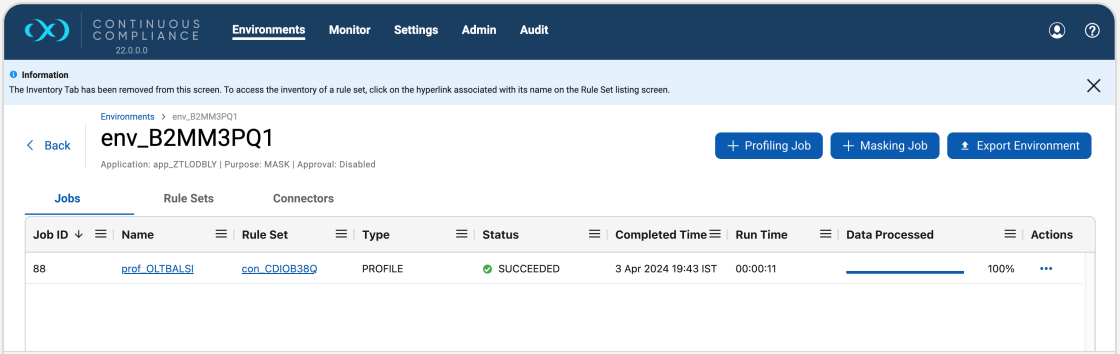
Two technical notes from the docs: PATH and REGEX use Java 8 regular expressions, with separate options for whole-input matching and case sensitivity, so you should not wrap patterns in `^(...)$`. List classifiers can "tokenize" input so that a full name is matched word by word against first-name and last-name lists, with the confidence being the sum of matching token strengths divided by the number of tokens.

8.45 Creating and managing a profiling job

Profiling jobs live inside an **environment**, next to masking jobs. Click an environment and you land on its Jobs tab. The documentation says profiling jobs can be created for relational databases (examples: Oracle, PostgreSQL) and for XML, JSON, mainframe, delimited and fixed-width rule sets. It is not supported for XML or JSON documents stored inside database columns, or for NoSQL databases .

Which jobs an environment offers depends on its purpose. A masking environment supports Profiling and Masking jobs. A tokenize environment supports Profiling, Tokenization and Re-Identification jobs . In both, the Jobs tab can be sorted and filtered by Job ID, Name, Rule Set, Type, Status and Completed Time, and *Export Environment* packs the environment's connectors, rule sets and jobs into a file (with an optional passphrase). Tokenization is explained in the algorithms and masking jobs chapter.

> **A DOCUMENTATION DISAGREEMENT TO BE AWARE OF**
>
> The profiling jobs page lists JSON rule sets as supported, while the inventory page says "Profiling is not supported on the JSON file rule sets". The text does not settle this. For delimited, fixed-width and database rule sets there is no conflict. If you must profile JSON, test it on your engine version.



Official screenshot: An environment's Jobs tab (engine 22.0.0.0). On screen: buttons "+ Profiling Job", "+ Masking Job" and "Export Environment" top right; tabs Jobs, Rule Sets and Connectors; one job of Type PROFILE with status SUCCEEDED, 100% data processed and a "..." Actions menu.

`Source page ↗`

8.45.1 Create a profiling job, step by step

1. **In the environment, open the Jobs tab and click *+ Profiling Job*.**

The Create Profile Job wizard opens with three steps: Details, Configuration, Summary.

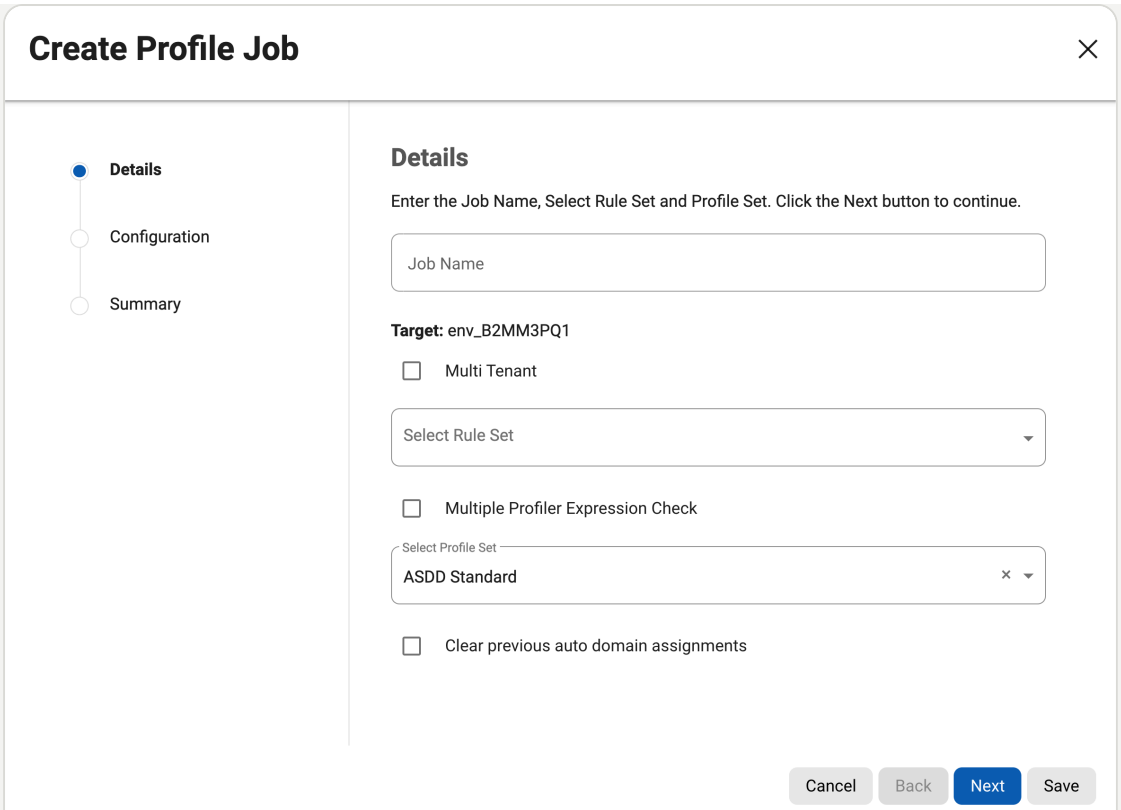
2. **Details: enter a unique Job Name and pick the Rule Set.**
The rule set is the scope of the scan. Tables not in it are never examined.

3. **Details: choose the Profile Set.**
The UI pre-selects the profile set named in `DefaultProfileSetName`, which is ASDD Standard in the screenshot below.

4. **Details: decide on the two checkboxes.**
Multiple Profiler Expression Check assigns a fallback algorithm when classifiers in different domains tie at the same confidence; the report will mention the multiple matches. *Clear previous auto domain assignments* wipes domains assigned by earlier runs for columns whose automatic updates are on, so the run starts fresh. A third box, *Multi Tenant*, lets one rule set be reused for identical schemas through different connectors, with the connector picked at run time.

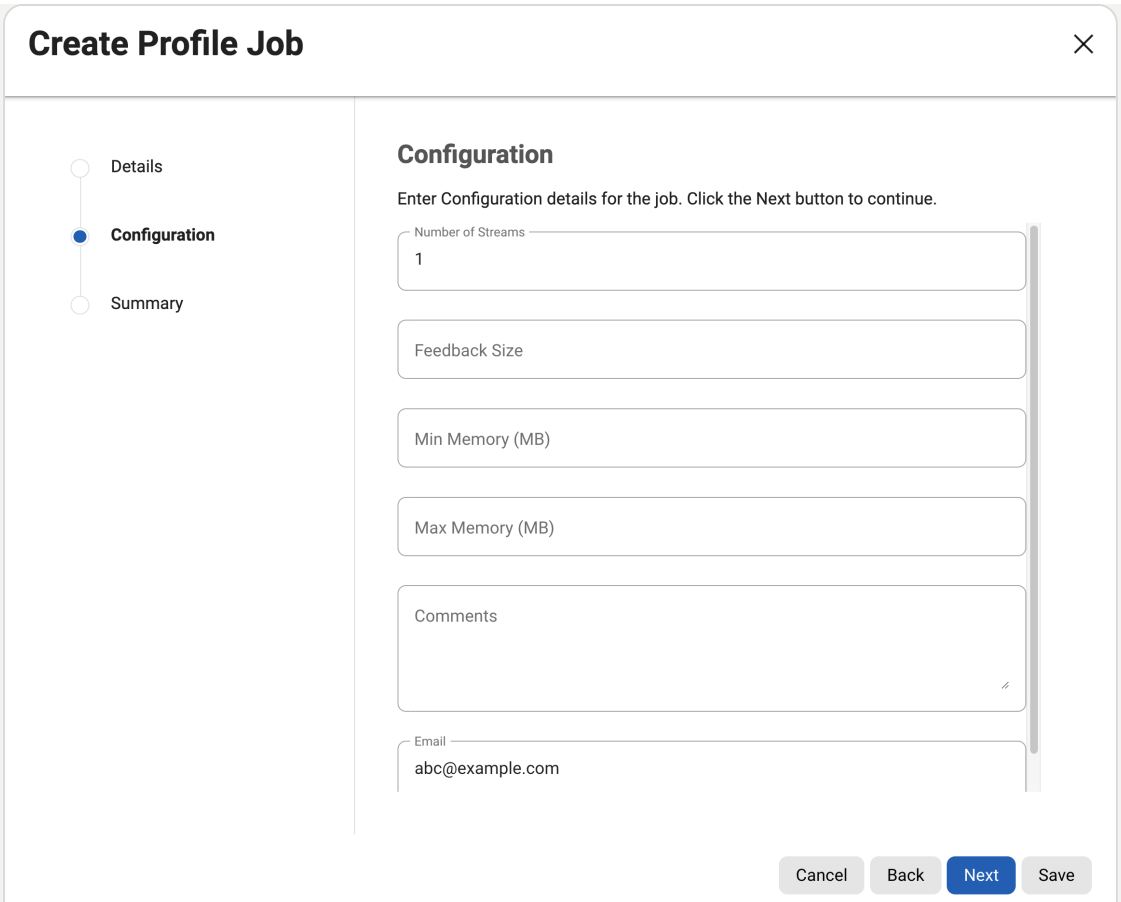
5. **Click Next. Configuration: set streams, memory and notes.**
See the table after the screenshots.

6. **Click Next to reach Summary, check it, and click *Save*.**
Expected result: The job appears in the Jobs list with no status until you run it.



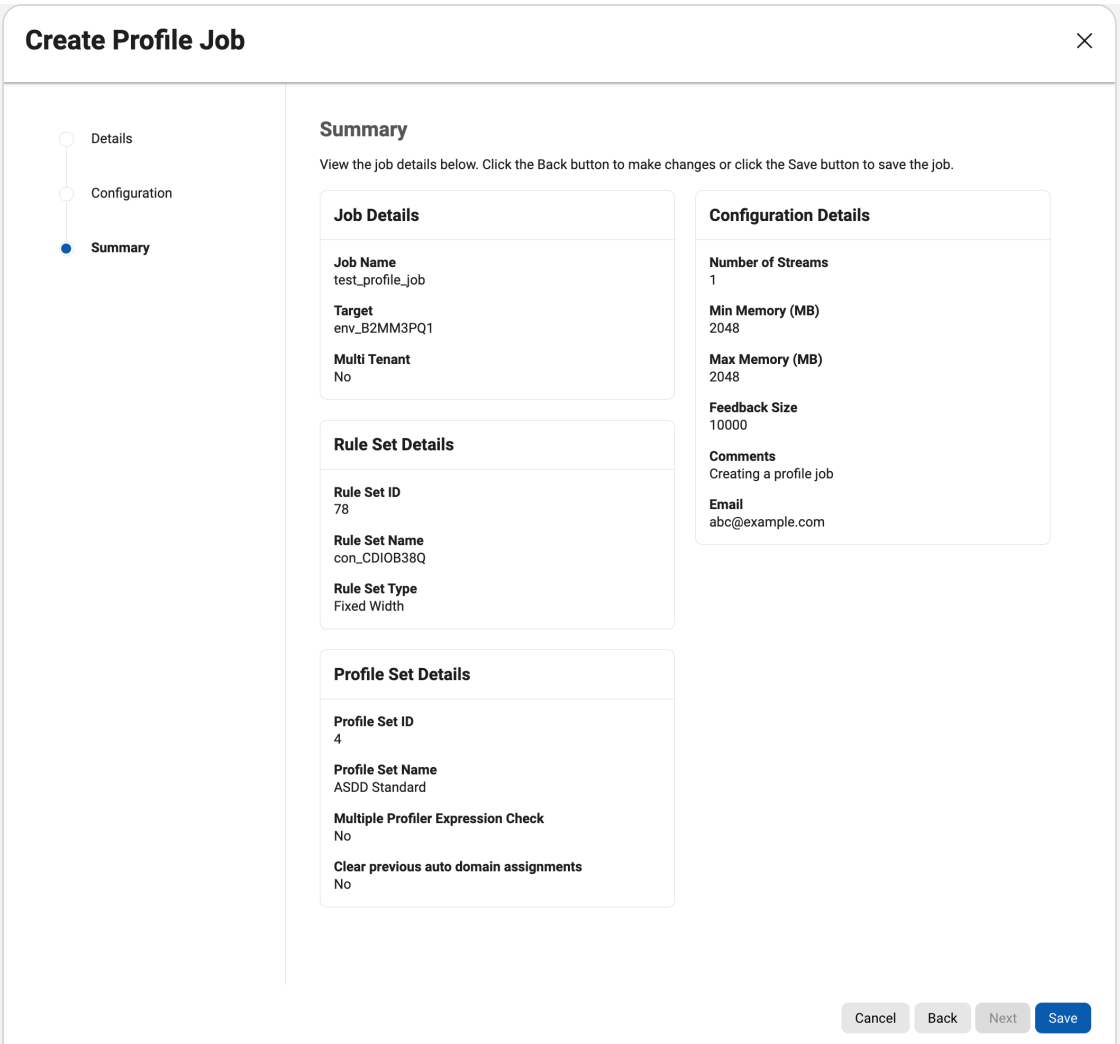
Official screenshot: Step 1, Details. On screen: Job Name, a "Target" environment label, Multi Tenant, Select Rule Set, Multiple Profiler Expression Check, a Select Profile Set drop-down showing ASDD Standard, and Clear previous auto domain assignments.

`Source page ↗`



Official screenshot: Step 2, Configuration. On screen: Number of Streams (showing 1), Feedback Size, Min Memory (MB), Max Memory (MB), Comments and Email (the example address is a placeholder).

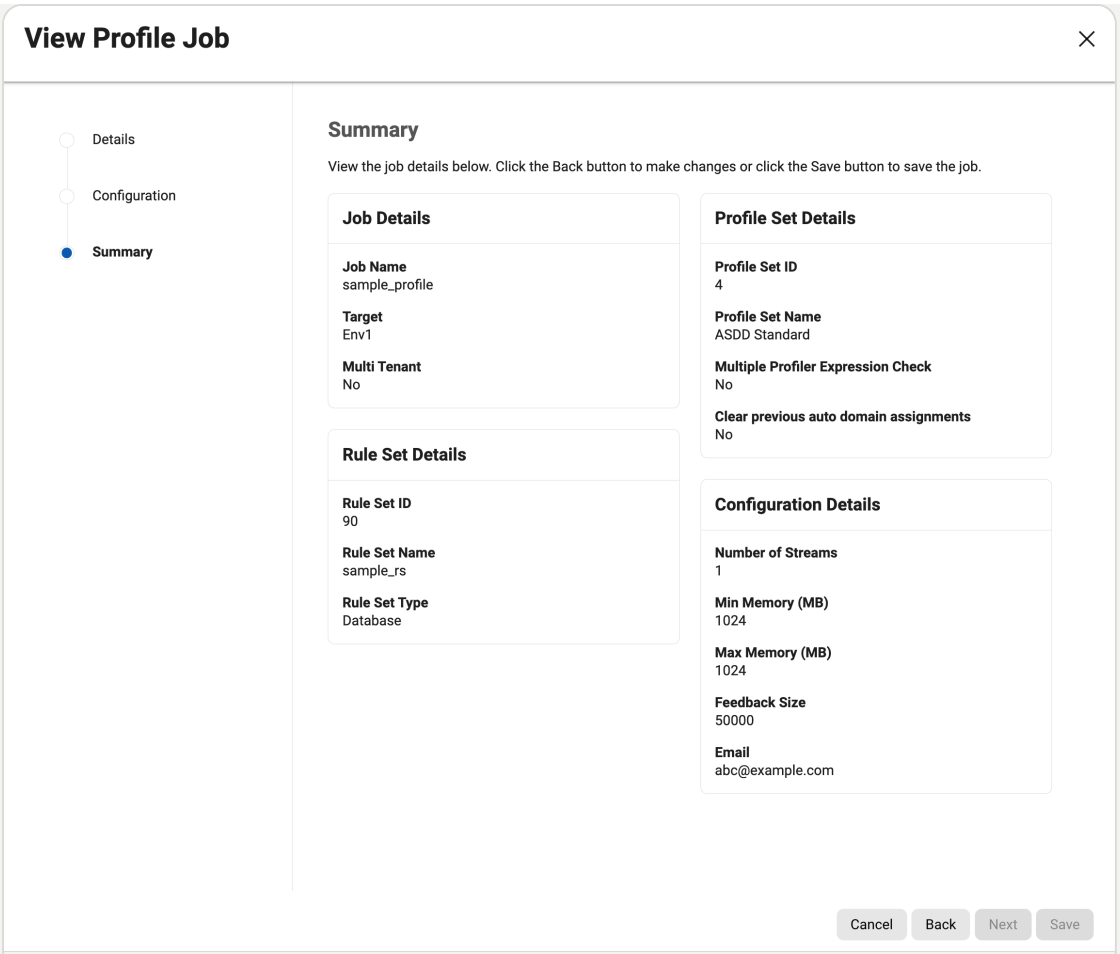
`Source page ↗`



Official screenshot: Step 3, Summary: job, rule set, profile set and configuration panels. The numbers (memory 2048, feedback size 10000) are the example's own, not recommendations.

`Source page ↗`

Field	Meaning	Notes from the docs
Number of Streams	How many tables are profiled in parallel	Default comes from an application setting (the screenshot shows 1)
Feedback Size (optional)	Rows processed between log messages	If set far above the real row count, progress only shows 0% or 100%
Min Memory / Max Memory (MB, optional)	Memory the job may use	With an ASDD profile set, Max Memory must be at least 1024 MB per stream (4 streams means 4096 or more)
Comments (optional)	Free text note	
Email (optional)	Addresses for status messages	Separate with commas



Official screenshot: The View action: the same Summary panels as creation, with Next and Save greyed out because nothing can be changed.

`Source page ↗`

The managing profiling jobs page calls the fallback setting `DefaultMultiphiAlgorithm`, while the Masking API client page calls it `DefaultMultipleProfilerExpressionAlgorithm` (default NullValueLookup). The second name is used here. The docs do not say how a tie is resolved when the option is off.

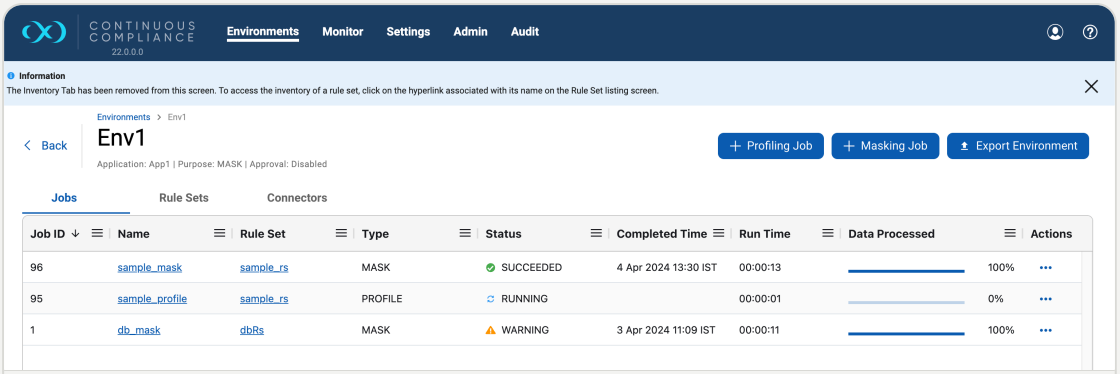
8.45.2 Edit, view and delete

Each job row has a **...** button in the Actions column. *Edit* opens the same wizard; the job name is locked after creation, and Edit is disabled if you lack permission. *View* opens the wizard on the Summary step. *Delete* asks for confirmation (click Confirm) and is also permission controlled .

8.46 Running a profiling job and reading the results

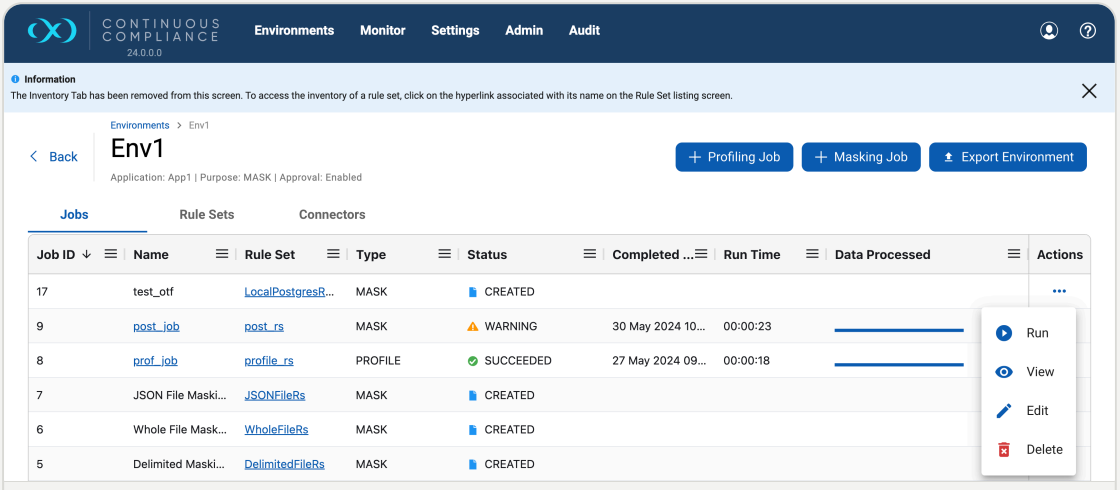
8.46.1 Run it

Click the **...** button on the job row and choose **Run**. Before it starts, the engine checks two resources: total memory allowed for all running jobs, and the number of jobs allowed to run at once (the application setting `NumSimulJobsAllowed`, default 7). If either limit would be exceeded, the job waits in a queue . You cannot edit or delete a job while it is running. To stop it, use *Cancel* and confirm. The docs' warning about semi-masked data is about cancelling a running *masking* job (see the algorithms and masking jobs chapter); a profiling job does not mask anything.



Official screenshot: Job statuses on the Jobs tab. On screen: job 95 of Type PROFILE shows RUNNING at 0%, job 96 MASK shows SUCCEEDED at 100%, and job 1 MASK shows a yellow WARNING at 100%. Each row has a "..." Actions menu.

Source page ↗



Official screenshot: The row Actions menu. On screen: on an environment Jobs tab (engine 24.0.0.0) the "..." menu of a job row is open with four items, Run, View, Edit and Delete. Other rows show statuses CREATED (never run), WARNING and SUCCEEDED; job 8 has Type PROFILE.

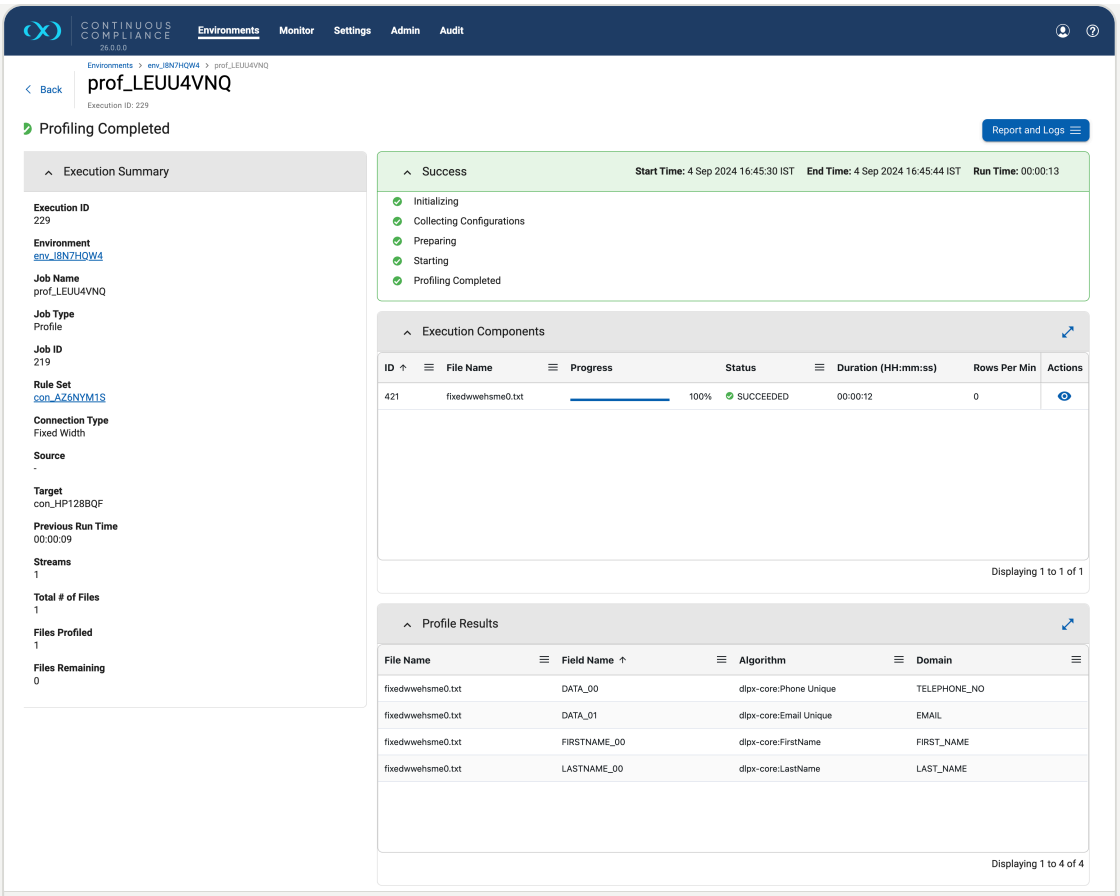
Source page ↗

8.46.2 Monitor it

Go to **Monitor > Job Executions**, or click the job name from anywhere. The list can be sorted and filtered by Environment, Job Name, Status, Submit Time, Start Time and Type. A job moves through these states :

Status	Meaning
Queued	Waiting for resources; events have not started
Running	In progress
Succeeded	Finished successfully
Warning (Non-Conformant)	Finished, but with a warning
Failed	An event in the execution had an error
Cancelled	A user cancelled it
Skipped	An event was skipped and the process moved on

Click the job name for details. The *Execution Summary* shows environment, job name, job type (Mask, Profile, Tokenize or Re-Identify), rule set, connection type, number of streams and, for profiling, files or tables profiled and remaining. The *Events* box lists stages such as Initializing, Collecting Configurations, Preparing, Starting and Profiling Completed. *Execution Components* shows each table or file with progress, duration and a View Logs action.



Official screenshot: A finished profiling job on a fixed-width file (engine 26.0.0.0). The Profile Results panel lists fields with domain and algorithm: TELEPHONE_NO, EMAIL, FIRST_NAME and LAST_NAME.

`Source page ↗`

The **Profile Results** panel (only for profiling jobs) is the quickest read of the findings, one row per table-column or file-field. Its grid supports sorting and filtering on every column.

8.46.3 Four ways to share the findings

1. **Job page, Profile Results panel.**
   Fast, but it lives in the UI.

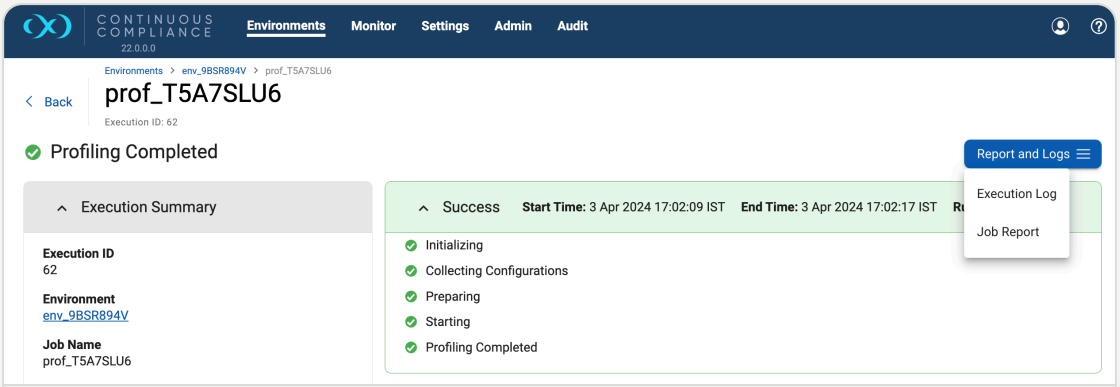
2. **PDF report.**
   On the job page click *Report and Logs* and choose *Job Report*. Good for sending to a compliance officer. *Execution Log* is in the same menu.

3. **Rule set inventory.**

Open the rule set. For databases, the assigned Algorithm shows the result; for file rule sets, the Domain and Algorithm per field.

4 **CSV round trip.**

On the rule set page click *Actions* > *Export Rule Set* for a CSV. Edit Is Masked, Algorithm or Domain in the spreadsheet, then use *Actions* > *Import Rule Set* to load it back. A reviewer without engine access can therefore correct findings in Excel.



Official screenshot: Where the PDF lives. On screen: "Profiling Completed", the Execution Summary, and the open "Report and Logs" menu with two entries, Execution Log and Job Report.

`Source page ↗`

Delphix

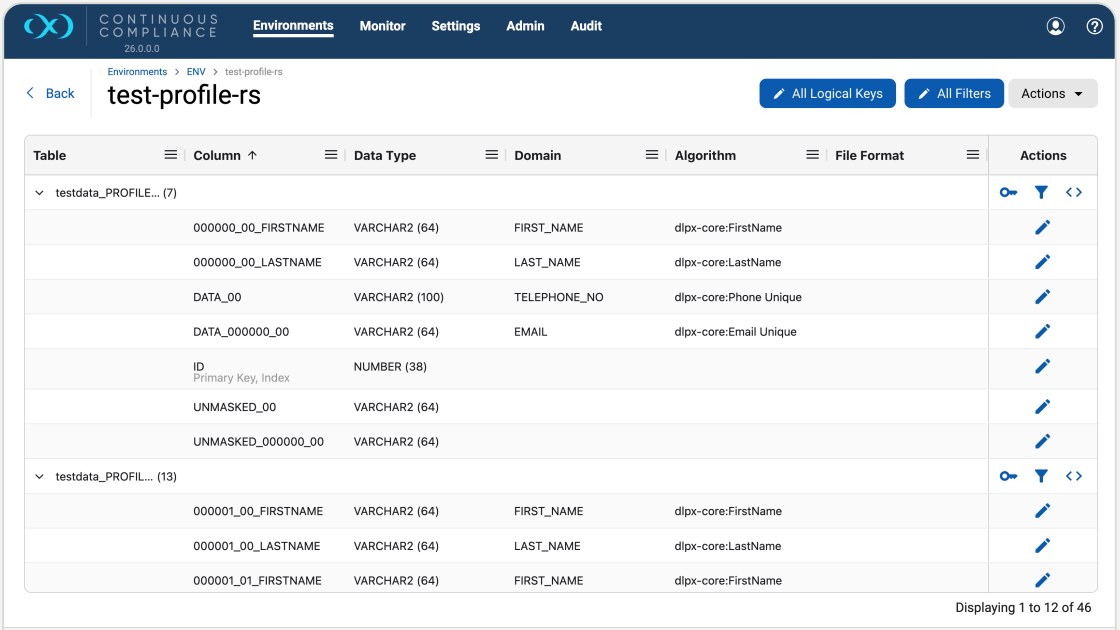
Profiling Report

Job Profile		Job Status	
Name :	prof_T5A7SLU6	Current Status :	SUCCEEDED
Type :	Profiling	Start Time :	2024-04-03 11:32:09.906
Environment :	env_9BSR894V	End Time :	2024-04-03 11:32:17.14
Schema :	DLPXDBORA		

Table	Column	Domain	Algorithm	Data Type	Confidence	Auto ID
testdata_PROFILE_000000	DATA_00	TELEPHONE_NO	dlpx-core:Phone Unique	VARCHAR2	80	true
testdata_PROFILE_000000	DATA_000000_00	EMAIL	dlpx-core:Email Unique	VARCHAR2	80	true
testdata_PROFILE_000001	DATA_00	TELEPHONE_NO	dlpx-core:Phone Unique	VARCHAR2	80	true
testdata_PROFILE_000001	DATA_01	TELEPHONE_NO	dlpx-core:Phone Unique	VARCHAR2	80	true
testdata_PROFILE_000001	DATA_000001_01	EMAIL	dlpx-core:Email Unique	VARCHAR2	80	true
testdata_PROFILE_000001	DATA_000001_00	EMAIL	dlpx-core:Email Unique	VARCHAR2	80	true
testdata_PROFILE_000002	DATA_000002_00	EMAIL	dlpx-core:Email Unique	VARCHAR2	80	true

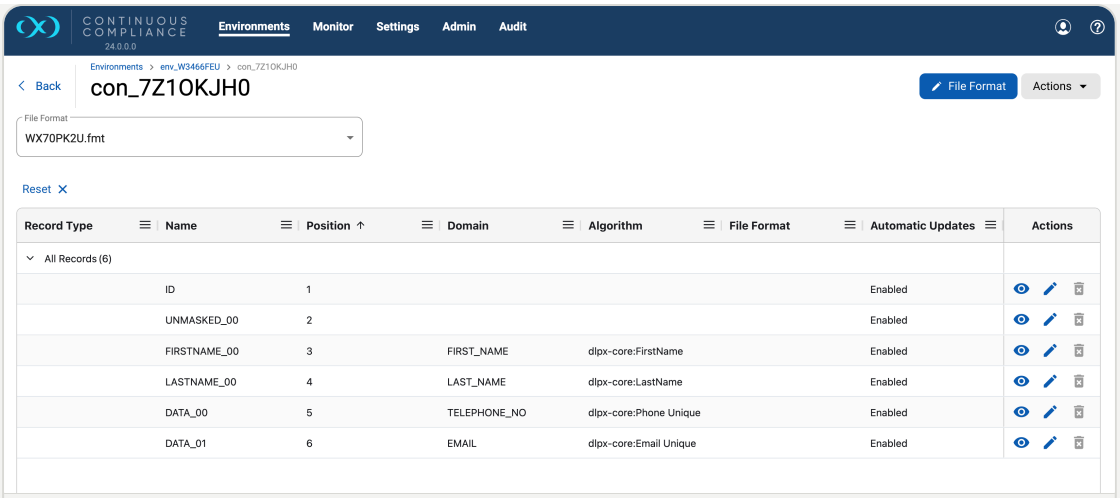
Official screenshot: The PDF Profiling Report: job and status boxes, then Table, Column, Domain, Algorithm, Data Type, Confidence and Auto ID. The visible rows are TELEPHONE_NO and EMAIL at confidence 80.

Source page ↗



Official screenshot: A database rule set after profiling (engine 26.0.0.0). Flagged columns have a domain and algorithm; ID (Primary Key, Index) and the UNMASKED_ columns have none.

Source page ↗



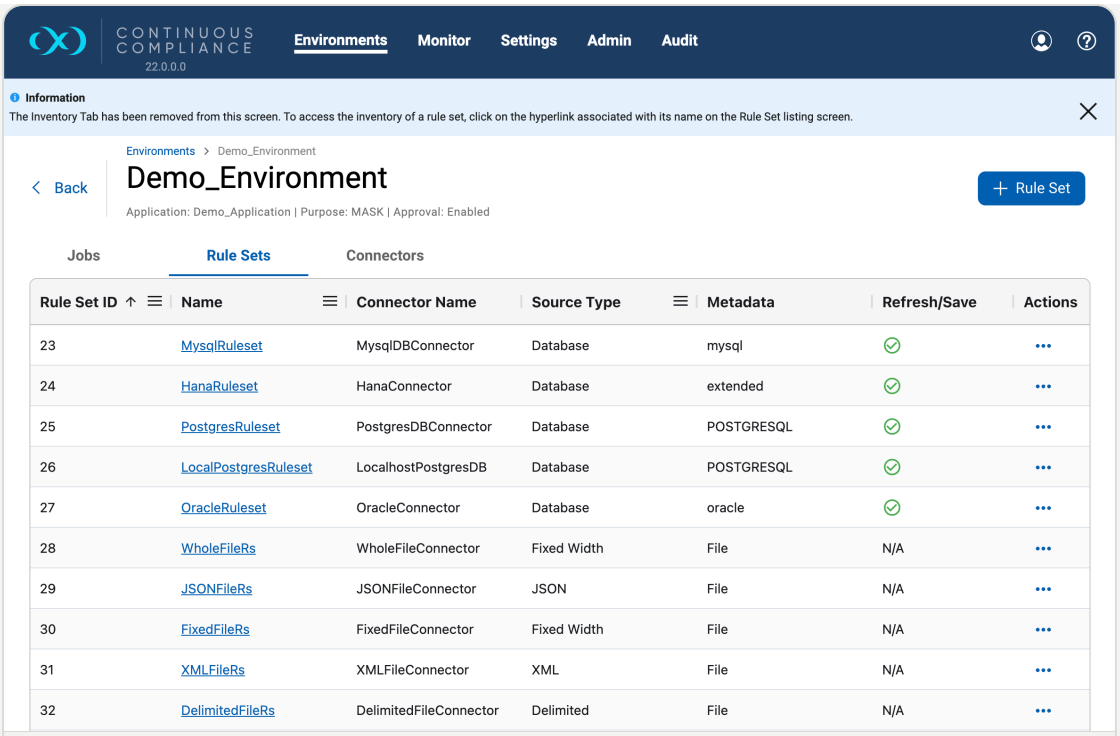
Official screenshot: A file rule set after profiling (engine 24.0.0.0): four of six fields have domains, and Automatic Updates is Enabled on every field.
`Source page ↗`

For database rule sets only, the API endpoint `/profileResultDatabase/{executionId}` returns the same results as JSON (column, table, domain, algorithm, data type, confidence). It does not work for other connection types . When profiling several mainframe data sets that share one format, ASDD averages results across the files.

> **READING THE CONFIDENCE NUMBER**
>
> In the PDF screenshot the confidence is 80, from classifiers that carry strengths like 0.67 and 0.7. The API example in the docs shows 100 for some columns, without explaining how that arises. Do not treat these as probabilities you could audit; the docs say the combining formula may change. Use them to sort your review: look first at low-confidence findings.

8.47 From findings to a reviewed inventory

The profiler writes its decisions into the rule set's *inventory*. Reaching it takes one click: **Environment > Rule Sets tab > click the rule set's name**. The "Inventory" tab that older versions had was removed, and the banner in the screenshots says so .



Official screenshot: The Rule Sets tab (engine 22.0.0.0). Rule set names are links; clicking one opens its inventory.

`Source page ↗`

A database inventory lists every column of every table. The number beside a table name is its column count. Primary key, foreign key and index flags appear under the column name, and data type and length are shown read-only. By default the visible columns are Table Name, Column Name, Data Type, Domain, Algorithm and File Format. For fixed-width and delimited files, fields are grouped under record types; see the file masking chapter.

8.47.1 Correcting one column: the Edit dialog

1. **Click the pencil (Edit) icon in the Actions column of the column's row.**
   The Edit Column (or Edit Field) dialog opens. From the inventory screen only the Masking section is editable.

2. **Decide about *Enable Automatic Updates*.**
   When ticked (the default), a profiling job may change this column's masking decision. When unticked, your choice wins over the profiler. If you are about to override the profiler, untick it first, otherwise the next profiling run may undo your work.

3 **Pick the *Domain* and the *Algorithm*.**

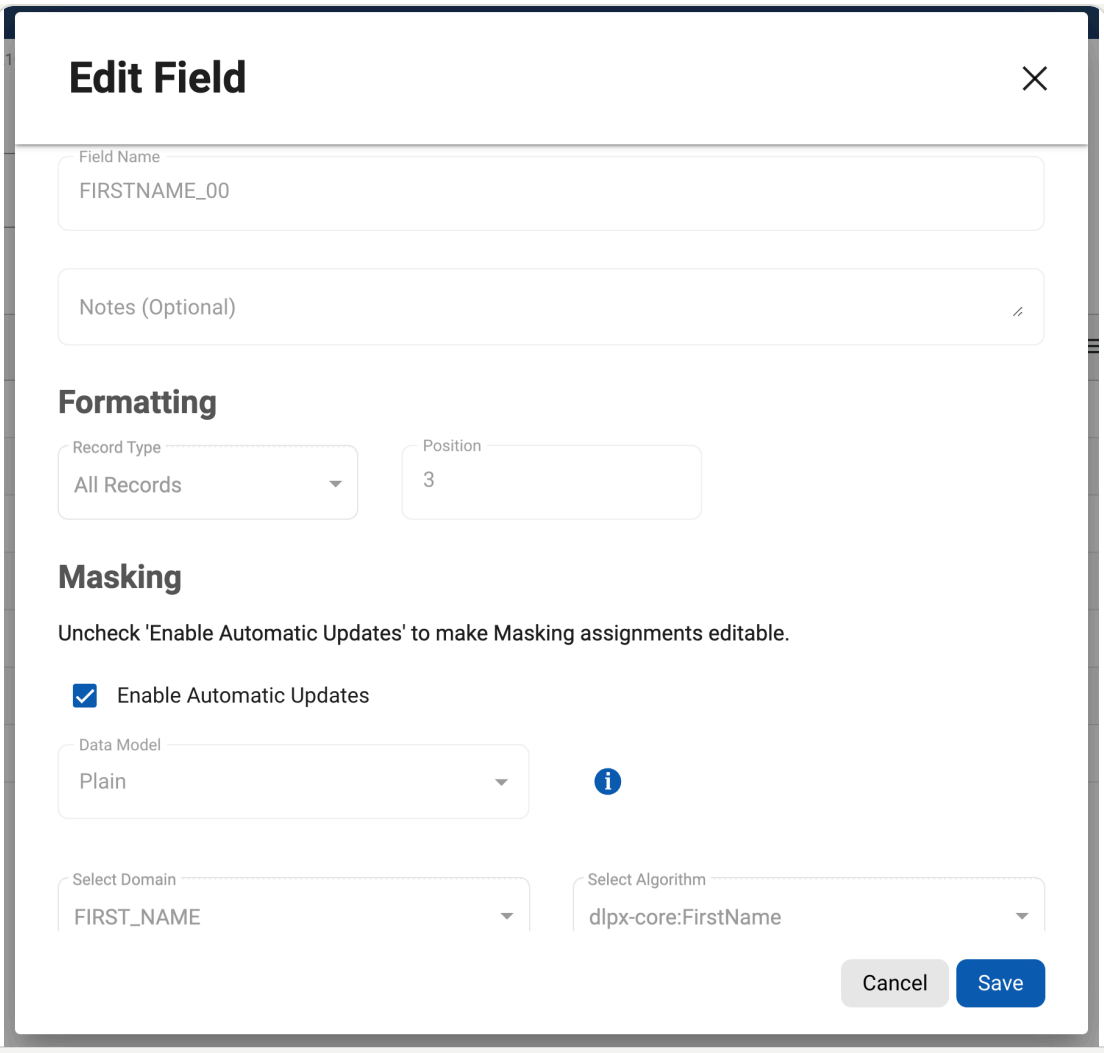
The Domain drop-down fills in that domain's default algorithm; you may replace it with any other algorithm.

4 **If you chose a DATESHIFT algorithm on a column that is not a native date or datetime, fill in the Date Format.**

The default is `yyyy-MM-dd`. The letters are case sensitive: `D` means day of the year, `d` day of the month, `MM` month, `yyyy` four-digit year.

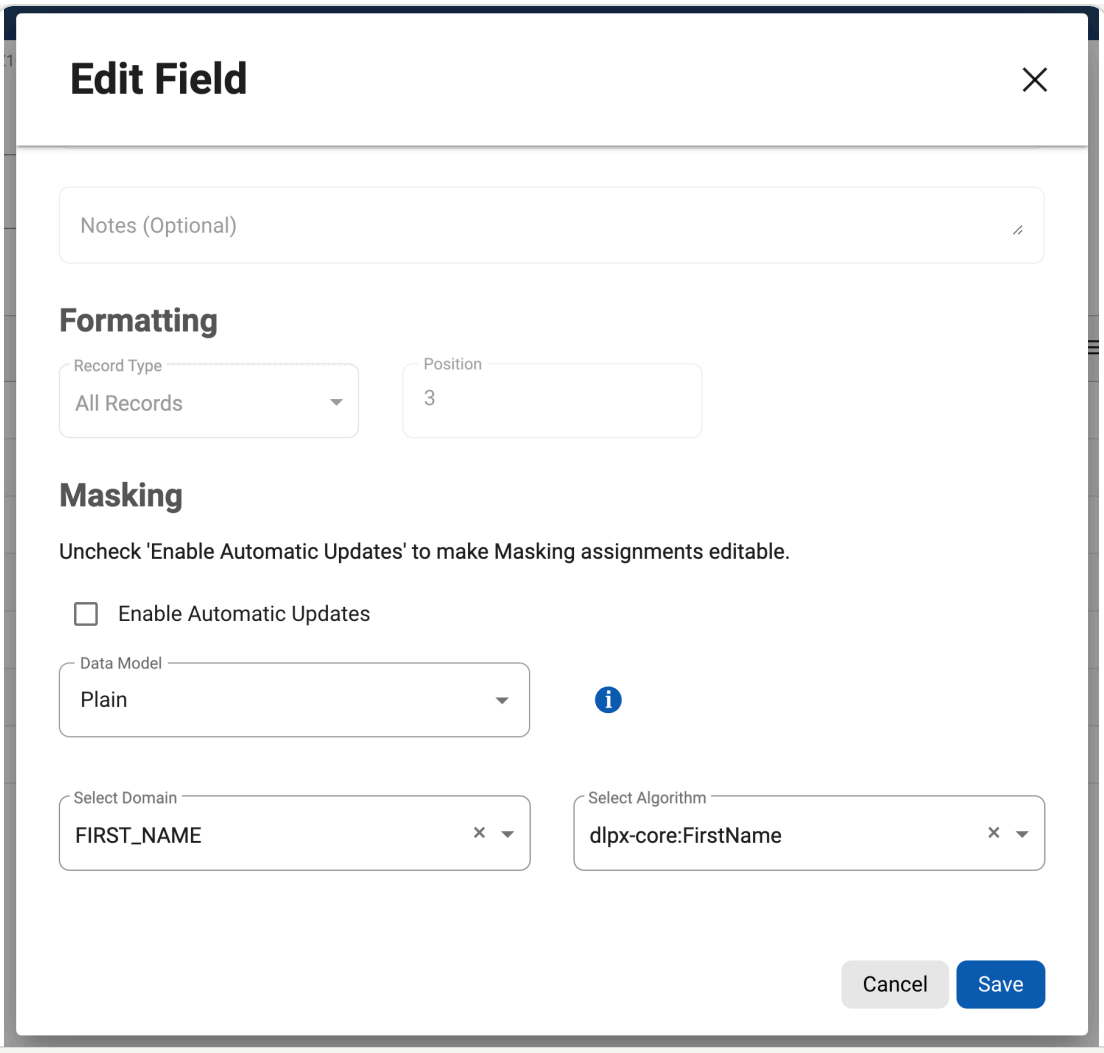
5 **Add a note if useful, then click *Save*.**

Expected result: Nothing takes effect until you save.



Official screenshot: Edit Field with automatic updates ON. On screen: field FIRSTNAME_00, Record Type "All Records", Position 3, "Enable Automatic Updates" ticked, Data Model Plain, domain FIRST_NAME, algorithm dlpx-core:FirstName. The instruction line reads "Uncheck 'Enable Automatic Updates' to make Masking assignments editable".

`Source page ↗`



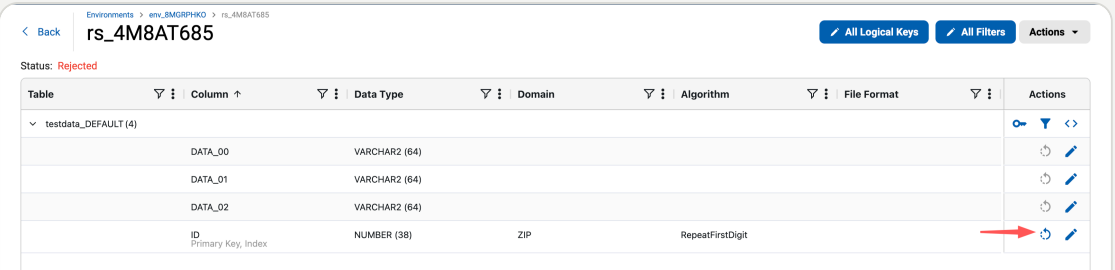
Official screenshot: The same dialog with automatic updates OFF. On screen: the checkbox is clear and the Select Domain and Select Algorithm boxes now show clear (x) and drop-down controls, i.e. they can be edited.

`Source page ↗`

Three more inventory tools worth knowing :

- **Clear Masking Assignment** (the circular-arrow icon in a row's Actions column, enabled only when that column or field has an assignment) removes the applied masking assignment and, in the same click, switches automatic updates off for it. A confirmation bar appears for a few seconds; if you do nothing, the clearing goes ahead, and Cancel keeps your assignment.
- **Nulling algorithms.** If you pick an algorithm that can return null (such as Null SL) for a column that has a NOT NULL constraint, the UI shows an error.
- **Inventory Approval Workflow** (database rule sets only, enabled per environment). Every edit sets the status to NEW; the editor clicks Submit; an Admin or a user with the Inventory Approval privilege clicks Approve or Reject. A

masking job for that rule set will not run unless the status is Approved. This gives a data steward a formal sign-off point.



Official screenshot: A database inventory under the approval workflow. On screen: the header shows "Status: Rejected" in red, buttons "All Logical Keys", "All Filters" and "Actions", and column rows with domain ZIP and algorithm RepeatFirstDigit on the ID column. The key, filter and code icons on the table row are for logical key, filter condition and custom SQL.

`Source page ↗`

Review every row in this order: is the domain right, is the algorithm right for the application (see the algorithms chapter), is confidence low (worth a second look, not necessarily wrong), and what is missing. Rows with no domain, such as UNMASKED_00 in the database screenshot, matter as much as flagged ones. Put the reason for each manual change in the Notes field for the audit trail.

Everything above is also available through the Masking API. The documentation's `createInventory` example assigns an algorithm and a domain to a column with `PUT /column-metadata/{id}` and a body such as `{"algorithmName": "AddrLine2Lookup", "domainName": "ADDRESS_LINE2"}` . The script needs helper files that are not shown, so it cannot be run as printed.

> **WHY THE CHECKBOX MATTERS IN TEAMS**
>
> A TDM engineer who fixes ten columns by hand, and a colleague who re-runs profiling the next day, are a common cause of "my changes disappeared". Untick automatic updates on every column you corrected deliberately, and let the profiler keep updating the rest.

8.48 ShopSphere: what should the profiler flag?

Predicting the profiler's answer from the classifier definitions makes a wrong result easier to spot. The ShopSphere tables are in the data basics chapter. Each column name was checked against the published ASDD Standard PATH regexes (the 21.0.0.0

listing) with an ordinary regular-expression engine. The real engine also applies type checks, data sampling and its own matching options, so read the table as a **prediction**, not as engine output.

PREDICTED ASDD STANDARD RESULT FOR SHOPSPHERE (PREDICTED, NOT ENGINE OUTPUT)

Table.column	Expected domain	Why (evidence)	Human check
CUSTOMERS.customer_id	none	The Customer Number PATH rule wants "number", "num", "nbr" or "no" after "customer"; `customer_id` has none of them	Primary key. Usually left alone or masked consistently as a key, see the rule sets and keys chapter
CUSTOMERS.first_name	FIRST_NAME	PATH `f(irst)?_?-??(na?me?)` matches; a first-name list also checks values	Confirm algorithm is `dlpx-core:FirstName`
CUSTOMERS.last_name	LAST_NAME	PATH for "last name" matches; last-name list checks values	Confirm `dlpx-core:LastName`
CUSTOMERS.email	EMAIL	PATH "email"; value pattern with strength 0.9	Check column length is not tiny; the EMAIL type rule lists a minimum length of 20
CUSTOMERS.phone	TELEPHONE_NO	PATH "phone". Plain 10-digit values like 9876543210 only reach the weakest value pattern (0.05), so this is mostly a name match	Check the algorithm suits 10-digit numbers

Table.column	Expected domain	Why (evidence)	Human check
CUSTOMERS.ssn	SSN	PATH ends with "ssn"; dashed values like 123-45-6789 fit the 0.7 pattern	Algorithm choice is in the algorithms chapter
CUSTOMERS.date_of_birth	DOB	PATH matches "date of birth"; type must be Date or a string of 6+ characters	Check which date algorithm and date format
CUSTOMERS.zip_code	ZIP	PATH "zip code" matches. The value patterns are US shaped (5 digits, or 5+4); ShopSphere's 6-digit PINs such as 560001 do not fit them	Confirm the ZIP assignment survived, and that the ZIP algorithm accepts 6 digits
CUSTOMERS.tax_id	TAX_ID	PATH "tax id"; type String or Number, 6+ long	Check how it handles letters and digits
ORDERS.total_amount	PAYMENT_AMOUNT (false alarm)	The Payment Amount PATH rule matches names ending in amount, amnt, amt, balance or price	Business decision: an order total is not personal data. Clear it
ORDER_ITEMS.unit_price, PRODUCTS.price	PAYMENT_AMOUNT (false alarm)	Name ends in "price"	Catalogue prices are public. Clear them

Table.column	Expected domain	Why (evidence)	Human check
ORDERS.order_date, ORDERS.customer_id, ORDER_ITEMS.*_id and quantity, PRODUCTS.product_code and product_name	none	No rule matches these names	Keep as is. Foreign keys must follow the key policy

UNCERTAIN CLAIM

The listing gives each TYPE classifier a `minimumLength` (for example 20 for email, 10 for first name). The documentation does not define it exactly; it is probably the declared length of the column. If so, a very short `first_name` column might be rejected. Do not take this as fact: run the job and look.

If the prediction holds, 8 of the 9 CUSTOMERS columns and 3 price or amount columns (11 in total) will carry a domain after the first run. Three of those are false alarms and one real column (`customer_id`) is deliberately left alone.

8.49 Lab: profile ShopSphere

Lab: Create a profiling job for ShopSphere, run it, and correct the inventory

Goal: scan the four ShopSphere tables with the ASDD Standard profile set, compare the result with the prediction table above, and fix what is wrong. This lab assumes you already created the environment, the connector and the rule set (all four tables) in the connectors chapter and the rule sets chapter. Names such as `ShopSphere_Profile_01` are examples, not Delphix defaults.

1. **Open your ShopSphere environment, Jobs tab, and click *+ Profiling Job*.**
   Profiling jobs are grouped with masking jobs inside the environment.
   Expected result: The Create Profile Job wizard opens at Details.

2 **Name it `ShopSphere_Profile_01`, pick the ShopSphere rule set, keep profile set *ASDD Standard*, and leave both checkboxes clear.**
The rule set limits the scan to your four tables. Leaving "Clear previous auto domain assignments" clear means nothing from older runs is wiped.

3 **On Configuration, keep 1 stream and set Max Memory to at least 1024 MB.**
The docs require 1024 MB per stream for ASDD profile sets. Add your own email address if SMTP is configured.

Expected result: Summary shows ASDD Standard and your memory values.

4 **Save, then ... > *Run*.**
The engine may queue the job if too many are running.

Expected result: Status goes Queued or Running, then Succeeded. Warning means it finished with caveats; open the logs.

5 **Click the job name and read *Profile Results*.**
Write down every table, column, domain and algorithm. Count the rows.

6 **Compare with the prediction table.**
Mark each column as: as predicted, extra (profiler flagged more), or missing (profiler flagged less). Explain any difference using classifier evidence, for example "column name matched PATH but the type was rejected".

7 **Open the rule set (Rule Sets tab, click its name) and fix the inventory.**
For the three price or amount columns: click the Clear Masking Assignment icon on each row (it removes the assignment and switches automatic updates off), and confirm the bar. For each real finding, confirm the algorithm. For `customer_id` and the foreign keys, apply the key policy from the rule sets and keys chapter. For any column the profiler missed that you know is sensitive, assign domain and algorithm by hand and untick automatic updates.

Expected result: Only truly sensitive columns carry an algorithm.

8 **Export the rule set to CSV (*Actions > Export Rule Set*).**
This is your evidence file. Open it in a spreadsheet and check that the Is Masked, Algorithm and Domains values match your intent.

9 **Re-run the profiling job and check the inventory again.**
Expected result: The columns you corrected with automatic updates off should keep your choices. If they changed, the checkbox was still ticked.

10 **Download the PDF (*Report and Logs > Job Report*) and keep it with the CSV.**
These two files are what you would hand to a data steward.

▶ **Show a worked answer**

8.50 Common mistakes and gotchas

WATCH OUT FOR THESE

- **Treating profiling as the finish line.** The output is a draft. Review every flagged and every unflagged column, especially free-text and oddly named ones.
- **Forgetting that only the rule set is scanned.** A table added to the database after the rule set was built is invisible to profiling.
- **Editing assignments and leaving automatic updates ticked.** The next profiling run can replace your edits. Untick it on columns you decided by hand.
- **Changing a domain's default algorithm and expecting old inventories to change.** They do not; only future profiling runs are affected.
- **Max Memory below 1024 MB per stream with an ASDD profile set.** The docs say the job needs at least that. Four streams need 4096 MB or more.
- **Trusting name matching alone.** A column called `ssn_present` is a flag, not an SSN; TYPE classifiers exist to reject these, but they only help when the data type is right.
- **Using legacy profile sets for new work.** They are deprecated and give less coverage.
- **Expecting value-level detection in languages other than English.** The built-in lists are English only.
- **Sharing the PDF report or CSV without thinking.** They list table and column names with their domains, which tells a reader exactly where your sensitive data lives, so store them with care.

8.51 Recap

KEY POINTS

- Profiling scans a rule set with a profile set and writes domain and algorithm assignments into the inventory.
- Classifiers come in four frameworks: PATH and TYPE look at columns; REGEX and LIST look at values. Scores combine into a confidence compared with the assignment threshold (default 1).
- A domain is a kind of sensitive data with one default algorithm. Changing it affects only future runs.
- Create the job in an environment: rule set, profile set (ASDD Standard by default), streams, memory (at least 1024 MB per stream for ASDD).
- Read results on the job page, in the PDF, in the rule set inventory or in an exported CSV.
- Correct findings in the Edit dialog and untick Enable Automatic Updates so that the profiler does not overwrite you.
- For ShopSphere, expect the customer columns to be flagged and price or amount columns to be false alarms.